Proceedings of the 9th Ph. D. Retreat of the HPI Research School on Service-oriented Systems Engineering

Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch, Tobias Friedrich (Hrsg.)

Technische Berichte Nr. 100

des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 100

Christoph Meinel | Hasso Plattner | Jürgen Döllner | Mathias Weske |
Andreas Polze | Robert Hirschfeld | Felix Naumann | Holger Giese |
Patrick Baudisch | Tobias Friedrich (Hrsg.)

Proceedings of the 9th Ph. D. Retreat of the HPI Research School on Service-oriented Systems Engineering

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2016
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URN urn:nbn:de:kobv:517-opus4-83347
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-83347

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-345-9

Contents

Preface

This is the 100th technical report of the Hasso Plattner Institute (HPI) and the proceedings of the 2015 edition of the annual retreat of HPI's research school on "Service-Oriented Systems Engineering". Within the past 10 years, more than 100 Ph. D. students have graduated from HPI, with more than 30 graduates from the HPI research school among them.

The topic "Service-Oriented Systems Engineering" serves as an underlying theme for many dissertation projects at HPI. Design and implementation of service-oriented architectures present numerous research questions from the fields of software engineering, system analysis and modeling, adaptability, and application integration. Service-oriented systems engineering represents a symbiosis of best practices in object orientation, component-based development, distributed computing, and business process management. It allows for integration of business and IT concerns and denotes an important research topic with high potential in academic research and industrial application.

Looking back at a history of 10 years, the annual retreat of HPI's research school offers to all of its members the opportunity to present their ongoing research. Due to the interdisciplinary structure of the research school, this technical report covers a wide range of topics. These include but are not limited to: Human Computer Interaction and Computer Vision as Service; Service-oriented Geovisualization Systems; Algorithm Engineering for Service-oriented Systems; Modeling and Verification of Self-adaptive Service-oriented Systems; Tools and Methods for Software Engineering in Service-oriented Systems; Security Engineering of Service-based IT Systems; Service-oriented Information Systems; Evolutionary Transition of Enterprise Applications to Service Orientation; Operating System Abstractions for Service-oriented Computing; and Services Specification, Composition, and Enactment.

Prof. Dr. Andreas Polze, Speaker HPI Research School
Prof. Dr. Robert Hirschfeld, Coordinator HPI Research School

Towards Analysis of Public Social Data to Improve Situational Awareness

Aragats Amirkhanyan

Internet Technologies and Systems
Hasso-Plattner-Institut
Aragats.Amirkhanyan@hpi.de

Nowadays, social networks are essential parts of modern life. People posts everything what happens with them and what happens around them. Such data are widely used for analyzing and detecting global events and trends in social networks. But since the amount of data producing by social networks increases dramatically every year, and users start to post geo-tagged messages more often, we are getting a possibility to detect more local and geo-spatial events.

In the report, I provide my view of how to use publicly available data to detect geo-spatial events. Especially, I am interested in detecting crime events, and by this way, improve situational awareness. In this report, I provide my progress of doing research in this area including a description of methods for collecting data, processing data, advanced searching and a description of the web application that is aimed to provide a possibility to investigate collected geodata.

1 Introduction

Nowadays, Social Media services produce a huge amount of public data. And researchers are very interested in analyzing such data for different purposes. And many research papers of past years confirm that public data can be very valuable in different scenarios. These data become more valuable when they are mapped to the online geographic map that gives us wide range of possibilities for analyzing.

Sakaki et al. [12] use Twitter users as social sensors to detect earthquake shakes. They investigated the real-time interaction of events, such as earthquakes, in Twitter and proposed an algorithm to monitor tweets and to detect a earthquake target event.

Another use-case of using public data to detect threats was presented by De Longueville et al. [3]. In the paper, they showed how location-based social networks (LBSN) can be used as a reliable source of spatio-temporal information, by analyzing the temporal, spatial and social dynamics of Twitter activity during a major forest fire event in the South of France in July 2009.

Also, there are many papers that are aimed to use location-based social networks to understand user behavior around some area or to explore the area around in order to improve situational awareness [9, 11, 13, 15, 16].

Social networks are very reflective to global events, such as earthquakes or forest fires, but the amount of data, produced by social media services, increases every year dramatically, and it gives us a possibility to use these data to detect also local events. In 2013, Walther et al. [14] presented the paper, where they have shown how to detect local geo-spatial events in the twitter stream. According to the authors, they are interested in detecting local events, such as house fires, on-going baseball games, bomb threats, parties, traffic jams, Broadway premiers, conferences, gatherings and demonstrations in the area they monitor.

The idea of detecting the local geo-spatial events is quite interesting. And with the trend of increasing the popularity of location-based social networks and geo-tagged content, we can suppose that in near future all events happening around will be reflected in location-based social networks and blogs. Therefore, I am interested in using the idea of detecting local geo-spatial events as a basement and go further, and I would like to use publicly available data to detect geo-spatial crime events [6, 10]. Now, I am going in this direction and this report presents the current state of the research project.

The remainder of the paper is organized as follows: section 2 reflects the main part of the report. This section contains three subsections. Section 2.1 provides information about data, which I collect and use. In subsection 2.2, I describe the overview of the developed web application that is used for displaying, searching and analyzing data. This overview includes the overview of the architecture and the overview of the user interface. Section 2.3 briefly describes the method of geo-clustering since it is a quite important part of data processing for displaying and analyzing. I conclude the report and provide future directions of research in section 5.

2 Public Data Analyzing

2.1 Data

Data is an essential part of research. Therefore, initially, to prove the concept I decided to use public data from Twitter. It is a quite reasonable decision because, nowadays, Twitter is something more than just a social network. Twitter is the source of world news [8] and it is the place, where breaking news about important events appear firstly. Therefore, I can suppose that if something important happens around some area then tweets about it will almost immediately appear in Twitter.

Twitter provides a quite powerful API to fetch required data. The only way to access 100 % of all tweets in real-time is through the Twitter “Firehose”, but the access to Twitter “Firehose” is very expensive. The other option for accessing tweets is using one of Twitter’s direct API offerings [2]. The main disadvantage of using free API is that Twitter does not guarantee that you will get all requested data. But studies have estimated that using Twitter’s Streaming API users can expect to receive anywhere from 1 % of the tweets to over 40 % of tweets in near real-time [2]. Also, you can increase the percentage of receiving tweets by applying more strict criteria. The criteria can be keywords, usernames, locations, named places, etc. For example, if you ask Twitter to provide only geo-tagged tweets from some concrete city then with the high level of likelihood you will get almost all geo-tagged tweets from the specified area. For my experiments, I use the area around London. Firstly, because London is one of the most active Twitter cities [4] and, secondly, because the main language of tweets posting in London is English.

2.2 Application

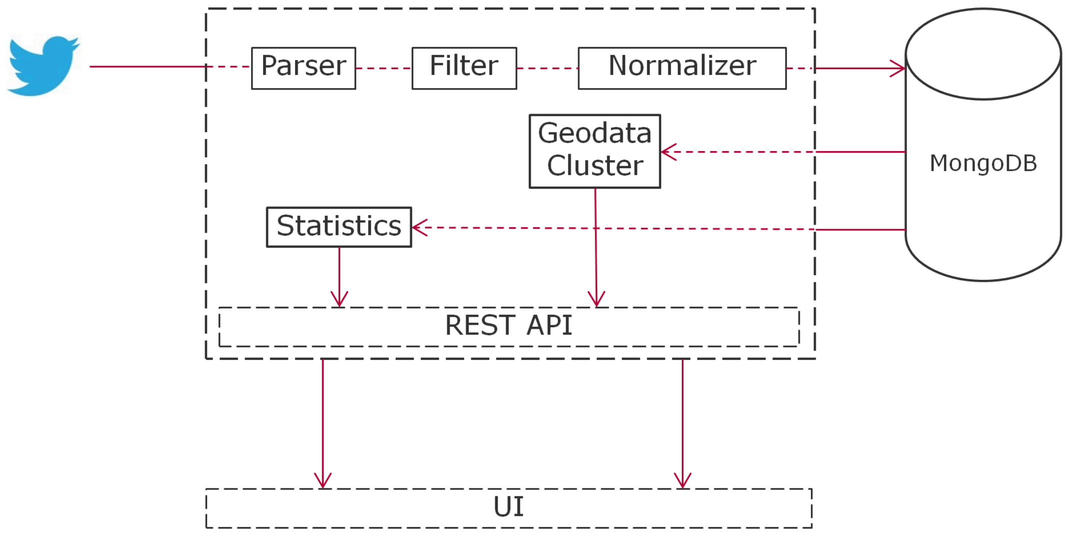


Figure 1: Application architecture

Once we collected data, we need a tool to work with data. Therefore, I developed a web application to provide methods for displaying, searching and analyzing collected data. The application is written in the Java programming language with some additional frameworks. To store geodata, I used MongoDB, because this database provides geo-spatial index and suitable to store geodata.

You can find the overview of the application architecture in Figure 1. The architecture is quite simple. The application connects to the Twitter stream and receives all tweets in real-time, which meet subscribed criteria. Tweets are presented in the JSON format, therefore, firstly, application parses new incoming tweets, then it filters them according to the predefined rules (for example, we would like to accept only English tweets). After that, we normalize the tweet messages and build the database entity *Post* that contains all needed fields for further analysis.

Application has the full access to the database and it provides the access to data by REST API. User can request data according to the search criteria, which could be dates, location, keywords and others. Once the user requested data, the system retrieves data from the database and then passes them to the *Cluster* and *Statistics* modules. The first one clusters geodata based on their location and the zoom parameter (the zoom level of the online map). By other words, the module groups close posts and present them as one post with some weight that reflects the number of grouped posts. Meanwhile, the second module calculates some statistics, such as the intensity of posts, the most popular languages and hashtags. The processed result is returned to the user interface (UI) through REST API, and after that, the user can see the result and analyze it manually.

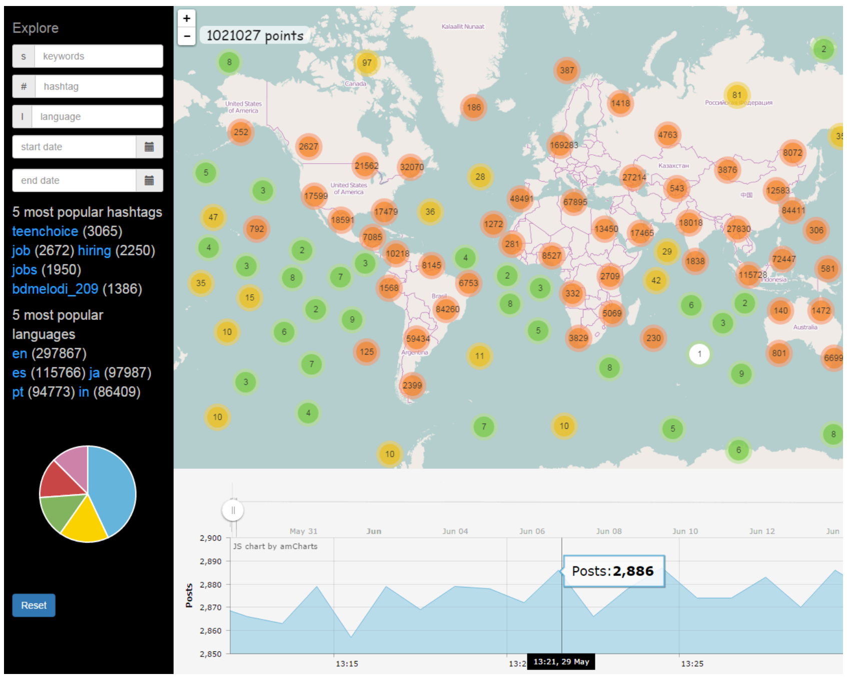


Figure 2: User interface of the web application

The screenshot of the user interface you can find in Figure 2. As you can see from this screenshot, the application provides the possibilities to cluster and display tweets on the OpenStreetMap. Also, application provides some basic statistics, such as the five most popular languages, the five most popular hashtags (on the left sidebar) and the intensity of posts per minute (timeline under the map). Also, on the left sidebar, you can apply different search criteria to find and analyze particular dataset. For example, you can apply full text search by keywords and search by hashtag and language.

The main goal of providing different statistics, displaying posts on the online map and advanced methods for searching is to use them to detect local threats, especially, crime threats. So far, it is possible only manually to monitor the area by applying keywords (for example, bomb, crime, fire and so on) to be aware about the public safety.

2.3 Clustering

In this section, I briefly show how geo clustering works. It is the important part of the research work and the report, because displaying huge amount of data on the online map is a challenge. The proposed approach is fully presented in the paper [1], which is accepted by the IIT'15 conference. But in this report, I provide only brief description of the approach.

The proposed approach is the grid and density-based geo clustering that clusters data in two steps: (1) geohash preprocessing and (2) real-time density-based geo clustering. The first step is quite simple and can work piece-by-piece of data and it is based on geohash function. Geohash is a latitude/longitude geocode system invented by Gustavo Niemeyer [5]. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape [5]. Geohash is the from 1 up to 12 characters value and it means that you can specify how accurate you want to calculate the hash value and with that how accurate you want to determine the location.

In the first step of geo clustering, we take coordinates of the points, calculate the geohash values with the specified precision and group them according to the same geohash values. So, as a result, we have some filled hash table. The key point is which level of the precision to use. And it depends on the zoom level, but the idea is that we do not need fully delegate clustering to the geohash function because clustering algorithms based on the geohash function (the grid-based clustering) suffer from bad clustering of points, that are located near to the border of grids [1]. Therefore, these recommended values of the precision come from experiments and your desire of accuracy. Some recommendations you can find the paper [1]. For example, if we want to cluster geodata with a zoom level equaled 5 then probably the best precision will be 3. Using these recommendations and applying geohash preprocessing, we reduce the number of points by grouping very close points, which will be grouped into the one cluster in any clustering algorithm.

The next step of clustering is the density-based geo clustering. Proposed density-based algorithm can process its input data piece-by-piece in a serial mode and it does not need the entire data to start the process. We can pass points by batches and have the result at any time. Just one assumption is that, in order to cluster new batch of points, the algorithm should know about existing clusters.

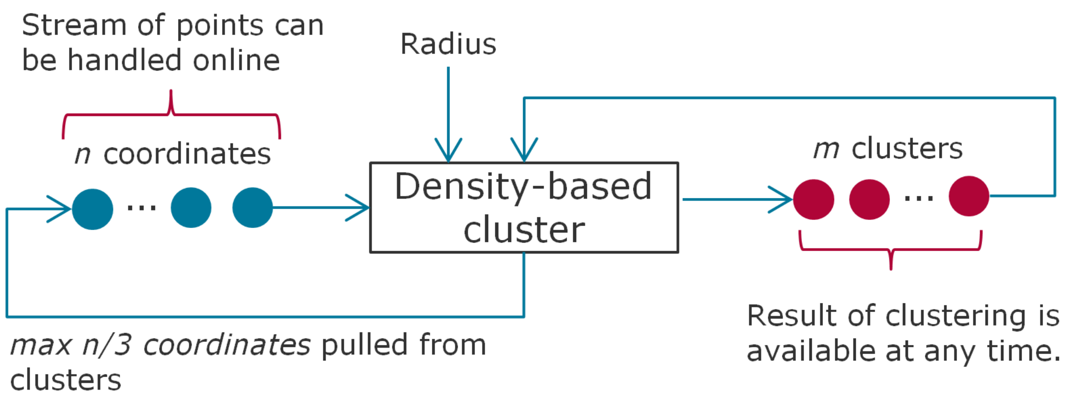

Stream of points can be handled online
Radius
n coordinates
Density-based cluster
m clusters
Result of clustering is available at any time.
max n/3 coordinates pulled from clusters

Figure 3: Density-based geo clustering

Figure 3 shows the simplified view of the density-based geo clustering. The algorithm produces the cluster with the center in the most densest area. The workflow is quite simple. For every incoming point, we try to find the closest cluster among the existing clusters based on the radius of the cluster, which we must specify as the input parameter. If we can not find the closest cluster, we need to create new one that contains this considering point. So, the center of the new cluster is the coordinates of the point. But if we found the closest cluster, we need to attach the point to the cluster and then recalculate the center of the cluster. The center of the cluster is specified algorithmically through iteration of the existing points and the center is located in the densest area [7]. After recalculating the center of the cluster, it could be that some points (maximum n/3 points) become out of the range of the cluster. In this case, we need to pull such points from the cluster and pass again to input of the algorithm. It is showed by the arrow from clusters to the *Density-based cluster* box.

3 Conclusion

In this report, I provided my research progress for the past semester. And the main contribution of the work is developing a system that collects data from public sources and provides statistics and advanced methods for searching and analyzing collected geodata. And since the main goal of the research project is the crime events detection to improve situation awareness, as future work, I would like to integrate additional sources of public data including some crime statistics and crime reports that can improve data analysis. Also now, I am working under the approach of time-spatial clustering that can be used to cluster potential events based on anomalies around some specified area during specific time. Another very important challenge is to integrate machine learning classifiers to detect posts that reflect crimes and threats in their content.

References

[1] A. Amirkhanyan, F. Cheng, and C. Meinel. *Real-Time Clustering of Massive Geodata for Online Maps to Improve Visual Analysis*. Accepted by the 11th International Conference on Innovations in Information Technology (IIT'15).

[2] *Bright Planet*. URL: http://www.brightplanet.com/2013/06/twitter-firehose-vs-twitter-api-whats-the-difference-and-why-should-you-care/ (last accessed 2015-10-01).

[3] B. De Longueville, R. S. Smith, and G. Luraschi. ""OMG, from Here, I Can See the Flames!": A Use Case of Mining Location Based Social Networks to Acquire Spatio-temporal Data on Forest Fires". In: *Proceedings of the 2009 International Workshop on Location Based Social Networks*. LBSN '09. Seattle, Washington: ACM, 2009, pages 73–80. DOI: 10.1145/1629890.1629907.

[4] *Forbes. The most active Twitter Cities*. URL: http://www.forbes.com/sites/victorlipman/2012/12/30/the-worlds-most-active-twitter-city-you-wont-guess-it/ (last accessed 2015-10-01).

[5] *Geohash*. URL: https://en.wikipedia.org/wiki/Geohash (last accessed 2015-10-01).

[6] M. Gerber. "Predicting Crime using Twitter and Kernel Density Estimation". In: *Decision Support Systems (Elsevier)* 61 (2014), pages 115–125. DOI: 10.1016/j.dss.2014.02.003.

[7] *Google's recommendations for clustering geodata*. URL: https://developers.google.com/maps/articles/toomanymarkers (last accessed 2015-10-01).

[8] H. Kwak, C. Lee, H. Park, and S. Moon. "What is Twitter, a Social Network or a News Media?" In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pages 591–600. DOI: 10.1145/1772690.1772751.

[9] H. J. Miller and J. Han. *Geographic Data Mining and Knowledge Discovery*. Bristol, PA, USA: Taylor & Francis, Inc., 2001.

[10] S. V. Nath. "Crime Pattern Detection Using Data Mining". In: *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. WI-IATW '06. Washington, DC, USA: IEEE Computer Society, 2006, pages 41–44. DOI: 10.1109/WI-IATW.2006.55.

[11] D. Preoţiuc-Pietro and T. Cohn. "Mining User Behaviours: A Study of Check-in Patterns in Location Based Social Networks". In: *Proceedings of the 5th Annual ACM Web Science Conference*. WebSci '13. Paris, France: ACM, 2013, pages 306–315. DOI: 10.1145/2464464.2464479.

[12] T. Sakaki, M. Okazaki, and Y. Matsuo. "Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pages 851–860. DOI: 10.1145/1772690.1772777.

[13] S. Scellato, A. Noulas, and C. Mascolo. "Exploiting Place Features in Link Prediction on Location-based Social Networks". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: ACM, 2011, pages 1046–1054. DOI: 10.1145/2020408.2020575.

[14] M. Walther and M. Kaisser. "Geo-spatial Event Detection in the Twitter Stream". In: *Proceedings of the 35th European Conference on Advances in Information Retrieval*. ECIR'13. Moscow, Russia: Springer-Verlag, 2013, pages 356–367. DOI: 10.1007/978-3-642-36973-5_30.

[15] M. Ye, D. Shou, W.-C. Lee, P. Yin, and K. Janowicz. "On the Semantic Annotation of Places in Location-based Social Networks". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: ACM, 2011, pages 520–528. DOI: 10.1145/2020408.2020491.

[16] Y. Zheng, Y. Chen, X. Xie, and W.-Y. Ma. "GeoLife2.0: A Location-Based Social Networking Service". In: *Mobile Data Management: Systems, Services and Middleware, 2009. MDM '09. Tenth International Conference on*. May 2009, pages 357–358. DOI: 10.1109/MDM.2009.50.

Optimization of Decision Making in Business Processes

Ekaterina Bazhenova

Business Process Technology Group
Hasso Plattner Institute
Ekaterina.Bazhenova@hpi.de

Working faster and smarter has become a necessity for companies. A firm's value chain is directly affected by how well it designs and coordinates the decision making. In order to serve this purpose, it is important for decision models and decision logic to be well grounded in formal and precise semantics. According to recently emerged Decision Model and Notation (DMN) standard from OMG group, the decision logic should be separated from the process logic. With that, while the business process modeling and execution perspective is well understood, there are still no mature approaches for modeling and execution of decisions complementary to business processes. To overcome this gap, we conducted research work presented in this report in 2 parts: (1) a short overview of our earlier work on detecting and optimizing decisions incorporated within process models; (2) A more extensive outlook on our recent work on separation of the decision and process logic and planned work on evaluating the benefits of such separation for a business process performance.

1 Introduction

The value of business processes management (BPM) has been acknowledged as an essential asset to drive a company. One of the most important and challenging BPM aspects is decision making. Adding decision management perspective improves the process by focusing on both the way decisions are made and directing the process that must carry out the decisions [8, 11].

An intense interest of both scientific, and commercial communities towards exploring the possibilities of decision support in organizations is demonstrated by a large number of emerging approaches on different decision ontologies [9], decision service platforms [6, 17, 19], and decision modeling notations [13]. The limitations of immediate start of usage of the above mentioned methods is that they all come up with new semantics or platforms for expressing decisions. Thus, extra effort is needed to identify and implement their integration with already existing and widely used process modeling and execution approaches. In our earlier research works, we addressed this issue by introducing an approach which consists of detecting and improving the decision subprocesses by preserving the process logic [3].

Following the "separation of concerns" paradigm, we investigated handling the compounding complexity of the business process models by externalizing the events, operational conditions and decisions in separate decision models, e.g. which is supported by the recent Decision Model and Notation (DMN) [13]. However, the applicability of such kind of a standard is limited in the current moment, as it is quite challenging for enterprises to identify which parts of business processes comprise process logic, and which parts of them should be externalized to a separate decision model. This served for us to provide the formal framework for semi-automatical detection of decision logic in business process models and for generation of decision models and corresponding decision tables from process models' decision logic. These decision models conform to DMN standard [13] for decision modeling. Process models are represented with the industry standard: the Business Process Model and Notation (BPMN) [12]. More details of this approach are presented in section 2 based on our paper [2].

The advent of knowledge discovery in data technology has created new opportunities to extract powerful knowledge from the stored data using data mining algorithms. Thus, another way of deriving the decision logic from process logic is through analysis of the execution log of the process model. In our most recent paper, we addressed this issue by introducing an approach consisting of derivation of decision models from process models and execution logs on the examples of DMN and BPMN [12]. In particular, we derive the decision logic in the form of decision tables from the event logs of a process model. Based on the derived decision logic, we show the way to construct a corresponding DMN decision model. The approached is summarized in section 3 based on our paper [4].

The remainder of the report is structured as follows: section 2 and section 3 representing our works on the extraction of decision logic from process models, and process models and execution log respectively. The limitations of the approaches and future work are provided in section 4 which is followed by Conclusions and References sections.

2 Extracting Control-Flow Based Decision Logic from Process Models

2.1 Motivation and Exemplary Extraction of Decision Logic from Process Models

Business process modeling is well established in business organizations and is highly supported by modern tools. Modeling languages as BPMN [12] are well suited for process experts as well as — thanks to tool support — end users. Process models allow documentation of business operations, enactment of business processes following the given execution semantics, and (automatic) process analysis regarding correctness, improvement, compliance, etc. In the course of process execution, multiple decisions are taken that influence the mentioned areas [8]. Analysis of industry processes reveals that such decisions include the assignment of actual resources to activities answering the question who shall execute a specific activity or evaluating a given set of data to calculate a decision indicating which path shall be followed at decision points (branching behavior). Furthermore, exceptional behavior is handled by pre-specified procedures for each case. Especially, in the insurance and banking domains, regulatory compliance highly influences process execution by specifying which guidelines must be followed.

Based on the analysis of real world process models, we recognized that part of the logic leading to decisions is often encoded in process models resulting in models that are hard to read and to maintain. BPMN allows to represent decisions and their impact or consequence respectively. However, BPMN is not meant to represent the detailed decision logic since modeling the decision logic often results in spaghetti like models (see Figure 1 for an abstract example) or extensive natural language descriptions explaining the decision taking process that are not analyzable automatically. Thus, decision logic modeling is out of scope for BPMN. Instead, decision tables [18] and further decision modeling concepts [9, 19] are a reasonable and compact method to describe decision logic in scenarios with many input parameters. The upcoming DMN standard is meant to supplement BPMN and allows the "separation of concerns" [14] between process logic and decision logic modeling based on the actual scope of both modeling techniques.

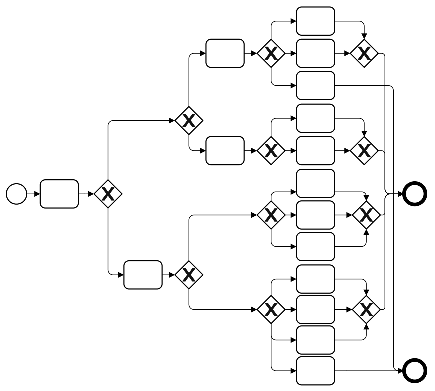


Figure 1: Misuse of BPMN for decision logic modeling

BPMN's scope comprises the business logic containing information on what activities need to be executed in which order by which resource utilizing which data objects while DMN covers the decision logic modeling by specifying which decision is taken based on which information, where to find it, and how to process this information to actually derive the decision result. Since both worlds existed long without proper integration, organizations misused BPMN by including decision logic into process models.

Thereby, data-based decisions are most common. A data-based decision is represented by a decision structure consisting of single and compound decision nodes we refer to as split gateways representing exclusive or inclusive alternatives based on external information. These decisions can be classified into three types:

(i) An explicit decision task with succeeding branching behavior, e.g. in a task, the decision about a customer's loyalty is taken and based on the result, different actions (e.g. give or deny discount) are taken.

(ii) Branching behavior is encoded in decision points, e.g. split gateways with decision logic (about the customer's loyalty) encoded in annotations to the gateways or to edges originating from such gateway.

(iii) There exists a decision task without succeeding branching behavior, e.g. set discount for a customer based on her loyalty.

In Figure 2 an example of extraction of a decision model from process models is presented, which corresponds to extraction of the decision models using the process fragment satisfying one of the patterns (P3) introduced in our paper. On the left side of Figure 2, one can see the process fragment, whereas on the right side the decision model is shown. The latter can be divided into the decision requirements level (top)

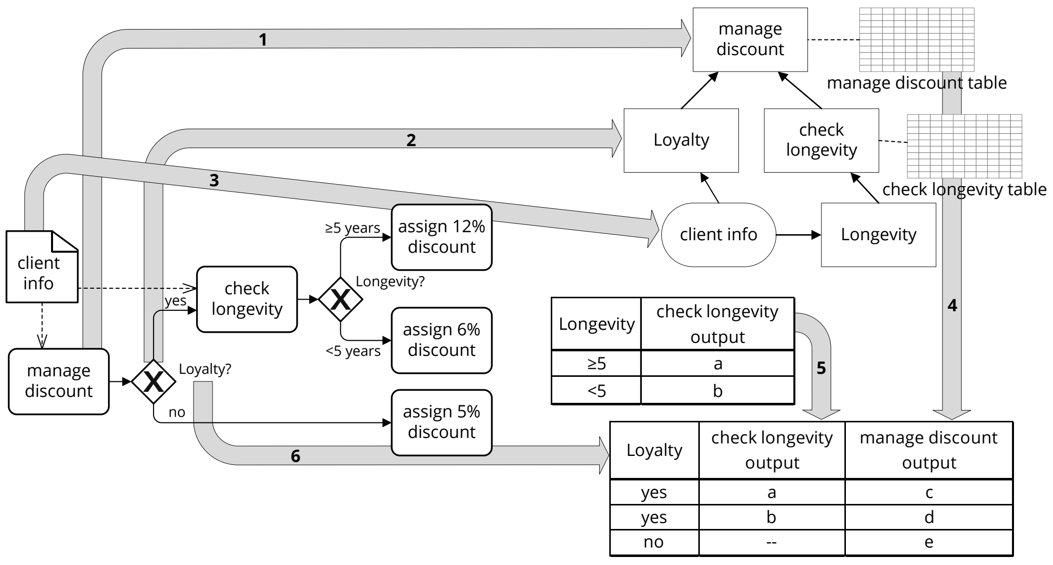


Figure 2: Exemplary mapping for pattern P3: 1—from BPMN activity to DMN decision; 2—from BPMN gateway to DMN decision; 3—from BPMN data node to DMN input data; 4—from DMN decision table reference to actual DMN decision table; 5—from DMN rule conclusions of sub-decision to DMN rule conditions; 6—from BPMN gateway to DMN rule conditions

and the decision logic level (bottom) consisting of decision tables. Also, we inserted arrows to point out the correspondences of the two models' elements. For the sake of clarity, we omitted arrows when the correspondence was already shown by another arrow. For example, arrow 1 shows that the process model's decision task *manage discount* corresponds to the decision element *manage discount* in the decision model. Consequently, we did not draw an arrow for the decision task *check longevity*.

Arrow 2 illustrates the mapping of the gateway labeled *Loyalty?* to an equivalent decision element. The correspondence between BPMN data nodes and DMN input data elements is demonstrated by arrow 3. Note that the connections between the data node and the tasks in the process model result in connections between the gateway decision elements and the input data in the decision model. Furthermore, the mapping of the *manage discount* decision is demonstrated through placing the *Loyalty* decision as an input requirement. Similarly, corresponding to the left part of row three of that table, since the task *check longevity* succeeds *manage discount*, the DMN decision *manage discount* also requires *check longevity* as an input.

Arrow 4 shows that decisions are connected to decision tables if the decision logic is visible in the process model and arrow 5 shows that the output column of the sub-decision is used as input column of the dependent decision. Arrow 6 visualizes that the headers of the condition columns correspond to the labels of the gateways following the decision task and the cell values equal the edge conditions.

2.2 Adaptation of Input Process Model

After extracting the decision logic from a process model to a decision model, the process model needs to be adapted in order to be usable together with the decision model. Basically, the entire decision logic is hidden inside of the first decision task of the pattern. For that purpose, BPMN offers *business rule tasks* that can be linked to decision models and that will output the value of the top-level decision of the decision model.

Thus, for the adaptation we transform the task corresponding to this top-level decision to a business rule task. Since this decision potentially subsumes the decisions corresponding to following decision tasks, these tasks will not be required anymore in the adapted process model. Consequently, we delete each decision task other than the first from the process fragment. Basically, this means that also the gateways succeeding the deleted decision tasks can be removed, such that only the first decision task, the gateway succeeding it and the end nodes of the process fragment are kept. For each end node the gateway has an outgoing edge connected to it and the conditions with which the edges are annotated equal the row conclusions of the top-level decision table. This situation is illustrated in Figure 3. It is important to assign the correct conditions to the different edges originating from the split gateway. For example, the end node *assign* 12 % *discount* in Figure 3 is connected to an edge annotated with *c*. This is because in the original process fragment in Figure 2 the conjunction of the conditions leading from the start node to this end node equals *yes* ∧ (≥ 5 *years*) and the table row representing this conjunction has *c* as its output value.

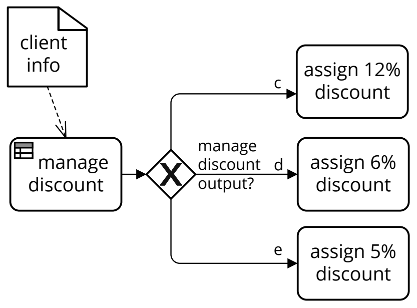


Figure 3: Refactored process fragment for pattern P3

The rest of the details on our work on extracting the control-flow decision logic can be found in our paper [2], in which we introduce a semi-automatic approach to identify decision logic in business processes pattern-based, with consequential mapping of these patterns into DMN models and adaptation of the input process model structure accordingly before we allow configuration of the results in model post-processing. The separation especially fosters the differentation of stable process models to be changed if the business model changes and dynamic decision models allowing flexible configuration of the currently applied business models. Since organizations long misused BPMN as decision modeling languages also it is not meant to capture these aspects, organizations require migration capabilities from misused, spaghetti like process models to a separation that we build of an adapted BPMN model and a DMN model.

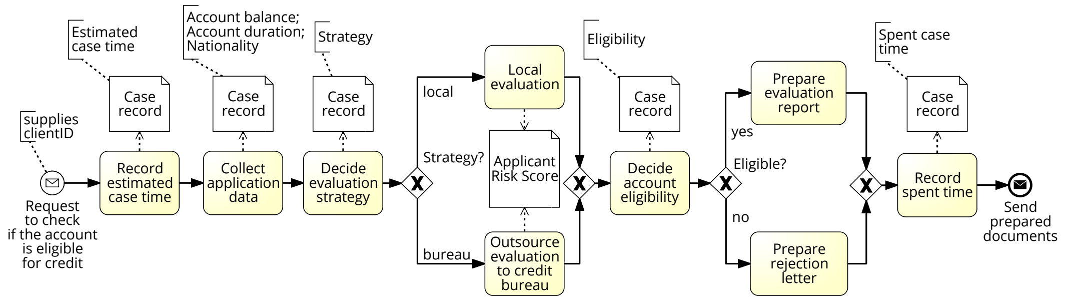


Figure 4: Process model showing the example business process of assessing the credit

3 Extracting Control-Flow and Event Log Based Decision Logic from Process Models

3.1 Motivation and Exemplary Extration of Decision Logic from both Process Models an Execution Data

In this section we provide a description of the business process from the banking domain, to which we refer throughout the rest of the paper. The corresponding business process model is presented in Figure 4; it is based on a step-by-step approach that is generally followed while assessing a personal loan [16].

Upon receiving the request to check if the account is eligible for a credit, the bank worker firstly enters the estimated case time in the case record and then collects application data such as account balance, account duration, nationality, and registers it in the same case record. Afterwards, an expert decision is made about the evaluation strategy of the case: should it be done locally, or outsourced to a credit bureau. Presumably, there are guidelines identifying how the application risk score is assigned in the case of local evaluation; in the case of the credit bureau evaluation, the algorithm is not known. The process follows with an account eligibility decision, noted by a corresponding case recording. In case of a positive decision, the worker prepares an evaluation report, otherwise, a rejection letter. The process finishes after the worker sets the spent case time in the case record.

Figure 4 is a typical representation of a process model that is not aware of decisions. For example, the model contains textual annotations and decision activities which incorporate the business logic that is not represented explicitly in the model. It is not difficult to imagine that upon expanding, such process models quickly become complex, and the business logic management becomes tedious.

For understanding what would be a decision-aware process model in this case, we need to refer to the context of the business process. Nowadays there exist two main types of credit evaluation: (1) judgmental credit-evaluation systems, which may rely on the subjective evaluation of loan officers; and (2) credit-scoring systems that are empirically derived [5]. Basically, it has been settled in literature that using scoring in credit evaluation rules out personal judgment. In credit scoring systems, the decision are taken, depending on the applicant's total score, whilst in personal judgment this issue is neglected. The decision here depends on decision-makers' personal experience and other cultural issues, which vary from market to market [1]. With that, the brute force empiricism that characterizes the development of credit scoring systems can lead to a treatment of the individual applicant in an unjust manner; an example of such an occasion is judging the applicant's creditability by the first letter of a person's last name [7]. Thus, it seems reasonable to use automated credit scoring systems complemented with an explanatory model. The derivation of DMN decision model corresponding to this process model could provide such an explanatory model considering the past experiences, as the goal of the DMN standard is to provide basic principles and rules of thumb for organizing process models to maximize agility, reuse, and governance of decision logic in BPM solutions [18]. We demonstrate the derivation of such a model in the paper [4].

With respect for deriving the decision model from the process model and its execution log, we firstly review in the paper the most common ways to express decisions in process models. One of the most widely used decision construct is the exclusive gateway. It has pass-through semantics for exactly one out of a set of outgoing paths if the associated condition, a Boolean expression, evaluates to true [12]. Thus, a gateway does not "decide" anything; it just specifies the path taken by a process instance based on existing process data. For example, in Fig. 1 the evaluation strategy decision is taken in the activity preceding the corresponding gateway, which only uses the result of this decision. Additionally, in our previous work [2] we identified further decision structures in process models, through analysis of 956 industry process models from multiple domains. We discovered that a single splitway pattern is a common decision pattern, which was used in 59 % of the models. For this paper, we use the notion of a Decision point, which corresponds to this single split gateway from our previous work. With that, we assume that the decision task preceding any decision gateway refers to the same business decision (the omission of this decision task is a common mistake made by inexperienced BPMN users [18]).

In our paper [4] we introduce the following step-by-step approach to derive decision models from process models, which is demonstrated by us on the examples of BPMN process models and DMN decision models. The algorithm is expressed through BPMN process model, as shown in Figure 5. The inputs to the algorithm are the process model m and the corresponding event log which contains the values of data object attributes assigned during the process execution. The goal of the algorithm is to get the outputs: (1) the decision model consisting of one decision requirements diagram and corresponding decision tables; and (2) the adapted process model m', in which the described decisions are referenced by a *business rule* task. In order to simplify the algorithm explanation, we assume for the rest of this chapter, that each decision point of the input process model m contains only two outgoing control flow paths and, correspondingly, two alternative tasks to be executed. The extrapolation of the algorithm for an arbitrary finite set of control flow paths and corresponding tasks will be presented in future work. Below we describe the actual algorithm step-by-step.

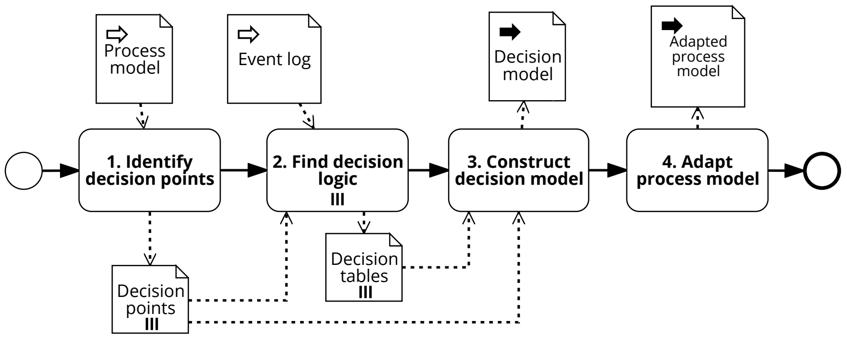


Figure 5: Algorithm of derivation of the decision model and adaptation of the process model

For concrete example of logs for the presented use case, see Section 5 in our paper [4]. The consequent application of the suggestions from the previous section, yields the following decision model as shown in Figure 6 for our use case. The skeleton of the decision points was built according to step 2 of the algorithm presented above. Then, it was established during the decision logic extraction (which we partly omit) that there are data dependencies between decision points dp_1 (Strategy) and dp_2 (Eligibility). Therefore, an information requirement is added to the decision model (directed edge between "Decide evaluation strategy" and "Eligibility" decisions). Also, input data "case record" was replaced by the attributes which statistically were correlated based on the records from the event log to the corresponding decisions (*Account balance*, *Account duration*, and *Nationality*). At last, the labels of the gateway data-based decisions *Strategy* and *Eligibility* were concatenated with the strings "recommended" so as to reflect that the decision logic in the form of business rules was related to them.

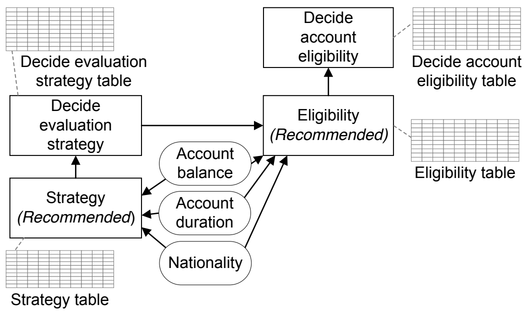

Decide evaluation strategy table
Decide account eligibility
Decide account eligibility table
Decide evaluation strategy
Eligibility (Recommended)
Account balance
Eligibility table
Strategy (Recommended)
Account duration
Nationality
Strategy table

Figure 6: Constructed decision model referring to Figure 4

3.2 Output Decision Model and Adaptation of Input Process Model

Adaptation of the process model in our use case works as follows: in the Figure 4, the types of the tasks *Decide evaluation strategy* and *Decide account eligibility* will be changed into *business rules* tasks. These business rules will be referencing the corresponding decision tables (examples of decision papers can be found in our paper [4]).

The extracted decision model in Figure 6 reflects explicitly the decisions corresponding to the process model from Figure 4, and therefore, could be served as an explanatory model, e.g., for compliance checks. Another advantage of the derived decision model is that it permits changes in the decision model without changing the process model, and vice versa, which supports the principle of separation of concerns [18]. Further details of the formal grounding distinguished into process modeling and decision modeling can be found further in the paper [4].

4 Conclusions and Planned Work

In this report we presented a short overview of our earlier work on the decision modeling and improving in the processes where the decisions are incorporated within process models. As well, we provided a more extensive outlook on our recent work on separation of the decision and process logic and planned work on evaluating the benefits of such separation for a business process performance. We firstly investigated extraction of control-flow based derivation of decision logic from process models. Secondly, we provided the algorithm for extracting control-flow and event log based decision logic from process models.

The limitations of the presented work and corresponding planned activities are presented below:

1. Advanced Derivation of Control-Flow Based Decisions in Process Models In future work, we will extend the approach presented in section 2 by analyzing further process model collections and reduce the assumption of control flow decision structures to identify more patterns and to provide a complete overview about decision logic modeling in process models. The mapping will be adjusted accordingly. Furthermore, we extend the configuration capabilities by, for instance, including label analysis. Finally, we plan to publish best practice guidelines on how to model processes and decisions separately for optimal business usage. Later, the approach can be extended with the other representations of decisions in BPMN, for example, with the involvement of data objects analysis etc. In future it should be, however, taken into account, that the models can be underspecified, and then the "corrected" decision patterns should be considered.

2. Enhanced Decision Model Mining from Process Model and Event Log The approach in section 3 is our first step in investigating the possibility of derivation of DMN process model from process model and its execution log. For that, we assumed that each decision point of the input process model contains only two outgoing control flow paths and, correspondingly, two alternative tasks to be executed. In future, we plan to generalize this for an arbitrary finite set of control flow paths and corresponding tasks. Secondly, we plan to include into the framework the extraction of the decision rules by discovering branching conditions where the atoms are linear equations or inequalities involving multiple variables and arithmetic operators, as in [10]. Further, we will do an implementation of the provided framework and provide the extended approach evaluation by applying it to business processes from multiple domains.

3. Improvement of Business Processes through Decision Optimization In order to improve a given business process, we plan to generate the recommendations on changing the actual structure of the process. Such transformation should increase the expected payoff of the process. For example, the separation of the decision logic in the credit assessment process should improve the process flexibility—one of the most important quality measures of the business process [15].

4. Further Challenges in Process and Decision Management Compared to the extensive stream of BPM research, the decision perspective of business processes up to now has received by far less attention. To overcome this gap, we are currently working on an extensive overview (a planned journal paper) with the aim to identify the challenges of decision modeling with respect to business processes. To this end, we plan to introduce the concept of a decision lifecycle at the enterprise with respect to business process perspective, which consists of five stages such as design, analysis, configuration, enactment and evaluation of decisions. Then, for each stage we will establish the challenges coming emerging from theory and industrial experience. It is planned that this work will summarize our experience on modeling and optimizing decisions in business processes.

5 Publications Summer Term 2015

Workshop papers:

- *Deriving Decision Models from Process Models By Enhanced Decision Mining* [4];

Conference papers:

- *Extracting Decision Logic from Process Models* [2].

References

[1] H. A. Abdou and J. Pointon. "Credit Scoring, Statistical Techniques and Evaluation Criteria: A Review of the Literature." In: *Int. Syst. in Accounting, Finance and Management* 18.2-3 (2011), pages 59–88.

[2] K. Batoulis, A. Meyer, E. Bazhenova, G. Decker, and M. Weske. "Extracting Decision Logic from Process Models". In: *Proceedings of the 27th International Conference on Advanced Information Systems Engineering (CAiSE '15)*. Springer, 2015, pages 349–366.

[3] E. Bazhenova and M. Weske. "A data-centric approach for business process improvement based on decision theory". In: *Proceedings of the 15th International Conference on Business Process Modeling, Development and Support (BPMDS 2014)*. Volume 175. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2014, pages 242–256. DOI: 10.1007/978-3-662-43745-2_17.

[4] E. Bazhenova and M. Weske. "Deriving Decision Models from Process Models Through Enhanced Decision Mining". In: *Proceedings of 3th International Workshop on Decision Mining and Modeling for Business Processes (DeMiMoP '15)*. Innsbruck, Austria, 2015.

[5] Board of Governors of the Federal Reserve System. *Report to the congress on credit scoring and its effects on the availability and affordability of credit*. Technical report. Aug. 2007.

[6] A. Bock, H. Kattenstroth, and S. Overbeek. "Towards a modeling method for supporting the management of organizational decision processes". In: *Modellierung 2014*. Volume 225. Lecture Notes in Informatics. Gesellschaft fuer Informatik, 2014, pages 49–64.

[7] N. Capon. "Credit Scoring Systems: A Critical Analysis." In: *Journal of Marketing* 46.2 (1982).

[8] S. Catalkaya, D. Knuplesch, C. Chiao, and M. Reichert. "Enriching Business Process Models with Decision Rules". In: *BPM 2013 International Workshops*. Springer, Sept. 2013.

[9] E. Kornyshova and R. Deneckère. "Decision-making Ontology for Information System Engineering". In: *ER*. Springer-Verlag, 2010.

[10] M. de Leoni, M. Dumas, and L. García-Bañuelos. "Discovering Branching Conditions from Business Process Execution Logs". In: *FASE*. Springer, 2013. DOI: 10.1007/978-3-642-37057-1_9.

[11] M. Lohrmann and M. Reichert. "Modeling Business Objectives for Business Process Management." In: *S-BPM ONE*. Springer, 2012, pages 106–126.

[12] OMG. *Business Process Model and Notation (BPMN) v. 2.0.2*. 2013.

[13] OMG. *Decision Model and Notation (DMN) v.1*. 2014.

[14] D. L. Parnas. "On the Criteria to be Used in Decomposing Systems into Modules". In: *Communications of the ACM* 15.12 (1972), pages 1053–1058.

[15] H. A. Reijers and S. Liman Mansar. "Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics". In: *Omega* 33.4 (2005), pages 283–306.

[16] M. Sathye, J. Bartle, M. Vincent, and R. Boffey. *Credit Analysis and Lending Management*. Wiley, 2003.

[17] W. Schaper. *SAP Decision Service Management*. SAP AG. Sept. 12, 2014. URL: http://scn.sap.com/docs/DOC-29158 (last accessed 2014-11-13).

[18] B. Von Halle and L. Goldberg. *The Decision Model: A Business Logic Framework Linking Business and Technology*. Taylor and Francis Group, 2010.

[19] A. Zarghami, B. Sapkota, M. Z. Eslami, and M. van Sinderen. "Decision as a Service: Separating Decision-making from Application Process Logic." In: *EDOC*. IEEE, 2012.

Model Synchronization for Complex Industrial Systems

Dominique Blouin

System Analysis and Modeling
Hasso-Plattner-Institut
Dominique.Blouin@hpi.uni-potsdam.de

This document reports on my work since I joined the research school at the Hasso-Plattner-Institute in June 2015. The purpose of this work is to improve the current approaches and tools for model transformation/synchronization developed at the System Analysis and Modeling group, so that they perform well for the rich languages used for modeling complex industrial systems. For this, we propose new strategies exploiting meta-model cardinality constraints and model statistics to improve performances when matching complex patterns. We also plan to develop new concepts for improving the usability of the Triple Graph Grammars (TGG) language in order to ease the specification of model transformations.

1 Introduction

In order to be able to develop nowadays and future complex systems at affordable costs, model-based engineering has been proposed. It consists of modeling the system to be developed with dedicated languages thus replacing natural language specifications with models supporting automated analyses of the design and code generation. Model analysis allows discovering design flaws early before implementation of the system starts. The objective is to avoid as much as possible the costly rework of systems due to the discovery of defects late in the cycle such as at operation time.

In general, several modeling languages and tools must be used and combined to be able to represent all aspects of a system. In addition, it is often the case that the same information about a system is represented with models of different languages leading to information overlaps. Hence, it must be ensured that the models remain consistent with each others as they are often changed during design.

Inconsistencies in system specifications can indeed lead to extremely costly rework and production delays. For example, when the first Airbus A380 was assembled, it was discovered that the cables made in Hamburg were too short for the cabin made in Toulouse [1]. Such inconsistency only discovered late at integration time lead to more than six billion dollars in additional costs. The problem was due to differences in the CAD tools (and processed specifications) used by the German and French teams, which allowed this inconsistency in the specifications to be undetected.

Such problems are expected to occur more and more often as system complexity is increasing with the heterogeneous system specifications being more and more interrelated.[1] In this context, ensuring system specifications are consistent is challenging and becomes even more crucial due to concurrent engineering [15], where systems are being developed more and more in parallel as opposed to the traditional sequential process.

While model-based engineering is being introduced in industry, this problem of maintaining consistency between models is not yet solved. In order to account for it, Airbus has been developing for more than a decade its own development environment ACE (Airbus Concurrent Engineering) that intents to impose the same CAD tools to all its engineering sites. However, it is not always the case that specific CAD tools can be imposed, especially when systems are developed by several independent subcontractors. Therefore, developing automated means to preserve consistency or at least detect inconsistencies between heterogeneous specifications (models) is essential.

In that direction, the System Analysis and Modeling (SAM) group of the HPI has been developing the MoTE (Model Transformation Engine) tool [16], which is based on the Triple Graph Grammar (TGG) language [21]. In a former project that occurred before I joined the HPI [5], I have used MoTE to synchronize the models of two different editors for the SAE AADL (Architecture Analysis and Design Language) modeling language [20]. Since each editor was storing its data in its own format (meta-model), the objective was to to automatically synchronize their data to preserve consistency and allow users to use any of the editors transparently.

During this experiment, several scalability and expressiveness shortcomings were discovered when MoTE was used with the complex and rich AADL language, and in particular for AADL models of industrial case studies. Such case studies are now used to evaluate our current work and are therefore introduced in the next section.

2 Case Studies

In my former work before joining the HPI, I have proposed the RDAL (Requirements Definition and Analysis Language) [18]. RDAL is a modular language supporting Requirements Engineering (RE) that is meant to be combined with other languages for modeling concerns such as architecture and use cases. The language also embeds constructs to support specific RE best practices inspired from both RE and industry practices [14]. In a first step, I have applied a combination of the RDAL, AADL and URN [13] languages for the modeling and analysis of a simple isolette system, used to maintain newborns at a safe and comfortable temperatures (left part of figure 1). The work consisted of converting an existing natural language requirements specification for the system into a set of combined models. This process allow the discovery of several design flaws thus showing important benefits of the approach.

[1]For example, for the A380, typically 80 % of changes in the specifications may have an impact on the cable network [2].

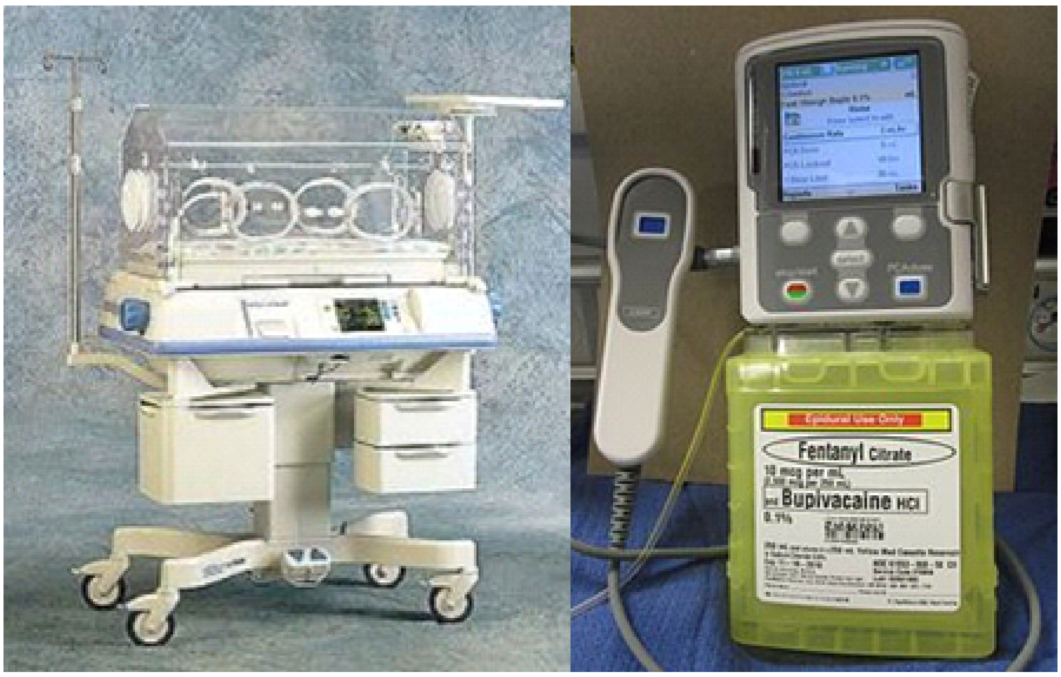


Figure 1: Isolette and Patient-Controlled Analgesia (PCA) infusion pump devices

Later on, a team from the Kansas State University (KSU) applied the same approach to a much more complex and realistic case study consisting of a Patient Controlled Analgesia (PCA) infusion pump system (right part of figure 1). Infusion pumps are used to provide a high level of control, accuracy, and precision drug delivery to patients and their malfunction can lead to severe injuries or death. However, these devices have been associated with persistent safety problems, so much that in 2010, the US Food and Drug Administration (FDA) undertook an initiative to help improve their design [10].

Like for the simple isolette case study, modeling the PCA Pump system with the same combined RDAL, AADL and URN languages allowed to significantly improve the specifications. However, this process of combining languages revealed several shortcomings in model integration [3], thus making these case studies ideal for studying and evaluating model integration techniques. In particular, these case studies made use of MoTE for synchronizing the AADL graphical and textual editors, and now serve to evaluate our current research work on model synchronization described below.

3 Current Work

3.1 Current Issues

The MoTE tool developed at SAM allows specifying relations between two models of different modeling languages. These relations can be used to transform one of the related models into the other one and vice versa, thanks to bi-directionality. Besides, an important advantage of MoTE is that it can transform the models in an incremental fashion. This means that when a model is changed, only the parts of the related model that need to be changed to preserve consistency are updated. This allows better scalability for large models, but also to preserve information present in only one of the models by avoiding unnecessary re-instantiation of model elements.

These relations between models in MoTE are specified in a declarative way using the TGG formalism [21]. However, to be executable, these TGGs must be compiled into models of the Story Diagram language (SDM) [12], which is another model transformation language developed in the SAM group but implementing an operational paradigm. SDM specifications are interpreted to perform the actual transformations/synchronizations. This interpretation makes extensive use of pattern matching, which consists of identifying predefined patterns in the graphs of the related model elements. Patterns are then pre-conditions for applying a subsequent transformation to the matched models elements to realize the transformations.

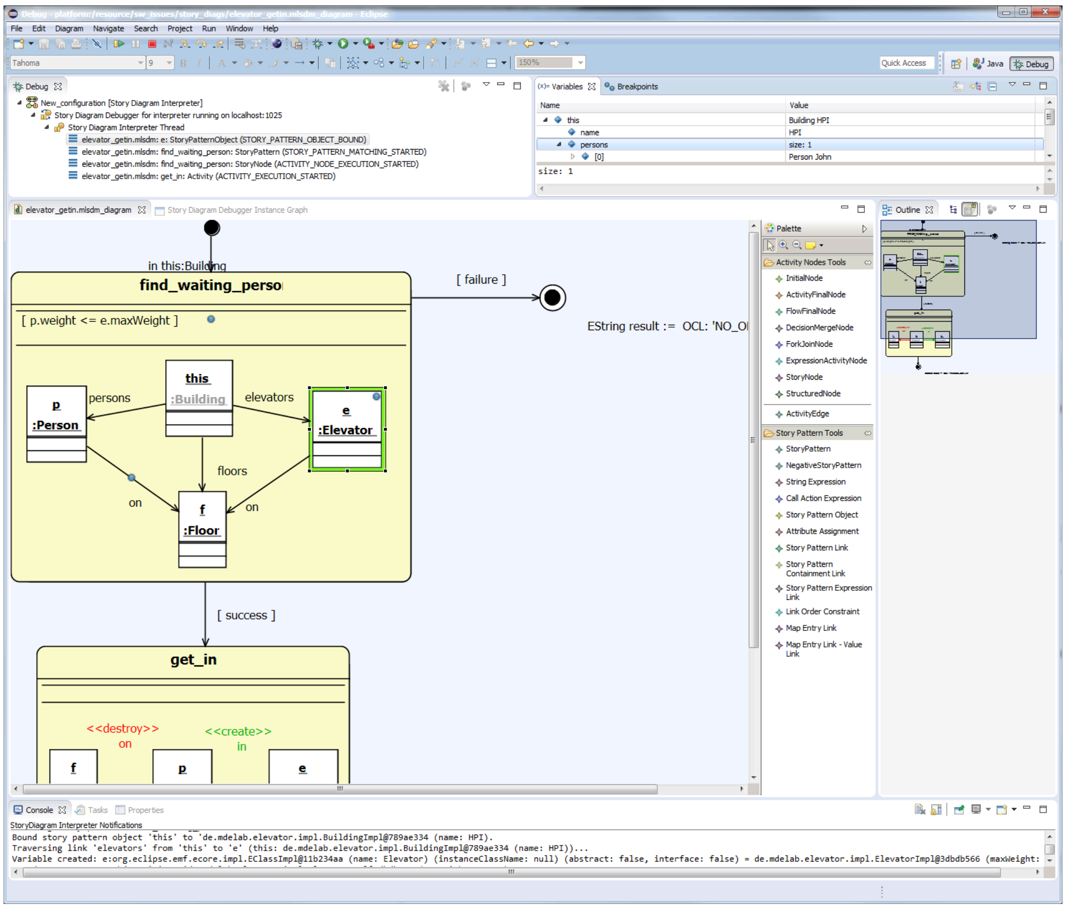
A key advantage of the SDM interpreter over other similar tools and approaches [22] is that in order to optimize performances, it is able to adapt its matching plan at runtime according to the actual data structures that have to be matched. This is in contrast with other story diagram tools for which the matching strategy is hard coded in generated code and cannot adapt at runtime. It has been shown that this runtime adaptation can avoid extremely costly matching processes that result hard coded search plans not taking into account the actual size and structure of data.

In SDM, the way the matching plan is computed is encoded into a *matching strategy* component that can easily be replaced at runtime to adapt to the specific models to be transformed. The current matching strategy simply consists of first trying to match the references between model elements having the least number of elements in order to reduce the matching time. However, this it was shown in my experiments that this strategy does not always perform well for the complex structures of the AADL models of the complex PCA Pump case study. Indeed, such models have several complex components which may have as many as 90 connections among. Connection patterns are the most complex and expensive to match and actually cause several performance problems on our cases study.

3.2 A Debugger for Story Diagrams

In order to better understand these performance problems, we have first worked on providing means to better analyze the performances of SDM. For this, we have completed the development a graphical debugger for story diagrams (figure 2). This debugger, for which a first version ad been developed during a master project in the SAM group required significant subsequent development efforts to become usable.

The debugger allows for tracing the execution of story diagrams by highlighting in the graphical editor the elements that are currently executed, as illustrated by the *elevator* pattern object in Figure 2 being tested for a match. Variables used during the match can be inspected to help better understand the behavior of the matcher. Breakpoints can also be declared to stop the execution when a specific model element is processed by the interpreter, and optionally when a condition attached to the breakpoint evaluates to true.



```
New_configuration [Story Diagram Interpreter]
  Story Diagram Debugger for interpreter running on localhost:1025
    Story Diagram Interpreter Thread
      elevator_getin.mlsdm: e: StoryPatternObject (STORY_PATTERN_OBJECT_BOUND)
      elevator_getin.mlsdm: find_waiting_person: StoryPattern (STORY_PATTERN_MATCHING_STARTED)
      elevator_getin.mlsdm: find_waiting_person: StoryNode (ACTIVITY_NODE_EXECUTION_STARTED)
      elevator_getin.mlsdm: get_in: Activity (ACTIVITY_EXECUTION_STARTED)
```

Name	Value
this	Building HPI
name	HPI
persons	size: 1
[0]	Person John

```
size: 1
```

StoryDiagram Interpreter Notifications

```
Bound story pattern object 'this' to 'de.mdelab.elevator.impl.BuildingImpl@789ae334 (name: HPI).
Traversing link 'elevators' from 'this' to 'e' (this: de.mdelab.elevator.impl.BuildingImpl@789ae334 (name: HPI))...
Variable created: e:org.eclipse.emf.ecore.impl.EClassImpl@11b234aa (name: Elevator) (instanceClassName: null) (abstract: false, interface: false) = de.mdelab.elevator.impl.ElevatorImpl@3dbdb566 (maxWeight:
```

Figure 2: Story Diagram Graphical Debugger

In addition, an execution trace model can be created automatically during the execution of a story diagram, from which various data such as detailed execution times can be computed to help identify bottlenecks and benchmark different matching strategies.

3.3 Currently Explored Pattern Matching Approaches

Our current work consists of exploring new matching strategies to solve the performance problems experienced for the PCA Pump complex models. For this, we are investigating strategies that will make use of more information from the models such as cardinality constraints indicating the maximum number of elements a reference may have. Furthermore, we are also exploring the possibility to make use of statistical data from the models specific to the languages of the models being transformed. For example, while the cardinality of several references may be unbounded in the meta-model of a language, it is often the case that in practice such statistical bound there does exist, due to the nature of the actual real objects of the domain being represented by the models.

Furthermore, we aim to characterize the limits of our dynamic pattern matching approach. In other words, we are investigating means to determine precise conditions under which we can guarantee that our matching strategy will execute in an acceptable time, acceptable being related to the time users can wait for models to be synchronized while they are working with the tools.

Altogether, this work on pattern matching embraces a much larger scope than model synchronization. Pattern matching is indeed a widely spread problem and as a matter of fact, our SDM tool could be used for many other applications far beyond those of the SAM group.

4 Future Work

Once we have completed our work on pattern matching, we will work on other shortcomings that have been identified during my experiments with MoTE. There are related to the declarative TGG language used to specify model transformation/ synchronization between languages.

These shortcomings mostly concern the expressivity, structure and verification/ validation of TGGs specifications from which MoTE produces operational story diagrams.

4.1 Reuse of Model Elements

A challenging task in model synchronization is finding algorithms to avoid information losses due to model elements being destroyed and re-instantiated during changes propagation. In this sense, a first improvement was made to MoTE by implementing the algorithm presented in [11], which avoids re-instantiating the entire set of objects created in the sub-tree of the changed object. However, during my project on synchronization with the AADL, I found that changing a reference from an object to another model object was still causing re-instantiation of the object. I could provide a first solution for this problem [5], but further work is required to ensure its correctness. In addition, it relies on additional constraints that must be imposed to specific sets of TGG rules describing the creation of a model object of a specific type. Further research must be pursued to develop means to formally identify these sets of rules, either through the introduction of a new concept in the TGG language or with appropriate validation constraints pertaining to these sets of TGG rules.

Finally, research on developing a more general strategy is required to avoid object deletion in general. This could be achieved by for example storing the objects to be destroyed temporarily in case they are later needed or interact with the user during synchronization and let him handle non-deterministic cases.

4.2 Usability Improvements

Several shortcomings regarding the usability of TGGs were identified during my former work with MoTE. This calls for a review of the TGG language to ease the development of model synchronization layers in general. The proposed improvements are briefly summarized in the following.

4.2.1 Structure of TGGs

Several improvements are needed to better structure TGGs to favor reuse of elements across specifications and ease maintenance. We will investigate new concepts such as TGG rule extension/refinement, global rule variables, and potentially reusable TGG sides to help retarget one side of an existing TGG specification to another language to be synchronized.

4.2.2 Query Languages

Queries can be defined at various places in a TGG to express attribute assignments or constraints for model objects. While the architecture of MoTE provides hooks to integrate arbitrary query languages, only OCL is currently available. Other languages could help improve performances on large models such as EMF-IncQuery [9], which makes use of an incremental evaluation scheme. In any case, we will provide a generic query language infrastructure allowing to easily integrate new languages to better meet our needs.

4.3 Verification/Validation of TGGs

One difficulty when defining a TGG is insufficient means for verification and validation of specifications. Inexperienced users can easily define incorrect TGGs and only discover the problems at runtime. Many simple rules could be added to the existing set of validation rules to detect problems as early as possible.

Other analyses are also needed to verify properties such as TGG completeness, taking into account the languages of the models to be synchronized. For example, it is not easy to ensure a rule is defined for all contexts in which a model object can be created, and missing rules will lead to incorrect models being generated during transformation.

Further research is also required to develop automated analyses to verify completeness, for example using graph theory taking into account not only the structure of a meta-model but also its additional well-formedness constraints. This work fits well into the CorMorant project of the SAM group [7], which aims at providing methods and tools for the formal verification of correctness of model transformations specified with TGGs.

5 Other Research Activities

5.1 Publications

Since I joined the research school, I have submitted a paper about the PCA Pump case study mentioned above to the REFSQ conference that will be held next year [3]. In addition, I have published 2 papers on some of my former work. The first paper [4] proposes a model-based approach and tool for the synthesis of FPGA systems. It makes use the MoTE tool developed at the SAM group to generate VHDL code. I was invited to present this work at a special session on HLS (High Level Synthesis) during the NASA/ESA Adaptive Hardware and Systems conference in Montreal. The second paper presents a framework for exploration and synthesis of multiprocessor architectures on FPGAs [8], again making use of a modeling approach and tool based on AADL. This paper has been published in the ACM Transactions in Embedded Computing Systems journal.

5.2 PhD Thesis Jury

I have accepted to be part of the jury for the PhD thesis of Cuauhtemoc Castellanos [6] and Elie Richa [19] at the Telecom Paris-Tech engineering school in Paris, France.

5.3 Committees

I have been a member of the program committee of the Multi-Paradigm workshop at the MODELS conference and of the EUROMICRO SEAA conference for the Embedded Software Engineering track.

I have also pursued my activities as a member of the SAE AS-2C standards committee for the AADL language [20] and been involved in the European COST action on Multi-Paradigm Modeling for Cyber-physical Systems (MPM4CPS) [17], for which Prof Dr. Giese leads the working group 1 responsible for providing foundations for MPM4CPS.

5.4 Master Projects

With Prof Dr. Giese, we have proposed a project for HPI master students to extend the Eclipse meta-modeling framework for global model management. Unfortunately, we did not get any candidate.

I am also currently discussing potential collaboration between the Lina laboratory of the University of Nantes in France and the University of East Anglia in the UK for a master project on inferring traceability links between legacy systems modeled with the AADL and requirements. Work on automated traceability management from the SAM group may be an interesting starting point in this project.

6 Conclusion

With these improvements of both the SDM pattern matching strategy and MoTE TGG language, we hope to contribute to a better handling of model-based specifications of complex industrial systems. In particular, by allowing automated model synchronization in an incremental scalable manner, our approach aim at limiting the introduction of inconsistencies in system specifications or at least at detecting them during development, thus reducing development costs significantly.

Furthermore, this research will serve as a basis of a future longer term project for establishing new modeling foundations for modular incremental global model management.

References

[1] *Why do Projects Fail: Airbus A380*. URL: http://calleam.com/WTPF/?p=4700 (last accessed 2015-10-01).

[2] *A 380, quand la CAO s'emmêle*. URL: http://www.usinenouvelle.com/article/a-380-quand-la-cao-s-emmele.N17079 (last accessed 2015-10-01).

[3] D. Blouin, B. Larson, E. Vasserman, and H. Giese. *Combining Requirements, Use Case Maps, and Architectural Models for Medical Device Design*. Submitted to the Requirements Engineering: Foundation for Software Quality (REFSQ) 2016 conference.

[4] D. Blouin, G. Ochoa Ruiz, Y. Eustache, and J.-P. Diguet. "Kaolin: a System-level AADL Tool for FPGA Design Reuse, Upgrade and Migration". In: *NASA/ESA International Conference on Adaptive Hardware and Systems (AHS)*. Montréal, Canada, June 2015.

[5] D. Blouin, A. Plantec, P. Dissaux, F. Singhoff, and J.-P. Diguet. "Synchronization of Models of Rich Languages with Triple Graph Grammars: An Experience Report". In: *Theory and Practice of Model Transformations: 7th International Conference ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings*. 2014, pages 106–121. DOI: 10.1007/978-3-319-08789-4_8.

[6] C. Castellanos. "Conception de Systèmes Sûrs et Sécurisés à partir d'une Modélisation Orientée Composant". PhD thesis. Telecom Paris-Tech School, 2015.

[7] *The CorMorant Project*. URL: http://www.hpi.uni-potsdam.de/giese/projects/cormorant (last accessed 2015-10-01).

[8] Y. Corre, J.-P. Diguet, D. Heller, D. Blouin, and L. Lagadec. "TBES: Template-Based Exploration and Synthesis of Heterogeneous Multiprocessor Architectures on FPGA". In: *ACM Transactions in Embedded Computing Systems* (2015).

[9] *EMF-IncQuery Project Homepage*. URL: https://www.eclipse.org/incquery/ (last accessed 2015-10-01).

[10] *U.S. Food and Drug Administration Infusion Pumps Improvement Initiative*. URL: http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm202501.htm (last accessed 2015-10-01).

[11] H. Giese and S. Hildebrandt. *Efficient Model Synchronization of Large-Scale Models*. Technical report 28. Hasso Plattner Institute at the University of Potsdam, 2009.

[12] H. Giese, S. Hildebrandt, and A. Seibel. "Improved Flexibility and Scalability by Interpreting Story Diagrams". In: *Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)*. Edited by T. Magaria, J. Padberg, and G. Taentzer. Volume 18. Electronic Communications of the EASST, 2009.

[13] *ITU URN Specification*. URL: http://www.itu.int/rec/T-REC-Z.150/en (last accessed 2015-10-01).

[14] D. Lempia and S. Miller. *Requirements Engineering Management Handbook*. Technical report. Federal Aviation Administration (FAA), 2009.

[15] F. Mas, J. Menendez, M. Oliva, and J. Rios. "Collaborative Engineering: An Airbus Case Study". In: *Procedia Engineering* 63 (2013). The Manufacturing Engineering Society International Conference, MESIC 2013, pages 336–345. DOI: 10.1016/j.proeng.2013.08.180.

[16] *MoTE Project Homepage*. URL: http://www.mdelab.org/mdelab-projects/mote-a-tgg-based-model-transformation-engine/ (last accessed 2015-10-01).

[17] *The Multi-Paradigm Modeling for Cyber-Physical Systems (MPM4CPS) European COST Action*. URL: http://www.cost.eu/COST_Actions/ict/IC1404 (last accessed 2015-10-01).

[18] *RDAL SAE Draft Specification*. URL: https://wiki.sei.cmu.edu/aadl/images/0/0e/RDAL_annex_draft_v161.pdf (last accessed 2015-10-01).

[19] E. Richa. "Qualification of Source Code Generators in the Avionics Domain: Automated Testing of Model Transformation Chain". PhD thesis. Telecom Paris-Tech School, 2015.

[20] *SAE AADL Specification*. URL: http://standards.sae.org/as5506b/ (last accessed 2015-10-01).

[21] A. Schürr. "Specification of graph translators with triple graph grammars". In: *Proc. of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*. Edited by E. W. Mayr, G. Schmidt, and G. Tinhofer. Volume 903. Lecture Notes in Computer Science. Herrsching, Germany: Spinger Verlag, June 1994, pages 151–163.

[22] G. Varro, A. Anjorin, and A. Schurr. “Unification of Compiled and Interpreter-Based Pattern Matching Techniques”. In: *Modelling Foundations and Applications*. Edited by A. Vallecillo, J.-P. Tolvanen, E. Kindler, H. Storrle, and D. Kolovos. Volume 7349. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 368–383. DOI: 10.1007/978-3-642-31491-9_28.

Runtime data-driven software evolution in enterprise software ecosystems

Thomas Brand

System Analysis and Modeling
Hasso-Plattner-Institut
thomas.brand@hpi.uni-potsdam.de

The study presented in this report explored how runtime data already supports enterprise software evolution decisions at SAP to make the software continuously fit the requirements. The study helped answering questions and to confirm or rebut assumptions made when first approaching this research field. Also it revealed additional research challenges.

1 Introduction

When software is adapted to the needs of a changing context through the periodic release of modified versions then this is considered as software evolution [4] for this study. The modifications performed by the software manufacturer may affect the source code, the documentation as well as the recommended configuration and finally result in a new software package release version. The term software package is used in the study to distinguish a deployable release of software artifacts from a software system, which in contrast shall be the term for a software instance operating in a particular environment. Software packages are built for software products such as SAP software and custom developed applications.

For this study any data generated or collected during the operation of a software system is considered as runtime data. Runtime data is already used by self-adaptive autonomous software systems for their individual adaptation [3]. If necessary such a system can to a certain extend automatically adjust to the events which it is capable to sense and process as runtime data. Those can be events for example regarding the system itself or its given goals and its environment, which shall be considered here as context.

Especially the following current circumstances suggest asking how runtime data of systems, which are based on the same software package, can also support the evolutionary adaptation of this particular underlying software package in the sense of the earlier given definition of software evolution: 1. The emerging support for software deployments in cloud environments and the increase in software as a service offerings making it technically easier for the software manufacturer to access the runtime data as well as 2. the currently available technologies for mass data processing to analyze the accruing runtime data.

Figure 1 depicts the automated steps of the control loop to accomplish self-adaptation of an individual software system. The strengths of such an automated control loop can comprise fast reaction times, great complexity handling [1] and

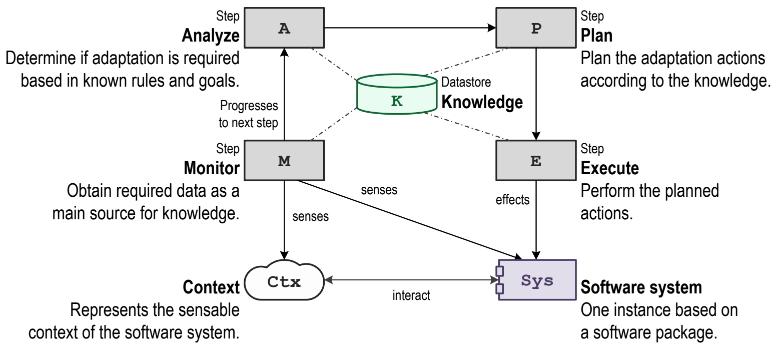

Step
Analyze
Determine if adaptation is required based in known rules and goals.
A
P
Step
Plan
Plan the adaptation actions according to the knowledge.
K
Datastore
Knowledge
Progresses to next step
Step
Monitor
Obtain required data as a main source for knowledge.
M
E
Step
Execute
Perform the planned actions.
senses
senses
effects
Context
Represents the sensable context of the software system.
Ctx
interact
Sys
Software system
One instance based on a software package.

Figure 1: Control loop to automatically perform the individual adaptation of a software system based on its knowledge, which can comprise for example rules, goals, system self- and context awareness.

objective decisions based on specified goals and measured data. The very narrow context comprehension and the limited adaptation solution space can be assessed as weaknesses of the loop. Both of those weaknesses are less challenging for the human-controlled development cycle that drives the evolutionary adaptation of a software package. But utilizing ideas from the automated self-adaptive systems control loop especially the consideration of runtime data may help to further improve the results of this development cycle. An enhanced version of the development cycle, which considers runtime data to produce additional knowledge for software evolution decisions, is depicted in Figure 2.

This study investigated the development cycle for SAP configuration recommendations. The recommendations are provided to customers as SAP Best Practices packages, sometimes also simply called content packages. A SAP Best Practices package is typically developed for a particular software package release. With the best practices packages the software manufacturer SAP intents to help its customers to implement business best practices in their SAP software systems faster and easier. The packages consist of a variety of items including business process descriptions, planning aids, configuration guides, test scripts and learning materials.

The goal behind understanding the development cycle for SAP Best Practices packages was to learn about the challenges associated with decisions regarding the creation and evolutionary adaptation of the content. Furthermore during the study I sought for insights about how runtime data is or could be utilized to support those decisions but also how the data could support the content provisioning towards the customer. In order to get to know more about the use cases for runtime data in those regards and about techniques to obtain and process the data, I also got in touch with the SAP Usage Measurement Program. Eventually understanding software respectively content evolution relevant questions better shall help to evaluate system architecture options required for answers based on runtime data.

For this explorative empirical study I discussed with subject matter experts and additionally conducted interviews. I recorded and processed nine interviews adhering to scientific practices recommended for example in [5] and [2] to compile

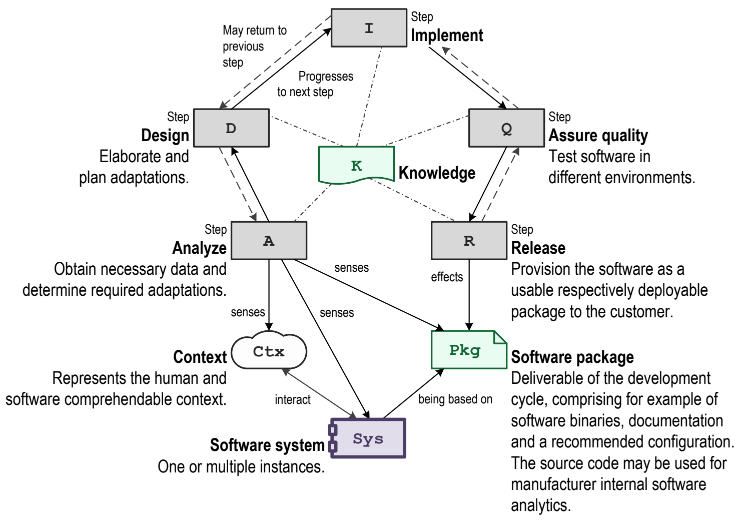

Step
Implement
I
May return to previous step
Progresses to next step
Step
Design
D
Elaborate and plan adaptations.
K
Knowledge
Step
Assure quality
Q
Test software in different environments.
Step
Analyze
A
Obtain necessary data and determine required adaptations.
Step
Release
R
Provision the software as a usable respectively deployable package to the customer.
senses
senses
senses
effects
Context
Ctx
Represents the human and software comprehendable context.
Pkg
Software package
Deliverable of the development cycle, comprising for example of software binaries, documentation and a recommended configuration. The source code may be used for manufacturer internal software analytics.
interact
being based on
Software system
Sys
One or multiple instances.

Figure 2: Human-controlled development cycle driving the evolutionary adaptation of a software package by producing new package releases, which are the basis for one or more software systems.

summaries which were approved by the interviewees and are quoted in this report. The following sections of this report present major insights I gained from the study.

2 Reporting on conducted interviews

2.1 Development cycle

This section shall provide an overview about the content development cycle for SAP Best Practices packages and related insights gained through the study.

Role model Several roles from within SAP and its software ecosystem are involved in the development cycle for SAP Best Practices packages. At SAP the solution package owner has the responsibility for the content package as a product and is basically involved in all steps of the development cycle and also coordinates the activities before and afterwards. The package build lead is responsible that the content and finally the package are actually created. He controls the development cycle and manages a team of package build experts. The package build expert with his detailed business and software package knowledge is mainly involved in the design, implementation and quality assurance steps. The software packages for which the SAP Best Practices packages are built, are provided by the SAP software development team. Besides customers the SAP software ecosystem comprises several types of partners. Those partner companies are most relevant for the development cycle, which have distinguished business process and industry expertise and conduct content based software system implementation projects with customers. Thus consultants from partners or SAP Consulting are regularly members of the package build team. The collaboration seems to work well without an explicit role model for the software ecosystem at SAP [9, § 15, 9, § 4].

Analyze step After the portfolio decisions about which SAP Best Practices packages shall be created the development process starts with the identification of required changes and new content features. This comprises trying to anticipate future requirements and considering feedback about existing content. The conducted interviews indicate that the focus of content development lies on the identification of additionally required content for further business processes, process steps and integration scenarios. Insights into customer systems are considered as less helpful when trying to understand what the customers' requirements are or will be in the future [10, § 3] and thus those insights are likely not considered today [9, § 6].

Input about what content shall be developed is gathered through several channels [10, § 6]. On the one hand from outside of the software manufacturer based on relationships with customers and partners [10, § 2]. On the other hand from within the organization, such as from the support [9, § 7] and the go-to-market team [10, § 2]. But most of the content changes and enhancements seem to be triggered through changes of the software package performed by the software development team [9, § 9, 10, § 2, 7, § 3]. The content is also reviewed about once a year [13, § 9]. But as the content creation is carefully deliberated, the existing content seems to rarely become obsolete [10, § 13]. The package owner and the package build team together prioritize both external and technical requirements for the SAP Best Practices package releases and add them to the backlog from where they are processed [10, § 2].

With the SAP Best Practices packages the goal is to cover 80 % of the functionality, which every customer needs independent of his size [9, § 17, 10, § 2]. A challenge for the package build team is to comprehend how many customers actually require the process being built and to forecast the usage frequency. The team's corresponding assessment is sometimes not right. It happens that a request of one customer is generalized and becomes the customers' demand [10, § 5]. The interviews also suggest that additional support would be very appreciated for assumption validation and understanding what customers actually use [8, § 14]. This could for example help to determine which processes are used most often [9, § 8] and which products customers integrate as part of their solutions [9, § 5].

To summarize: there are many sources from where requests for changed or additional content reach the content development team. Among those sources runtime data from customer systems could not be found. But as the content shall serve the vast majority of customers as best practice it is exceedingly important to validate the received requests and to objectively prioritizing them. Thus helping to answer the following question seems to be especially valuable: how is the already existing content actually being used and how well will requested content meet the needs of most customers?

Design step This step comprises for example designing blueprints, performing feasibility checks, specification tasks and doing the detailed project planning. The work deals mainly with the question how to do something. Creativity and knowledge about usage patterns are essential [9, § 9, 13, § 12]. While SAP has strong expertise in base processes and cross components, partners are favored for industry specific processes, like retail, banking or hi-tech [10, § 10]. First the required features are specified as well as the bill of material, which determines the content artifacts to be created. Then a project plan is defined containing the timelines, work assignments and required systems [10, § 8]. Also the network to the software development team needs to be established as it changes from time to time [10, § 8]. To summarize: designing for content development and the corresponding planning requires creativity and sophisticated technical and industry business expertise.

Implement step During this step all content assets are created. It lasts typically four to eight weeks [10, § 8]. When the package build team defines a process based on their knowledge and the fragmentary product documentation then it quickly runs into problems. Therefor the collaboration with the software development is very close. Regular calls take place and the package build team also takes part in the program management meetings so it is aware of the features the software development is working on and the problems the developers encounter. [10, § 9]

Also consultants from SAP and SAP partners contribute to the implementation of the SAP Best Practices packages with their experience from customer projects [10, § 11]. Additionally partners create and market their own best practices packages with a major goal being a high level of consistency and integration with SAP content.

To summarize: developing content requires tight collaboration with the software development team. In the future also partners shall get more involved in content development efforts. This might require allowing them to participate in the content development infrastructure.

Assure quality step In this step content consolidation tests are followed by acceptance tests. [10, § 8]. Also performance tests are conducted and documentation assets are checked for quality. While the content is tested by SAP in simulated customer implementation projects, acceptance testing is also done together with partners [10, § 8].

Release step This step is about getting the content published and rolled out to the customer. Three release modes can be distinguished: 1. A SAP Best Practices package is released sometime during the lifetime of the corresponding software package release. This is a rare case, today. 2. The best practices package is released shortly after the software package became available. 3. Particular software packages and their best practices packages are developed simultaneously.

The SAP Hybris e-commerce software is an example for the second mode. The Hybris software package gets released every three months. About four to six weeks later the corresponding SAP Best Practices package is released. [10, § 7]

The reason for this delay in the second mode is that the content development is not done simultaneously but may only start when the software enters the solution validation phase, which is only about four weeks before the software release [10, § 7].

In this mode SAP Best Practices packages must also only be tested with already released software, too. Thus the second mode does not allow including a mechanism in the software setup routine to automatically tailor the SAP Best Practices integration configuration content according to the scenario which the customer earlier specified during the setup in a questionnaire. [9, § 19]

If the content is released for downloading then only the entire best practices package can be obtained as a single archive file by the consumer. This circumstance makes it currently impossible to measure how often individual content assets are actually accessed [9, § 5]. It is planned that in the future content created by SAP and by partners is provided through one common database [10, § 11]. This may then also allow for a more detailed access tracking.

Finally it is the responsibility of the package owner that the released content reaches the partners and customers, for example by providing enablement material, conducting partner workshops and presenting it at fairs. The package owner's main target group are the partners and not the customers. [9, § 1]

To summarize: if the content development is separated from the software development process as with the first and second release mode then it is hard for the content development to benefit from or influence the runtime data which the software collects and analyzes. Thus the data also can not be utilized for example for the content provisioning or content adaptation. Also the coarse granularity for accessing and obtaining the content on package instead of asset level has been recognized as hindrance for understanding asset user access frequencies as an indicator for content asset relevance.

2.2 Question properties

The study investigated questions relevant for deciding about new features and evolutionary adaptation. It turned out that some of the questions' properties seem to be useful for deriving requirements for the system architecture, which can support answering those questions with the help of runtime data. For example the following two questions probably require different architectural components to be answered: 1. Where do long waiting times between two business process steps occur as they may indicate missing functionality within the software executing this business process [9, § 12]? 2. How many customer software systems are based on SAP Best Practices packages and how does their setup differ from the originally recommended configuration [13, § 2]? This section presents some of the question properties and discusses related question examples.

Response time This is the duration from the point in time when the question occurred until a suitable answer is available. For example if an error occurs then access to a customer system would be helpful for the development team [10, § 3] to analyze the error as well as provide support and later a corresponding bug fix. The developer would need responses to his questions within support case timeframes—if not immediately then likely within a few hours.

In comparison the question about how often which of the software manufacturer's software packages get integrated by customers [9, § 5] might not be as urgent. But an system architecture and related processes requiring up to a year or longer [14, § 5, 12, § 6] to provide this information might be less valuable.

Also customers are interested in a feature to compare their common key performance indicators against aggregated benchmark values to identify potential for improvement [11, § 8, 12, § 5]. Even though gathering and analyzing the data for this use case might take some time, the answers to such a set of pre-defined standardized questions can be provided with a very short response time as the required data can be pre-collected and pre-processed.

Questioner Data about the execution of software is not only interesting for software evolution decisions but also for other purposes. Thus besides developers and product owners other roles are also interested in accessing software runtime data ideally using the same infrastructure, such as a customer license auditor [11, § 3] or customers wanting to justify their software investments [11, § 8]. Different roles may also have different requirements on how which data is presented. Thus there are currently efforts at SAP to build a dashboard providing information to different roles [11, § 3]. Receivers of answers may also be other software components. Such a component might automatically correlate measurement results with development artifacts or combine them with results from software analytics. But the interviews showed that the content artifacts are not prepared for this use case, yet and no demand for this kind of support was recognized [7, § 1, 14, § 7, 13, § 12].

Object of the question Software evolution questions are related to different objects, for example parameter settings and how they deviate from default or best practice configurations [13, § 2]. Also system setups are of interest, for example which software packages are integrated [9, § 5, 15, § 4] and which versions are used [14, § 11]. The interviews also indicate that usage patterns are particularly relevant [13, § 12], from functionality invocation frequencies [14, § 2] to actual business process workflows [9, § 13, 8, § 1].

Answer accuracy The accuracy of an answer to a question can be determined by several factors. For example analyzing runtime data from all customer systems would provide an exact picture of how the software is used. Nevertheless analyzing the data of a comparatively very small subset of customers might already yield a sufficiently representative result [13, § 7] while consuming significantly less resources. Also the measurement frequency can contribute to the accuracy ranging from one-time queries over regular samplings to permanent observation. Close related to the frequency is also the required duration of the observation, which can be from a week [13, § 8] to several years [14, § 5].

Validity of the question The validity of a question shall mean if it is generally applicable to several software packages or if it is very specific to a particular package. It seems that the product owners' questions about software usage are very package specific [14, § 6]. Also general usage patterns have not been identified [12, § 2] which could have been used to provide guidelines on how to measure for software evolution decisions. Ultimately it has not been systematically examined, how data about the execution of software can influence and enhance software development, yet [12, § 1].

Measure versus discover The fact whether the question itself is known before the data collection or not has a big impact on the amount of required data and the infrastructure coping with it. In the first case when the question is known only structured data is upfront goal-oriented collected. For the second case, where the question is unknown, the rule can be applied the more data is collected the better. In this case also unstructured data may be gathered, for example entire log files, to later recognize new patterns and search for correlations. According to the interviews the first case is currently more relevant to understand software usage [12, § 3] while the second case is promising [11, § 4].

3 Conclusion and outlook

The development of SAP Best Practices packages content is mainly driven by new software packages and software package features as well as by established relationships with partner companies and customers, who provide feedback or directly submit change requests. The results of the conducted empirical study indicate that runtime data currently does not play a role when for example (a) identifying the need for new content and (b) validating assumptions as part of the development cycle and (c) obtaining feedback for evolutionary adaptation of existing content, neither from delivering the content to the customers nor from its utilization in the their software systems. The study results also indicate that further support for assumption validation and understanding actual usage within customers systems is considered as especially helpful. In both cases runtime data should be a valuable additional source for insights to help with decisions concerning content and software evolution.

Investigating questions related to evolutionary adaptation adumbrated a set of particular question properties. It appears that those properties determine requirements for the system architecture which is useful to answer those questions with the help of runtime data. Further work is required to specify a question classification system. In conjunction with such a classification system then a maturity model is thinkable which would allow rating system architectures by their support for different question classes. The intend behind such a maturity model is to help software manufacturers to derive steps to evolve their software package to a higher maturity level and afterwards to enable customers to adapt their software systems individually to the level which fits their needs.

Additionally due to allowing flexibility and handling complexity the adaptive monitoring described in [6] seems to be a very interesting approach to gather runtime data not only for the individual adaptation of a software system but also for evolutionary adaptation of its underlying software package. Furthermore analyzing the knowledge of multiple operating monitoring adaptation engines could be helpful to determine or improve the usefulness of the recommended monitoring configuration, which can be part of a SAP Best Practices package for all customers. This is just one example for the very appealing idea that the knowledge gained by multiple self-adaptable systems during their individual adaptation can be utilized as runtime data by the software manufacturer for the evolutionary adaptation of the underlying software package.

References

[1] C. Gacek, H. Giese, and E. Hadar. "Friends or Foes? – A Conceptual Analysis of Self-Adaptation and IT Change Management". In: *Proceedings SEAMS '08, Leipzig, Germany*. ACM Press, 2008.

[2] J. Gläser and G. Laudel. *Experteninterviews und qualitative Inhaltsanalyse*. 4. Aufl. Wiesbaden: VS Verlag für Sozialwissenschaften, 2010.

[3] IBM. *An architectural blueprint for autonomic computing*. USA, 2005.

[4] M. M. Lehman and L. A. Belady. *Program Evolution: Processes of Software Change*. London: Academic Press, 1985.

[5] F. Shull, J. Singer, and D. I. K. Sjøberg. *Guide to advanced empirical software engineering: Forrest Shull, Janice Singer, Dag I.K. Sjøberg*. London: Springer, 2008.

[6] G. Tamura, N. M. Villegas, H. A. Müller, L. Duchien, and L. Seinturier. "Improving Context-Awareness in Self-Adaptation using the DYNAMICO Reference Model". In: *Proceedings of SEAMS '13, San Francisco, CA, USA*. IEEE Press, 2013.

[7] Thomas Brand. *Interview summary - IS01*. Walldorf, 28.04.2015.

[8] Thomas Brand. *Interview summary - IS02*. Walldorf, 28.04.2015.

[9] Thomas Brand. *Interview summary - IS03*. Walldorf, 28.04.2015.

[10] Thomas Brand. *Interview summary - IS04*. Walldorf, 29.06.2015.

[11] Thomas Brand. *Interview summary - IS05*. Walldorf, 30.06.2015.

[12] Thomas Brand. *Interview summary - IS06*. Walldorf, 1.07.2015.

[13] Thomas Brand. *Interview summary - IS07*. Walldorf, 1.07.2015.

[14] Thomas Brand. *Interview summary - IS08*. Walldorf, 4.08.2015.

[15] Thomas Brand. *Interview summary - IS09*. Walldorf, 5.08.2015.

See-Through Lenses for Massive 3D Point Clouds

Sören Discher

Computer Graphics Systems Group
Hasso Plattner Institute
soeren.discher@hpi.de

3D point clouds as a digital representation of our world are used in a variety of applications. They are captured with LiDAR or derived by image-matching approaches to get surface information of objects, e.g., indoor scenes, buildings, infrastructures, cities, and landscapes. To allow for the efficient exploration of heterogeneous, time variant, and semantically rich 3D point clouds novel interaction and visualization techniques are required. In this report, interactive and view-dependent see-through lenses are introduced as exploration tools to enhance recognition of objects, semantics, and temporal changes within 3D point cloud depictions. Novel filtering and highlighting techniques are presented that allow to dissolve occlusion to give context-specific insights. All techniques can be combined with existing point-based rendering approaches as exemplified on an out-of-core real-time rendering system for massive 3D point clouds.

1 Introduction

Applications for environmental monitoring [5], urban planning and development [13], as well as disaster and risk management [1] require precise and up-to-date surface information for objects, sites, and landscapes. 3D point clouds are digital discrete surface representations that fulfill these requirements. They can be created automatically and efficiently by means of in-situ and remote sensing technology (e.g., laser scanning or photogrammetric approaches) with high density (e.g., 400 points per m^2) and high capturing frequency (e.g., once a month). Due to the massive amount of data managing, processing, and visualizing 3D point clouds poses challenges for hard- and software systems [7].

Traditionally, geoinformation applications and systems reduce the density and precision, or extract generalized, mesh-based 3D models in a time-consuming process [6]. Hence, they do not use the full potential of the data. Out-of-core and external memory algorithms on the other hand are designed to cope with massive amounts of data [8]. As an example, arbitrarily large 3D point clouds can be explored in real-time by combining specialized spatial data structures with Level-of-Detail concepts [4, 10]. Most of these approaches render all points in a uniform way giving an equal amount of significance to every point. However, points within a 3D point cloud typically differ in terms of relevance. The relevance of a point depends on the kind of information it carries (e.g., timestamp, surface category, color, position) as well as the current use case and visualization task. Common use cases are for example:

Different Acquisition Types Surface information for a site could be available from different surveys such as an airborne, mobile mapping, terrestrial, outdoor and

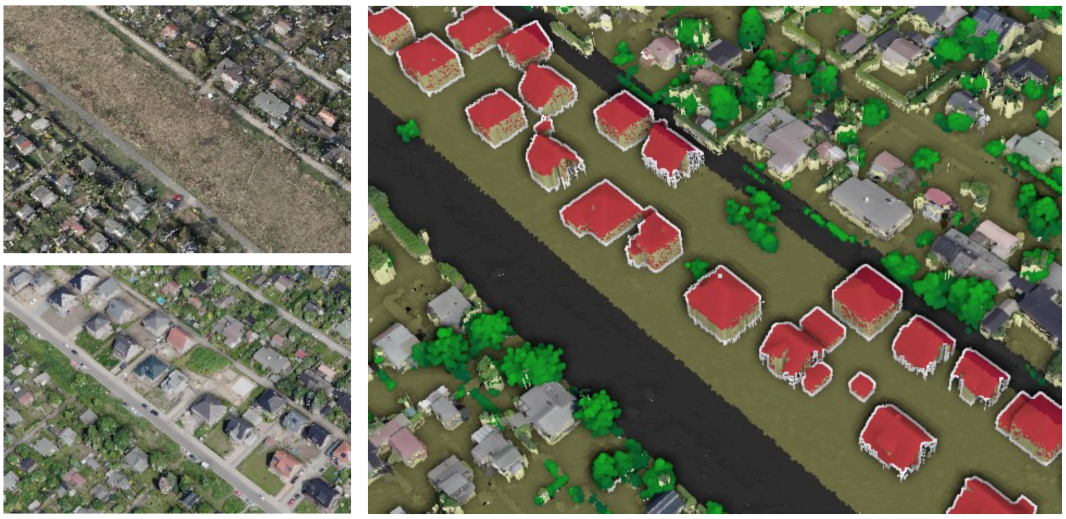


Figure 1: Multi-temporal 3D point cloud: building points that are not present in the old scan (upper left) but captured in the new scan (lower right) are highlighted with a red color scheme and by tracing their boundaries.

indoor data acquisition. On a broader scale, 3D point clouds of spatial environments might consist of overground as well as subterranean structures (e.g., mine shafts, sewers). By allowing users to see through occluding structures (i.e., by masking out corresponding points) the in-depth exploration of such 3D point clouds can be facilitated.

Multi-temporal Data Many applications require frequent scans and simultaneous use of 3D point clouds taken at different points in time. To assist users in exploring differences and structural changes of the captured site within such multi-temporal 3D point clouds (e.g., constructed, demolished, or modified buildings), points indicating such changes should be highlighted (Figure 1).

Classification-dependent Rendering Typically, points represent different surface categories (e.g., ground, building, vegetation, water, city furniture interior, façade). By applying different point-based rendering techniques and color schemes to each point, taking into account characteristics of its surface category (e.g., fuzzyness of vegetation, smooth ground surfaces or planar building roofs), different objects within a 3D point cloud depiction and relations between these objects can be distinguished more efficiently [3, 10]. However, relevant objects might still be not visible due to occlusion by less relevant objects (e.g., buildings below vegetation). To facilitate the identification of full and partly occluded objects, users need to see through occluding structures (Figure 2).

To filter points within a 3D point cloud based on their relevance to the given use case, see-through lenses for massive 3D point clouds are defined as follows:

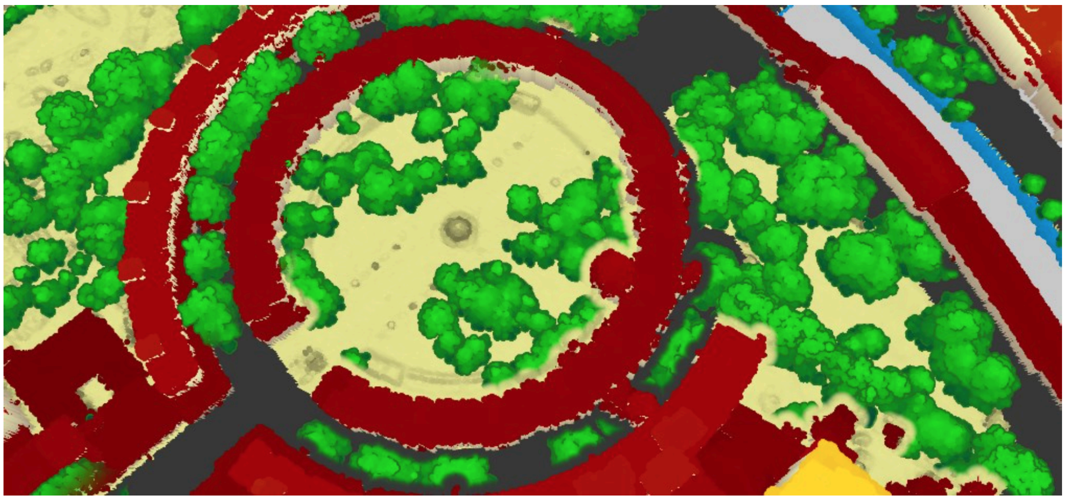


Figure 2: 3D point cloud with semantics: vegetation points are masked out in the lower right part to show hidden building points.

- A see-through lens defines a space within a 3D point cloud, in which points of higher relevance are emphasized by masking out less relevant points completely or in parts.
- The area of a see-through lens can be defined interactively by the user or automatically with respect to the current view position (e.g., center of the screen).

In the following several interactive and view-dependent see-through lenses for massive 3D point clouds are presented and their specific advantages and disadvantages in different scenarios are discussed (Section 2). Building upon an existing out-of-core rendering approach for the classification-dependent visualization of massive 3D point clouds, it is shown how see-through lenses can be efficiently combined with existing point-based rendering approaches as a post processing step (Section 3).

2 Overview

Points may feature different, precomputed attributes such as an individual color, temporal information describing the date of collection, or the most likely surface category a point represents. These surface categories may range from basic ones such as indoor, outdoor, overground, subterranean, airborne, or terrestrial to more specific ones such as building, ground, or vegetation. These per-point attributes can be used to customize the appearance of points individually at runtime [10]. Hence, points that are related to each other (e.g., representing the same surface category or being created at the same point in time) can be identified more easily. If points are occluding each other, these attributes can also serve as a means to assess and highlight the significance of each point for a given use case.

2.1 Priority Levels

Depending on the application, use-case, and kind of information that needs to be explored, the significance of a point may vary. To describe the significance of a point the following priority levels are used:

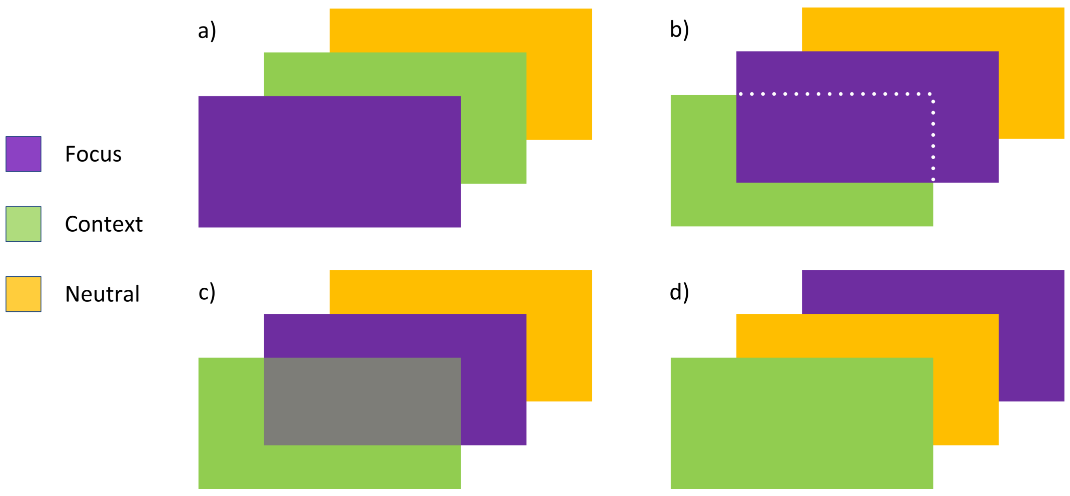


Figure 3: Illustration of compositions for different priority levels in occlusion situations. Context information is (b) masked out in favor of or (c) blended with focus information. Neutral information is neither highlighted nor filtered.

Focus Essential information of interest and exploration aim (e.g., interior objects occluded by walls, subterranean structures or buildings that have been demolished between consecutive scans). The more points carrying such information are occluded, the less information can be gathered, i.e., the less effective the exploration becomes (Figure 3).

Context Information that increases the overall realism of the visualization without being the main focus of the exploration (e.g., vegetation in densely forested areas). Points carrying such information can be safely masked out in favor of focus information (Figure 3).

Neutral Depending on the use case, users might want to focus on specific occlusion scenarios, such as solely on buildings that are occluded by vegetation. This requires to define all other information as neutral, i.e., points representing such information are treated as solid geometry that is neither masked out or highlighted (Figure 3).

These priority levels are used to define the composition of the data for the rendering and visualization of different occlusion scenarios (Figure 3): occlusions with the same priority level or including points carrying neutral information are solved by displaying the nearest point to the view position (i.e., similar to the default behavior in 3D rendering). Points carrying context information on the other hand (so-called context points) are masked if focus points are occluded.

2.2 Interactive and View-Dependent See-Through Lenses

Simply masking out all context points in occlusion scenarios limits the correct estimation of depth differences and does not provide information about the object shape and boundaries within a 3D point cloud depiction (Figure 3a). Blending context points and focus points by a certain factor addresses these limitations, however, areas with blended structures might be difficult to recognize during the exploration (Figure 3b).

2.2.1 Blueprints

Blueprints are a traditional form of technical drawings and known for their characteristic style which originating from the historical contact print process [9]. Construction elements are visualized by tracing outlines using different line widths and a color contrasting favorably with the background. Thus, the focus of the viewer is directed towards the most significant construction elements. This concept can be adapted to highlight areas where focus and context points have been merged by tracing the boundaries of those areas with a configurable line width and color (Figure 3c). This approach is especially effective to highlight changes within multi-temporal 3D point clouds (Figure 1).

2.2.2 Halos

A stronger emphasis on focus points can be obtained by masking out additional context points in a defined local proximity (Figure 5d). As a result, groups of neighboring focus points are surrounded with a halo effect, similarly to the real-world phenomenon and the technique used by artists throughout history to emphasize certain individuals. Techniques that highlight objects by removing occluders are known as cut-away-views [12, 14]. So far, they have not been applied to point-based rendering. The typical use case to apply this technique is the exploration of complex structures that are completely occluded by their surroundings, such as subterranean structures.

2.2.3 Interactive See-Through Lenses

All techniques that have been introduced are applied automatically to the data where focus information is occluded by context information. Naturally, the number, position, and extent of these areas varies depending on the view position. As opposed to that automated, view dependent approach, interactive see-through lenses (also commonly known as 'magic lenses') can be moved freely across the screen. Within an interactive see-through lens, context points are masked out completely, whereas in the surrounding no blending is applied at all (Figure 5f+e). This is required to focus on occlusions within certain areas whose position is known beforehand (e.g., to explore and show a former state for a certain area). However, if those areas are

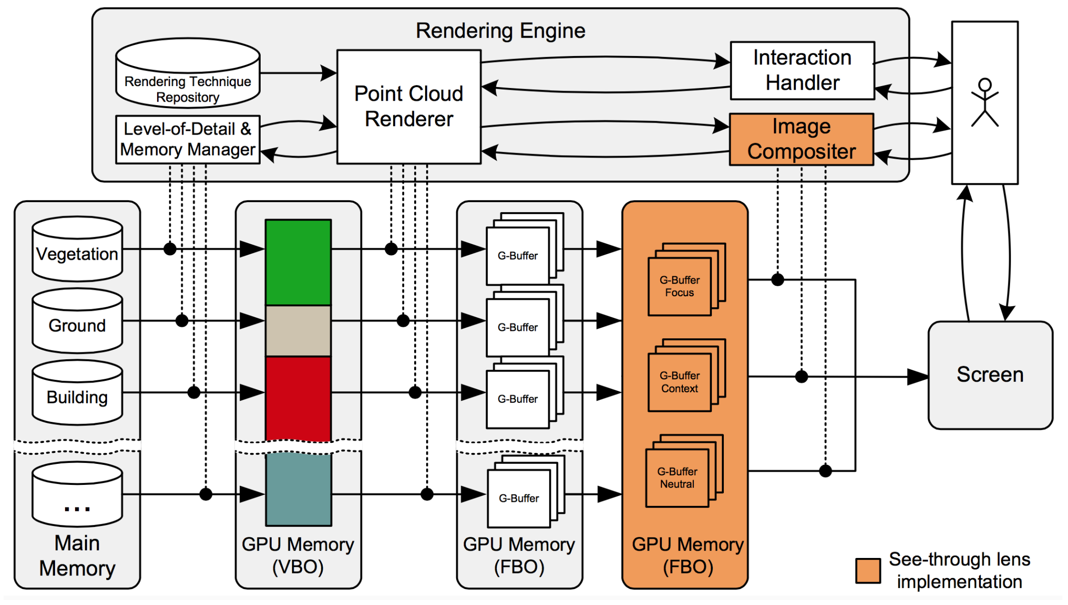


Figure 4: Schematic overview of the classification-dependent point-based rendering system by Richter et al. [10] and our modifications. Categorized by surface categories, points are transferred to GPU memory and rendered into separate G-Buffers that are merged based on the respective priority levels before being composed to synthesize the final image.

unknown, interactive see-through lenses are inefficient as the entire 3D point cloud has to be traversed manually to identify all areas.

3 Compatibility to Existing Point-Based Rendering Approaches

The proposed see-through lenses can be integrated into existing rendering systems for 3D point clouds. This is demonstrated by extending a rendering approach for massive 3D point clouds with surface category information introduced by Richter et al. [10] Figure 4. The approach is based on a layered, multi-resolution kd-tree to efficiently select relevant points of different surface categories for a given view position and rendering budget. To render selected points multi-pass rendering in combination with G-Buffers [11], i.e., specialized frame buffer objects (FBO), is used. Multiple 3D textures store information for color, depth, or normal values. For each surface category a separate rendering pass is applied, allowing to combine different rendering techniques. Originally, Richter et al. [10] use a separate G-Buffer for each surface category and apply a compositing pass to combine these G-Buffers into a final image. To apply see-through lenses, the compositing logic is extended. See-through lenses are implemented via programmable fragment shaders and can be activated and configured at runtime. To guarantee interactive frame rates even for massive 3D point clouds with a variety of thematic attributes and surface categories, the original approach was modified to have only one G-Buffer for each priority level (instead of one for each surface category). Thus, the compositing pass always combines three G-Buffers, independently from the overall number of surface categories within a 3D point cloud depiction. A performance evaluation conducted by Discher et al. [2] based on 3D point clouds from several real-world data sets of different size and point density showed the applicability of the proposed see-through lenses for arbitrarily large 3D point clouds.

4 Conclusion and Outlook

As shown in this report, the exploration of heterogeneous, time-variant, and semantically rich 3D point clouds can be facilitated by taking into account the relevance of a point for different use cases. The relevance of a point can be assessed based on a point's spatial position or any additional per-point attribute such as its surface category or timestamp. By introducing the concept of see-through lenses to interactive point-based rendering the visual recognition of relevant, occluded objects within a 3D point cloud depiction can be facilitated. The proposed see-through lenses can be integrated into existing rendering systems for massive 3D point clouds as additional post processing effects and offer many degrees of freedom for graphics and interaction design as they can be configured and selected at runtime. Therefore, they are highly adaptive and applicable to a variety of use cases and visualization tasks from different domains. Future work includes investigating how to efficiently apply alternative techniques to highlight occluded objects (e.g., multiple views or multi-perspective projections) to point-based rendering.

5 Acknowledgements

This work was funded by the Federal Ministry of Education and Research (BMBF), Germany within the InnoProfile Transfer research group "4DnD-Vis". I would like to thank virtualcitySYSTEMS and the Faculty of Architecture at the Cologne University of Applied Sciences for providing datasets.

References

[1] E. Canli, B. Thiebes, B. Höfle, and T. Glade. "Permanent 3D Laser Scanning System for Alpine Hillslope Instabilities". In: *6th International Conference on Debris-Flow Hazards Mitigation: Mechanics, Prediction and Assessment*. 2015.

[2] S. Discher, R. Richter, and J. Döllner. "Interactive and View-Dependent See-Through Lenses for Massive 3D Point Clouds". In: *Proceedings of the 3D GeoInfo 2015*. Accepted. 2015.

[3] Z. Gao, L. Nocera, M. Wang, and U. Neumann. "Visualizing aerial LiDAR cities with hierarchical hybrid point-polygon structures". In: *Proceedings of the 2014 Graphics Interface Conference*. 2014, pages 137–144.

[4] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. "An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees". In: *The Visual Computer* 29.1 (2013), pages 69–83.

[5] K. König, B. Höfle, M. Hämmerle, T. Jarmer, B. Siegmann, and H. Lilienthal. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *IISPRS Journal of Photogrammetry and Remote Sensing* 104 (2015), pages 112–125.

[6] F. Lafarge and C. Mallet. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *International journal of computer vision* 99.1 (2012), pages 69–85.

[7] F. Leberl, A. Irschara, T. Pock, P. Meixner, M. Gruber, S. Scholz, and A. Wiechert. "Point Clouds: Lidar versus 3D Vision". In: *Photogrammetric Engineering & Remote Sensing* 76.10 (2010), pages 1123–1134.

[8] S. Nebiker, S. Bleisch, and M. Christen. "Rich point clouds in virtual globes–A new paradigm in city modeling?" In: *Computers, Environment and Urban Systems* 34.6 (2010), pages 508–517.

[9] M. Nienhaus and J. Döllner. "Blueprint Rendering and 'Sketchy Drawings'". In: *GPU Gems II: Programming Techniques for High Performance Graphics and General-Purpose Computation*. 2004, pages 235–252.

[10] R. Richter, S. Discher, and J. Döllner. "Out-of-Core Visualization of Classified 3D Point Clouds". In: *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014*. 2015, pages 227–242.

[11] T. Saito and T. Takahashi. "Comprehensible rendering of 3-D shapes". In: *ACM SIGGRAPH Computer Graphics* 24.4 (1990), pages 197–206.

[12] S. Sigg, R. Fuchs, R. Carnecky, and R. Peikert. "Intelligent cutaway illustrations". In: *IEEE Pacific Visualization Symposium 2012*. 2012, pages 185–192.

[13] I. Tomljenovic, B. Höfle, D. Tiede, and T. Blaschke. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *Remote Sensing* 7.4 (2015), pages 3826–3862.

[14] M. Vaaraniemi, M. Freidank, and R. Westermann. "Enhancing the Visibility of Labels in 3D Navigation Maps". In: *Progress and New Trends in 3D Geoinformation Sciences*. 2013, pages 23–40.

Formal Approaches and Failure Cause Models for Software Dependability

Lena Feinbube

Operating Systems and Middleware
Hasso-Plattner-Institut
lena.feinbube@hpi.uni-potsdam.de

The importance of software dependability grows as we increasingly rely on complex software systems in our daily lives. This article compares four different approaches to formally guaranteeing that a software system obeys its specification. After highlighting the advantages and drawbacks of state-of-the-art formal methods, we shift our focus to failure cause models for real world software. The importance of understanding the details of environment-dependent fault activation is discussed in the context of fault injection testing.

1 Introduction

As software is becoming more and more ubiquitous in our daily lives, our dependence on it has also grown. Increased effort is needed to ensure its dependability. Software complexity is human-made, but nevertheless increasingly hard to handle. Modern software systems are next to impossible to grasp as a whole, let alone by a single person. The consequences of even the smallest bug may lead to a failure which is arbitrarily related in space, time and severity to the original fault [3].

The complexity of software arises from the following three factors: First, there is the internal state space of the program, given by all variables and the values they can possibly have. Second, there is the space of input and output values of the program, which may be infinite. Third, programs never execute in isolation but interact with their environment, being influenced for instance by scheduling, resource states and interfaces to other software components.

Formal approaches to software dependability research try to demonstrate the correctness of a software system in all cases. More precisely, *software verification* aims at proving that a program (an implementation) obeys a specification. Desirable properties of a system can be formulated either as *safety properties*: something bad should never happen, or as *liveness properties*: eventually, something good should happen.

Many efforts towards complete and sound software verification have been made, despite seemingly discouraging theoretical results: It is well established that not only is program termination undecidable [36], so is any nontrivial program property [33], which holds even for while-programs containing only two counter variables [29]. Thus, the quest for proving software correct may seem futile. In practice, much progress has nevertheless been made by reducing the state space under consideration. In contrast to hardware systems, the state space of software is infinite – leading to the above mentioned theoretical impossibility results. However, some realistic optimizing assumptions can be made to abstract and constrain the state space, making verification feasible.

This article attempts to provide a broad overview of existing formal approaches to software dependability. Sections 2 to 5 are largely based on the proceedings and insights from two Marktoberdorf Summer Schools [23, 24]. There, four broad classes of formal approaches are discussed with a focus on their applicability and impact in real software systems.

The article concludes by presenting own research on the topic of software failure cause models (section 6). In the author's opinion, a better understanding of software failure causes can aid in making the existing software dependability means – formal as well was experimental ones – more targeted and applicable in real-world scenarios.

2 Static Analysis

In the most general sense, static analysis is the reasoning about the behaviour of programs without running them. Since not all information about the execution environment is available before runtime, static analysis is always an over-approximation of the actual program behaviour. This means that some behaviours which can never occur in practice are assumed to be possible. In contrast, dynamic analysis, which builds upon traces from actual execution(s) is an under-approximation: some possible behaviours may not be captured.

Reasoning statically about the behaviour of programs is desirable because it can efficiently uncover bugs in early phases of software development, because it is relatively cheap and because details about the execution environment may not be known beforehand. Static analysis can find violations of safety properties and is also used to assert the reachability of desirable liveness properties.

A typical approach to achieve this is by *abstract interpretation* [8]:

1. Define an abstract domain where the concrete program values are mapped to. For the sake of performance, this domain is usually much smaller than the original domain. Termination is guaranteed for finite abstract domains, but infinite domains can also be used (together with additional convergence theorems).

2. Define an abstract language semantics in this domain. It needs to be clear how operators in the program transform the abstract values obtained from the concrete variables.

3. Apply the above defined operators on the abstract input until a fix point is reached, i.e. until no new abstract states are visited anymore.

The example in Figure 1 illustrates this process.

As already mentioned, the beauty of static analysis lies in the fact that it can be applied without running the program. This makes the approach both scalable and

```
1  int foo(int i) {
2    int x = 3;
3
4    while (i >= 0) {
5      i--; x++;
6    }
7    assert(x >= 0);
8  }
```

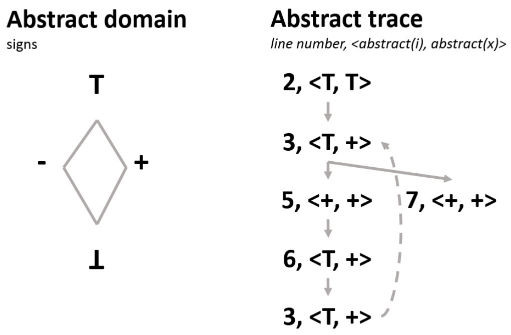


Figure 1: Example for abstract interpretation: in this program, the property in line 7 can be checked by mapping the concrete domain of integer values to the abstract domain of signs. Initially, i and x are both of unknown sign. After the initialization this changes and it becomes known that x is positive. Thus, transformers according to the program operators are applied to the symbolic values, until all states have been explored. It can be observed that no state where x is negative in line 7 is reachable, therefore the assertion holds true.

applicable to small code fragments early in the development process. Static analysis can be found in various bug-hunting tools [10].

Device drivers are a popular application area for most formal methods: they are obviously relevant to the dependability of the operating system, and they have a comparatively small state space. A prominent success story of static analysis is the SLAM project [2], where correct API usage in Windows device drivers is verified.

Because static analysis is inherently over-approximating, false positives are an issue which can hinder the applicability of purely static tools in practice [25].

Static and Dynamic Analysis It has been argued that static and dynamic approaches to program analysis should be combined more often, in order to leverage the advantages of both: the scalability and locality of static analysis, and the increased precision and avoidance of false positives of dynamic analysis.

One challenge here is that dynamic analysis requires the program or code snippet of interest to be executed easily. *Micro Execution* [13] is a technology which allows for the execution of arbitrary code without writing dedicated test drivers. The code is run inside a VM which intercepts all memory operations at the binary level and controls memory allocation. Custom memory policies can be defined. Therefore, micro execution is also suitable as a fault injection or fuzzing tool which works on any compilable code.

3 Software Model Checking

In contrast to static analysis, model checking assumes that a system is described formally – for instance, as an automaton or petri net. Model checkers can then answer the question whether this system satisfies a formal specification. If it does not, a counterexample is provided. A systematic, exhaustive exploration of the entire state space takes place, returning the counterexample as soon as a violating state is reached.

Model checkers inherently suffer from the *state space explosion* problem – the exponential increase in the state space caused by all possible combinations of variable assignments. Model checking originates from hardware research, where systems are usually finite state. Software, at least in theory, is inherently infinite state. Therefore, traditional model checking approaches need to be adapted before being usable for complex software systems.

Software model checking is the application of model checking to "real code". This means that source code written in commonly used languages is checked without requiring a manual translation to an automaton representation. There are two general classes of approaches towards automatically extracting a state space model from a piece of software [12]: The first class uses static analysis on the source code, whereas the second class explores the state space dynamically, e.g. by employing a runtime scheduler to cover many interleavings.

3.1 Verifying Concurrent Systems

With the advent of multi-core and parallel programming, verification of concurrent software has become an increasingly important topic. Model checking a concurrent program is computationally even more challenging because the possibility of many execution orders, depending on scheduling, further enlarges the state space. In this aspect, shared memory concurrency is harder to handle than message passing paradigms: while it is not obvious from the code which variables in shared memory are indeed modified by multiple threads, the length and amount of messages is generally well-defined and kept low to avoid latencies.

Current approaches to model checking concurrent software employ partial order reduction techniques (discussed below in 3.2) to handle the large number of possible interleavings, one example being the *Impact* algorithm [38]. Another idea – *bounded model checking* – is to search for property violations heuristically only to a certain depth in the transition system [28].

3.2 Partial Order Reduction

In contrast to hardware, software exhibits a much larger, theoretically infinite state space. Consequently, much effort is put into finding finite state abstractions. *Partial-order reduction* (POR) [1] is one such technique which reduces the state space. It is based on the observation there may be different orderings of concurrently enabled transitions which are equivalent. When exploring the state space to find property violations, it is sufficient to check one ordering from each equivalence class. Finding equivalent orderings is usually based on a notion of *dependence* between transitions. For instance, two write operations on a shared variable are mutually dependent, but two read operations are independent. Methods for pruning equivalent orderings during partial order reduction are based on stubborn sets [37], persistent sets [16] or ample sets [31]. Partial-order reduction is implemented in state of the art model checkers such as SPIN [4] and Verisoft [14].

Detecting dependence between variables can be hard in higher level programming languages or in the presence of pointer arithmetics. Either, a sophisticated aliasing model is needed, which complicates and slows down analysis, or conservative dependency analyses quickly make all variables potentially dependent, which hinders partial order reduction. Therefore, *dynamic partial-order reduction* [12] has been introduced, which monitors accesses to shared memory locations in a multi-threaded program during runtime. A backtracking algorithm then determines which additional schedules should be explored, while avoiding equivalent execution traces.

3.3 Compositional Approaches

To tackle the state space explosion problem, divide-and-conquer approaches to model checking have been discussed. One such approach [7] is based on the *Assume-Guarantee framework*. Here, the system is modeled as a composition of various subsystems. Thus, the model checking question becomes: does a composite system $M1||M2$ consisting of subsystems $M1$ and $M2$ satisfy a property p? Since software components rarely work correctly in isolation, assumptions about the environment need to be introduced to decompose the problem as follows:

1. Does $M1$ satisfy p under an assumption A?
2. Does $M2$ satisfy A?
3. Then, $M1||M2$ satisfy p.

To perform these checks, the system is usually modeled as a labeled transition system (LTS). The approach is limited to safety properties, which are also transformed to LTSs. Another central question is how to find the assumption A. Such assumptions are learned incrementally. A model checker performs the two checks mentioned above, starting with an empty assumption. If a counterexample is found in the first step, the assumption can be strengthened, meaning that fewer traces are allowed by A. On the other hand, if a counterexample is found in the second step, it is checked whether the counterexample composed with $M1$ satisfies the property. This would indicate a real violation of the property in $M1||M2$. Otherwise, the counterexample is used to weaken the assumption.

Compositional model checking is attractive because it reduces the state space which needs to be kept in memory at one point in time. Its application to behavioural UML models – which often represent larger software systems – has also been demonstrated [20].

4 Test Case Generation and Fault Injection

In software engineering practice, most faults are still found and removed by testing. Testing is an under-approximation of the system behaviour – meaning that some faulty behaviours may never be encountered by testing. Hence, one well-known issue is how to increase and maintain adequate test coverage. To achieve this, formal methods can be applied to build test drivers which systematically maximize coverage of the input space.

In large systems, software model checking can be applied as a testing and bug-hunting tool, which finds errors (counterexamples) even before it has exhaustively traversed the entire state space. Here, the term "model checking" is used in a much broader sense: the state space is not checked as a whole, but explored systematically. Triggering unlikely execution paths, as during dynamic partial order reduction, can also be viewed as a form of fault injection.

Verisoft [14] is an early tool for systematic testing which maintains a model of the system behaviour based on dynamically gathered information. Here, the assumption is that all externally relevant system behaviour is manifested as system calls, which are intercepted during runtime and analysed. The Verisoft tool then drives the system towards yet unexplored execution paths. Deadlocks, livelocks and violations of assertions are detected automatically. Verisoft is restricted to safety properties (no liveness properties), where a violation is detected when the system reaches a forbidden state during runtime. Verisoft has been applied with success in the telecommunication domain – both to exhaustively check smaller systems [15], and to inject many potentially fault-triggering schedules into a large phone call processing system [5].

Symbolic execution [26] maximizes path coverage during testing by harnessing SAT solving. The program is executed with symbolic instead of concrete values. At each branching condition, a path condition for the different paths is derived, from which new input values are generated. Under certain assumptions, this reduces the undecidable question of finding a test input covering some program statement to the decidable problem of finding a satisfying assignment for the path conditions. However, some functions, such as cryptographic ones, random number generation or simply unknown code, are too complex to be executed on symbolic values. To solve this problem, *concolic execution* techniques [34] combine symbolic with concrete execution, resorting to concrete execution when no symbolic value can be computed. Symbolic execution is implemented (among many others) in the prominent white-box fuzzing tool SAGE [17], which has been useful in identifying many Windows application bugs.

5 Program Synthesis

Instead of finding flaws in an existing program, program synthesis attempts to generate a correct implementation automatically from a formal specification. While program synthesis is becoming an attractive research topic due to the increased computational power available, its theoretical complexity is still restrictively high for most cases. The synthesis of a system from an LTL specification is $2EXPTIME$-complete. This currently prohibits synthesizing systems larger than a single controller, or a protocol implementation. One recent success has been the synthesis of the state-of-the-art AMBA bus protocol for embedded systems [18].

Many synthesis problems can be mapped to the solving of 2-player-games against an adversarial environment. State changes are represented by the players taking turns to decide their next action. A winning strategy against the environment corresponds to a feasible implementation [9]. Quantitative games make it possible to synthesize not only any program satisfying the specification, but programs which optimize for certain values, for instance minimizing the number of message sends in a protocol implementation.

A divergent, pragmatic approach to program synthesis can be found in the domain of end-user programming. Based on the observation that non-programmers should be aided more in completing repetitive tasks, *programming by examples* is the approach of synthesizing a small program from user-provided examples, which serve as a partial specification. It is mainly applicable in "data wrangling" scenarios, such as the generation of Excel scripts [21]. In order to synthesize correct software, i.e., code which corresponds to the users' expectations, the focus in this synthesis domain lies upon appropriate user interfaces and the design of simple yet sufficiently expressive DSLs from which to synthesize.

6 Fault Injection and Software Failure Cause Models

When designing dependable software, three aspects need to be considered:

- The **specification** of the desired system behaviour,
- its **implementation** – as code, which can be executed,
- the assumed **fault model**, describing what can go wrong during execution.

The previously discussed approaches largely focus on ensuring that the implementation obeys the specification. In practice, however, formulating complete and sound specifications is extremely demanding – one might argue that it is just as hard as writing correct implementations in the first place. In the formal techniques mentioned above, the fault model is often implicit or simply the "negation of the specification". In reality, however, specifications are usually partial in the sense that they do not cover all aspects of the desired system behaviour. The simplest and most common forms of specification in real-world software are a couple of assertions in the source code, or some API documentation.

The degree of specification and test coverage ultimately becomes question of budget: how much are we willing to pay for software dependability in terms of development and specification effort, computational complexity and hardware resources? Non-exhaustive dependability means such as fault injection, dependability modeling including uncertainty, and testing are cheaper alternatives to formal methods.

Rather than relying on a formal specification, such approaches most importantly require a well-defined fault model.

Even if exhaustive formal methods are applied to one software component, its interaction with foreign code, errors in the execution environment or malformed user input can still cause it to fail. Consequently, understanding *failure cause models* is an important step towards dealing with a major class of software failures – those caused by complex interaction, Heisenbugs [19] or code which is outside one's control. It has been noted that every test case selection strategy corresponds to a fault model [32]. For example, boundary testing assumes that faults tend to be activated at the boundaries of the input range, whereas random testing is based on the heuristic that fault occurrence is uniformly distributed across all inputs. We believe that models of software failure causes need to go beyond fault models which are purely input- or code-based. Besides the bug (fault) in the source code, the conditions leading to its activation, the resulting error state and failure severity should be studied for more efficient error detection, fault tolerance and targeted fault injection experiments. Table 1 illustrates the importance of distinguishing between fault, activation condition, error and failure.

To make this distinction more explicit, we have proposed an extension of the classical fault-error-failure concepts which also accommodates environment-dependent activation conditions [35]. Based on this terminology, a systematic literature study of software failure cause models [11] has been conducted, which reveals that models of static code flaws are prevalent in literature. Research on fault-activating state conditions and error states is rare.

In practice, even a thoroughly verified software system usually depends on environment components which are outside one's control and potentially untrustworthy. The *seL4* project [27], which built a fully verified micro-kernel operating system, has shown that thorough verification of an entire software stack requires immense development and maintenance effort. Consequently, experimentally evaluating system dependability, for instance using fault injection, will always remain relevant.

As Table 1 shows, the circumstances leading to an observable software failure can be complicated. *Fault representativeness* has been identified as a major challenge for fault injection [30]: which faults should be injected, and when? Should the focus lie on high coverage – potentially uncovering irrelevant corner-case bugs, or on realism and a notion of severity and failure modes?

We have implemented a fault injector [22] which operates at the interfaces to third-party libraries. Based on the community-maintained CWE database [6], several error classes, which are relevant in software engineering, are injected into a dynamically linked library. This library may not be available as source code, and misbehave in arbitrary ways.

In future research, I intend to use further fault injection experiments and software failure case studies to investigate the detailed mechanics of fault activation and error propagation in complex software systems.

Table 1: Examples of software faults, their activation conditions, resulting error states and failure modes

Name	Description	Fault (code)	Activation Condition	Error	Failure

CWE: Stack-based Buffer Overflow http://cwe.mitre.org/data/definitions/121.html	Missing range check: Omission	`#define BUFSIZE 256`
		`int main(int argc, char **argv)`
		`{`
		`  char buf[BUFSIZE];`
		`  // No check that input data`
		`  // does not exceed BUFSIZE`
		`  strcpy(buf, argv[1]);`
		`}`
			Input data exceeds intended buffer size	Overwritten data on the stack	Data corruption, security problem

CWE: Memory Leak http://cwe.mitre.org/data/definitions/401.html	Insufficient tracking of allocated memory	`char* str = new char [256];`
		`// ...`
		`// missing delete []`
			Leaving scope without deallocation	Memory allocated which can never be freed	Out of memory, loss of performance

CWE: Race Condition http://cwe.mitre.org/data/definitions/362.html	Missing or improper synchronization primitives	`// thread 1`
		`  if (global != 0) {`
		`  local = global;`
		`  //...`
		`}`
		`// thread 2`
		`if (someCondition) global = 0;`
			Problematic interleaving of concurrent operations scheduled	Unexpected state of shared data	Arbitrary

CWE: Improper Initialization http://cwe.mitre.org/data/definitions/665.html	Missing or improper initialization code of a variable	`char str[20]; // no initialization`
		`strcat(str, ”hello world”);`
		`printf(”%s”, str);`
			The runtime does not initialize the variable as expected	Unexpected variable content, undefined behaviour, or overflow	Arbitrary

GnuRadio: Error allocating memory https://gnuradio.org/redmine/issues/692	Missing handling of architecture-dependent memory alignment	`// missing check: alignment can be 1`
		`int err = posix_memalign(&ptr,`
		`                alignment, size);`
			Execution on ARM architecture	Invalid parameter passed to `posix_memalign`	Crash (Segmentation fault)

Linux kernel: Kernel panic on highly loaded webserver in `task_rq_lock` https://bugzilla.kernel.org/show_bug.cgi?id=27142	Missing wait, making race condition possible	`sma = sem_lock(ns, semid);`
		`// missing wait until`
		`// wake_up_sem_queue_do()`
		`// missing wait`
		`if (IS_ERR(sma)) {`
		`  error = -EIDRM;`
		`  goto out_free;`
		`}`
		
		`error = get_queue_result(&queue);`
			A semaphore array is removed and in parallel a sleeping task is woken up	Stale pointer	Crash/Kernel panic

Gnome: Pango segfaults on Korean example from testgtk https://bugzilla.gnome.org/show_bug.cgi?id=388702	No handling of corner case: unknown glyph in fallback code for missing font	`if (n_jamos > 0)`
		`  render_syllable (font, start,`
		`           n_jamos, glyphs,`
		`           &n_glyphs,`
		`           start - text);`
		`// missing handling of corner case`
			Input includes a specific character sequence and no Hangul fonts are installed	Invalid state in `render_syllable`	Crash

Firefox OS: Crash when turning on Bluetooth in Settings app https://bugzilla.mozilla.org/show_bug.cgi?id=875251	Missing call to push JSContext		User enables bluetooth	Passing of JSContext to a function in a bad state	Crash

PostgreSQL: Weak memory ordering bug http://www.postgresql.org/message-id/24241.1312739269@sss.pgh.pa.us	Race condition due to missing memory barriers	`for (;;)`
		`{`
		`  ResetLatch();`
		`  // missing barrier`
		`  if (/*work to do*/)`
		`    DoStuff();`
		`  WaitLatch();`
		`}`
			“a lot of contention for the cache line containing the flag, but not for the cache line containing the latch”	write to flag propagates to memory later than write to latch	Failure of the token ring protocol

References

[1] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. "Partial-order reduction in symbolic state-space exploration". In: *Formal Methods in System Design* 18.2 (2001), pages 97–116.

[2] T. Ball and S. K. Rajamani. "The SLAM Project: Debugging System Software via Static Analysis". In: *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '02. Portland, Oregon: ACM, 2002, pages 1–3. DOI: 10.1145/503272.503274.

[3] B. Beizer. "Software is Different". In: *Annals of Software Engineering* 10.1-4 (Jan. 2000), pages 293–310. DOI: 10.1023/A:1018999919169.

[4] M. Ben-Ari. *Principles of the Spin model checker*. Springer Science & Business Media, 2008.

[5] S. Chandra, P. Godefroid, and C. Palm. "Software model checking in practice: an industrial case study". In: *Proceedings of the 24th International Conference on Software Engineering*. ACM. 2002, pages 431–441.

[6] S. Christey, J. Kenderdine, J. Mazella, and B. Miles. *Common Weakness Enumeration*.

[7] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. "Learning assumptions for compositional verification". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2003, pages 331–346.

[8] P. Cousot and R. Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM. 1977, pages 238–252.

[9] L. Doyen and J.-F. Raskin. "Games with imperfect information: Theory and algorithms". In: *Lecture Notes in Game Theory for Computer Scientists* (2011), pages 185–212.

[10] P. Emanuelsson and U. Nilsson. "A comparative study of industrial static analysis tools". In: *Electronic notes in theoretical computer science* 217 (2008), pages 5–21.

[11] L. Feinbube, P. Tröger, and A. Polze. "The landscape of software failure cause models". submitted.

[12] C. Flanagan and P. Godefroid. "Dynamic Partial-order Reduction for Model Checking Software". In: *SIGPLAN Not.* 40.1 (Jan. 2005), pages 110–121. DOI: 10.1145/1047659.1040315.

[13] P. Godefroid. "Micro execution". In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pages 539–549.

[14] P. Godefroid. "Model checking for programming languages using VeriSoft". In: *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM. 1997, pages 174–186.

[15] P. Godefroid, R. S. Hanmer, and L. J. Jagadeesan. "Systematic software testing using VeriSoft—An analysis of the 4ESS™ heart-beat monitor". In: *Bell Labs Technical Journal* 3.2 (1998), pages 32–46.

[16] P. Godefroid, J. van Leeuwen, J. Hartmanis, G. Goos, and P. Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*. Volume 1032. Springer Heidelberg, 1996.

[17] P. Godefroid, M. Y. Levin, D. A. Molnar, et al. "Automated Whitebox Fuzz Testing." In: *NDSS*. Volume 8. 2008, pages 151–166.

[18] Y. Godhal, K. Chatterjee, and T. A. Henzinger. "Synthesis of AMBA AHB from formal specification: a case study". In: *International Journal on Software Tools for Technology Transfer* 15.5-6 (2013), pages 585–601.

[19] M. Grottke and K. S. Trivedi. "A classification of software faults". In: *Journal of Reliability Engineering Association of Japan* 27.7 (2005), pages 425–438.

[20] O. Grumberg, Y. Meller, and K. Yorav. "Applying Software Model Checking Techniques for Behavioral UML Models". In: *FM 2012: Formal Methods: 18th International Symposium, Paris, France, August 27–31, 2012. Proceedings*. 2012, pages 277–292.

[21] S. Gulwani. "Automating String Processing in Spreadsheets Using Input-output Examples". In: *SIGPLAN Not.* 46.1 (Jan. 2011), pages 317–330. DOI: 10.1145/1925844.1926423.

[22] L. Herscheid, D. Richter, and A. Polze. "Hovac: A Configurable Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications". In: *Proceedings of the 2015 International Conference on Software Quality, Reliability, and Security*. IEEE, 2015.

[23] M. Irlbeck, D. Peled, and A. Pretschner, editors. *Dependable Software Systems Engineering (Summer School Marktoberdorf 2014)*. NATO Science for Peace and Security Series — D: Information and Communication Security. IOS Press, 2015.

[24] *Verification and Synthesis of Correct and Secure Systems (Summer School Marktoberdorf 2015)*. https://asimod.in.tum.de/2015/index.shtml, to appear.

[25] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. "Why don't software developers use static analysis tools to find bugs?" In: *Software Engineering (ICSE), 2013 35th International Conference on*. May 2013, pages 672–681. DOI: 10.1109/ICSE.2013.6606613.

[26] J. C. King. "Symbolic execution and program testing". In: *Communications of the ACM* 19.7 (1976), pages 385–394.

[27] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. "seL4: Formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM. 2009, pages 207–220.

[28] D. Kroening and M. Tautschnig. "CBMC–C bounded model checker". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pages 389–391.

[29] M. L. Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.

[30] R. Natella, D. Cotroneo, J. Duraes, H. S. Madeira, et al. "On fault representativeness of software fault injection". In: *Software Engineering, IEEE Transactions on* 39.1 (2013), pages 80–96.

[31] D. Peled. "All from one, one for all: on model checking using representatives". In: *Computer Aided Verification*. Springer. 1993, pages 409–423.

[32] A. Pretschner, D. Holling, R. Eschbach, and M. Gemmar. "A generic fault model for quality assurance". In: *Model-Driven Engineering Languages and Systems*. Springer, 2013, pages 87–103.

[33] H. G. Rice. "Classes of Recursively Enumerable Sets and Their Decision Problems". In: *Transactions of the American Mathematical Society* 74.2 (Feb. 1953), pages 358–366.

[34] K. Sen. "Concolic testing". In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM. 2007, pages 571–572.

[35] P. Tröger, L. Feinbube, and M. Werner. "What activates a bug? A refinement of the Laprie terminology model". In: *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*. to appear. IEEE. 2015.

[36] A. M. Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *J. of Math* 58.345-363 (1936), page 5.

[37] A. Valmari. "Stubborn sets for reduced state space generation". In: *Advances in Petri Nets 1990*. Springer, 1991, pages 491–515.

[38] B. Wachter, D. Kroening, and J. Ouaknine. "Verifying multi-threaded software with Impact". In: *Formal Methods in Computer-Aided Design (FMCAD), 2013*. IEEE. 2013, pages 210–217.

Checks and Balances: Object-Constraints Without Surprises

Tim Felgentreff

Software Architecture Group
Hasso-Plattner-Institut
Tim.Felgentreff@hpi.uni-potsdam.de

Systems that integrate constraint solving with general purpose programming languages must strike a balance between exposing the full power of solvers and how aware programmers need to be of the solving process to write understandable and correct programs. To integrate with imperative languages, this balancing act extends to how freely programmers can express constraints versus how aware they need to be of the transition between the imperative and declarative parts of the program in order to understand its limits. Babelsberg is a framework that integrates constraints with a standard object-oriented language. Experience with Babelsberg, as well as with earlier constraint imperative languages, revealed the potential for complex interactions between constraints and the object-oriented core, in particular powerful constraints involving object identity, multiple types, and change over time.

In this work we present design principles that tame the power of the constraint solver and of the translation between imperative and declarative expressions, avoiding difficult corner cases and surprising solutions while retaining the key features and capabilities of the approach. We provide an informal description of these principles, with the full semantics given in a standalone technical report. We validate the utility of our principles by applying them to existing Babelsberg programs.

1 Introduction

Babelsberg is a family of object-constraint languages, with current instances being Babelsberg/R (a Ruby extension) [1], Babelsberg/JS (a JavaScript extension) [2], and Babelsberg/S (a Squeak extension) [5]. The Babelsberg design integrates constraint satisfaction with common object-oriented constructs such as inheritance and polymorphism, and inherits many properties from the earlier Kaleidoscope [10] and Turtle [6] constraint imperative programming languages. It is motivated by the observation that a number of key aspects of interactive applications can be concisely specified using constraints, but that other aspects are more easily specified using standard imperative constructs (assignment, loops, and so forth). Its goal is to provide both paradigms in a cleanly integrated fashion, within an object-oriented framework that respects standard object-oriented features such as messages, encapsulation, and inheritance. However, experience with Babelsberg has shown that interactions between the object-oriented core and constraints can be difficult to understand — it is easy to formulate constraints that allow the system to come up with surprising solutions, but also to write constraints that produce no solutions at all, because they would need to be formulated differently for the translation from imperative code to constraints to work as intended.

In this paper we propose a set of design principles to avoid such surprising solutions, along with a set of restrictions on the behavior of object-constraint programming systems that enforce these principles. The principles and restrictions are as follows:

- *Structure preservation*: Asserted constraints cannot change the structure of any objects (meaning the number and names of their fields). This property is enforced through a form of dynamic structural typechecking, along with implicitly generated extra constraints that implement frame axioms.
- *Identity preservation*: Similarly, newly asserted constraints cannot change what object a particular variable or field stores (such changes can only flow from assignments). This property is enforced by requiring constraints on object identity to already be satisfied at the point where they are asserted.
- *Structural/identity determinism*: The structure of objects as well as the particular objects stored by variables and fields must be allowed to change through imperative updates, and such changes can in turn cause existing constraints to be re-solved. However, the results of constraint solving are deterministic in terms of the final structures of objects and the identities of objects stored in each variable/field. This property is enforced through a novel two-phase solving process that first deterministically solves identity constraints and then solves the remaining constraints in the updated environment and heap.

2 Background

As an example to illustrate some of the kinds of surprising behavior that can arise when integrating constraints with an object-oriented language, consider a bank account application in which we want to prevent changes that would make the account balance drop below a certain threshold. Additionally, we want to track the current interest the account produces, but not allow it to rise above 1000 units of currency. (This example is simple enough that we can show the listing easily, but is designed to illustrate some complexities around object identity, as well as constraint features such as soft constraints and read-only annotations on variables.)

Listing 1: Ensuring domain requirements on a bank account

```
1 var acc = new Account(), minBalance = 100, curInterest = 0;
2 always: { acc.balance >= minBalance? }
3 always: { priority: "medium"
4        curInterest == acc.balance? * 0.01 }
5 always: { curInterest <= 1000 }
```

Here, the solver will ensure the property `balance` remains larger than `minBalance` after line 2. The read-only annotation on `minBalance` (indicated by the question mark) prevents the solver from changing it to satisfy the constraint. Thus, any change to the account that would drop the result of `balance` below `minBalance` is rejected by the system. Conversely, the system is not allowed to change the result of `balance` to satisfy the second constraint. The current interest is always determined from the current balance, but not vice versa. Note also that we assign a priority to the second constraint. This tells the system that it should satisfy this relation if possible, but it is no error not to do so. Due to the third constraint, if the interest were to rise above 1000, the system can leave the second constraint unsatisfied.

We discovered that Babelsberg would sometimes find unintuitive solutions that the human programmer did not even consider. Its design requires programmers to consider the solving as well as the translation process of the framework to use constraints effectively; even the small examples above contain the potential for surprising solutions.

To help address this, we define a set of restrictions on the language to rule out a significant number of such solutions. For example, in Listing 1, we intuitively expect the solver to reason about the balance of the newly created `Account` object. However, in the absence of additional restrictions there are other valid solutions. First, the solver might change the variable `acc` to point to another object that has the right balance. Second, the solver may correct the balance, but also add or remove fields from the account that are not relevant to the constraint. Third, even if the initial balance was already higher than `minBalance` and a change is unnecessary, the solver may still choose to make the balance equal to `minBalance`, as this also satisfies the constraint.

All three solutions are clearly undesired: imperative programmers do not expect the identity of the `acc` variable to change or the solver to change the object structure. Neither do they expect the property to change, if the constraint is already satisfied.

To address this problem, we restrict the power of the solver so it finds the expected solution in cases such as these. First, we separate constraints involving object identity from those involving only values, and solve those independently, so that a solution that changes the identity of `acc` would not be valid. Second, we use a non-standard form of structural type-checking at run-time to ensure the solver cannot add or remove fields from objects. Third, we implicitly add a low priority "stay" constraint to each variable, so that its value only changes if required by some higher-priority constraint.[1]

The Kaleidoscope language, particularly the early versions, was arguably too complex in part because it was too powerful in ways that were interesting, but not useful in practice. This made it difficult to understand what the result of a program might be and also difficult to implement efficiently. A heuristic for the present work is that when there is a choice, we favor simplicity over power.

[1]The formulation of the first two restrictions is new in this work; the third has been present in Babelsberg since the original language definition, as well as in the earlier Kaleidoscope design.

3 Solution Overview

A goal of this work is to clarify design decisions of Babelsberg to guide language users and implementers. However, it is not intended to be a complete description of a practical object-constraint language, and we omit aspects of the language that are inherited from the host language and that are not core to the interaction between constraints and object-oriented programming, such as class and method definitions, exception handling, and IO.

A guiding principle in our design has been to adhere to the original goal of Babelsberg: in the absence of constraints, it should behave like a regular object-oriented language. This work focuses on how object-oriented constructs need to be modified to support the Babelsberg design. Where there is a choice, we aim to make the smallest possible changes while still accommodating constraints in a clean and powerful way. Beyond this, we favor simplicity of the rules over support for interesting features that are rarely used in practice.

At the core of our design is the idea that the solver determines the entire heap and environment after each statement. However, for an imperative programmer it would be surprising if variables change more than they have to, or if objects unnecessarily lose or gain fields or change their types — the solver should always pick a solution that is "close" to the current state of the system.

3.1 Object Structure

Listing 1 includes a constraint on a field. However, in the absence of other restrictions, nothing prevents the solver from finding a solution that adds or removes fields of the object. Consider setting the `acc` variable to an empty object and asserting the constraint. Should the solver invent a balance field and constrain it?

To tame the power of the solver so that it does not (for example) invent new fields for objects, we add *structural compatibility checks* that are dynamically asserted before sending the constraints involving objects to the solver. They ensure, for example, that field accesses occur only on objects (and not primitive values), and that those objects have the necessary fields. These assertions, unlike constraints, are only checked — if one is violated it is an error and the constraint solving fails. The programmer must ensure that objects used in constraints have all the referenced fields.

3.2 Object Identity

A central issue in the design of an object-constraint language is the interaction between constraints and object identity. Object identity plays an essential role when using an object-oriented language to model aspects of the real world and so, in a language that integrates constraints with object-oriented features, we need to resolve the tensions between these fundamental features [9]. Additionally, making explicit identity constraints available to the programmer allows for useful capabilities such as specifying that two variables refer to the same identical object, or describing circular structures. However, experience with Kaleidoscope in particular suggests that allowing powerful constraints on object identity and types can lead to non-obvious consequences. As an example, consider the following program:

Listing 2: Constraining two variables to be identical

```
1 var account1 = {credit:  10}, account2 = {debit:  100};
2 always: { account1 === account2 }
3 account1.withdraw(1);
4 always: { account1.debit >= 0 }
```

This adds a constraint on line 2 that `account1` and `account2` be identical. However, it is not clear if `account1` and `account2` will afterwards refer to an account with debit balance or an account with credit balance (or both, or none). It is thus not clear which account we withdraw from in line 3 (or even if this variable still refers to an account object — the solver might let both variables might refer to `null`). If both now refer to a debit account, the constraint on line 4 will not pass our structural compatibility checks. If we allow the solver to change the identities again to make the second constraint valid, we may withdraw from the credit account first and then assert that the debit account be above 0. The choice of identity in one place would thus determine the meaning of the second constraint and introduce non-determinism and thus make the programs more difficult to understand.

To tame the power of constraints on object identity and type, we set the following goals for this aspect of our design:

a) Constraints on object identity must be declared explicitly using the host languages identity check method (=== in the case of JavaScript). They cannot be hidden within a method.

b) The solution to the constraints is deterministic as far as object identity and type are concerned — there should never be multiple correct solutions in which a given variable refers to objects with different identities in the different solutions. Thus, the programmer must first ensure the identity constraint is already satisfied, before asserting it.

c) Any change to the identities of the objects referred to by variables can only flow from an assignment statement — the constraint solver may not otherwise alter object identities.

d) To make Babelsberg programs more understandable for programmers, the identities of objects in value constraints should be clear. We thus first solve all the identity constraints (with a deterministic solution, in keeping with goal b)), and then the value constraints. In the second step, identities are fixed, and thus method lookup follows regular object-oriented semantics. This requires all identity constraints to be declared separately from value constraints.

To preview the results of our rules, the program in Listing 2 would then be illegal, since the identity constraint `account1 === account2` is not satisfied at the time it is asserted. If the programmer uses an assignment to first make both `account1` and `account2` refer to the debit account, the program succeeds; if the programmer makes them refer to the credit account, it halts with a structural compatibility check error; in either case it becomes deterministic and straightforward to reason about.

3.3 Structural and Identity Determinism with Stay Constraints

As in Kaleidoscope, Babelsberg adds an implicit `weak` stay constraint to each variable to keep it at its current value, if possible. This satisfies the desire that the solver should not change a variable, if no constraint on it is violated.

4 Evaluation

Our aim in developing these principles has been primarily a practical one, including clarifying the desired behavior of the language, providing a guide for language implementors, and proving useful language properties that then can be relied on by programmers.

As one form of evaluation, we have adapted the currently existing suite of example programs for Babelsberg/JS to the revised language. We demonstrate that nearly all of the programs continue to work without modification, or work with minimal adjustments. (These adjustments are described below.) Those programs that do not work anymore stopped doing so because they use constructs we explicitly decided to disallow. These programs demonstrate various aspects of constraint use in Babelsberg/JS, and include a simulation for a radial engine, a simple temperature converter, a simulator for electrical circuits, a generator for color schemes, and a layouting example.

Argument Checking One issue we encountered is that many LivelyKernel methods have, besides a return statement, some statements that check the number, types, or structure of arguments. Because allowing the solver to reason about the number and types of arguments would be a source of surprising solutions, such methods do not work multi-directionally. As an example, consider the frequently used method to add two points in LivelyKernel:

Listing 3: The `lively.Point.addPt` method

```
function addPt(p) {
  if (arguments.length != 1) throw ('addPt() only takes 1 parameter');
  return new lively.Point(this.x + p.x, this.y + p.y);
}
```

A future improvement to our design that may increase compatibility with existing applications may allow such statements that simply do argument checking, without adding them to the constraint. For now, although it is not semantically clean, the practical implementation allows such tests on arguments. The resulting constraints are not truly multi-directional, but in practice we think they exhibit the expected behavior.

Branching We encountered a similar issue with methods that return one of two expressions, depending on a test. Our new design principles does not allow branching in constraints, because the effect of solving and which branch is solved for is not always clear from looking at the constraint. One such method encountered frequently is `getPosition`:

Listing 4: The `lively.morphic.Morph.getPosition` method

```
function getPosition() {
   if (!this.hasFixedPosition() || !this.world()) {
      return this.morphicGetter('Position');
   } else {
      return this.world().getScrollOffset().
            addPt(this.morphicGetter('Position'));
   }
}
```

Moving the test outside of the method and adding the constraint only for the branch that is chosen fixes the issue for cases where the branch condition does not change.

Benign Side Effects Lazy initialization and caching are used in the constraints in some of the example applications. Prior work has explicitly allowed such benign side effects [1, 2], but we now disallow them, because we found their effect in constraints is often confusing. For example, the LivelyKernel method `Morph.getBounds` is used (often indirectly) in many of the examples (the code below was adapted to focus on the caching):

Listing 5: Shortened `lively.morphic.Morph.getBounds` method

```
function getBounds() {
   if (this.cachedBounds && !this.hasFixedPosition())
      return this.cachedBounds;
   // ... other code paths
   return this.cachedBounds = this.innerBounds();
}
```

A workaround that we use here is to call `innerBounds` directly, and circumvent the caching. For Morphs for which all child Morphs are contained within the bounds of their parent, this returns the same result.

5 Application to Related Work

There are a large number of related systems that integrate constraints with imperative languages, ranging from DSLs, to constraint satisfaction libraries, and to syntactic and semantic integration similar to Babelsberg. How these systems choose to balance the power of solvers and how aware the programmers need to be of the solving process differs greatly, as does their balance between how freely programmers can express new constraints versus how aware they need to be of the fact that they are writing constraints rather than ordinary statements in the host language.

Kaleidoscope [4] was an early constraint-imperative programming (CIP) language, and as such had to address many of the issues that arise when integrating declarative constraints with an imperative system that includes mutable state. Kaleidoscope uses soft constraints to ensure that, in the absence of other constraints, the solver does not change the values of existing variables. We adapted this in our principles. In contrast to OCP, Kaleidoscope separates methods that generate constraints from ordinary methods. This avoids having to give rules for when a method may work multi-directionally or only in the forward direction as we have done. The disadvantage is that it puts the burden on the programmer to develop and maintain two sets of interfaces, one for use in constraints and one for use in imperative statements.

Kaleidoscope'93 [10] also adds support for identity constraints, to express the distinction in object-oriented languages between object equality and identity. However, the Kaleidoscope system still allowed the solver to determine the identity of method arguments when solving value constraints, and it used multi-method dispatch to decide which methods to call in a constraint. In combination, while this gave the system much power, it also made it sometimes hard in practice to understand which solution the system might come up with. Babelsberg does not use multi-method dispatch, removing one dimension of freedom from the solver (as well as conforming to a more standard semantics for an object-oriented language). Furthermore, with our principles, the solver cannot change the identity or structure of arguments for value constraints, because identities are fixed and structure is typechecked before value constraints are interpreted. We believe the same solution could be applied to the Kaleidoscope design.

Turtle [6] is a more recent CIP language developed from scratch, while Kaplan [8] provides constraints in Scala. Both separate the declaration of *constrainable* variables from ordinary variables to make it clearer what may happen when a variable is used. Neither addresses issues of object identity, however. Like Babelsberg, the Turtle system provides constraint priorities; Kaplan does not. Because ordinary variables in Turtle are not determined by the solver, only constrainable variables have low-priority stay constraints on them. (Kaplan does not currently support constraints over mutable data types, so stay constraints are not relevant for it.) Analogous to Kaleidoscope and in contrast to OCP, both languages separate constraint functions from ordinary functions. In Kaplan, only such specifically annotated functions can be used in constraints. Turtle does allow ordinary methods and variables to be used in constraints; however, their values are treated as constants, making all ordinary methods work only in the forward direction. We believe the simple rules for using methods in constraints used in OCP could be used with these systems as well to allow a restricted set of ordinary object-oriented methods to be used in constraints.

BackTalk [13, 14] and SQUANDER [12] explicitly allow the constraint solver to determine object structure, and even to create new objects that satisfy a set of constraints.

While these systems have many differences, they are more similar to each other than to Babelsberg. Both use object-oriented methods in constraints merely as tests for the backtracking algorithm as it tries different objects and object structures as assignments for the constrained variables. Thus, it is a basic property of these systems to allow actions that we prohibit in Babelsberg. However, this does not represent such a problem in those systems, because the transition from declarative variables to ordinary variables is explicit, so the programmer can check for surprising results at that time.

Finally, *α*Rby [11] is a language that embeds the Alloy specification language [7] in Ruby. Its goal is to allow Alloy users to easily pre- or post-process their models using imperative libraries, for example to experiment with visualizations for Alloy models. *α*Rby translates Ruby programs into Alloy, but the programs are written in a DSL that closely mimics the Alloy language rather than using ordinary Ruby classes and methods. In contrast to Babelsberg, *α*Rby aims to provide imperative constructs to Alloy users, whereas Babelsberg provides declarative constructs to object-oriented programmers.

6 Conclusion and Future Work

We have presented design principles to control the power of the solver in object-constraint programming languages to avoid surprising or non-deterministic behavior by ensuring that object structure and identity are preserved when adding constraints, and that any changes to them are deterministic. A formalization of our principles in a natural semantics is about to appear [3]. With this semantics we also include two theorems that formalize two key properties of the first principle, namely that we never allow the solver to find solutions that would needlessly add or remove fields from existing objects. Our rules also ensure that the only way for a variable to change its structure is through an assignment statement — thus, solving constraints outside of an assignment can never lead to solutions in which variables change their structure.

These restrictions are stronger than the earlier versions we used in the practical implementations of OCP. They do allow us to express a wide range of useful programs, but we suspect we can do better, and a direction for future work is to relax some of them while still retaining clarity and precision.

References

[1] T. Felgentreff, A. Borning, and R. Hirschfeld. “Specifying and Solving Constraints on Object Behavior”. In: *Journal of Object Technology* 13.4 (Sept. 2014), 1:1–38. DOI: 10.5381/jot.2014.13.4.a1.

[2] T. Felgentreff, A. Borning, R. Hirschfeld, J. Lincke, Y. Ohshima, B. Freudenberg, and R. Krahn. "Babelsberg/JS". In: *ECOOP*. Springer, 2014, pages 411–436. DOI: 10.1007/978-3-662-44202-9_17.

[3] T. Felgentreff, T. Millstein, A. Borning, and R. Hirschfeld. "Checks and Balances: Constraint Solving without Surprises in Object-Constraint Programming Languages". In: *Proceedings of the 2015 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA'15. Pittsburgh, Pennsylvania, USA: ACM, 2015, pages 771–786. DOI: 10.1145/2814270.2814311.

[4] B. N. Freeman-Benson. "Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming". In: *ACM SIGPLAN Notices* 25.10 (1990), pages 77–88.

[5] M. Graber, T. Felgentreff, R. Hirschfeld, and A. Borning. "Solving Interactive Logic Puzzles With Object-Constraints: An Experience Report Using Babelsberg/S for Squeak/Smalltalk". In: *REBLS*. ACM, 2014, 1:1–1:5.

[6] M. Grabmüller and P. Hofstedt. "Turtle: A constraint imperative programming language". In: *RDIS*. Springer, 2004, pages 185–198. DOI: 10.1007/978-0-85729-412-8_14.

[7] D. Jackson. "Alloy: A Lightweight Object Modelling Notation". In: *ACM Transactions on Software Engineering and Methodology* 11.2 (2002), pages 256–290.

[8] A. S. Köksal, V. Kuncak, and P. Suter. "Constraints as control". In: *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*. ACM, 2012, pages 151–164.

[9] G. Lopez, B. Freeman-Benson, and A. Borning. "Constraints and Object Identity". In: *Proceedings of the 1994 European Conference on Object-Oriented Programming (ECOOP'94)*. Springer. July 1994, pages 260–279. DOI: 10.1007/BFb0052187.

[10] G. Lopez, B. Freeman-Benson, and A. Borning. "Kaleidoscope: A Constraint Imperative Programming Language". In: *Constraint Programming*. Volume 131. NATO Advanced Science Institute Series, Series F: Computer and System Sciences. Springer-Verlag, 1994, pages 313–329. DOI: 10.1007/978-3-642-85983-0_12.

[11] A. Milicevic, I. Efrati, and D. Jackson. "αRby–An Embedding of Alloy in Ruby". In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Volume 8477. Lecture Notes in Computer Science. Springer, 2014, pages 56–71.

[12] A. Milicevic, D. Rayside, K. Yessenov, and D. Jackson. "Unifying Execution of Imperative and Declarative Code". In: *33rd International Conference on Software Engineering (ICSE)*. May 2011, pages 511–520.

[13] F. Pachet and P. Roy. "Integrating constraint satisfaction techniques with complex object structures". In: *15th Annual Conference of the British Computer Society Specialist Group on Expert Systems*. Dec. 1995, pages 11–22.

[14] P. Roy, A. Liret, and F. Pachet. "A Framework for Object-Oriented Constraint Satisfaction Problems". In: *Computing Surveys Symposium on Object-Oriented Application Frameworks*. ACM, 2000, pages 1–22.

Towards Efficient Processing of Multi-Temporal 3D Point Clouds: Refactoring the Processing Workflow

Dietmar Funck

Computer Graphics Systems Group
Hasso-Plattner-Institut
dietmar.funck@hpi.de

Recent developments in remote-sensing technologies lead to an increased capturing frequency and wider coverage of our environment, making 3D point clouds more available and up-to-date. Applications such as change detection or other kinds of difference analysis benefit from the availability of 3D point clouds representing the same area, created at different points in time—typically denoted as *multi-temporal 3D point clouds*. A side effect is the large amount of data since multi-temporal 3D point clouds typically contain billions of points for each point in time and each point has a set of attributes. Service-oriented computing is a reasonable opportunity to face the trade-off between high requirements on processing hardware and keeping the costs of processing software and hardware within reasonable limits. In this report, concepts are presented to exploit the processing hardware and to increase maintainability of processing services to deploy and establish a variety of applications. The results of a case study for improving the efficiency of neighbourhood analysis are presented as well.

1 Introduction

Remote-sensing technologies are used to capture areas of interest for further analyses such as urban planning, deriving building or terrain models and change detection since decades. A large variety of systems is used for capturing, such as airborne (e.g., by using a plane), mobile (e.g., by using a car) or terrestrial (e.g., by using a tripod) systems. While laser-scanning (e.g., *Light Detection and Ranging*, *LIDAR*) was almost the only popular remote-sensing technology to generate 3D point clouds, in recent years dense image-matching approaches [5, 6, 7, 13, 16] are more and more in use. Although LIDAR technology has been improving with respect to accuracy and point cloud density since the early days, there are still some advantages of dense-image matching over LIDAR [10]: lower acquisition costs, higher point density and simultaneous acquisition of orthophotos and 3D point clouds. Therefore, dense image-matching makes even capturing of whole countries feasible and the capturing frequency can be increased. The availability of orthophotos and 3D point clouds enables analyses of forests [2, 21]., such as determining the species, health and age of trees, as well. The major disadvantage of dense-image matching to LIDAR is that dense-image matching tends to represent forest canopies overly smooth and ground under dense forest canopies is not captured [2, 17, 21]. Apart from that, remote-sensing technologies typically capture not much more than the point coordinates, which is why most applications of 3D point clouds require deriving of additional per point attributes and meta information (e.g., connectivity information).

A subset of 3D point cloud applications are based on difference analyses, which detect changes (e.g., in buildings, trees or landscapes in general) or differentiate between dynamic and temporary objects, on the one hand, and static structures on the other hand. The basis are *multi-temporal 3D point clouds*: a set of 3D point clouds representing the same area, created at different points in time. Since a single 3D point cloud typically consists of billions of points, algorithms and hardware for processing multi-temporal 3D point clouds have to deal with a large amount of data. However, customers are typically not interested in maintaining their own high-end processing software and hardware just for the processing of 3D point clouds. Service-oriented computing architectures offer a more flexible way for processing large amounts of point data since no software and hardware infrastructure specialized on 3D point clouds has to be maintained and idle times are avoided. However, traditional processing software is monolithic and developed for running on a local machine.

This report presents concepts to fulfil the following main requirements:

1. Providing processing on local as well as remote (e.g., server) machines.
2. Reusing frequently used tasks (e.g., neighbourhood analysis).
3. Reusing of frequently used derived attributes (e.g., neighbourhood information).
4. Providing solutions for different optimization goals such as quality and short processing time.
5. Providing concurrent processing whenever possible.

The central concept is to introduce a modular architecture. The first requirement aims at improving the maintainability and code reuse of processing software since the only difference between a local running software and a remote running processing service is the type of user interface. The back-end, e.g., importers, exporters, converters and analysers, can have the same implementations. Using a modular architecture it is easier to separate the user interface and the back-end.

The second and third requirement target different kinds of reusability: related to tasks, reusability has the meaning of *can be used multiple times and freely within a processing workflow by calling it whenever needed*. There should be as few restrictions for combining tasks as possible. The loose coupling between modules, which implement the tasks, within a modular architecture supports this kind of flexibility. Related to attributes reusability has the meaning of *is calculated once and can be used multiple times*. While the processing time can be reduced significantly, the memory consumption may increase. Therefore, the applicability of reusing attributes strongly depends on the specific attributes and applications.

The fourth requirement aims at providing flexibility and customizability for the construction of processing workflows. Depending on the available hardware and optimization goals, such as quality and short processing time, different kinds of algorithms may be suitable to fulfil a task within a processing workflow. The modular architecture can provide different and exchangeable variants of tasks, while minimizing the additional implementation effort. The fifth requirement points out the goal to provide for all tasks concurrent variants. Therefore, it is closely related to requirement four.

In the following the modular 3D point cloud processing architecture is described in further detail (section 2). The results of a case study for improving the efficiency of the fundamental, frequently used neighbourhood analysis are presented as well (section 3).

2 Modular Architecture for Processing 3D Point Clouds

This section describes the modular architecture for processing 3D point clouds in detail. An overview is shown in Figure 1. A processing workflow, e.g., classification or change detection, may consist of an arbitrary number of major steps, denoted as *tasks*. For example, a classification requires usually preprocessing including tasks such as outlier filtering or point normals estimation. Tasks are implemented as modules. Each module defines data constrains for input and output, e.g., point coordinates and normals are required as input and segment IDs are provided as output. Modules can be combined freely as long as the input and output constrains of the modules are met. Therefore, the input and output constrains specify the limits within modules can be combined. Tasks may contain subtasks which are unique (e.g., *Analysing Local Surface Membership*) and are implemented in the module of the corresponding task, as well as reusable and frequently used subtasks (e.g., *Gathering Neighbours*), which are implemented as stand-alone, auxiliary objects.

The tasks provided by the modular architecture can be grouped into three different major categories: importing & exporting, transforming and analysing. The following subsections provide further details for these three major categories.

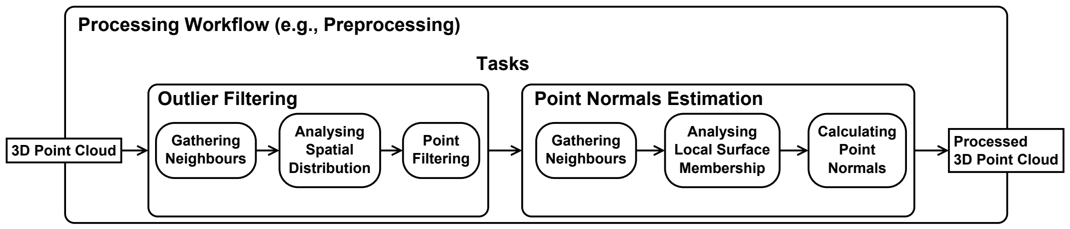


Figure 1: With the proposed modular architecture we can create processing workflows by combining often used activities called tasks (e.g. *Outlier Filtering*), which are implemented as modules. Tasks may contain subtasks which are either unique (e.g., *Analysing Local Surface Membership*) or reusable (e.g., *Gathering Neighbours*) to provide the desired result.

2.1 Importing & Exporting Tasks

The objective of the I/O tasks is to serialize and deserialize data as well as to translate between different file formats. There is a variety of file formats for 3D point clouds, some are specialized for the serialization or the CPU-based, respectively GPU-based analysis. The most popular file formats are XYZ, PLY and LAS. The XYZ file format stores point coordinates and optional per point colour in a plain ASCII mode. The PLY file format supports almost the same features as the XYZ file format: in addition, it supports a binary mode and storing per point normals. The LAS file format[1] is standardize by the *ASPRS* (*American Society for Photogrammetry and Remote Sensing*) and cannot only store point coordinates or per point colour, but also attributes such as acquisition dates or pulse information. Since all these file formats for 3D point clouds have their shortcomings, the modular architecture uses internally its own format. Therefore, importers are supported for all common file formats for 3D point clouds. There are importers for file formats which are often used for supplementary data (e.g., building footprints) as well.

Exporters are used to store the results in the original file formats or in a file format based on the internal format. Although exporting to external formats is required for compatibility with other processing software or viewers, the file format based on the internal format offers additional opportunities for visualizing 3D point clouds and their supplementary data. For example, there are attributes (e.g., distance to another 3D point cloud) which are directly supported by other file formats and there is no standardized file format for visualizing 3D point clouds out-of-core.

2.2 Transforming Tasks

Transforming tasks are a group of smaller, rather schematic, auxiliary tasks. The most important task is converting between different georeference systems since 3D point clouds as well as their supplementary data may be referenced by different approaches. First time georeferencing is usually already conducted during data acquisition. Therefore, the presented modular architecture assumes that the input data is already georeferenced. Further examples are bounding box based splitting of 3D point clouds, merging without consideration of point semantics, removing obsolete per point attributes or inserting the points into a spatial hierarchy to support out-of-core rendering and processing.

2.3 Analysing Tasks

Analysing tasks derive additional per point attributes or general information for 3D point clouds. The main categories are data filtering, general attribute generation

[1]http://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf (last accessed 2015-10-01)

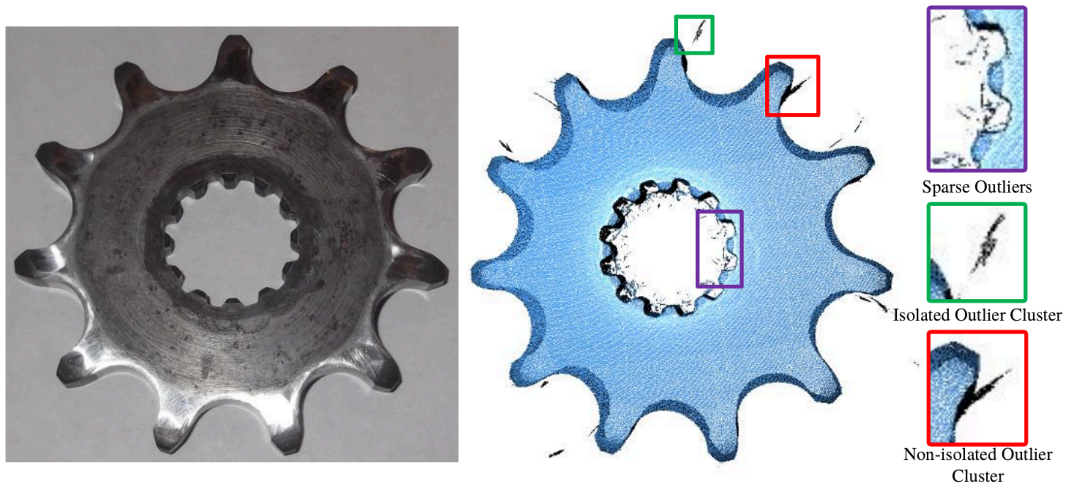


Figure 2: Outliers in a point cloud scanned from a gear model [20]

and difference analysis. The following subsections discuss these categories in more detail.

2.3.1 Data Filtering

There are often noise, outliers and duplicates in 3D point clouds due to inherent inaccuracies of remote-sensing technologies. While the impact of outliers and duplicates can be significantly reduced by data filtering during a preprocessing step, noise has to be treated by each analysing algorithm.

There is no generally adopted, strict definition of outliers in the context of 3D point clouds. Geometrically outliers can be roughly defined as follows: *"outliers in a point cloud are false measurement points that do not belong to the scanned surface"* [20]. Whether outliers can be filtered out depends on the type of outliers (Figure 2). Sparse outliers are isolated points which can be easily detected. Isolated outlier clusters are groups of points which are not close to the scanned surface and can be detected by using this information. However, detecting outlier-clusters which are close to the scanned surface is almost not possible since the object surface is typically unknown and estimation is inaccurate in the presence of extensive outliers. The difference from noise to outliers is that noise originates from measurement errors with small magnitude and is local, whereas outliers are large deviations from the surface [20]. A common approach to outlier filtering is to search for neighbouring points in a local point proximity (e.g., 2 m) [14]. If not enough other points are found, the point is defined as an outlier and not considered for further processing. A more sophisticated approach is presented by Yutao Wang [20].

Duplicates typically occur in overlapping captured areas since 3D point clouds are usually acquired by capturing the environment in several stripes. The filtering is conducted by comparing the positions of points within a small proximity (e.g., 0.01 m) [14].

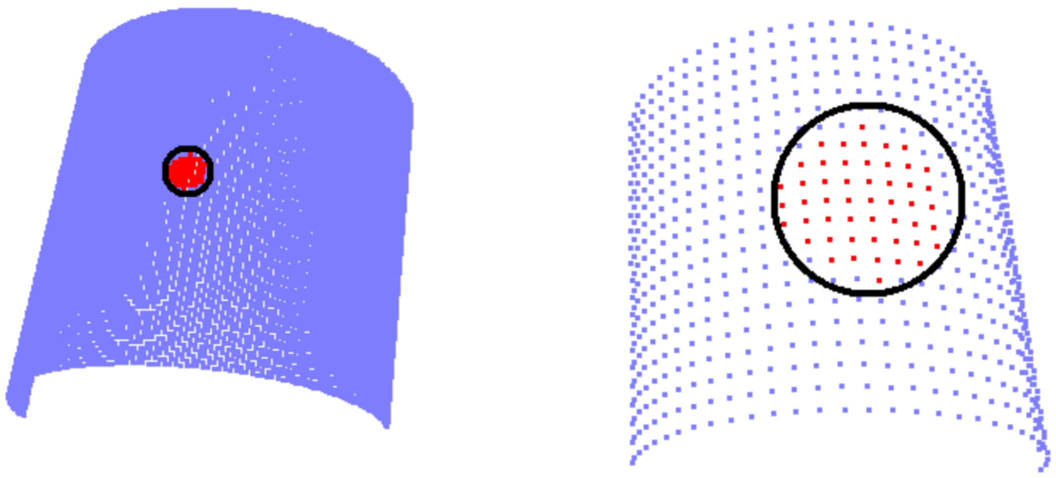


Figure 3: KNN adapts the region of interest for k-neighbours: areas with high density (left) result in a smaller region of interest, while areas with low density (right) result in a larger region of interest. [12]

Outlier as well as duplicates filtering are suitable for concurrent processing on the GPU.

2.4 General Attribute Generation

The neighbourhood analysis is the fundamental task of 3D point cloud analysis. Almost every analysis requires information about the neighbourhood of a point. Therefore, neighbourhood analysis is implemented as a reusable subtask, which can be used by every task. The most basic variants of neighbourhood information are the *K-Nearest neighbours* (*KNN*) and the *Fixed Distance Neighbours* (*FDN*) [12]. Since KNN uses a fixed k, the region of interest adapts to the local point density (Figure 3). Related to 3D point clouds FDN uses a fixed volume size for the region of interest. Typical volumes are spheres and cylinders [4]. While cylinders weight the upwards direction most and are suitable to detect upstanding structures, spheres are more computation efficient and favour no direction. In contrast to KNN, FDN variants are more suitable if the point density is almost constant within the 3D point cloud. Although KNN adapts to the local point density, the k has to be adapted to the average point density of the 3D point cloud to get a sufficient number of points, but no more. Besides, almost every kind of neighbourhood can be derived from a KNN neighbourhood of an appropriate size and shows the benefit of reusing results of a KNN search instead of recalculating it.

Especially robust point normal estimation algorithms [1, 19, 24] have their own definition of neighbourhood. In this context robust means that sharp features are preserved (Figure 4) and the influence of outliers and noise is minimized. The neighbourhood is defined as points belong to the same local surface and the point normal denotes the estimated local surface normal at the position of the point. Since the

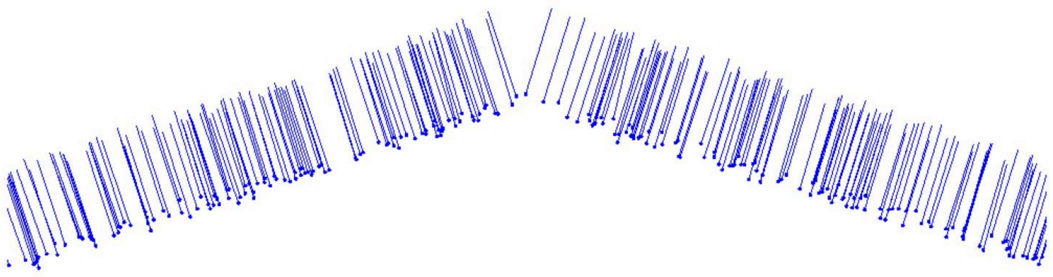


Figure 4: A sharp features is preserved by a robust point normals estimation approach [24].

local surface is usually unknown, estimation approaches are used. In the case of robust point normal estimation algorithms heuristically and stochastically approaches (e.g., the *Randomized Hough Transform* (*RHT*) [22]) are used. The main disadvantage of robust point normal estimation approaches is the larger processing time. It depends on the current application whether the quality of robust point normal estimation approaches or the short processing time of ordinary point normal estimation approaches are sufficient. Providing a variety of modules implementing different kinds of point normal estimation approaches, the most suitable approaches can be used.

Point normals are often used as an input for segmentation approaches. "Segmentation is the process of labeling each measurement in a point cloud, so that the points belonging to the same surface or region are given the same label." [12]. Therefore, segmentation is closely related to point normal estimation. The difference is the scale of surface estimation. While point normals estimation focuses on the local surface of a point, segmentation intends to group points of whole surfaces together. The estimated point normals are used as the main indicator for surface membership. Spatial proximity is usually used as an indicator as well. Region-growing based approaches are usually used to group points together. They require a neighbourhood analysis to collect spatially close candidates.

Segments are often used by object category detectors [14, 23] for examining the object category of a point since for a group of points which belong to the same surface more meaningful attributes can be derived than for a single point [18]. Typical object categories are building, vegetation and ground. Not for all segments can reliable object categories be estimated, e.g., for small segments. Therefore, the spatial relationship between segments with unreliable and reliable object categories is examined. This requires an iterative approach as well as a neighbourhood analysis.

Overall, the different flavours of neighbourhood analysis are the basis for almost all analysing tasks.

2.5 Difference Analysis

Difference analyses are specific to multi-temporal 3D point clouds. Typical kinds of difference analyses are change detection, merging and detecting different types of structures and objects regarding time. Point-based as well as segment-based approaches are used for difference analyses. For example, there is a point-based change detection approach to derive the difference between 3D point clouds measured in spatial distance per point [15] and a GPU-based implementation is suggested. However, results of point-based change detection are often not sufficient to be used directly for further analyses [8]. Segment information is combined with the point-based change detection results to derive more meaningful information, e.g., changes of buildings or landscapes.

Since multi-temporal 3D point clouds contain billions of points and require a large amount of memory, redundant structures are identified to reduce memory requirements. To detect redundant structures per point cloud segmentation and the spatial relationship amongst all segments is used. Voxelgrids [3, 9] aid by the examination of the spatial relationship. Typically redundant structures are static as well and dynamic objects are only part of one or very few 3D point clouds. Therefore, detecting redundant structures is linked to the detection of static structures and dynamic objects. Especially mobile acquired 3D point clouds may contain dynamic objects such as driving cars or pedestrians. Processing workflows such as deriving of building facades may require to leave dynamic objects aside.

3 Case Study

The neighbourhood analysis is a fundamental and frequently used task. Therefore, the question arises if a naive GPU-based implementation is sufficient to reduce computation time compared with a CPU-based implementation. The case study was implemented within a student project and a algorithms for counting neighbours within a spherical neighbourhood was used. Two GPU-based implementations were developed: a CUDA-based as well as an OpenCL-based. The CPU-based implementations is based on the *Point Cloud Library* (*PCL*)[2] and uses a spatial datastructure.

The test data is a 3D point cloud acquired in 2013 by using airborne dense image-matching with 100 points/m^2. The prototypical GPU-based implementation has no out-of-core mechanism. Therefore, a small region with ~142,000 points and an area of ~1,420 m^2 was used. The processing tasks was evaluated on a system with an Intel Xeon W3530 CPU with 4 x 2.8 GHz, 24 GB main memory, and a NVIDIA GeForce GTX 660 with 2 GB device memory and Debian Stretch 64-Bit. In Table 1, the average computation times for the three implementations and varying radii are shown. Since the kernel is updated automatically everytime when the code or the parameters (not the input data) have changed, the computation time of the first execution differs

[2]http://pointclouds.org (last accessed 2015-10-01)

Table 1: Average computation times (in ms) for counting neighbours within a spherical neighbourhood for the test area with different implementations and varying radii (in m)

Radius	CUDA	OpenCL	CPU
0.5	~ 635	~ 660	~ 1,020
1.0	~ 635	~ 660	~ 3,000
2.0	~ 635	~ 660	~ 11,300
5.0	~ 635	~ 660	~ 69,000

from the average computation time of subsequent executions. Therefore, the average computation times shown in Table 1 are only based on the subsequent executions. On average, the first execution took about 50 ms for CUDA and 1,800 ms for OpenCL. The reason for the large difference between both first execution times is that the OpenCL kernel is compiled during runtime, while the CUDA kernel is almost precompiled.

The computation times of the CUDA-based and the OpenCL-based implementation are only slightly different. Increasing the radius has no impact to both of the GPU-based implementations. In contrast, the CPU-based implementation is not only slower for all radii, but also significantly increasing. For the selected radii the CPU-based implementation is about 1.7 to 107 times slower than the GPU-based implementations.

To enable experiments with large datasets, out-of-core concepts have to be incorporated. There is a CUDA-based approach for detecting differences between points of multi-temporal 3D point clouds [15]. This approach transfers chunks of appropriate size to the GPU. It is stated as well, that the chunk size has to be chosen carefully. At first the computation times decrease while the chunk size increases until a minimum is reached. Afterwards, the computation times increase together with the chunk size.

4 Conclusions and Outlook

In this report concepts for the analysis of multi-temporal 3D point clouds were presented. It is shown that 3D point cloud processing workflows contain frequently used tasks, such as neighbourhood analysis, point normal estimation and segmentation. The presented modular architecture provides means to reuse and flexibly combine tasks for specifying processing workflows.

Future steps include the implementation of all tasks required for typical difference analysis workflows as well as providing them as a *web processing service* (*WPS*) [11]. An important issue is to integrate out-of-core concepts.

References

[1] A. Boulch and R. Marlet. "Fast and Robust Normal Estimation for Point Clouds with Sharp Features". In: *Computer Graphics Forum* 31.5 (2012), pages 1765–1774.

[2] J. Breidenbach and R. Astrup. "Small area estimation of forest attributes in the Norwegian National Forest Inventory". In: *European Journal of Forest Research* 131.4 (2012), pages 1255–1267.

[3] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. 1996, pages 303–312.

[4] S. Filin and N. Pfeifer. "Neigborhood Systems for Airborne Laser Data". In: *Photogrammetric Engineering & Remote Sensing* 71.6 (2005), pages 743–755.

[5] S. Gehrke, K. Morin, M. Downey, N. Boehrer, and T. Fuchs. "Semi-global matching: An alternative to LIDAR for DSM generation". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XXXVIII (2010).

[6] A. Grün. "Development and Status of Image Matching in Photogrammetry". In: *The Photogrammetric Record* 27 (137 2012), pages 36–57.

[7] H. Hirschmüller. "Semi-Global Matching — Motivation, Developments and Applications". In: *Photogrammetrische Woche* (2011), pages 173–184.

[8] Z. Kang and Z. Lu. "The Change Detection of Building Models Using Epochs of Terrestrial Point Clouds". In: *International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping (M2RSM)*. 2011, pages 1–6.

[9] L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. "Feature Sensitive Surface Extraction from Volume Data". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. 2001, pages 57–66.

[10] F. Leberl, A. Irschara, T. Pock, P. Meixner, M. Grubber, S. Scholz, and A. Wiechert. "Point Clouds: Lidar versus 3D Vision". In: *Photogrammetric Engineering and Remote Sensing* 76 (2010), pages 1123–1134.

[11] M. Müller. *OGC WPS 2.0 Interface Standard*. Open Geospatial Consortium. 2015.

[12] T. Rabbani, F. A. van den Heuvel, and G. Vosselman. "Segmentation of Point Clouds using Smoothness Constraint". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 36 (2006), pages 248–253.

[13] F. Remondino, M. Spera, E. Nocerino, F. Menna, and F. Nex. "State of the art in high density image matching". In: *The Photogrammetric Record* 29 (146 2014), pages 144–166.

[14] R. Richter, M. Behrens, and J. Döllner. "Object class segmentation of massive 3D point clouds of urban areas using point cloud topology". In: *International Journal of Remote Sensing* 34.23 (2013), pages 8394–8410.

[15] R. Richter, J. Kyprianidis, and J. Döllner. "Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds". In: *Transactions in GIS* 17.5 (2013), pages 724–741.

[16] T. Rosnell and E. Honkavaara. "Point Cloud Generation from Aerial Image Data Acquired by a Quadrocopter Type Micro Unmanned Aerial Vehicle and a Digital Still Camera". In: *Sensors* 12.1 (2012), pages 453–480.

[17] M. Vastaranta, M. Wulder, J. White, A. Pekkarinen, S. Tuominen, C. Ginzler, V. Kankare, M. Holopainen, J. Hyyppä, and H. Hyyppä. "Airborne laser scanning and digital stereo imagery measures of forest structure: comparative results and implications to forest mapping and inventory update". In: *Canadian Journal of Remote Sensing* 39.5 (2013), pages 382–395.

[18] G. Vosselman. "Point cloud segmentation for urban scene classification". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-7/W2 (2013), pages 257–262.

[19] Y. Wang, H.-Y. Feng, F.-É. Delorme, and S. Engin. "An adaptive normal estimation method for scanned point clouds with sharp features". In: *Computer-Aided Design* 45.11 (2013), pages 1333–1348.

[20] Y. Wang. "Outlier Formation and Removal in 3D Laser Scanned Point Clouds". PhD thesis. University of British Columbia, 2014.

[21] J. White, M. Wulder, M. Vastaranta, N. Coops, D. Pitt, and M. Woods. "The Utility of Image-Based Point Clouds for Forest Inventory: A Comparison with Airborne Laser Scanning". In: *Forests* 4.3 (2013), pages 518–536.

[22] L. Xu and E. Oja. "Randomized Hough Transform (RHT): Basic Mechanisms, Algorithms, and Computational Complexities". In: *CVGIP: Image Understanding* 57.2 (1993), pages 131–154.

[23] S. Xu, S. Oude Elberink, and G. Vosselman. "Entities and Features for Classifcation of Airborne Laser Scanning Data in Urban Area". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* I-4 (2012), pages 257–262.

[24] J. Zhang, J. Cao, X. Liu, J. Wang, J. Liu, and X. Shi. "Point cloud normal estimation via low-rank subspace clustering". In: *Computers & Graphics* 37.6 (2013), pages 697–706.

Utility-Driven Modularized MAPE-K loop architectures for Self-adaptive systems

Sona Ghahremani

System Analysis and Modeling Group
Hasso-Plattner-Institut
sona.ghahremani@hpi.uni-potsdam.de

A self-adaptive software is capable of evolving and modifying itself at runtime to obtain a specific level of quality of service and achieve certain functional goals. A MAPE-K loop, consisted of four adaptation activities (Monitor/Analyze/Plan/Execute) supported by a knowledge repository is an efficient way to employ a control loop in the adaptation engine. The adaptation strategies applying feedback loops are categorized as either rule-based approaches which follow an event-condition-action policy or search-based approaches which have certain goals they are banned to fulfill. Here we propose the idea of pursuing a hybrid adaptation strategy in which we benefit from the strong points of each approach and let them compensate for each others' weaknesses. We run a fast rule-based approach on the architecture model of the adaptable software and a utility-driven planning phase also runs in parallel to the rule-based approach and makes it possible to ensure some nice property for the executed rules such as optimality and highest expected utility of the system. We use the rule-based planning as a tool to conform system goals in a safe and fast manner. Triggering the adaptation process according to the occurrences of events rather than the state of the system also leads to faster adaptation of the software.

1 Introduction

A self-adaptive software is capable of evolving and modifying itself at runtime to obtain a specific level of quality of service and achieve certain functional goals [4, 7]. Therefore, self-adaptive systems that are able to adjust their behavior in response to their perception of the environment and the system itself, has become a trending issue in software engineering domain. Self-adaptation capabilities are recognized as an effective methodology to manage the increasing complexity and runtime challenges of many modern software systems which are either changing themselves, interacting with dynamic environments or their requirements are changing constantly which can not be forseen at design time. [3] models the self-adaptive software architecture as a hyper graph using graph grammars. [6] models self-healing systems by typed graph grammars applying systems rules. We follow similar approach and model the self-adaptive software as a graph. State of the system is represented as a specific graph structure and each occurrence of events in the system or the environment indicates formation of most likely new graph patterns in the architecture.

A multi-purpose self-adaptive software can be modularized into smaller single-purpose components each representing certain behavior of the system. In a modularized self-adaptive system, modules can trigger reconfiguration based on either the current state (current graph) or history or can adapt independent of the system state and only based on occurrences of events. Modularization can be beneficial when applying incremental adaptation in which changes only affect restricted components and are local. Here, modularity helps to avoid excess processing and this is important because in self-adaptive systems the whole reconfiguration process occurs at runtime and incrementality is the key solution [8]. Modules in an event-based self-adaptive system keep track of the changes occurring at each reconfiguration and do not require to be aware of the current state of neither the system nor the environment. In this study, we pursue an event-based adaptation of a modularized self-adaptive system. The adaptation policies however can be based on utility optimization or goal conformance which will be discussed further in the text.

2 Backround

2.1 Basics in self-adaptive systems

Self-adaptive software has developed the ability to observe the changes at runtime, reason about itself and autonomously evolve accordingly to fulfill its goals and expected requirements. To be able to do so, software needs to have certain characteristics which [10] describes them as "self-* properties". These properties are supposed to enable the software to evolve towards achieving the expected goals in spite of dynamic/runtime changes, which are the basis for self-adaptive systems. Considering the high cost of software management at runtime and the fact that anticipating and coding all the probable adaptation steps at design time is hardly feasible, applying a proper monitoring approach to trace the changes which is later followed by an adaptation plan to cope with new circumstances is the main idea behind self-adaptation [10].

[9] introduces an influential approach which structures the adaptation engine as feedback loop consisted of four adaptation activities (Monitor/Analyze/Plan/Execute) supported by a knowledge repository as depicted in figure 1. The monitoring component monitors the system through the sensors and gathers information on the occurring events and updates set of models in the knowledge repository accordingly. The analyze module investigates the models created in the monitoring phase and based on the system goals, decides whether there is need for adaptation or not, if so, the planning component is triggered to come up with an adaptation plan which includes one or several actions to deal with the rising software issues and the execute component is in charge of taking the plan to action and reconfiguring the software [14].

2.2 Goals and utility in search-based self-adaptive systems

A *search-based* self-adaptive system includes several goals which can be either *soft* or *hard*. [1] describes *soft goals* as those which there is no clear-cut condition to de-

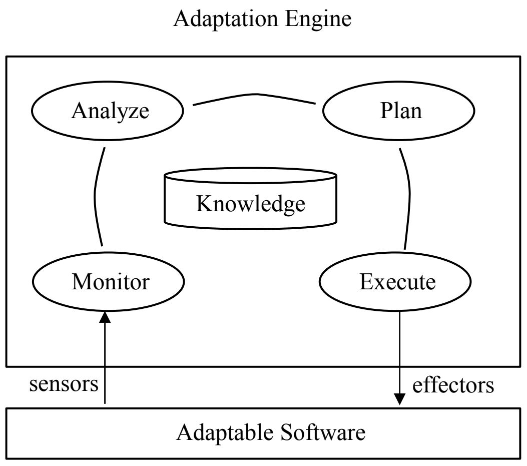


Figure 1: MAPE-K architecture

cide whether they have been satisfied such as the utility of a software or the energy consumption of a factory. *Soft goals* are typically used to model non-functional requirements of a system and we could always identify *how much* or *up to which degree* they have been satisfied/violated. The utility of a software system is a measure of relative satisfaction which tells us how good the system is operating and to which degree the *soft goals* have been satisfied, it can be representative of high level specification of system behavior [13].

Hard goals are the strict system goals which act like boolean variables, they are either satisfied or not, there is no *degree* in the level of fulfillment for *hard goals* in self-adaptive systems. In order to avoid any confusion and for the purpose of simplicity in the rest of this document, *goals* refer to *hard goals*.

2.3 Options for utility optimization

In a utility-based adaptation, all the reconfiguration efforts aim to maximize the total utility of a system so that the software can meet the required criteria. When the solution space is well defined and there are not many local optima, we can guarantee that local search-based approaches lead us to optimal utility, therefor *Hill climbing*-based approaches work. On the other hand, when there are multiple local optima, local search-based approaches do not work and we need other search heuristics to find the global optimum solution.

Defining the proper and precise utility function is of high importance since in an optimization problem, it is always the utility function that is maximized not the real utility value of the components. *Utility function* assigns a real-valued scalar desirability to each state which identifies how good or appropriate that specific state or configuration is as opposed to others. There have been plenty of research on *utility functions* and *utility-driven* decision-making policies, the notion of *multi attribute* utility function is a common term in the optimization context refers to a multi-variant utility function and is often considered as an implementation bottleneck for intelligent systems since it requires complex preference analysis among number of most possibly uncertain outcomes. So far plenty of schemes have been proposed for *multi attribute* utility functions but those following the Bayesian and Markov models seem to have achieved the most success in doing so. [2] introduces the notion of *conditional utility* and applies it for defining utility difference networks by satisfying additive analogues of the chain rule and *bays* rule which rely on their concept of *conditional independence* and results in a natural utility function elicitation.

[5] proposed an utility function elicitation method and tool, they measured both application-level and environment-level characteristics and generated multi-dimensional graph through automatically mapping the vector of application-level attributes which provide utility, and characteristics of its runtime environment and the curve which fits the best, provides the utility function. [15] evolved the utility function through genetic algorithm in a self-healing task, the evolution process included the training of specific genomes to identify the anomalous (undesirable) states.

There are also some design schemas on how to apply a *utility-driven* approach on control loops of a system, [13] presented a two-layer architecture for distributed self-adaptive systems consisting of independent modules. At the lower level, the scheme includes several separated *application environments* each having a *service-level utility function* providing particular service. The utility function of each module acts independently of the one of the others but sharing the same scale quantification. The system goal is to optimize the overall utility of the modules which means that the global optimization is distributed among several autonomic modules. Each module manages its own behavior and collaboration policy with other modules. At the higher level of the architecture, there is a global *resource arbiter* which is not aware of the details in the lower level and only takes as input the utility functions of the lower level modules. It acts as a global coordinator and periodically computes the optimum solution (resource allocation R^*) that maximizes the global utility of the system.

2.4 Modularization

Being a growing and complex system requires handling many runtime issues which means the adaptation engine must include several feedback loops each responsible for one issue at a time to be able to keep up with the speed of environment changes and software demands. However, control engineering defines the systems with only one feedback loop easier to reason about, but still the multiple feedback loops are more common [4]. In cases where the software is complex and requires many feedback loops, it makes sense to run them as separate individual modules and not as one big module with a central control on top of them. Following a proper engineering practice, the multiple feedback loops need to either be reduced to a single one ore be considered as independent individuals modules. Each module is a complete MAPE-K loop including monitoring, analyzing, planning and executing phase which operates on a knowledge repository which is normally a reflection model that reflects the adaptable software and its environment [12].

Decomposing a multi-concern module into several smaller ones requires determination of a proper ordering policy which schedules the modules in a way that they do not interfere with each other's output in an unwanted manner. There could also exist several implicit dependencies among modules which requires a particular order for executing the modules to achieve the desired output. These dependencies are either *static* (known at design time) or *dynamic* (rise at runtime). Inter-module dependencies lead to situations where if one module is executed before other one, the latter would interfere the output of the former. These dependencies can be interpreted as set of execute commands which need to be applied in the correct order. Here the system can benefit from adding a scheduler element to the system architecture which would be responsible to develop the proper ordering policy to execute modules or specific elements of a module.

3 Approach

3.1 graph architecture

Consider a sequence of architectural models of a complex modular self adaptive software where each of its modules include multiple components in different layers of architecture. What happens is that as a result of the software interacting with the observable domain (environment) it observes series of changes in the domain and some local changes in the software itself referred to as *regular behavior*. The software, being a self-adaptive system, expresses series of configuration changes as a response to the regular behavior which is called *adaptation behavior*. The architectural models can be presented as graphs in abstract level and the occurrence of certain patterns in this architecture indicates certain events and carries some observable information for us to notice. In this domain, the *state* is one concrete architecture pattern.

3.2 Goal conformance vs. Utility optimization

Goals are described as strict achievements system is required to gain, otherwise it would fail to meet its requirements. Goal-driven or goal-based self-adaptive systems are those which follow one or several goals and adaptation is required whenever a goal violation occurs. If a self-adaptive system is pursuing a *goal conforming* approach, it has a goal model which wishes to conform. In case of the graph structures, system goals are defined as avoiding/keeping certain graph structures identified by the general goal model. The reflection model needs to be constantly updated and monitored to see if the goals are fulfilled or not and in case of any goal violation, the adaptation is required to bring the system in the goal conformance zone where there are no banned structures in the graph representation of the system.

As described earlier, utility function of a self-adaptive system is an objective policy which expresses the value of each possible state of the system in its domain and generalizes the *goal policies* in a sense that they do not classify the system states, or in our case the possible system configurations in a binary manner into *desired* vs. *undesired* states. Following an utility-based adaptation requires monitoring the graph structures and ranking them based on their desirability and likelihood of occurrence. When there is the chance of applying a more desirable graph structure which is also most likely to result in an increase in the system utility, the adaptation engine is triggered to for a reconfiguration.

In the utility-based adaptation there is a utility function which maps any possible configuration of the system to a scalar value belonging to $[-\infty, +\infty]$. Here the optimal configuration gains the maximum utility and failing of constraints or violations of hard goals result in negative utilities which implies that the goal-conformance dominates utility optimization.

3.3 Modular utility-driven adaption

A behavior is a sequence of changes and considering the size of adaptable softwares we are focusing on, which are finite but not necessarily small architectural models, it is not possible to guarantee that we can be fully aware of the whole adaptation behavior of the system and all the possible sequence of changes proper for each probable situation. Therefor we are looking for some restrictions on choosing and applying the adaptation strategies to make it possible to apply them at the architectural level. Most complex software systems have scalability issues, in order to cope with them, optimization based approaches can be applied. These approaches use utility functions to conform the high level goals of the system and they to not explicitly capture adaptation rules. In our approach we pursue a hybrid adaptation approach in which we explicitly capture adaptation rules and in the meanwhile the utility of each rule is computed and later considered as dynamic weight to rank the rules based on their effectiveness and desirability regarding the circumstances under which they are triggered.

3.3.1 State-based vs. event-based adaptation

In general there can be strategies which are pure state-based, event-based or consider an accumulative past experience. In the state-based view, through considering the current state of the adaptation engine, the optimal strategy can be chosen among all possible options, but in the pure event-based view, it gets more complicated since each event has different effects in each state. Therefor it depends on the information each event provides and if there is enough, we can be pure state-independent. Being pure event-based or state-based also depends very much on how the utility function is defined. If an event occurs and decreases the utility of the system, we need to have a strategy which guarantees if we act to that event by applying a certain adaptation action, the local increase in utility out weights other local effects. In context of the graph structure, addressing the undesired graph pattern which leads to highest increase in the overall utility is the preferable choice. Here, having some given knowledge about the current situation enables us to determine which adaptation action has better effect (The greedy perspective). Switching between state-based to event based also requires a change in the monitoring policy, in the state-based adaptation we need to be aware of the global state of the whole system but in the event-based case we are only required to observe the *changes* in the system and environment and compute the delta.

3.3.2 General Formalization

We have a certain configuration and some environment or local changes represented as certain graph patterns which occur and it is desired to find out (1) whether the system is still optimal or it should adapt and if so, (2) how to proceed? In the worst case scenario, the cost of finding out whether the current configuration is the optimal one can be almost the same as finding better configuration.

We suggest to look for some phenomena or certain graph structures which tell us to optimize. We look for patterns that although we cannot know for sure that the performance is not optimal, but they still indicate there are some problems with the performance. It could also be the other way around, which means if we do not observe any signs triggering occurrence of certain pattern, we can conclude that the performance is good enough.

We define the overall utility function as a weighted sum of multiple sub-utility functions. The architectural model is presented as a graph and we are looking for certain graph patterns which are interpreted as occurrences of undesired events in the observable domain/system itself. For each module we define its related utility as a function of number of occurrences of the undesired patterns in its underlying components. As any change occurs in the architecture, we need to be able to detect whether there has been any change in the count of these certain monitored patterns or not. We count the occurrences of certain patterns and define the utility in proportion to that. Therefor $u = f(count)$ in which the variable *count* defines the number of occurrences. There are several scenarios to define function f:

- Utility is a monotonous function of count without any weight. $u = count$.
- Utility is defined as $u = \beta \times count$ where β is a negative factor which identifies the weight (importance) of the occurred pattern which later will be used as a factor indicating the dominance of the related component to others.
- Instead of having a fixed ordering defined by β in the second scenario, there can be a *re-ordering* or scheduling phase each time a change in the counts occurs. This scenario could be interpreted in various ways, it could be the case that frequent changes in a certain count means that the changes in the status of the related component are not that important and after four or five occurrence of change in the count we might ignore the sixth one. The frequent changes in the count can lead to increasing the weight of the related component and hence its priority because it might become a more serious issue as the count increases.

Each occurrence of the certain patterns in the model increases the count by one and applying the proper adaptation action to resolve it reduces the count by one. There for in this formalization, the maximum utility of a module would be $u = 0$ which is when there are no undesired patterns in that module or its observable domain. Under the assumption that resolving one pattern does not affect other patterns, we can say that the pattern with highest weight is the one which needs to be resolved first (this is of course when we do not take time into consideration). Therefor based on this formalization, a certain dominance relation should hold. The factor β multiply the *change* in the utility ($\beta \times \Delta u$), is always larger or equal other changes. In other words, we claim that if a change event occurs and there is a sequence of adaptation actions, action a is the local dominant if in this sequence the effect on utility caused by a is higher than or equal to other possible actions. In this formalization regardless of the state of the system, if any of these certain patterns occur, utility drops in proportion to the frequency of the occurrences, this allows us to assume that by applying the idea of counting certain structures we can be *state-independent*.

To guarantee optimality of the proposed approach we need to present some assumptions :

- After the occurrence of each event (pattern) we know which adaptation actions are applicable to resolve the issue.
- It is inherently assumed that applying one adaptation action does not affect more than one utility function.
- If the more than one pattern are occurring and we apply an adaptation action, that action would only resolve one issue at a time and alter the one specific related utility function.
- For each adaptation action we assume that we know how that will affect the utility.
- We assume that the future environment behavior and evolution of the system is not responding to the adaptation in a particular way and they are independent.
- The occurring patterns also should be independent from each other which means that one pattern cannot be included in another because then the applied adaptation action which resolves one would simultaneously resolve other one as well which would violate the first and second assumption. Therefor we assume that all the patterns are disjoint cases.

4 Planed Experiments

The rule-based adaptation can be developed as an event-based or state-based process. In case of event-based, occurrence of the events or symptoms trigger rules that eventually/directly result in reconfiguration requests. In the case of state-based, symptoms attached to the reflection models finally result in pre-planned reconfiguration in a reflection model by means of executing rules. In the state-based rule application, rules are triggered based on the current state of the system. A version of the state-based adaptation considers a history of the system states instead of one current state. Implementation wise, system's history can be encoded in the model. In this section we present our solution along with other adaptation scenarios which are planned to to be conducted, in each case a second dimension regarding the rule selection policy is involved.

4.1 Experimental setting

The adaptable software we use for experimental results is mRUBiS developed by [11], which is an auction site benchmark modeled after eBay. mRUBiS has a modularized architecture containing multiple components (modules) and implements the core functionality of an auction site: selling, browsing and bidding. This adaptable currently consist of to modules: self-healing and self-optimization both applying rule-based MAPE-K feedback loops. The applied adaptation strategy is focused on fulfilling the soft goals of high availability and low response time. The self-healing capabilities aim for automatically identifying and analyzing runtime failures in the running shops and for automatically planning and enacting an adaptation to heal these failures. On the other hand, self-optimization capabilities aim for automatically identifying and analyzing performance issues such as bottlenecks in the running shops and for automatically planning and enacting an adaptation to resolve these issues.

4.2 None incremental state-based adaptation

In none incremental state-based adaptation, for every change in the system, we have to run over the complete model and check if any of the rules are applicable. After finding the matches there are three scenarios to execute the the matched rules; Our approach suggests a utility driven selection of the rules, here we order these matches according to their utility values or the impact they they have on the overall utility which is represented as a dynamic weight for each rule and execute them.The alternative scenarios are applying the matched rules in random order or based on an static ordering which is defined at design time.

4.3 Type-incremental event-based adaptation

Here we have modular incrementality or as [8] describes it, syntax-driven incrementality where each event that occurs can be categorized as one of the recognized event types and then the rule checking process is limited tho class of rules which can only be triggered by these type of events. After a new exception occurs, the related rule set is checked to find and mark the triggered rules, the same scenarios as presented in the state-based adaptation will be applied here.

4.4 Incremental event-based adaptation

In this scenario we only look at the events occurring in the system or its environment regardless of the state of the system. Occurrence of events tigers the adaptation in this case, we search the rule set for the matches that only address that specific event and execute the rules following the three utility-driven, random or static ordering scenarios.

5 Conclusion and Outlook

The proposed approach is an incremental/type-incremental event-based adaptation approach which benefits from applying a utility function to optimize the reconfiguration of the adaptable software. Compared to the state-based adaptation, it is expected to have the benefit of less efforts and in case of the complex architectures where the scalability is a serious issue, our approach can show its benefits the best. The improvement with incrementally will make our approach noticeably faster. Regarding the utility-driven approach we proposed, which falls into the category of hybrid adaptation approaches, we introduce a utility function and use it as a dynamic weight to rate the rules. Comparing our approach with more straight forward solutions like static ordering of the rules or the random ordering, we claim that the dynamic weighting provides the opportunity to actively update the preferences among the rules and the orthogonality of all the overhead in computing and using the utility function is valuable. Finally we argue that in systems with simple architectures our solution will be competitive with other alternatives while in complex cases it will definitely perform better. Currently we assume all the rules are orthogonal and applying one does not change the applicability or the impact of the others which makes things much easier to handle, as the next step we would like to consider the case where this orthogonality does not exist anymore. In this case we know that a number of rules are not independent and the order in which we apply them has an impact on the overall utility of the system or applying one rule would change the improvement or the utility increase which is expected from another rule.

References

[1] L. Baresi, L. Pasquale, and P. Spoletini. "Fuzzy Goals for Requirements-Driven Adaptation". In: *Requirements Engineering, IEEE International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pages 125–134.

[2] R. Brafman and Y. Engel. "Directional Decomposition of Multiattribute Utility Functions". In: *Algorithmic Decision Theory*. Edited by F. Rossi and A. Tsoukias. Volume 5783. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pages 192–202.

[3] A. Bucchiarone, P. Pelliccione, C. Vattani, and O. Runge. "Self-Repairing systems modeling and verification using AGG". In: *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. 2009, pages 181–190.

[4] B. H. Cheng et al. "Software Engineering for Self-Adaptive Systems: A Research Roadmap". In: *Software Engineering for Self-Adaptive Systems*. Edited by B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee. Volume 5525. Lecture Notes in Computer Science (LNCS). Springer, 2009, pages 1–26.

[5] P. deGrandis. "Elicitation and Utilization of Application-level Utility Functions". In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC '09. New York, NY, USA: ACM, 2009, pages 107–116.

[6] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, and P. Pelliccione. "Formal Analysis and Verification of Self-Healing Systems". In: *Fundamental Approaches to Software Engineering, 13th International Conference, FASE 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010 Paphos, Cyprus, March 20-28, 2010. Proceedings*. Edited by D. S. Rosenblum and G. Taentzer. Volume 6013. Lecture Notes in Computer Science. Springer, 2010, pages 139–153.

[7] A. Elkhodary, N. Esfahani, and S. Malek. "FUSION: a framework for engineering self-tuning self-adaptive software systems". In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10)*. New York, NY, USA: ACM, 2010, pages 7–16.

[8] C. Ghezzi. "Evolution, Adaptation, and the Quest for Incrementality". In: *Large-Scale Complex IT Systems. Development, Operation and Management*. Edited by R. Calinescu and D. Garlan. Volume 7539. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 369–379.

[9] J. O. Kephart and D. Chess. "The Vision of Autonomic Computing". In: *Computer* 36.1 (Jan. 2003), pages 41–50.

[10] M. Salehie and L. Tahvildari. "Self-adaptive software: Landscape and research challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (2009), pages 1–42.

[11] T. Vogel. *Modular Rice University Bidding System (mRUBiS)*. 2013. URL: http://www.mdelab.de (last accessed 2013-11-02).

[12] T. Vogel and H. Giese. "Model-Driven Engineering of Self-Adaptive Software with EUREMA". In: *ACM Trans. Auton. Adapt. Syst.* 8.4 (Jan. 2014), 18:1–18:33.

[13] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. "Utility functions in autonomic systems". In: *Autonomic Computing, 2004. Proceedings. International Conference on*. May 2004, pages 70–77.

[14] D. Weyns and J. Andersson. "On the Challenges of Self-adaptation in Systems of Systems". In: *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*. SESoS '13. New York, NY, USA: ACM, 2013, pages 47–51.

[15] S. Wong, M. Aaron, J. Segall, K. Lynch, and S. Mancoridis. "Reverse Engineering Utility Functions Using Genetic Programming to Detect Anomalous Behavior in Software". In: *Reverse Engineering (WCRE), 2010 17th Working Conference on*. Oct. 2010, pages 141–149.

Editing Metamaterials, Creating Mechanisms

Alexandra Ion

Human Computer Interaction
Hasso-Plattner-Institut
alexandra.ion@hpi.uni-potsdam.de

Recent advances in fabrication technology, such as high-resolution 3D printers, allow fabricating objects with internal microstructure, also known as mechanical metamaterials. In this paper, we argue that the key to more complex metamaterials is to allow users to manipulate individual cells and sub-cell elements interactively. We make two main contributions. First, we present a simple system that converts metamaterials to (stacks of) images and back. This allows users to edit metamaterials using traditional raster graphics editors, such as Adobe Photoshop. This is efficient, but more importantly also allows users to engineer structures on per-cell level, giving them the fast and precise control required to develop new structures. Second, we use our own approach to create a new class of metamaterials: mechanisms, i.e., devices that transform forces and movement. Our mechanisms are all based on one specific type of asymmetric cell the only ability of which is to shear. We add these cells to our image processing approach; we achieve this by representing cells as color pixels instead. We then use our raster graphics approach to engineer a series of mechanisms each of which combines asymmetric cells in a different way, ranging from four-bars and hinges to door latches and pliers.

1 Introduction

Recently, researchers moved beyond merely designing the shape of 3D printed objects and started to also specify their interior. This has allowed them to create objects, for example, that stand [17] or spin [2]. Researchers in mechanical engineering and graphics have started to push the concept of internal structures even further. They explore how to design the microstructure of materials, typically in the form of repetitive cell patterns. These have been referred to as “metamaterials”. Researchers showed how to create objects with unusual behavior, such as objects that pull rather than resist when compressed (negative stiffness [6, 13]) or shrink in two dimensions upon one-dimensional compression [5, 11]. Unfortunately, designing new metamaterials is difficult and inefficient. Users currently structure by scripting it (e.g. using Matlab [13]) or by specifying forces or displacements [18]. While these approaches perform well for optimizing *already defined* structures, they are of limited use when exploring *new types* of metamaterials. In this paper, we tackle this issue. We demonstrate how to edit metamaterials interactively, using raster graphics editors. We demonstrate, how the resulting visual control allows us to implement a new class of metamaterials, i.e., mechanisms.

2 Editing Metamaterials, Creating Mechanisms

Figure 1a shows an example object we have created using the proposed image manipulation approach, a simple door latch made from a single block of metamaterial.

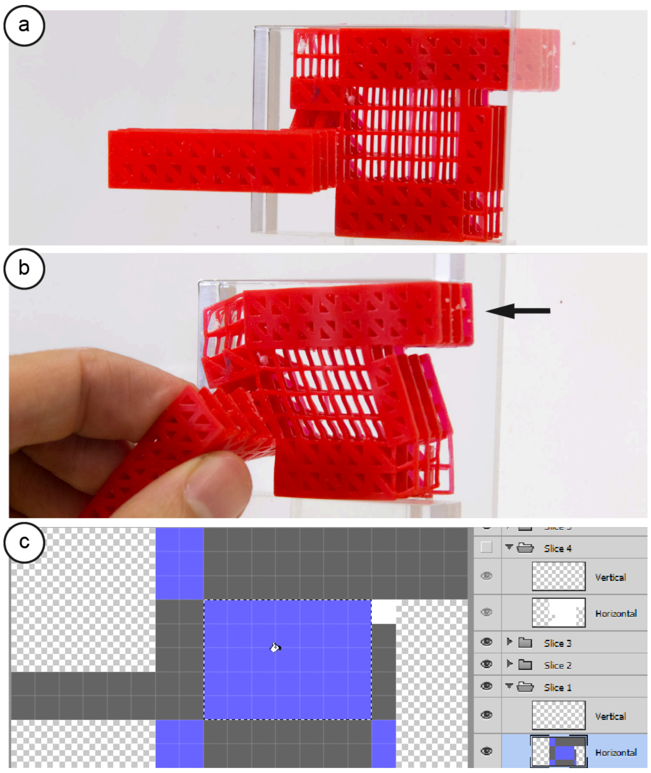


Figure 1: (a) A simple door latch made from a single block of metamaterial. (b) Rotating the handle causes the large cell rectangle in the middle to deform and pull the latch towards the left. (c) We created this mechanism by converting its outline to a stack of bitmap images, which allowed us to place with pixel accuracy in a bitmap raster graphs editor, here Adobe Photoshop. Here we just filled the center region with purple color, allowing it to shear left-right.

As shown in Figure 1b, the mechanism is functional, i.e., pushing the handle down causes it to rotate and pull the latch on the top right towards the left, unlocking the door. This behavior is the result of the interplay of several elements, a key element being the purple rectangular block of cells in the middle. It consists of cells with the sole ability to shear, which is what allows the mechanism to transform torque applied to the handle into the horizontal force on the latch required to unlock the door. The entire assembly consists of a single block of metamaterial—an arrangement of cube-shaped cells on a regular 3D grid. The functionality of the device is the result of the interplay between different types of cells. Figure 1c illustrates one step out of the process we used to create this mechanism. We are editing the latch in a bitmap raster graphics editor, here Adobe Photoshop. The latch is represented as a stack of layers (right) and we are currently editing the layer that contains the rectangular element discussed above. In the shown step, we are filling this rectangle with purple color. As we discuss in detail later, this eliminates the diagonal from these cells, allowing them to shear and thus perform their necessary mechanical function in the door latch mechanism. The raster graphics editor implicitly provides us with a visual overview of the inner workings of the device as we are editing it, allowing us to engineer the device directly in the editor. In addition, pixel-accurate tools allow us to assemble cells with the precision required to obtain a functional result. Figure 2 summarizes the workflow of our simple system. (a) Users start by modeling the outer shape of the latch mechanism in an arbitrary 3D modeling program, such as "Blender" (b) Users use our custom slicer to convert the 3D model to (multi-layer) bitmap file that contains one pixel for each voxel in the 3D model. (c) Users designs the mechanism using an arbitrary raster graphics editor, as discussed above. (d-e) Finally, users export the edited bitmap file through our custom mesh-generation pipeline based on OpenJSCAD [15] and (f) 3D print the model (we use a Titan 1 DLP printer with Spot-e resin).

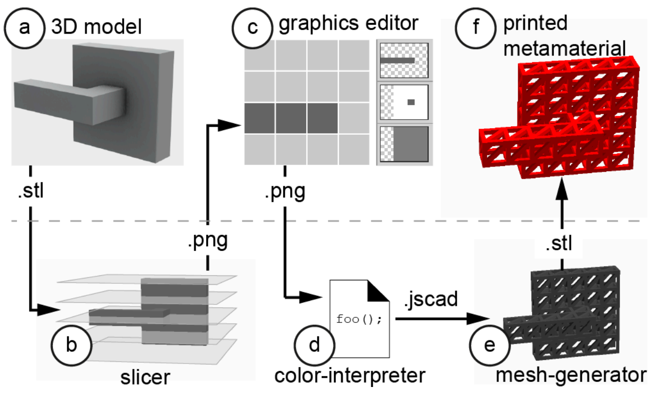


Figure 2: The workflow of our simple system

Although the door latch we printed is rather small due to the limited build size in current printers, we envision that in the future 3D printers would fabricate the whole door including the functional door latch in one piece.

3 Related Work

Our work builds on previous work in interactive personal fabrication, in particular on techniques that modify the internal structure of 3D printed objects, multi-material printing, and mechanical metamaterials.

3.1 Personal fabrication and User Interaction

Researchers in HCI and computer graphics explored how to define the behavior of printed objects. In particular they changed the rigid inside structure of objects in order to optimize the object's strength-to-weight ratio [10], to balance the object (e.g. "make it stand" [17]) or to allow it to spin [2]. One of the key contributions of our work is to allow users to interactively define the behavior of 3D printed objects. Since this requires users to interact with a large number of discrete elements, we are building on user interface concepts based on painting [3].

3.2 Compliant mechanisms and flexures

Our work builds on deformable struts that transfer forces. Similar mechanical structures have been examined in the context of compliant mechanisms, i.e. monolithic structures that transfer motion, force and energy without traditional hinges, but using flexure hinges (thin regions allowing for rotation) [9].

3.3 Mechanical Metamaterials

Metamaterials are artificial structures, usually repetitive patterns. Their unusual properties originate from their geometry, rather than the material they are made of. Researchers in the fields of mechanical engineering and material science discovered designs for negative stiffness materials through pattern transformation [13] (materials "pull" in the direction of compressing rather than resisting it) or materials with auxetic behaviour [5, 11, 19] (materials that compress in the orthogonal direction of the compression force too. These are properties that can only be achieved using metamaterials. These metamaterial structures were designed by a mathematical formulation of the desired behavior or empirical experimentation. The evolution of metamaterials has been further driven by recent advances in high-resolution 3D printing. An example of this is a printed material by Bickel et al. with structured pores that lowers the material's resistance to uniform compression, i.e. its bulk stiffness [4]. Recently, Schumacher et al. and Panetta et al. focused on elastic properties and created cell families where they computationally varied these properties [16, 18]. In these systems, users specify metamaterials either by applying a force to the object's boundaries in a 3D editor or by specifying forces (not cells) using color. Similarly, there are other editors that either optimize the stiffness of an homogenous internal structure [1, 12, 14] based on force vectors. We build on the idea of coloring objects with desired material behavior properties, but take it a step further, allowing for per-cell and per-edge control, which allows us to leverage powerful image-processing functionality.

4 Editing Metamaterials—in 2D

In this section, we present (1) the metamaterial structures we use in our tool, (2) how we map color values to material structure, and most importantly, (3) new types of metamaterials that the resulting framework suggests. We start our discussion with 2-dimensional materials.

4.1 Grayscale: Localized Compliance

A considerable amount of related work focuses on 2D materials, i.e. materials that consist of a 2-dimensional cell structure which can be extruded in the third dimension to obtain a desired thickness, e.g. [4].

A practical example is shown in Figure 3 where users produces a custom shoe sole, made from metamaterials, similar to [4], but using our bitmap representation: (a) users start from a 2D imprint of their foot from a pressure sensor. Then, they directly manipulate the cells by using standard image processing tools. (b) Users blur the image to create smooth transitions between pressure regions. (c) They add extra softness around the foot's corn by manually touching it up with various brushes. (d) At any time, users may export the bitmap image through the mesh-generator and obtain a 3D model of a custom sole with localized compliance. Note that this was done entirely using standard image manipulation (e.g. Adobe Photoshop) and did not require the typical scripting workflow currently involved in designing metamaterials.

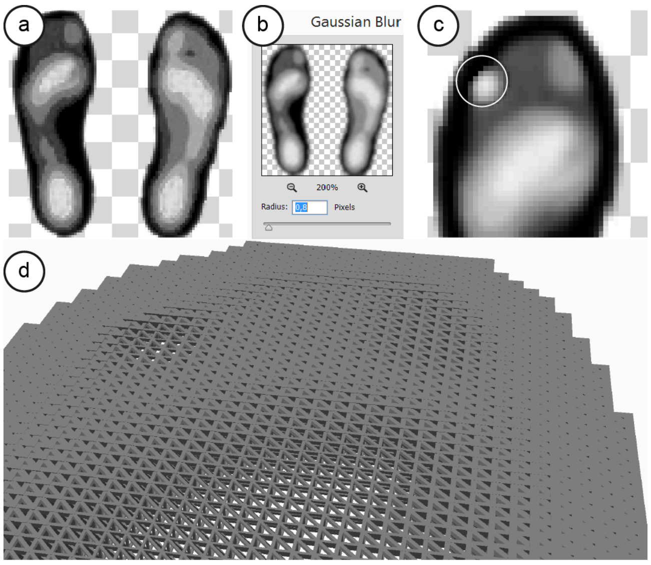


Figure 3: Users create a shoe sole by (a) importing pressure data. (b) They smooth it using Gaussian blur, and (c) retouch the toe area to relieve their corn. (d) Our mesh-generator create the 3D geometry of the metamaterial.

Because our approach leverages existing image manipulation tools, we can benefit from the many other image processing operations and macros that move pixels, such as scaling using different types of filters, translation by a sub-pixel amount, any type of band pass, including sharpening and directional or symmetric blurring, any type of distortion (e.g. "Liquify"), polar transforms, and so on. Our approach takes advantage of the correlation that adjacent pixels and cells share and leverages all these existing image processing tools as manipulators of physical matter for creating metamaterials. Image processing tools allow users to edit metamaterials efficiently. In addition, they allow for precision down to the edge level. This allows us to create new types of metamaterials, i.e. materials that implement mechanisms.

4.2 Color: Anisotropic Cells allow Making Mechanisms

Rather than manipulating the stiffness of cells as a whole, we will now use image processing to manipulate individual elements inside of cells, i.e. individual edges. We start out with a basic mesh structure that is stiff, i.e. resists compressive, tensile, and shearing forces. The minimal mesh that resists these forces consists of three edges per cell, as illustrated by Figure 4. To manipulate individual cell edges, we map each of the three edges to their own color channel; we thereby manipulate a piece of metamaterial by representing it as a RGB color image. We add degrees of freedom to our initially stiff mesh by weakening its beams as to allow it to deform. By manipulating the value of one of the color channels of a pixel, we manipulate the stiffness of one of its beam individually.

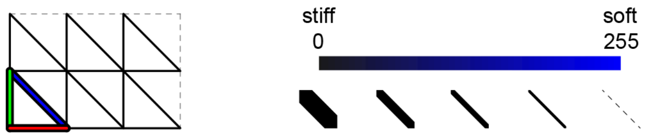


Figure 4: We map the three edges of our grid to a color channel each. The color value defines the stiffness of the corresponding beam, where the softest setting is not having the beam at all.

This is an important step, because it gives the user control over each beam individually and thus allows for creating anisotropic cells, i.e., cells that deform non-uniformly. As we demonstrate in the following, such cells are key to creating mechanisms, i.e., devices designed to transform input forces and movement into a desired set of output forces and movement. An interactive interface that offers access to individual cells is crucial here. While metamaterials with localized compliance can be designed by specifying a few control points and obtaining the rest through interpolation [1, 18], devising mechanisms from metamaterial requires per-cell and per-edge control.

4.3 The Four-Bar Cell Shears

Figure 5 shows the basic element for most mechanisms we have explored so far: the four-bar. The four-bar allows a piece of metamaterial to compress in a well-defined way. Since the individual beams in the cell support each other as they bend, the entire assembly shears in a controlled motion rather than buckling in an unpredictable way. Given that a cell of metamaterial may be arbitrarily small, a single cell cannot take much of a load. We concatenate multiple four-bars, resulting in an element that can take additional load. Similarly, we achieve additional compression by using multiple rows of such cells. The four-bar's ability to compress and shear in a controlled way matters, because it allows us to redirect a force, rather than simply turning it into deformation. This is the essence that allows us to create mechanisms.

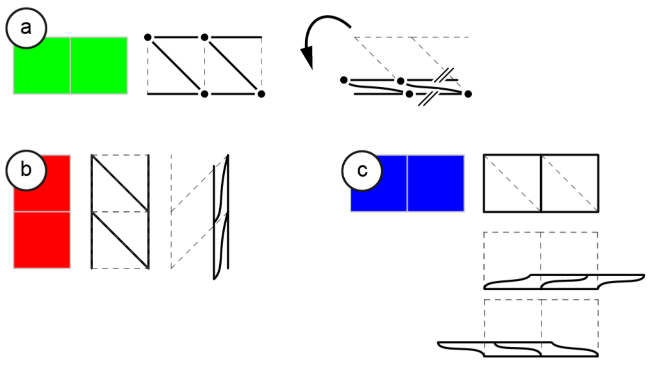


Figure 5: The four-bar. Its bitmap representation is always either red, green, or blue, as exactly one beam has been taken out, which we represent by setting the corresponding color component to 255.

4.4 Rotation: Hinges based on Four-bars

As illustrated by Figure 6, users can implement a hinge by using the four-bar element. (a) The naïve way of implementing a hinge is in the form of a single living hinge; however, with the same reasoning as before, a single living hinge can take only a small amount of force. (b) We address the issue by filling in larger region with four-bars. This replicates the hinge, allowing the resulting assembly to take a higher load.

In Figure 7 we used this concept to create a functional pair of pliers. This design connects each of four levers the pliers are made of to the four individual sides of the same four-bar array.

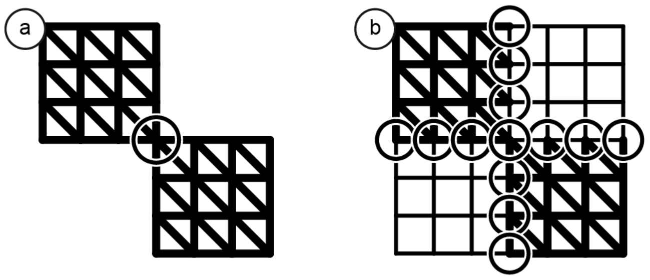


Figure 6: (a) Thin flexures allow for bending, (b) we add multiple flexure points to enable taking higher loads.

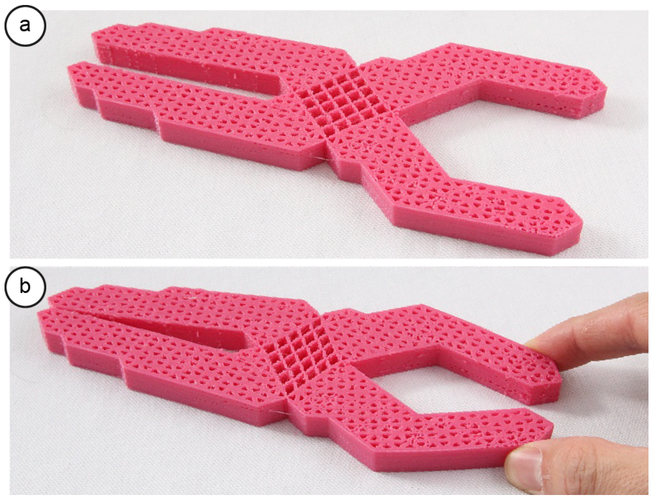


Figure 7: (a) This pair of pliers is based on a metamaterial hinge. (b) When we apply a force, the array of shearing cells in the center transmits the force, and the pliers close.

5 Editing Metamaterials—in 3D

We handle 3D metamaterials as a direct extension of 2D metamaterials. While editing a 2D metamaterial required manipulating a bitmap image, editing a 3D metamaterial requires manipulating a volumetric image, i.e. a 3D arrangement of pixels (or voxels). While there are a few volumetric editors in the graphics community (e.g. [21]), we would like to preserve the interaction model that served us well in 2D, i.e. use the powerful image processing tools. We thus instead slice 3D models into a stack of 2D layers. Figure 8 shows a simple example of a 3D object. Again, we are starting with a single parameter (stiffness) per cell. Here we use it to create a simplified notion of a button: essentially just a soft spot.

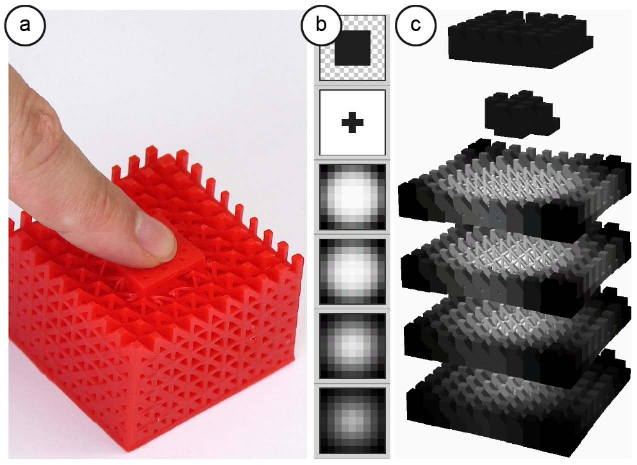


Figure 8: (b-c) Users create a simple push button by using the gradient tool to paint a soft (compressible) spot. Again, all image processing tool can be used. The layers are simply copied to add volume to the button. After exporting through our mesh-generation system, the user can (a) print the button.

5.1 Importing 3D Objects

As discussed earlier (Figure 2), our pipeline for 3D metamaterial objects imports 3D models by means of our custom slicer. As illustrated by Figure 9, this slicer breaks down a 3D model into equally spaced slices, creating a multi-layer bitmap image as output, i.e. one layer per slice.

The original purpose of layers in image processing tools is to allow users to break down the elements of an image (e.g. a collage) into individual layers, manipulate these individually, then collapse into a single layer upon export. In analogy, we use each layer to represent a different slice of a 3D model instead and at export we "collapse" the information that is contained in the layer into the 3-dimensional object. To represent the actual shape of the 3D object, our slicer generates a so-called layer

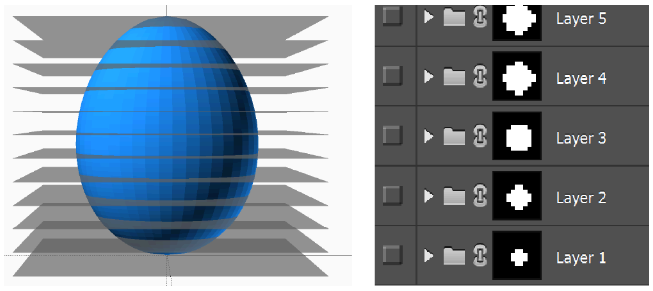


Figure 9: Our slicer slices an importaed 3D model along its z-direction and the outline of each slice is used as a layer mask.

mask to represent the actual shape of the object. Layer masks are a standard image processor feature; we use them here to prevent the user from creating visible contents in the regions located outside the object geometry, thus preserving the user's visual thinking. Our system's import functionality, including the generation of layer masks, also works for 2D objects.

5.2 Anisotropic 3D Metamaterials Based on Color

In order to create 3D mechanisms we again choose the model with the smallest number of edges. In 3D, this means six edges. As shown in Figure 10a, we have the three edges that together form a 2D square cell; on top of that there are three edges that connect the triangle, resulting in (b) a cube mesh with diagonals. In analogy to the three-edge cells in 2D, the 6-edge cells in 3D are minimal, i.e. removing any edge results in a material that is not stiff anymore. Ideally, one would use 6 color channels to store our 6-edge cells, as depicted in Figure 10a. However, we now have more edges than the number of color channels the human eye can perceive (Adobe Photoshop supports multi-channel images; unfortunately, these are restricted to a single layer though). Thus, to allow users to manipulate the 6 edges, we divide them into two layers: one for vertical axis and one for horizontal, each featuring three RGB channels as depicted in Figure 10b.

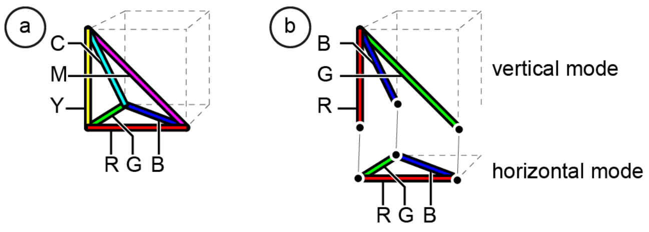


Figure 10: (a) Minimal 3D square cells are based on 6 edges and would ideally be stored in 6-channel bitmaps. (b) We store them in two RGB layers.

6 Extending to Meshes from Related Work

So far, we covered examples in which metamaterials were made from square/cubic or triangular/tetrahedral meshes that were edited using our image processing approach. However, the system we propose is modular and not limited to these meshes. By replacing the mesh-generator with one that, for instance, creates parametrized holes [6] (Figure 11), we can edit a very different mesh using the same image processing tools. This allows us to edit a variety of meshes shown in related work.

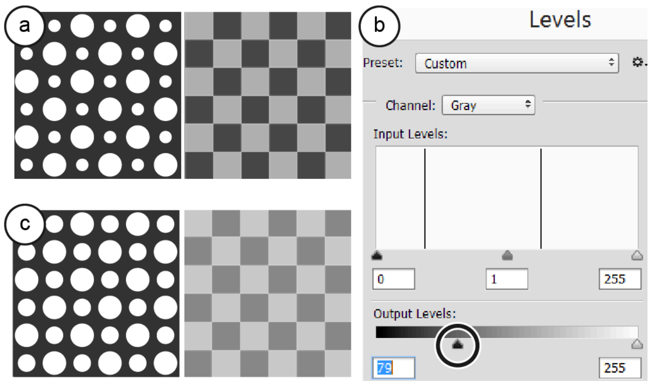


Figure 11: (a) The "auxetic" mesh [6] is characterized by an alternating pattern of small and big holes, which in our approach renders as a checkerboard. (b-c) Using the "Levels" dialog, we adjust the hole diameter to tune the material.

Figure 12 shows a range of exemplary meshes from the related work [11, 20] and the parameters that are mapped to color and varied by image processing.

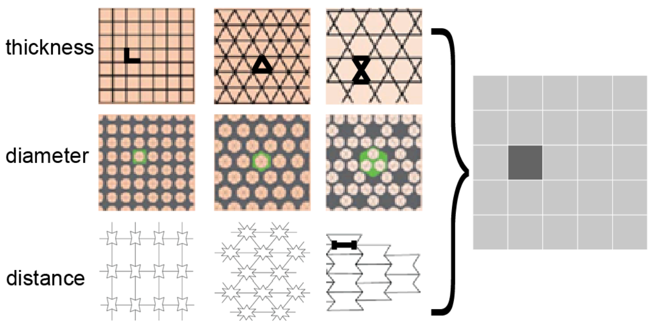


Figure 12: Examples of different meshes from the related work that can be generated with our system

7 System Implementation

To help readers replicate our approach, we use the following section to provide details of our software implementation. In the following we describe the internal processes implemented in our software to generate metamaterials from an image processing software, in our case Photoshop CC 2014. We offer an extension for Photoshop, which handles: import of .stl files, mesh selection, and export. Our extension builds on Adobe's Common Extensibility Platform 5.0 using a HTML5 engine and Node.js.

7.1 Import and Voxelize

Our system imports .stl files to be edited in Photoshop. With our mesh generator, we support two types of meshes, namely a cubical, where the voxels are stacked, and a triangular mesh, where cells undergo a phase shift. If the imported 3D object should be edited on a triangular mesh, we shear each vertex of the .stl model by

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x + \frac{y}{2} + \frac{z}{2} \\ y \cdot p + z \cdot r \\ z \cdot h \end{bmatrix}$$

where r is the inradius and p is the height of a equilateral triangle ($r = (\frac{1}{6}\sqrt{3})$, $p = (\frac{1}{2}\sqrt{3})$), and h is the height of a regular tetrahedron ($h = (\frac{1}{3}\sqrt{6})$). After shearing, we voxelize the model based on Bresenham's line algorithm, which we use to find voxels on each face edge in the .stl file. This way we find the voxels for the outer shape of the 3D model. We fill the inside of the model with voxels by evaluating the average of the face normals [22].

7.2 Voxels to Layered Bitmaps

After voxelization, we return the voxel data to our Photoshop extension. We iterate over the voxels layer-wise along the z-axis and generate black and white bitmaps consisting of white pixels for each voxel. This bitmap is masking the generated Photoshop layers. For each group of voxels found at the same height we generate a layer in Photoshop. In every pixel, each layer encodes three edges of the cube or tetrahedral mesh.

7.3 Image processing using Photoshop's tools

Once the layers are prepared for editing the 3D model, the user can use all standard Photoshop tools to edit the object's material including image adjustments and resampling, as discussed before.

7.4 Generating the metamaterial's internal structure

7.4.1 Interpret color and create beams

Our color-interpreter reads every pixel's color and writes a file containing function calls for our mesh-generator, which is based on OpenJSCAD [15]. The cell size, grid type (cubic or triangular), and the described edges per pixel (isotropic or anisotropic cell) are passed from the Photoshop extension through the color-interpreter to our mesh-generator. If a triangular mesh is selected, the generated beams are sheared back to form the original 3D model. We offer a simple library that creates parametric cells by taking a vector defining the stiffness of beams and the cell's position on the grid. We calculate the thickness of the beams to correspond to the user-defined linear stiffness values by

$$t_{desired} = \sqrt[4]{i} * t_{min},$$

where

$$i = x * i_{scaled},$$

and

$$x = \left(\frac{t_{max}}{t_{min}}\right)^{\frac{4}{100}}.$$

The user-defined stiffness i is given as a ratio of the desired stiffness over the minimum stiffness. We calculate the thickness of the beam t by scaling it by the fourth root for beams in bending [8] and in buckling [7]. We normalize the input stiffness i to cover the full range of stiffness the user's printer can produce, given by t_{min} and t_{max}. We now correct $t_{desired}$ because as the thickness grows, the length of a beam is subsumed into the thickness its neighbors. This shortens the apparent length, increasing the stiffness. We correct beams in bending (beams of triangles, diagonal of the square mesh) by

$$t_{corrected} = \left(\frac{l_{cellsize}}{l_{current}}\right)^{\frac{3}{4}} \cdot t_{desired},$$

and beams in buckling (the axes-aligned beams of the square mesh) by

$$t_{corrected} = \left(\frac{l_{cellsize}}{l_{current}}\right)^{\frac{1}{2}} \cdot t_{desired}.$$

7.4.2 Completing border cells

Since our cell definition contains only edges that are unique to a cell, the cells at the object's border are missing some edges (these would normally be specified by its neighboring cells). Our system automatically completes all border edges by adding a cell adjacent to the border cell and replicating the edge in the corresponding direction. Border cells are defined as being at the margin of the image, or as having adjacent cells that are empty (transparent or white). Note that only cells in the right and top side must be evaluated.

7.4.3 Generate a printable file

Finally, we generate a .stl file from the .jscad file using OpenJSCAD's built-in render engine (based on CGAL) which we invoke directly from our Photoshop extension.

8 Conclusions and Future Work

We made two main contributions. First, we presented a simple system that converts metamaterials to (stacks of) images and back. This allows users to edit metamaterials using traditional raster graphics editors, such as Adobe Photoshop. This is allows users to engineer structures on per-cell level, giving them the fast and precise control required to develop new structures. Second, we used our own approach to create a new class of metamaterials: mechanisms, i.e., devices that transform forces and movement. We then use our raster graphics approach to engineer a series of mechanisms each of which combines asymmetric cells in a different way, ranging from four-bars and hinges to door latches and pliers. For future work, we plan to investigate metamaterials that create mechanisms that possess dynamic properties.

8.1 Publications

- Alexandra Ion, Jack Lindsay, Louis Kirsch, Moritz Hilscher, David Stangl, Arthur Silber, Hsiang-Ting Chen, Pedro Lopes, Patrick Baudisch. Editing Metamaterials, Creating Mechanisms. Submitted to CHI'16. **Under review.**
- Pedro Lopes, Alexandra Ion, Patrick Baudisch. 2015. Impacto: Simulating Physical Impact by Combining Tactile Stimulation with Electrical Muscle Stimulation. In Proceedings of UIST'15, Charlotte, USA.

References

[1] *Autodesk Within*. URL: http://www.autodesk.com/products/within/overview (last accessed 2015-09-20).

[2] M. Bächer, E. Whiting, B. Bickel, and O. Sorkine-Hornung. "Spin-It: Optimizing Moment of Inertia for Spinnable Objects". In: *ACM Transactions on Graphics* 33.4 (July 2014), 96:1–96:10. DOI: 10.1145/2601097.2601157.

[3] P. Baudisch. "Don't click, paint! Using toggle maps to manipulate sets of toggle switches". In: *Proceedings of the 11th annual ACM symposium on User interface software and technology*. 1998, pages 65–66. DOI: 10.1145/288392.288574.

[4] B. Bickel, M. Bächer, M. Otaduy, H. R. Lee, H. Pfister, M. Gross, and W. Matusik. "Design and fabrication of materials with desired deformation behavior". In: *ACM Transactions on Graphics* 29.4 (2010), page 63. DOI: 10.1145/1833351.1778800.

[5] J. C. Á. Elipe and A. D. Lantada. "Comparative study of auxetic geometries by means of computer-aided design and engineering". In: *Smart Materials and Structures* 21.10 (2012), page 105004. DOI: 10.1088/0964-1726/21/10/105004.

[6] B. Florijn, C. Coulais, and M. van Hecke. "Programmable Mechanical Metamaterials". In: *Physical review letters* 113.17 (2014). DOI: 10.1103/PhysRevLett.113.175503. arXiv: 1407.4273.

[7] M. F. Gibson, Lorna J and Ashby. *Cellular solids: structure and properties*. Cambridge university press, 1997.

[8] R. C. Hibbeler. *Engineering Mechanics*. Prentice Hall, 2001.

[9] L. L. Howell, S. P. Magleby, and B. M. Olsen. *Handbook of Compliant Mechanisms*. John Wiley and Sons, 2013.

[10] L. LLu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, and B. Chen. "Build-to-Last : Strength to Weight 3D Printed Objects". In: *ACM Transactions on Graphics* 33.4 (2014), pages 1–10. DOI: 10.1145/2601097.2601168.

[11] M. Mir, M. N. Ali, J. Sami, and U. Ansari. "Review of Mechanics and Applications of Auxetic Structures". In: *Advances in Materials Science and Engineering* (2014), pages 1–17. DOI: 10.1155/2014/753496.

[12] *Monolith*. URL: http://www.monolith.zone/#introduction (last accessed 2015-09-20).

[13] T. Mullin, S. Deschanel, K. Bertoldi, and M. Boyce. "Pattern transformation triggered by deformation". In: *Physical Review Letters* 99.8 (2007), pages 1–4. DOI: 10.1103/PhysRevLett.99.084301.

[14] *netfabb: Selective Space Structures*. URL: http://www.netfabb.com/structure.php (last accessed 2015-09-20).

[15] *OpenJSCAD*. URL: http://www.openjscad.org (last accessed 2015-09-20).

[16] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin. "Elastic Textures for Additive Fabrication". In: *ACM Transactions on Graphics* 34.4 (2015), pages 1–12. DOI: 10.1145/2766937.

[17] R. Prévost, E. Whiting, S. Lefebvre, and O. Sorkine-Hornung. "Make It Stand: Balancing Shapes for 3D Fabrication". In: *ACM Transactions on Graphics* 32.4 (2013), pages 1–10. DOI: 10.1145/2461912.2461957.

[18] C. Schumacher, B. Bickel, J. Rys, S. Marschner, C. Daraio, and M. Gross. "Microstructures to Control Elasticity in 3D Printing". In: *ACM Transactions on Graphics*. 2015, pages 1–13. DOI: 10.1145/2766926.

[19] J. Shim, C. Perdigou, E. R. Chen, K. Bertoldi, and P. M. Reis. "Buckling-induced encapsulation of structured elastic shells under pressure". In: *Proceedings of the National Academy of Sciences* 109.16 (2012), pages 5978–5983. DOI: 10.1073/pnas.1115674109.

[20] J. Shim, S. Shan, A. Košmrlj, S. H. Kang, E. R. Chen, J. C. Weaver, and K. Bertoldi. "Harnessing instabilities for design of soft reconfigurable auxetic/chiral materials". In: *Soft Matter* 9.34 (2013), pages 8198–8202. DOI: 10.1039/c3sm51148k.

[21] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi. "Volumetric modeling with diffusion surfaces". In: *ACM Transactions on Graphics* 29.6 (2010), page 1. DOI: 10.1145/1882261.1866202.

[22] H. Widjaya. "Accurate incremental voxelization in common sampling lattices". PhD thesis. Simon Frasier University, 2006.

Profiling the Web of Data

Anja Jentzsch

Information Systems Group
Hasso-Plattner-Institut
anja.jentzsch@hpi.uni-potsdam.de

The Web of Data contains a large number of openly-available datasets covering a wide variety of topics. In order to benefit from this massive amount of open data such external datasets must be analyzed and understood already at the basic level of data types, constraints, value patterns, etc.

For Linked Datasets such meta information is currently very limited or not available at all. Data profiling techniques are needed to compute respective statistics and meta information. However, current state of the art approaches can either not be applied to Linked Data, or exhibit considerable performance problems. This paper presents my doctoral research which tackles these problems.

1 Problem Statement

Over the past years, an increasingly large number of data sources has been published as part of the Web of Data. This trend, together with the inherent heterogeneity of Linked Datasets and their schemata, makes it increasingly time-consuming to find and understand datasets that are relevant for integration. The true value of Linked Data becomes apparent when datasets are analyzed and understood already at the basic level of data types, constraints, value patterns etc. For Linked Datasets and other Web data meta information is currently quite limited or not available at all. Such *data profiling* is especially challenging for RDF data, the underlying data model on the Web of Data. In comparison to other data models, e.g., the relational model, RDF often lacks explicit schema information that precisely defines the types of entities and their attributes.

Existing work on data profiling often can not be applied to Linked Datasets due to their different nature. To overcome this gap we introduce a comprehensive list of data profiling tasks which compute the most important statistical properties along different groupings.

Finding information about Linked Datasets is an open issue on the constantly growing Web of Data. While most of the Linked Datasets are listed in registries as for instance at the Data Hub (`datahub.io`), these registries usually are manually curated. Existing means and standards for describing datasets are often limited in their depth of information. We present approaches and challenges for cataloging Linked Datasets and retrieving basic metadata.

Data profiling often exhibits considerable performance problems. We introduce three common techniques for improving performance, and present an approach that relies on parallelization and adapts multi-query optimization for relational data to optimize execution plans of Linked Data profiling tasks.

As Linked Datasets are usually sparsely populated, key candidates often consist of either multiple low-density properties or cannot be found at all. We present two approaches for key discovery, a traditional unique column combination adaption and an approach that tackles the sparsity on the Web of Data by combining the uniqueness and density of properties. Furthermore, since ontologies are topically clustered by their underlying ontologies, we analyze how to retrieve key candidates per topic cluster.

Graph analysis can be used to gain more insight into the data, induce schemas, or build indices. We present an approach for frequent graph pattern mining, and a set of common and re-occuring graph patterns that can be considered the core of most Linked Datasets. Finally, we analyze patterns that are dominant for certain class-combinations across multiple datasets.

All presented approaches are evaluated thoroughly on real-world datasets, and are implemented in the interactive Linked Data profiling suite ProLod++.

2 Related Work

While many general tools and algorithms already exist for data profiling, most of them cannot be used for graph datasets, because they assume a relational data structure, a well-defined schema, or simply cannot deal with very large datasets. Nonetheless, some Linked Data profiling tools already exist. Most of them focus on solving specific use cases instead of data profiling in general.

One relevant use case is schema induction, because the lack of a fixed and well-defined schema is a common problem with Linked Datasets. One example for this field of research is the ExpLOD tool [6]. ExpLOD creates summaries for RDF graphs based on class and property usage as well as statistics on the interlinking between datasets based on owl:sameAs links.

Li describes a tool that can induce the actual schema of an RDF dataset [8]. It gathers schema-relevant statistics like cardinalities for class and property usage, and presents the induced schema in a UML-based visualization. Its implementation is based on the execution of SPARQL queries against a local database. Like ExpLOD, the approach is not parallelized. Both solutions still take approximately 10h to process a 10 million triples dataset with 13 classes and 90 properties. These results illustrate that performance is a common problem with large Linked Datasets.

An example for the query optimization use-case is presented in [7]. The authors present RDFStats, which uses Jena's SPARQL processor to collect statistics on Linked Datasets. These statistics include histograms for subjects (URIs, blank nodes) and histograms for properties and associated ranges.

Others have worked more generally on generating statistics that describe datasets on the Web of Data and thereby help understanding them. LODStats computes statistical information for datasets from the Data Hub [2]. It calculates 32 simple statistical criteria, e.g. cardinalities for different schema elements and types of literal values (e.g. languages, value data types).

In [3] the authors automatically create VoID descriptions for large datasets using MapReduce. They manage to profile the BTC2010 dataset in about an hour on Amazon's EC2 cloud, showing that parallelization can be an effective approach to improve runtime when profiling large amounts of data.

3 Large Scale Data Profiling

The process of running data profiling tasks for large Linked Datasets can take hours to days, depending on the complexity of task and the size of the respective datasets. Data set characteristics highly influence the profiling task runtime. As an example, our *Property Cooccurrence by Resource* script runs 16 hours for only 1 million triples of the Web Data Commons RDFa dataset in contrast to 5 min on Freebase and 9 min on DBpedia.

We have compiled a list of 56 data profiling tasks implemented in Apache Pig to be executed on Hadoop. At this point Apache Pig only applies some basic logical optimization rules, like removing unused statements [5]. We present LODOP, a framework for executing, optimizing, and benchmarking such a set of profiling tasks, highlight reasons for poor performance when executing the scripts sequentially, and develop a number of optimization techniques. In particular, we developed and evaluated three multi-script optimization rules for combining logical operators in the execution plans of profiling scripts.

3.1 Multi-query optimization for Apache Pig

A prevalent goal for relational database optimization is to reduce the amount of required full table scans, which for file-based database systems effectively means reducing the amount of disk operations. Sellis introduces Multi-Query Optimization for relational databases as the process of optimizing a set of queries which may share common data [9]. The goal is to execute these queries together and reduce the overall effort by executing similar parts only once. The optimization process consists of two parts: identifying shared parts in multiple queries and finding a globally optimal execution plan that avoids superfluous computation.

Apache Pig[1] is a platform for performing large-scale data transformations on top of Hadoop clusters. It provides a high-level language (called *Pig Latin*) for specifying data transformations, e.g. selections, projections, joins, aggregations and sorting on datasets. Pig Latin scripts are compiled into a series of MapReduce tasks and executed on a cluster.

The main goals for our multi-query optimization rules for Pig are the following two: First, we attempt to minimize the dataflow between operators. In our evaluation we identified the dataflow between MapReduce jobs as a reasonable indicator for the performance of Pig scripts, as it is closely related to the amount of required

[1]http://pig.apache.org/ (last accessed 2015-10-01)

HDFS operations. Second, we try to avoid performing identical or similar operations multiple times. The idea behind this is to free up cluster resources for other tasks. All optimization rules presented in this section, are based on optimizing the logical plans of Pig scripts.

Three optimization rules have been implemented: Rule 1 merges identical operators in logical plans of different scripts, Rule 2 combines `FILTER` operators, and Rule 3 combines aggregations, i.e. `FOREACH` operators. Rule 1 is a prerequisite for the other two rules, which work on pairs of siblings operators, i.e. operators that have the same parent operator in a respective logical plan. For all optimization rules, it was important to make sure that their usage does not affect the intended output of scripts.

Rule 1 – Merge identical operators: In order to better utilize cluster resources, it makes sense to submit jobs to Hadoop in parallel. LODOP supports this by merging logical plans of different scripts into a single large plan. In our experiments, executing scripts in parallel as part of one large plan cuts execution time down to 25 % to 30 % of the time required to execute scripts sequentially. Once all plans have been merged together, it's possible to also merge identical operators. For 52 of our Pig scripts, this reduces the number of operators from 365 to 267.

Rule 2 – Combine filters: `FILTER` operators reduce the amount of data that needs to be processed in later steps of the execution pipeline. This optimization rule aims to avoid iterating over large sets multiple times. From our selection of profiling scripts, 25 scripts perform filtering operations on the full initial dataset.

First, we identify all suitable sibling filters, i.e. all `FILTER` operators that have the same parent operator. Second, a combined filter is created and we attach it to the same parent operator. This combined filter contains all boolean expressions of existing filters concatenated via `OR`. The expression of the combined filter is cleaned up by transforming it into disjunctive normal form. Finally, we re-arrange all previous filters and move them after the combined filter.

Rule 3 – Combine aggregations: `FOREACH` operators can be used for projections and aggregations. Some instances perform identical aggregations, but project different properties. This can happen, e.g. if the aggregation itself is only a preprocessing step to another aggregation. These operators are not exactly identical, so the rule for merging identical operators will not be able to merge them. However, these cases can be optimized by separating the aggregation from the projection, i.e. performing the aggregation only once with all projected columns, and then projecting the exact columns afterwards. For our set of scripts, this rule can be applied in seven different cases and combines varying numbers of `FOREACH` operators from the minimum of two to a maximum of eleven siblings operators.

While our goal is to optimize the performance of profiling tasks, the optimization rules can be applied on any Pig script.

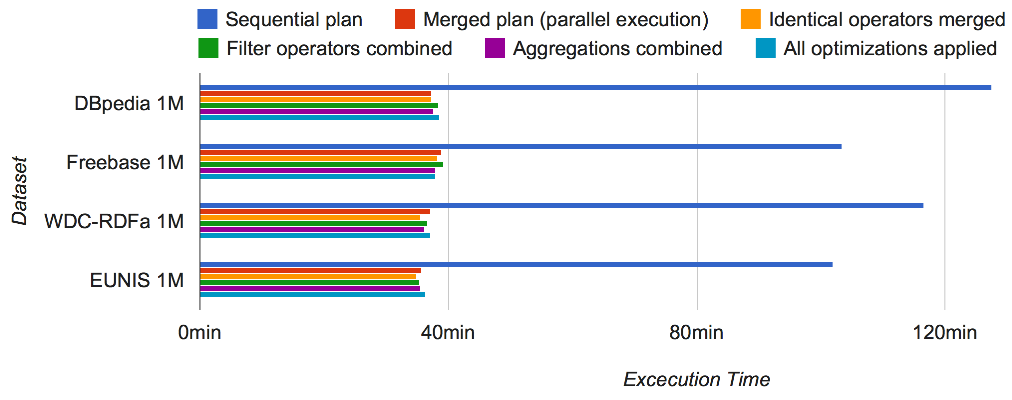


Figure 1: Execution time for all optimizations (52 scripts)

3.2 Evaluation

The number of MapReduce jobs and the amount of dataflow in the operator pipeline are good indicators for the performance of Apache Pig scripts. Our evaluation shows that improving only on these factors does not necessarily improve overall performance. Merging identical operators reduces both the total number of operators and the number of MapReduce jobs. It comes at the cost of less parallelism. Combining filter operators was shown to reduce the execution time of map/reduce functions (i.e. CPU time). Combining aggregations can reduce the amount of HDFS I/O, and improves overall execution time for certain combinations of scripts and datasets. Figure 1 shows execution times when optimizations are applied for all scripts. Overall in our experiments, executing scripts in parallel and applying all optimization rules cuts execution time down to 25 % to 30 % of the time required to execute scripts sequentially.

4 Uniqueness, Density, and Keyness of Data

As Linked Datasets are usually sparsely populated, minimal unique property combinations (key candidates) often consist of either multiple low-density properties or cannot be found at all. Novel property attributes, such as the uniqueness, density, and keyness of a property are needed to discover the set of properties that likely identifies an entity, the key candidates. Furthermore, since ontologies are topically clustered by their underlying ontologies, these attributes can be determined per cluster and give some detailed insights into the properties that serve as key candidates per topic.

A Linked Dataset's class hierarchy is the taxonomy defined by its ontology and therein the rdfs:subClassOf relations between the classes. A cluster C_c for a class c

consists of all the entities e that are of rdf:type c, which includes all subclasses of c.

$$C_c = \{e | e \xrightarrow{rdf:type} c\}$$

Clusters can contain entities e that are not in any of its subclusters d. We cluster these entities separately and call the resulting clusters *unspecialized clusters*, denoted as C'_c.

$$C'_c = C_c \setminus \{e \,|\, e \xrightarrow{rdf:type} d,\ d \xrightarrow{rdfs:subClassOf} c\}$$

We omit the c subscript where it is irrelevant in the context. As an additional complication, properties on the Web of Data can have multiple property values. E.g., in the DBpedia dataset we find the following four values for the property dbpedia:birthPlace for the entity of Albert Einstein:

```
dbpedia:Albert_Einstein dbpedia:birthPlace dbpedia:Ulm,
          dbpedia:Kingdom_of_Wuerttemberg,
          dbpedia:German_Empire
          dbpedia:Baden-Wuerttemberg .
```

We denote the set of property values of an entity e and property p as $V(e,p)$. To count the number of entities in a cluster C that have at least one value for p, we define $V(C,p) = \{e \,||V(e,p)| > 0, e \in C\}$. Property values of a property p and two entities $e1$ and $e2$ are equal if $V(e1,p) = V(e2,p)$, i.e., if the two sets are identical. With this definition we further define the *set of unique value sets* as $V_{uq}(C,p) = \{V(e,p) \,|\, e \in C\}$.

We are now ready to define the three attributes, uniqueness, density, and keyness, of a property. The *uniqueness uq* of a property p for a cluster C is the number of unique value sets $V_{uq}(C,p)$ per number of total value sets $V(C,p)$ for the given property.

$$\textbf{Uniqueness:} \qquad uq(C,p) = \frac{|V_{uq}(C,p)|}{|V(C,p)|} \tag{1}$$

The *density d* of a property p for a cluster C is the ratio of entities in C that have p to the overall number of entities in C.

$$\textbf{Density:} \qquad d(C,p) = \frac{|V(C,p)|}{|C|} \tag{2}$$

We call a property *full key candidate* if its density and uniqueness are both 1. For cases where they are not both 1 we define its keyness as a useful attribute. The *keyness k* of a property p for a cluster C is the harmonic mean of its uniqueness and density. The harmonic mean emphasizes that *both* parameters must be high to achieve an overall high keyness:

$$\textbf{Keyness:} \qquad k(C,p) = \frac{2 \cdot uq(C,p) \cdot d(C,p)}{uq(C,p) + d(C,p)} \tag{3}$$

We call a property *key candidate* if its keyness is above some threshold.

We investigate the three attributes of an RDF property, uniqueness, density, and keyness, for the given cluster types C, and C'. Determining uniqueness, density, and keyness for a property p in a cluster C_c requires analyzing *all* property value sets for *all* entities in the given cluster. We observe all kinds of specificities of properties for clusters and their subclusters that allow for a fine-grained, cluster-based retrieval of key candidates.

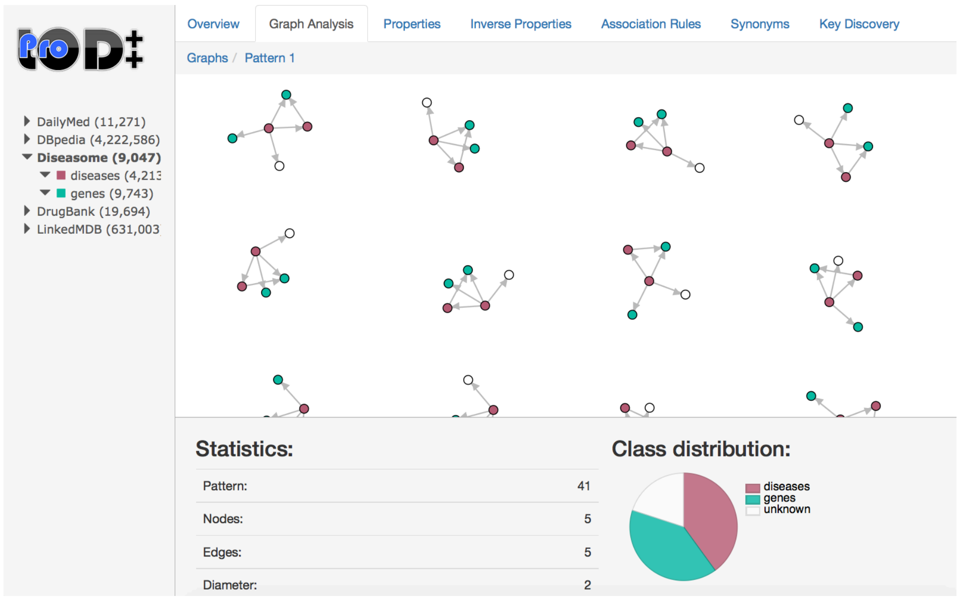
Our evaluation shows that the property keyness can help discovering key candidates for Linked Datasets. It also highlights the advantages of analyzing the class hierarchy in order to observe property behaviour for classes along it and make better choices when identifying key candidates for specific classes.

5 Graph Structures in Linked Datasets

Graph patterns are of interest to many communities, e.g. for protein structures, network traffic, crime detection, modeling object-oriented data, and querying RDF data. We leverage the graph pattern mining approaches gSpan [10] and GRAMI [4], to analyze Linked Datasets. To this end, we have significantly extended our prototype ProLOD++, which features many basic as well as specific profiling tasks for a given RDF dataset, such as schema discovery for user-generated attributes, association rule discovery to uncover synonymous predicates, and key discovery along ontology hierarchies [1]. ProLOD++ now is a Play application and allows easy extension by further techniques. It is available at http://prolod.org. We implemented and added the GraphLOD library, which provides the following new functionality:

- Basic graph statistics, such as the number of connected components and strongly connected components, their corresponding diameter, chromatic number, and node degree distribution.
- Connected components are visualized, and grouped if isomorphic.
- Three graph pattern mining algorithms.
- Visualization of mined patterns with class coloring.
- Interactive graph structure exploration in a faceted fashion.

ProLOD++ allows exploring the graphical structures of Linked Datasets by visualizing the connected components and the graph patterns mined from them. Given the underlying graph for a Linked Dataset, containing all entities as nodes and object properties between them as links, we detect graph patterns for its directed as well as undirected version. The latter allows for pattern mining on a more general level. Bigger graph components (> 1000 nodes) are mined for subgraph patterns using three different approaches: gSpan, GRAMI, and a new approach that mines for predefined patterns. Our goal is to define a set of graph patterns that can be considered the core of most Linked Datasets. We identify graph patterns such as paths, cycles, stars, siamese stars, antennas, caterpillars, and lobsters. Figure 2 is a screenshot of ProLOD++ showing all occurrences of a selected pattern and their class distribution along with some statistical information.



Statistics:

Pattern:	41
Nodes:	5
Edges:	5
Diameter:	2

Figure 2: Occurrences of a pattern in Diseasome visualized by ProLoD++

ProLoD++ allows faceted browsing through the graph patterns. Patterns are grouped when isomorphic, first based on their underlying structure and then based on the class membership (color). This allows for finding not only common, re-occurring patterns but also patterns that are dominant for certain class-combinations. E.g., astronomers in DBpedia are often to be found in star patterns, surrounded by their discovered astronomical objects.

Based on the graph features provided by ProLoD++ and its underlying GraphLoD library, an overall model for Linked Datasets can be given: We observe that most of the Linked Datasets consist of a number of small satellite graphs and a giant component that contains more than 80 % of the nodes and thus resemble scale-free networks as they occur in social networks.

When jointly profiling multiple datasets, ProLoD++ highlights the connectivity of connected components across them based on inter-dataset links. This, for instance, identifies the potential of dataset integration.

6 Reflections and Conclusion

The main difference in my approach with existing work on Linked Data profiling is to address the shortcomings mentioned in section 2, in particular gathering comprehensive metadata in an efficient way. Within my research I am building on existing profiling techniques for relational data and adapting them according to the different nature of Linked Datasets.

This paper has presented the outline and preliminary results of my doctoral research, in which I am focussing on profiling the Web of Data.

We have specified and implemented a comprehensive set of Linked Data profiling tasks and illustrated the Web of Data's diversity with the results for four different Linked Datasets. Furthermore we introduced three common techniques for improving performance of Linked Data profiling and implemented three multi-query optimization rules, reducing profiling task runtimes by 70 %.

We have introduced the concept of keyness (and therein uniqueness and density) of a property to address the sparsity on the Web of Data and thus create the possibility to find key candidates where traditional approaches fail. Our approach has been implemented in ProLod++ and provides users with the uniqueness, density, and keyness for all properties. Having these profiling results at hand helps users in finding key candidates and analyzing the relevance of properties along class hierarchies in Linked Datasets.

We presented the GraphLod extension for ProLod++, which offers Rdf *graph analysis* features. It allows for interactively exploring the graphical structures of Linked Datasets by visualizing the connected components and the graph patterns mined from them. Furthermore it offers basic graph statistics as node degree distribution, pattern diameter, and more. Furthermore we defined a set of graph patterns that can be considered the core of most Linked Datasets.

References

[1] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. "Mining and Profiling RDF Data with ProLOD++". In: *Proceedings of the International Conference on Data Engineering (ICDE)*. Demo. 2014.

[2] S. Auer, J. Demter, M. Martin, and J. Lehmann. "LODStats – an extensible framework for high-performance dataset analytics". In: *Proceedings of the Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*. 2012.

[3] C. Böhm, J. Lorey, and F. Naumann. "Creating VoiD Descriptions for Web-scale Data". In: *Journal of Web Semantics* 9.3 (2011), pages 339–345.

[4] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. "GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph". In: *PVLDB* 7.7 (2014), pages 517–528.

[5] A. Gates, J. Dai, and T. Nair. "Apache Pig's Optimizer". In: *IEEE Data Engineering Bulletin* 35.1 (2013), pages 34–45.

[6] S. Khatchadourian and M. P. Consens. "ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud". In: *Proceedings of the Extended Semantic Web Conference (ESWC)*. Heraklion, Greece, 2010.

[7] A. Langegger and W. Wöß. "RDFStats – An Extensible RDF Statistics Generator and Library". In: *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*. Los Alamitos, CA, USA, 2009, pages 79–83.

[8] H. Li. "Data Profiling for Semantic Web Data". In: *Proceedings of the International Conference on Web Information Systems and Mining (WISM)*. 2012.

[9] T. K. Sellis. "Multiple-query optimization". In: *ACM Transactions on Database Systems (TODS)* 13.1 (1988), pages 23–52.

[10] X. Yan and J. Han. "gSpan: Graph-Based Substructure Pattern Mining". In: *Proceedings of the International Conference on Data Mining (ICDM)*. 2002, pages 721–724.

BottlePrint: Scaling Personal Fabrication by Embedding Ready-Made Objects

Robert Kovacs

Human Computer Interaction Lab
Hasso-Plattner-Institut
robert.kovacs@hpi.de

BottlePrint is a fabrication system that allows users to produce large-scale objects on desktop-scale fabrication ma-chines. The key idea behind bottlePrint is to complement 3D printing with ready-made objects, in our case empty plastic bottles. BottlePrint considers 3D models as wireframe models; it then fabricates only the hubs of this wireframe model, while it implements all edges as bottles. The resulting large-scale objects are sturdy enough to carry human users. We present a furniture set with table and chairs, a truss strong enough to bridge a 3m gap while a human walks on it, and a functional boat sealed with tarp that seats two. To extend our approach to even larger ob-jects we fabricate hubs on a laser-cutter, which speeds up fabrication by another factor of 20. This allowed us to fabricate a 12 m2 tent. Benefits of our approach include (1) Scale: bottlePrint allows 3D printing of large-scale objects on desktop-scale devices. (2) Speed: as the bulk of the objects volume is ready-made. (3) Environ¬mentally conscious: as materials can be upcycled. (4) Ubiquitous: since plastic bottles can be acquired anywhere worldwide, users setting up large installations elsewhere can travel lightly, carrying only the hubs.

1 Introduction

Personal fabrication tools, such as 3D printers have achieved a desktop form factor. As a result, they have spread to the maker community, as well as increasingly also to consumers [10]. In contrast, the fabrication of large objects has remained a privilege of industry, which has access to specialized equipment, such as concrete printers to make houses [3], as well as fabrication robots [4]. The owners of the wide-spread desktop devices, in contrast, cannot participate in this step in evolution, as the underlying technology does not scale. Even if we break down large models into parts that fit into a desktop-scale device [5] fabricating a large model consumes time and material proportional to the size of the model, quickly rendering 3D printing and related techniques intractable for larger-than-desktop-scale models.

In this paper, we address this issue. We allow users of desk-top-size fabrication machines to fabricate objects as large as several meters by embedding ready-made objects into the fabricated objects.

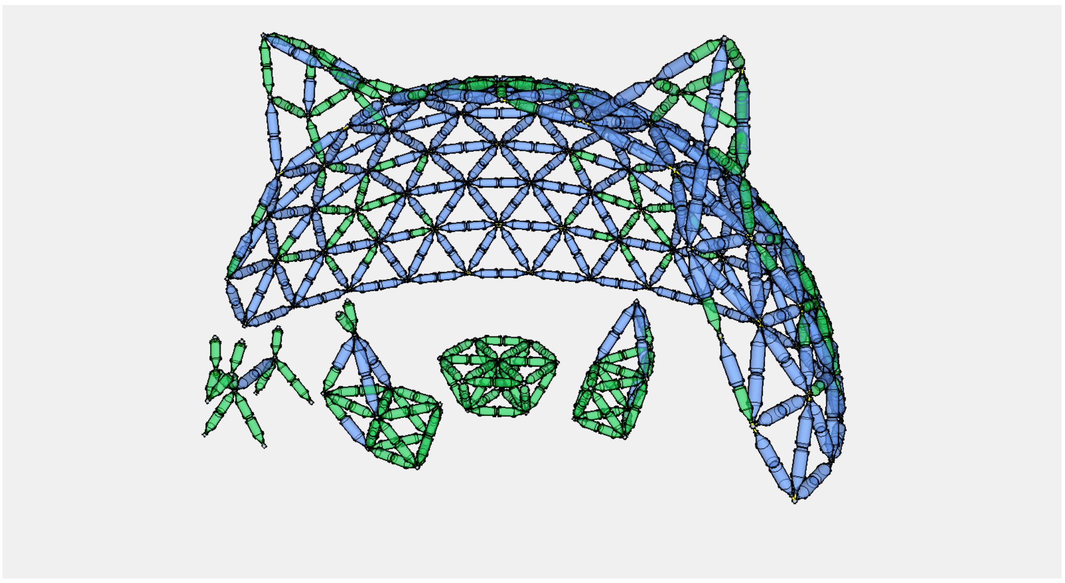


Figure 1: BottlePrint allows users to create large-scale objects using desktop-scale 3D printers by integrating ready-made objects, here up-cycled bottles, so that only the hubs are 3D printed.

2 BottlePrint

The key idea behind bottlePrint is to complement fabricated parts with ready-made objects, in our case empty plastic bottles, such as the ones used to sell water or soda.

Figure 1 shows a furniture set and a tent, all of which were created using bottlePrint. Figure 2 shows the resulting objects. The furniture set, comprised of a table, two chairs, and a hat stand was fabricated using desktop 3D printers. The tent was created from a mix of 3D printed and laser cut parts. As illustrated by these examples, bottlePrint considers 3D objects as wireframe models. It then fabricates only the hubs of these wireframe models, and implements all edges as bottles. The hubs, however, account for only a small fraction of the original volume of our structure; they are small enough to allow for fabrication using desktop ma-chines in a manageable amount of time.

2.1 Workflow

BottlePrint allows users to create structures either by modeling from scratch or by converting existing 3D models.

Step 0 Conversion (optional). One way to create bottlePrint structures is to convert an existing 3D model into a bottle mesh using our custom bottlePrint converter program. As illustrated by Figure 3, this results in a node link graph. The image shows a volume approximation, i.e., the entire volume of the model is filled with a honeycomb bottle structure, allowing it to bear a substantial load. Alternatively,

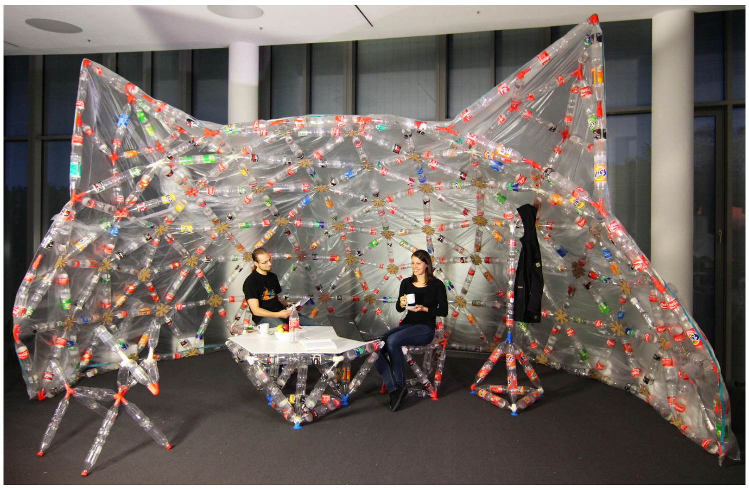


Figure 2: The objects from Figure 1 fabricated

users may choose shape approximation, in which case only the outer shape of the model is represented as a bottle mesh.

Step 1 Editing. Users may refine the result of the conversion in the bottlePrint editor (Figure 4); alternatively, they may start a new object in the editor from scratch. We have implemented the bottlePrint editor as an extension to the 3D editing software Trimble SketchUp[1] It offers all the functionalities of the original 3D editor, plus a range of bottle-specific functions, such as functions for placing predefined bottle primitives or automatic snapping geometry to bottle-compatible lengths.

Step 2: Hub generation. In order to fabricate, we first run the output of the bottlePrint editor (or converter) through bottlePrint's hub generator. This tool automatically generates the 3D models of all hubs required to fabricate the structure. The hub generator generates models in STL format for 3D printing and in SVG format for laser cutting. Figure 5 shows our three different connector types: (a) threaded, (b) snap-fit, and (c) laser-cut design.

Step 3 Fabrication. The output of the hub generator is sent to a 3D printer and/or laser cutter.

Step 4 Manual assembly. Finally, users assemble their object by matching up unique IDs embossed on each hub with the IDs they see in bottlePrint editor (Figure 6). These IDs are generated by the bottlePrint editor, which hands them down to the hub generator, which embosses them into the physical 3D print.

Figure 7 shows our assembled chair design from Figure 1. It consists of 8 hubs and 18 bottle pairs. To remove the load from the bottles on the ground, we place "pods" on the bottom hubs. The sitting plate is a piece of particle board, which is also supported by pods at the corners.

[1]http://www.sketchup.com (last accessed 2015-10-01)

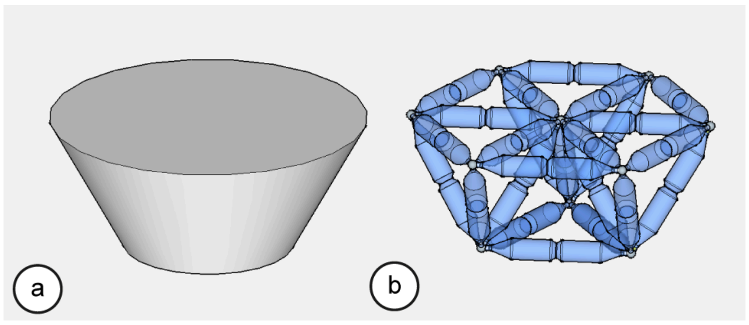


Figure 3: (a) Our custom bottlePrint converter software automatically turns this truncated cone, into (b) the table from Figure 1. Here the converter was configured to volumetric tessellation.

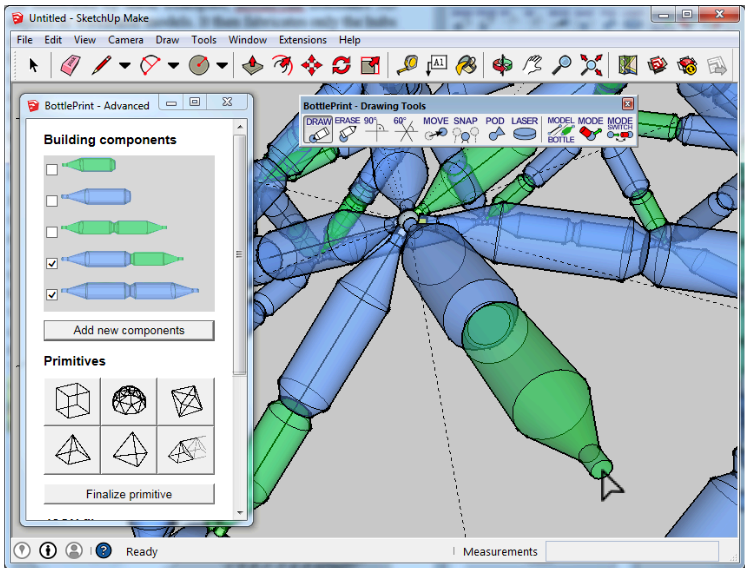

Untitled - SketchUp Make
File Edit View Camera Draw Tools Window Extensions Help
BottlePrint - Advanced
Building components
Add new components
Primitives
Finalize primitive
BottlePrint - Drawing Tools
DRAW ERASE 90° 60° MOVE SNAP POD LASER MODEL MODE BOTTLE MODE SWITCH
Ready
Measurements

Figure 4: User refining the curvature of the tent from Figure 1 in the bottlePrint editor.

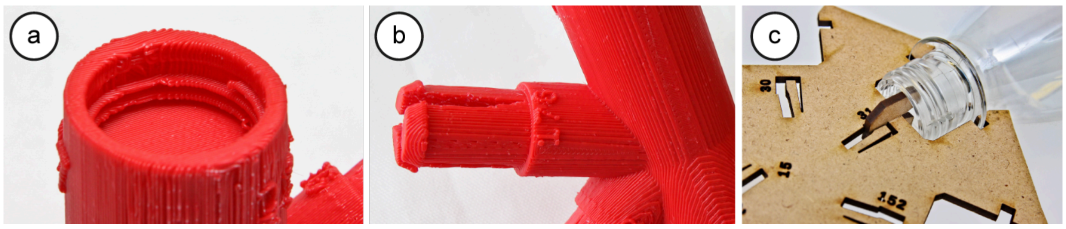


Figure 5: (a) Threaded bottle connector, (b) snap-fit connector, and (c) Laser cut connector

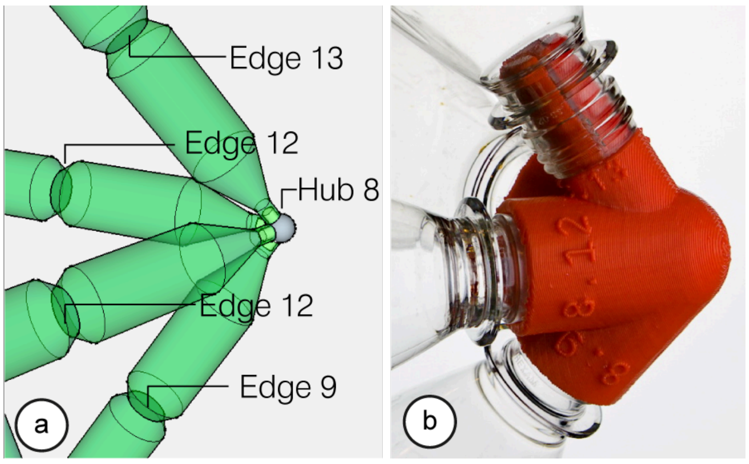


Figure 6: (a) Hub and edge IDs shown in the editor. (b) Embossed IDs help users assemble the structure.

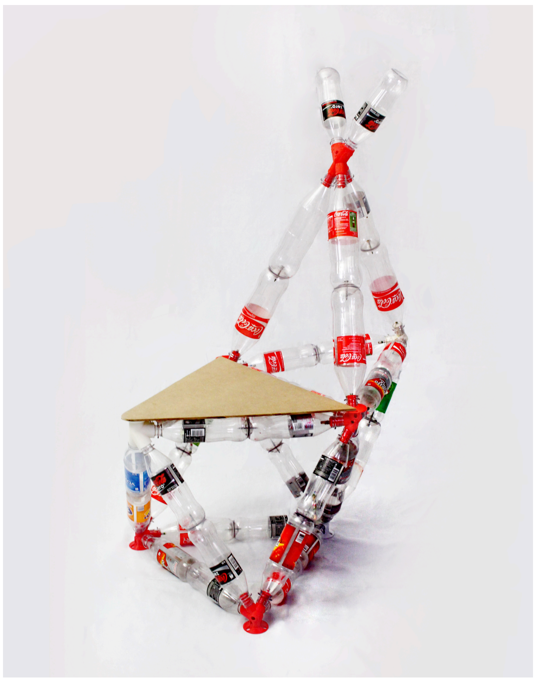


Figure 7: Assembled chair with backrest

3 Contribution, Benefits, and Limitations

Our main contribution is that we enable large-scale fabrication on desktop-size fabrication devices. The key idea is to include ready-made objects, in particular empty plastic bottles as the main building element. Our software system bottlePrint allows users to create bottle-based 3D objects by converting an existing 3D model or by modeling from scratch. We have created objects consisting of 30 to 500 bottles. We have validated their structural integrity through actual use (Figure 6). In addition to enabling users to fabricate large-scale objects on conventional desktop-scale devices, our approach offers the following benefits: **(1) Fast:** the bulk of the objects volume is ready-made. **(2) Light:** can be moved around even in assembled form. **(3) Modular:** modify, extend, combine, or fix objects with only local changes. **(4) Environmentally conscious:** most of the materials are up-cycled. **(5) Ubiquitous:** plastic bottles can be acquired anywhere worldwide, so users setting up large installations elsewhere can travel lightly, carrying just the hubs. For example, all the 3D printed hubs for the boat from Figure 6 fit into a backpack and weight less than 3 kg. Users may obtain everything else on site. The limitations of our technique include: (1) the creation of objects with solid surfaces requires additional material, such as the tarp that forms the boat's hull in Figure 6. (2) Our approach cannot reproduce details smaller than a bottle. (3) Only some of our techniques keep the bottles intact.

4 Implementation

To help readers replicate out results, we now describe the implementation of the three main components of the bottlePrint work flow: bottlePrint editor, 3D model converter and the hub generator. BottlePrint Editor

4.1 BottlePrint editor

We have implemented bottlePrint as a plug-in to the 3D editor SketchUp. The plug-in is written in Ruby and JavaScript. It allows users to create bottle models and to trigger the bottlePrint Hub Generator.

4.2 BottlePrint Converter

The Volumetric conversion procedure is in general similar to traditional voxelization methods. However, instead of intersecting the given 3D model against a regular cubical grid, bottlePrint intersects the model against a tetrahedral-octahedral honeycomb. All edges in this honeycomb are of equal length, so the result can always be fabricated using bottlePrint.

Our surface conversion procedure reproduces the object's facades as bottle, as illustrated by Figure 8. The main challenge here is to ensure that every edge of the

3D model either fits the length of one of the bottle primitives or is slightly longer, in which case the converter will lengthen the edge by extending the respective hub.

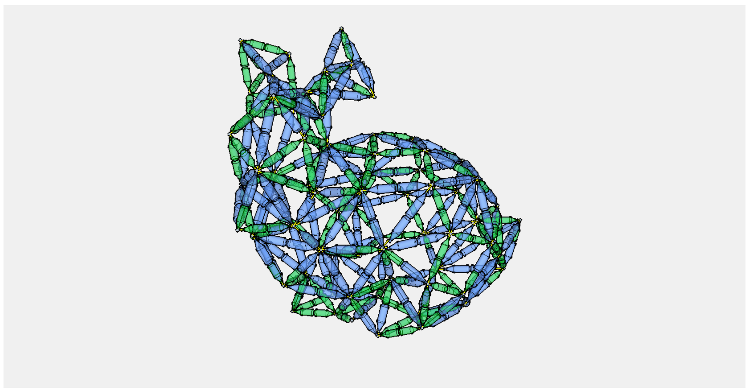


Figure 8: Stanford bunny converted using the bottlePrint converter in surface conversion mode

The conversion consists of two stages: mesh simplification and surface remeshing. In the mesh simplification stage, we use the quartic-based edge collapse function in MeshLab until it reaches the desired number of edges. In cases where we would like to preserve certain features, e.g., the ears of the bunny in Figure 8, we manually simplify the 3D model using the simplification brush in the Autodesk MeshMixer[2]. In the surface remeshing stage, we optimize the vertex position of the model so that all edges are of the valid length of bottle primitives and the distortion of the final mesh is minimized. The energy function has two terms, where the first term is the minimum distance between an edge and the bottle primitives and the second term is the distance between the vertex position and the original simplified mesh.

More specifically, the energy function is of the form:

$$E(V) = \sum_{i=1}^{m} min(|E_i - B|) + \alpha \sum_{i=1}^{n} Dist(v_i, S)$$

where V are the vertices of the simplified 3D model, n and m are the number of vertices and edges respectively, E_i is the length of the edge i, B is the set of valid length for all bottle primitives, $Dist(v_i, S)$ is the distance between vertex i and the surface S of the given 3D model. We calculate the optimized vertex positions using Powell's COBYLA optimization routine [8]. To reduce the computational load, our optimization does not check for self-intersections, which must be removed manually using the bottlePrint editor. The converted models are exported to the bottlePrint editor in the form of a JSON file.

[2]http://www.meshmixer.com (last accessed 2015-10-01)

4.3 BottlePrint Hub Generator

BottlePrint Hub Generator generates the 3D models of the hubs using a mathematical solid modeling tool called OpenSCAD[3]. OpenSCAD is an open source 3D-compiler that creates and renders solid 3D CAD objects from parametric functions using a textual description language. The Hub Generator receives its input from the bottlePrint editor one hub at a time in an OpenSCAD data file format. (1) For 3D printed hubs, this data file describes each connector using a vector annotated with connector type, elongation, and ID. (2) For Laser cut hubs, the plug-in projects all connections onto a plane before exporting all connectors as a 2D geometry. The bottlePrint Hub Generator generates hubs by arranging the individual connector primitives around a sphere. The connector geometry is loaded from separate modular files, allowing users to include their own ready-made objects by importing the respective geometry.

4.4 3D Printing

We fabricate hubs on a MakerBot 2X desktop FDM 3D printer. Each hub consumes about 50 g to 120 gof filament. Using a 0.5 mmnozzle, hubs print in about 1.5 h to 2.5 h. We mostly print ABS, but include recycled PET and ABS materials from Refil[4] as a means of environmentally-friendly fabrication.

5 Related Work

Large scale fabrication receives lots of attention in both academia and industry. We categorize previous efforts on scaling-up fabrication into three main branches: scaling fabrication with large-scale device, scaling fabrication with existing objects and scaling fabrication with construction kits.

5.1 Scaling Fabrication with Larger Printers

Architects and engineers have made efforts to scale up the additive manufacturing process for constructing large-scale structure such as houses or sculptures. These efforts mostly involve a scaled up version of the machinery [11]. For example, Minibuilders [4] and MeshMold [3] scales up the printing area of 3D printers by making printers mobile with wheels underneath and the Mataerial project [7] scales up a hand-held 3D filament extruder with an industry robot arm that extrudes a fast-curing polymer. Commercial companies also introduced specialized large-scale 3D printers into traditional manufacturing industries such as automobile and home building.

[3]http://www.openscad.com (last accessed 2015-10-01)
[4]http://www.refil.com (last accessed 2015-10-01)

5.2 Scaling Fabrication with Existing Objects

Aggregating existed objects into a larger object for decorative or functional purposes is an emerging research field due to economical and ecological reasons [1]. Yoshida et al. [12] proposes a computer-assisted fabrication method for large-scale architecture that combines a chopstick dispenser and a projector-based guiding system. Dierichs et al. [2] creates the desired form by aggregating a large quantity of spike-shaped elements. Song et al [9] describes a computational framework for creating reciprocal frame structures. Finally, the Annual Burning Man gathering famously demonstrates large-scale art pieces built from wood planks, recycled aluminum can and bottles.

5.3 Scaling Fabrication with Construction Kit

Construction kits, such as Lego and Geomag, are perhaps the most popular personal fabrication tools. Zimmer et al. [13] propose a geometry processing method that converts freeform surfaces into polygonal Zomes thereby enabling users to construct large-scale objects using the Zometool construction kit. FaBrickation [6] further combines Lego and 3D printing to accelerate design iterations.

6 Conclusion and Outlook

We presented bottlePrint, a system that allows users to produce large-scale objects on desktop-scale fabrication machines. The key idea behind bottlePrint is to complement 3D printing with ready-made objects, in our case empty plastic bottles. Our software system bottlePrint allows users to create bottle-based 3D objects by converting an existing 3D model or by modeling from scratch. As future work, we plan to extend bottlePrint with mechanisms, so as to allow users to design and build large-scale machines.

References

[1] E. Blevis, S. Bødker, J. Flach, J. Forlizzi, H. Jung, V. Kaptelinin, B. Nardi, and A. Rizzo. "Ecological Perspectives in HCI: Promise, Problems, and Potential". In: *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. ACM. 2015, pages 2401–2404.

[2] K. Dierichs and A. Menges. "Material computation in architectural aggregate systems". In: *Formation, Proceedings of the 30th Conference of the Association for Computer Aided Design (ACADIA)*. 2010, pages 372–378.

[3] N. Hack and W. V. Lauer. "Mesh-Mould: Robotically Fabricated Spatial Meshes as Reinforced Concrete Formwork". In: *Architectural Design* 84.3 (2014), pages 44–53.

[4] S. Jokic, P. Novikov, S. Maggs, D. Sadan, S. Jin, and C. Nan. *Robotic positioning device for three-dimensional printing*. 2014. arXiv: 1406.3400 [cs.RO].

[5] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. "Converting 3D furniture models to fabricatable parts and connectors". In: *ACM Transactions on Graphics (TOG)*. Volume 30. 4. ACM. 2011, page 85.

[6] S. Mueller, T. Mohr, K. Guenther, J. Frohnhofen, and P. Baudisch. "faBrickation: fast 3D printing of functional objects by integrating construction kit building blocks". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pages 3827–3834.

[7] P. Novikov and S. Jokić. *Mataerial*. URL: http://mataerial.com (last accessed 2015-09-25).

[8] M. J. Powell. "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The computer journal* 7.2 (1964), pages 155–162.

[9] P. Song, C.-W. Fu, P. Goswami, J. Zheng, N. J. Mitra, and D. Cohen-Or. "An Interactive Computational Design Tool for Large Reciprocal Frame Structures". In: *Nexus Network Journal* 16.1 (2014), pages 109–118.

[10] J. G. Tanenbaum, A. M. Williams, A. Desjardins, and K. Tanenbaum. "Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2013, pages 2603–2612.

[11] *Top 10 Biggest 3D Printers*. URL: http://3dprintingindustry.com/2015/08/19/top-10-largest-3d-printers (last accessed 2015-09-25).

[12] H. Yoshida, T. Igarashi, Y. Obuchi, Y. Takami, J. Sato, M. Araki, M. Miki, K. Nagata, K. Sakai, and S. Igarashi. "Architecture-scale human-assisted additive manufacturing". In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), page 88.

[13] H. Zimmer, F. Lafarge, P. Alliez, and L. Kobbelt. "Zometool shape approximation". In: *Graphical Models* 76.5 (2014), pages 390–401.

Robustness of Estimation of Distribution Algorithms to Noise

Martin Krejca

Algorithm Engineering
Hasso-Plattner-Institut
martin.krejca@hpi.uni-potsdam.de

Traditional optimization algorithms, for example for finding good solutions for the traveling salesperson problem, are designed by carefully analyzing the problem and then tailoring an algorithm for exploiting the problem structure. Research on optimization has progressed sufficiently so that, for many classic optimization problems, very good specific algorithms are available.

However, practical optimization problems frequently include uncertainty about the quality measure, for example due to noisy evaluations. Thus, they do not allow for a straightforward application of traditional optimization techniques. In these settings, meta-heuristics are a popular choice for deriving good optimization algorithms, most notably evolutionary algorithms, which mimic evolution in nature.

This report summarizes some of our results, published at GECCO'15 and ISAAC 2015, where we, overall, showed that a classical evolutionary algorithm explicitly storing a population fails in optimizing a simple noisy function, whereas certain Estimation of Distribution Algorithms, implicitly storing a population, succeed.

1 Introduction

In real-world optimization problems, there is sometimes a large degree of uncertainty present due to the complexity of candidate solution generation, noisy measurement processes, and rapidly changing problem environments. This is why heuristic optimization is widely used in practice for solving hard optimization problems for which no efficient problem-specific algorithm is known.

Jin and Branke [11] survey a number of sources of uncertainty that randomized search heuristics must often deal with in practice: (1) noisy objective functions, (2) dynamically changing problems, (3) approximation errors in the objective function, and (4) a requirement that an optimal solution must be robust to changes in design variables and environmental parameters that occur after optimization is complete.

Arguably, the two most important sources of uncertainty are (1) and (2), namely, *stochastic* problems and *dynamic* problems. In stochastic problems, the objective function value of a search point follows a random distribution, and that distribution does not change over time. In dynamic problems, the evaluation of the quality of a solution is deterministic but changes over time.

[2] shows that *Evolutionary Algorithms* (EAs) are very popular in settings including uncertainties. So, in order to address these practical issues, the theoretical analyses

of randomized search heuristics under uncertainty has recently gained momentum. For example, a number of recent papers rigorously analyzed the performance of EAs in stochastic environments [3, 8].

EAs have been successfully applied to a wide range of complex engineering and combinatorial problems [1, 7, 13]. Like Darwinian evolution in nature, evolutionary algorithms construct new solutions from old ones and select the fitter ones to continue to the next iteration. The set of solutions is also called the *population* of the algorithm.

We analyzed a popular simple fitness function, namely *OneMax*, perturbed by some additive zero-mean Gaussian noise. A cornerstone of the analysis of any search heuristic is an analysis of its performance on the OneMax function [5, 14], and studying the class of OneMax functions has also lead to several breakthroughs in the field of black-box complexity [4, 6].

We could show that the $(\mu+1)$-EA – a classical EA storing a population of μ solutions – is not able to optimize OneMax efficiently in our setting with a high probability. However, two other Algorithms, that just implicitly store a population, i.e., *Estimation of Distribution Algorithms* (EDAs), can cope with the noise and are able to optimize the function efficiently as long as the variance of the noise is polynomial.

2 Backround

We analyzed the search space $\{0,1\}^n$, i.e., all bit strings of length n, since it is the underlying search space of many optimization problems. Many problems (including combinatorial ones such as the minimum spanning tree problem) have a straightforward formulation as an optimization problem on $\{0,1\}^n$. Many evolutionary algorithms are applicable to this search space without further modification adaption, and most formal analyses of evolutionary algorithms consider this search space.

As mentioned in the introduction, we focused on analyzing the OneMax function, which simply yields the number of ones in a bit string of length n, with some zero-mean Gaussian noise added to it. More formally, our objective function, also called *fitness function*, is $f\colon \{0,1\}^n \to \boldsymbol{R}$ with $x \mapsto \mathrm{OneMax}(x) + N$, where $N \sim \mathcal{N}(0, \sigma^2)$.

We considered three different algorithms. The first one is a classical EA that only uses mutation and stores μ solutions each round: the $(\mu+1)$-EA. It initially generates a uniformly at random drawn population of μ solutions called *individuals*. Each iteration, it picks one of its μ individuals uniformly at random and mutates it. Then an individual with minimal fitness gets discarded; breaking ties uniformly at random as well. The pseudo-code of the $(\mu+1)$-EA can be seen in Algorithm 1.

The other two algorithms are EDAs. That means that they implicitly store a population via a distribution over $\{0,1\}^n$. Our two algorithms, namely MMAS-fp and cGA, assume independency of the single bit positions, and each just stores a vector $\tau \in [0,1]^n$ of n different probabilities, sometimes truncated to some interval $[\ell, u]$. Individuals are generated by τ in that fashion that an individual has a one at position i with probability τ_i, or it has a zero with probability $1 - \tau_i$. Each generation, a certain

Algorithm 1: The $(\mu+1)$-EA, for optimizing f

```
1 $P \leftarrow \mu$ elements of $\{0,1\}^n$ uniformly at random;
2 while optimum not found do
3     Select $x \in P$ uniformly at random;
4     Create $y$ by flipping each bit of $x$ independently with probability $\frac{1}{n}$;
5     $P \leftarrow P \cup \{y\}$;
6     Let $z \in P$ be chosen such that $\forall v \in P: f(z) \leq f(v)$;
7     $P \leftarrow P \setminus \{z\}$;
```

amount of individuals is generated. Their fitness is then analyzed, and τ is updated according to that outcome.

MMAS-fp and cGA differ from one another in how many offspring they produce each iteration and in how they update τ.

MMAS-fp is a special variant of the ant-inspired algorithm MMAS, which sends ants across a so-called *contrsuction graph*. These ants lay pheromones on the edges they traverse and choose edges with a preference relative to the amount of pheromone on each edge. MMAS enforces upper and lower bounds on the pheromone values of each edge. The name states these, since MMAS stands for *Max-Min Ant System*.

Our MMAS-fp sends each iteration just one ant along a construction graph which is a multi-path with two directed edges between two neighboring nodes; one edge standing for a zero in the bit string that gets constructed, the other edge standing for a one. Figure 1 shows an example of how such a construction graph in our setting looks like. However, this viewpoint does not show the similarities to cGA. So we follow with a description of constructing individuals via the probability vector τ.

MMAS-fp generates only one individual x each iteration and updates τ proportional to the fitness of x, $f(x)$. This is called *fitness-proportional*, hence the suffix *fp* in the name. It uses a parameter ρ, called the *evaporation factor*, that regulates the impact of the fitness-proportional update and can be thought of as the step size, i.e., the greatest difference that can occur per τ_i in one iteration. Algorithm 2 shows the pseudo-code of MMAS-fp.

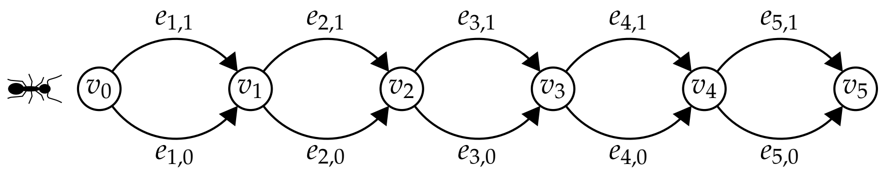


Figure 1: Construction graph for optimization of f with $n = 5$ bits

Algorithm 2: MMAS-fp with $\rho \in (0,1]$, for optimizing f

```
1  for $i \in \{1,\dots,n\}$ do
2      $\tau_i \leftarrow \frac{1}{2}$;
3  while optimum not found do
4      for $i \in \{1,\dots,n\}$ do
5          $x_i \leftarrow 1$ with probability $\tau_i$, $x_i \leftarrow 0$ with probability $1-\tau_i$;
6      for $i \in \{1,\dots,n\}$ do
7          if $x_i = 1$ then
8              $\tau_i \leftarrow \min\left\{\tau_i\left(1-\rho\frac{f(x)}{n}\right)+\rho\frac{f(x)}{n}, u\right\}$;
9          else
10             $\tau_i \leftarrow \max\left\{\tau_i\left(1-\rho\frac{f(x)}{n}\right), \ell\right\}$;
```

cGA stands for *compact Genetic Algorithm*. A Genetic Algorithm is an algorithm that produces a new individual out of more than one old individual. This process is often called *crossover*, *recombination*, or *sexual reproduction*. Since the cGA is an EDA and, thus, has no explicit population, it only imitates crossover in the following way: each iteration, it produces two offspring and compares their bits for each position. If the bits are the same, τ_i does not get updated. However, if the bits are different, τ_i gets adjusted in favor of the bit of the fitter individual, i.e., if the fitter individual has a one, τ_i gets increased, else decreased. The parameter K can be thought of as the population size of the algorithm. The pseudo-code is depicted in Algorithm 3.

Algorithm 3: cGA with parameter K, for optimizing f

```
1  for $i \in \{1,\dots,n\}$ do
2      $\tau_i \leftarrow \frac{1}{2}$;
3  while optimum not found do
4      for $i \in \{1,\dots,n\}$ do
5          $x_i \leftarrow 1$ with probability $\tau_i$, $x_i \leftarrow 0$ with probability $1-\tau_i$;
6          $y_i \leftarrow 1$ with probability $\tau_i$, $x_i \leftarrow 0$ with probability $1-\tau_i$;
7      if $f(x) < f(y)$ then swap $x$ and $y$;
8      for $i \in \{1,\dots,n\}$ do
9          if $x_i > y_i$ then $\tau_i \leftarrow \tau_i + 1/K$;
10         if $x_i < y_i$ then $\tau_i \leftarrow \tau_i - 1/K$;
11         if $x_i = y_i$ then $\tau_i \leftarrow \tau_i$;
```

We analyzed the expected time of each algorithm when optimizing f, that is, until the optimum is sampled the first time. The resulting run time is not only a function of the dimension of the search space, n, but also a function of the variance σ^2 of the noise. The analysis was thus a multivariate analysis, trying to see which levels of noise can be handled and which cannot.

We say that an algorithm optimizes f efficiently when the expected optimization time is a multivariate polynomial in n and σ^2. We coined this property *graceful scaling* to show that the algorithm can cope with any polynomial variance (in n) such that the expected run time still remains polynomial in n.

3 Results

One of our results is that the $(\mu + 1)$-EA is not able to efficiently optimize f if $\sigma^2 \geq n^3$. If the variance is this large, then it is quite likely that the fitness of one of the best individuals of the population is worse than the fitness of all other individuals. Thus, such an individual will be discarded, prolonging the optimization process. To be a bit more precise: for a constant fraction of the population, it holds that the number of good individuals decreases exponentially in n. So it is very unlikely for a good individual to be chosen for mutation and to survive long enough to reach the optimum within polynomial time. Therefore, the $(\mu + 1)$-EA does not scale gracefully with noise.

MMAS-fp and cGA, on the other hand, do scale gracefully with noise if their respective parameters ρ or K, respectively, get adjusted with respect to σ^2.

Our run time results for MMAS-fp are worse than for cGA, but they are also more general. MMAS-fp can basically cope with *any* additive posterior noise as long as the probability of generating very large polynomial amounts of noise (in n) decreases at some point exponentially. This is because of the fitness-proportional update rule. The main property of the algorithm that makes it succeed is that each bit independently contributes more to its corresponding τ_i if it is set correctly with respect to the optimum then otherwise. This property does not only hold for the OneMax function, so further research in that direction is desirable.

The fitness-proportional update scheme can be thought of as some kind of resampling. So in the long run, the resulting fitness value of a solution will be its noise-free fitness value with the added expected value of the noise. The addition of the expected value of the noise does not do any harm, since it does not destroy the ordering of the fitness values; better fitness values still remain better than worse ones. Because the update rule gains more from better fitness values, the τ_i get pushed in the correct direction, resulting in an efficient optimization.

One drawback of the fitness-proportional update scheme, however, is that the evaporation factor ρ has to be somewhat small to make sure that the update steps are not too large. If they were, some unlucky sampling of bad solutions could decrease some τ_i pretty fast, such that it would be unlikely for the algorithm to recover from that. That is why our run time results for MMAS-fp are worse than for cGA; a small step size results in more steps to reach the optimum.

Although MMAS-fp and cGA are both EDAs and roughly have the same layout, cGA optimizes f efficiently due to completely other reasons. A major difference between these two algorithms is that MMAS-fp always makes an update according to the last sampled solution, whereas cGA *compares* different solutions and then makes an update accordingly. Comparisons of noisy fitness values can be risky if the fitness of the best-so-far solution is stored [10]. If a bad solution is, by chance, evaluated as very good, updates are done with respect to this bad solution, thus making it even harder to generate a good solution to win against the bad one.

Even if the fitness is re-evaluated each time, efficient optimization may still fail, due to high variance. This can be seen in our result for the $(\mu + 1)$-EA: each individual gets re-evaluated each iteration, but if the variance is high enough, it is likely that a very good individual will lose against a bad one.

cGA, however, does not store a best-so-far solution. It just compares two individuals that were generated in the same iteration. So their noise-free fitness values are expected to be the same. In addition to that, cGA only updates τ_i if the two individuals differ at position i. This way, it makes sure to only update if there is a visible tendency to either preferring a zero over a one or vice versa. The whole update process can be thought of as a so-called *gene pool recombination*, as introduced in [12]. That is why cGA is called a Genetic Algorithm (GA); it performs some kind of crossover.

Since the beginning of EAs, it has been argued that GAs should be more powerful than pure EAs, which use only mutation [9]. This was debated for decades, but theoretical results and explanations on crossover are still scarce. The robustness of GAs to noise lies intuitively in the *diversity* of the population. However, the concept of diversity changes from problem to problem and follows no true definition. Additional, analyses of diversity and crossover operators is often hard. So our result for cGA could give an additional explanation why crossover can be so good. The concept of diversity is compactly hold in the probability vector τ. (Note that the $(\mu + 1)$-EA does not do crossover but only mutation.)

We also ran experiments with cGA, comparing it against a standard meta-heuristic that is not inspired by nature: RLS – Random Local Search. RLS used re-evaluations of function values, to kind of cancel out the noise, cGA just used our results and no further adjustments, since it already scales gracefully. The outcome of these experiments can bee seen in Figure 2. Note that we did not measure wall-clock time but the number of function evaluations, since this often is the point of interest in theoretical analyses. These results show empirically for a wide range of noise intensities that crossover is beneficial.

The NO-variant of each algorithm stands for the *noise-oblivious* version of the respective algorithm. The NO-scheme does not know of the real value of σ^2 and just starts an algorithm with the parameters set for $\sigma^2 = 1$. If the optimum is not found during an upper run time bound that should hold with a high probability, the algorithm doubles its guess for σ^2 and completely re-starts, including the resetting of the parameters. So the scheme basically makes an uninformed search for σ^2. Note that this scheme can be implemented for a variety of different algorithms, such as MMAS-fp.

This is the way to go in real-world applications, since the variance is usually not known. The theoretical run time of an NO-variant of an algorithm is just worse by a constant factor, compared to the original algorithm. The experiments, however,

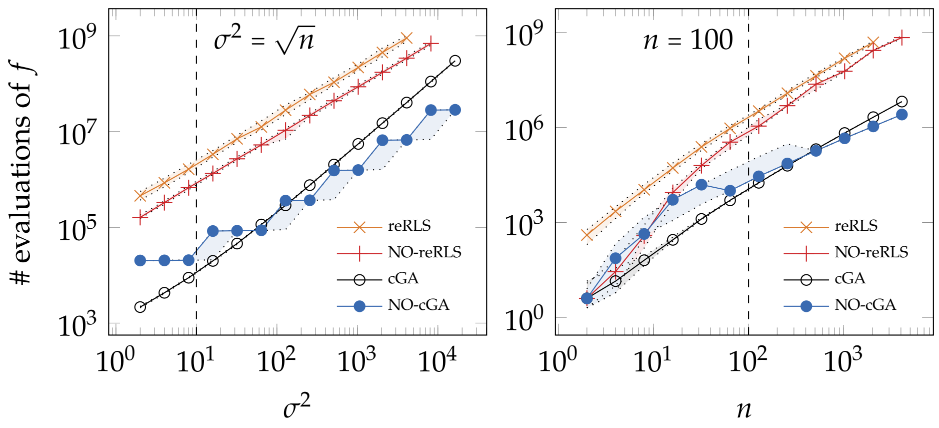


Figure 2: Median run time as a function of noise variance for $n = 100$ (left) and as a function of n for $\sigma^2 = \sqrt{n}$ (right). 100 runs at each point. Shaded area denotes interquartile range.

show that the NO-variant is usually even faster. This is likely a result from run time bounds for one run of an algorithm being worst case bounds and, thus, somewhat pessimistic. The NO-variant often succeeds with a rough approximation of σ^2 and therefore has a larger step size or does not do that many re-evaluations, resulting in a faster run time (with respect to number of function evaluations).

4 Conclusion and Outlook

We showed that the EDAs MMAS-fp and cGA scale gracefully for optimizing f, while this is not the case for the $(\mu+1)$-EA. Our results may hint that EDAs in general may be robust against noise in certain scenarios. The results for cGA show that crossover can help against noise.

It would be interesting to further investigate diversity and crossover in general to get an idea of why exactly it is preferable in many situations. This should ideally not only focus on EDAs but on GAs in general in a certain sense. Research in that area could answer questions that have been open for a long time.

Another extension to our work could be an even grander generalization of our results for MMAS-fp to not just optimizing f but a greater class of functions. It seems that the algorithm should be able to also optimize more functions. A generalization actually looks quite promising, since our proof concept is not that restrictive. Succeeding in doing so would result in a nice classification of robustness of MMAS-fp to a large noise model and function class, which would be nice to have.

Last, a closer look on EDAs in general, not necessarily under noise, would be interesting, because the framework can encapsulate quite an amount of algorithms. It would be good to know limits and merits and what properties suffice to achieve certain goals. This would shift the focus from very algorithm-specific analyses to a far more general viewpoint.

Overall, there are still many challenging opportunities left to be looked into.

References

[1] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. 1st. IOP Publishing Ltd., 1997.

[2] L. Bianchi, M. Dorigo, L. Gambardella, and W. Gutjahr. "A Survey on Metaheuristics for Stochastic Combinatorial Optimization". In: *Natural Computing* 8 (2009), pages 239–287.

[3] D.-C. Dang and P. K. Lehre. "Evolution under partial information". In: *Proc. of GECCO'14*. 2014, pages 1359–1366.

[4] B. Doerr and C. Winzen. "Playing Mastermind with Constant-Size Memory". In: *Proc. of STACS'12*. 2012, pages 441–452.

[5] S. Droste. "A rigorous analysis of the compact genetic algorithm for linear functions". In: *Natural Computing* 5.3 (Aug. 2006), pages 257–283. DOI: 10.1007/s11047-006-9001-0.

[6] S. Droste, T. Jansen, and I. Wegener. "Upper and Lower Bounds for Randomized Search Heuristics in Black-box Optimization". In: *Theory of Computing Systems* 39 (2006), pages 525–544.

[7] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[8] C. Gießen and T. Kötzing. "Robustness of populations in stochastic environments". In: *Proc. of GECCO'14*. 2014, pages 1383–1390.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[10] C. Horoba and D. Sudholt. "Ant Colony Optimization for Stochastic Shortest Path Problems". In: *Genetic and Evolutionary Computation Conference (GECCO'10)*. ACM, 2010, pages 1465–1472.

[11] Y. Jin and J. Branke. "Evolutionary optimization in uncertain environments — a survey". In: *IEEE Trans. on Evol. Comp.* 9 (2005), pages 303–317. DOI: 10.1109/TEVC.2005.846356.

[12] H. Mühlenbein and H.-M. Voigt. "Gene Pool Recombination in Genetic Algorithms". In: *Meta-Heuristics*. Springer US, 1996, pages 53–62. DOI: 10.1007/978-1-4613-1361-8_4.

[13] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[14] C. Witt. "Optimizing Linear Functions with Randomized Search Heuristics: The Robustness of Mutation". In: *Proc. of STACS'12*. 2012, pages 420–431.

Impacto: Simulating Physical Impact by Combining Tactile Stimulation with Electrical Muscle Stimulation

Pedro Lopes

Human Computer Interaction
Hasso-Plattner-Institut
pedro.lopes@hpi.uni-potsdam.de

We have advanced our research on wearable computing and proprioceptive interaction (i.e., interacting solely using the user's muscles for both input and output) by developing impacto, a device designed to render the haptic sensation of hitting and being hit in virtual reality. The key idea that allows the small and light impacto device to simulate a strong hit is that it decomposes the stimulus: it renders the tactile aspect of being hit by tapping the skin using a solenoid; it adds impulse to the hit by thrusting the user's arm backwards using electrical muscle stimulation. The device is self-contained, wireless, and small enough for wearable use, and thus leaves the user unencumbered and able to walk around freely in a virtual environment. The device is of generic shape, allowing it to also be worn on legs so as to enhance the experience of kicking, or merged into props, such as a baseball bat. We demonstrate how to assemble multiple impacto units into a simple haptic suit. Participants of our study rated impacts simulated using impacto's combination of a solenoid hit and electrical muscle stimulation as more realistic than either technique in isolation.

1 Introduction

The objective of virtual reality systems is to provide an immersive and realistic experience. While research in virtual reality has traditionally focused on the visual and auditory senses, many researchers argue that the next step towards immersion must include haptics, i.e., to allow users to experience the physical aspects of the world. In this paper we focus on one specific category of haptic sensation, namely impact, i.e., the sensation of hitting or being hit by an object. Impact plays a key role in many sports simulations such as boxing, fencing, football, etc.

Simulating impact is challenging though. Creating the impulse that is transferred when hit by a kilogram-scale object, such as a boxer's fist, requires getting a kilogram-scale object into motion and colliding it with the user. This requires a very heavy device. In addition, building up an impulse requires an anchor to push against (Newton's Third Law), typically resulting in a tethered device, e.g., SPIDAR [7]. Both clash with the notion that today's virtual reality hardware is already wearable and wireless.

In this paper, we propose a different approach. The key idea is to decompose the impact stimulus into two sub stimuli, each of which we can render effectively.

2 Impacto: electrical muscle & tactile stimuli

Impacto is designed to render the haptic sensation of hitting or being hit.

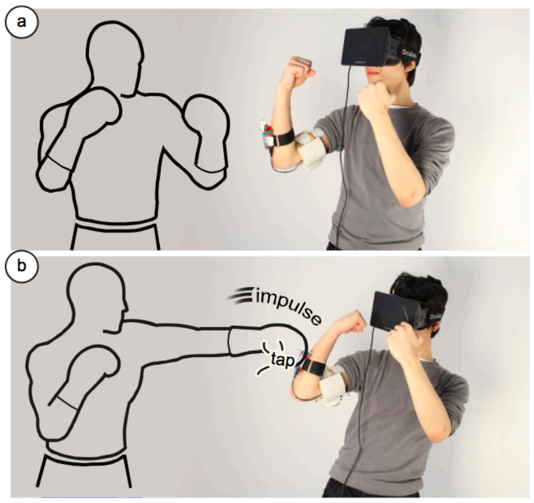


Figure 1: Impacto is wearable device designed to render the haptic sensation of hitting and being hit

Figure 1 illustrates our approach, here at the example of a boxing simulation. The key idea that allows the small and light impacto device to simulate a strong hit is that it decomposes the stimulus. It renders the tactile aspect of being hit by tapping the skin using a solenoid; it adds impulse to the hit by thrusting the user's arm backwards using electrical muscle stimulation. Both technologies are small enough for wearable use.

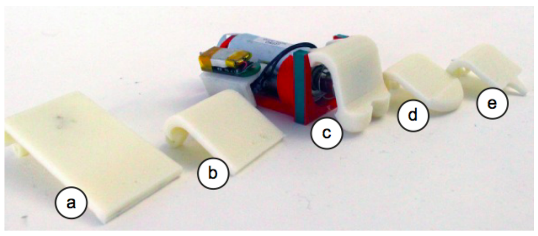


Figure 2: The solenoid component that we have in Impacto features a "knuckle" tip (c), which has a 90-degree lever to hit the skin orthogonally. The other four interchangeable tips are (a) generic surface, (b) small generic surface, (d) rounded, and (e) sharp.

Figure 2 shows the solenoid component in detail. To achieve a compact form factor, the solenoid is mounted parallel to the user's skin. A lever mechanism redirects its impact by 90 degrees, allowing it to hit the user's skin at a perpendicular angle.

Furthermore, we provide a set of exchangeable 3D printed tips to refine the desired tactile experience, e.g., to simulate boxing without gloves we use a tip that resembles human knuckles (Figure 2c). In addition to the knuckles, Figure 2 shows: (a) a generic surface, e.g., for punching a virtual avatar, (b) a small generic surface, e.g., for receiving a sharp impact, (d) a rounded surface, e.g., for jugging a ball, and (e) a sharp tip, e.g., for getting hit by a fencing weapon.

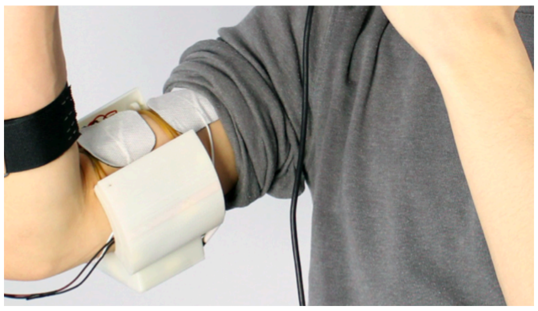


Figure 3: We show a detail of the electrical muscle stimulation component. Here, it stimulates the user's biceps brachii muscles causing an involuntary contraction that resembles force feedback.

Figure 3 shows the electrical muscle stimulation component. Its electrodes are mounted to the specific muscle that is able to render the impulse response that matches the solenoid. Here the solenoid is mounted to the outside of the arm, and therefore matches the impulse that would cause the arm to flex. Hence, we use the muscle that can flex the user's arm, i.e., we attach the EMS component to the user's biceps. When activated, the electrodes trigger an involuntary contraction of those muscles, simulating the transfer of impulse by thrusting the arm backwards.

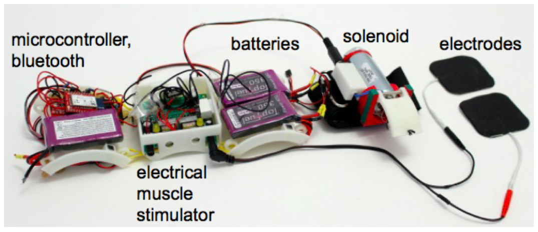


Figure 4: The Impacto bracelet opened to reveal its contents: Arduino microcontroller, bluetooth, electrical muscle stimulation, batteries, solenoid and electrodes.

Figure 4 shows the control unit that drives both solenoid and EMS components. We built impacto as a stand-alone and wearable device, with all electronics embedded in a bracelet. The solenoid module features a Velcro closure, allowing the device to be strapped to the user's upper arm, back of the hand, the user's leg and so forth.

2.1 Impacto's two components are mutually beneficial

Even though both technologies are small enough to allow for mobile or wearable use, it is their combination that creates a very strong sensation — in fact, stronger than either of the technologies by themselves (see "User Study"). However, solenoid and EMS play well together in more than one way:

1. Impacto simulates an impulse. The EMS component actually moves the arm. To create the required impulse, it creates a mechanical system between the limb and the user's torso. Given that the torso is comparably massive, there is very little effect on the torso and a strong effect on the limb.

2. The physical response produced by EMS is strong, despite the small form factor. It achieves this by leveraging the user's skeleton and muscles.

3. Because of the EMS, the solenoid can be small, wearable. Because the EMS is small but does the "heavy lifting", the task of the solenoid is limited to tapping the skin. This keeps the size of the solenoid down. With a small solenoid and EMS, we achieve a compelling simulation in a mobile/wearable form factor.

2.2 Benefits and Contribution

Our main contribution is the concept of impact simulation, its decomposition into tactile and impulse components, and the implementation of these two components using solenoid and electrical muscle stimulation. The main benefit of our approach is that it makes the simulation of a strong impact feasible in a small form factor. Our user study suggests that our approach generates a stronger sensation than either component in isolation. We demonstrate the use of our device in a series of virtual reality sport simulators.

On the flipside, simulating multiple impact locations requires multiple units, which places a natural limit on the spatial resolution of the simulation. Also, using a solenoid as a tactile feedback source adds inherent latency, which needs to be compensated for. Lastly, the use of EMS requires electrodes, which need to be manually placed by the user and calibrated prior to use.

3 Application Examples

We have implemented a few virtual reality sport simulators to demonstrate the potential use of impacto. All our examples use impacto for haptic feedback, an Oculus

Rift for visuals and a Kinect for tracking. We now describe the applications from the perspective of what the user feels.

3.1 Being hit—Boxing

Boxing is a sport for which the notion of impact is crucial. Figure 5 shows a screenshot of the simple boxing simulator we created to experiment with impacto. In this simulator, users can fight a virtual avatar by boxing. The avatar keeps its guard up and attacks periodically. Users must choose the right moment to unleash a successful attack. It takes ten successful hits to take down the avatar, which causes a new opponent to appear and the simulation continues.

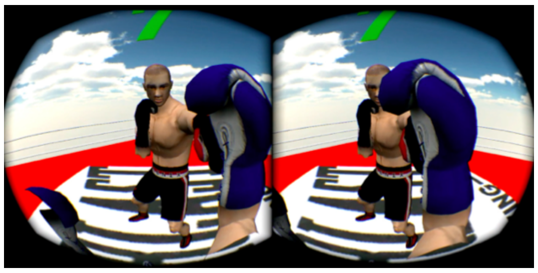


Figure 5: Stereo headset view from our simple boxing simulation. It allows users to attack the avatar and to block the avatar's attacks. Here, we see the avatar attacking the user's right arm.

Figure 6 illustrates how impacto adds a haptic component to the simulation: (a) the simulator provides haptic feedback when the user blocks the avatar, as discussed earlier. (b) The same impacto unit allows the user to hit the avatar using the part of the arm that wears the impacto unit, here the back of the arm.

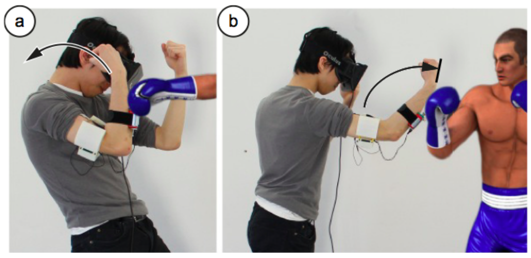


Figure 6: (a) Impacto allows users to feel the impact of blocking the avatar's hit by thrusting the user's arm backwards by operating the user's biceps. (b) The same impacto unit allows simulating the sensation of attacking. In both cases the impacto unit activates the user's biceps. This time, this causes the user's hand to stop in mid-air, as if it had hit the opponent.

3.2 Feeling Impact on Props—Baseball

The decomposition of impulse and tactile sensation transfers readily to hand-held props. Figure 7 illustrates this at the example of a simple baseball simulator.

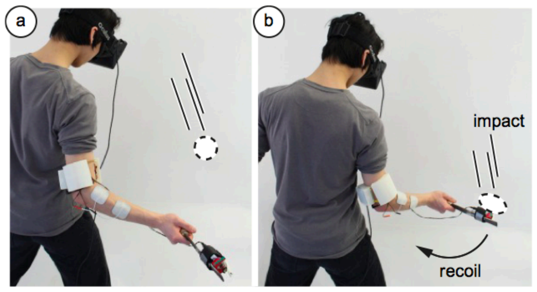


Figure 7: User hitting a virtual baseball.

In the baseball simulator, by wearing an impacto unit, the user experiences the impact of an incoming baseball against the bat. To enable the prop, here a stand-in for a baseball bat, we mount the solenoid onto the prop; the EMS unit, in contrast, stays with the user and stimulates the wrist extension muscle (extensor digitorium).

As illustrated by Figure 8, the same prop and electrode placement can power additional applications: by replacing the visuals in the virtual world and adjusting impacto's response, we can reuse the same prop to simulate a baseball bat, a fencing weapon, or a ping-pong paddle.

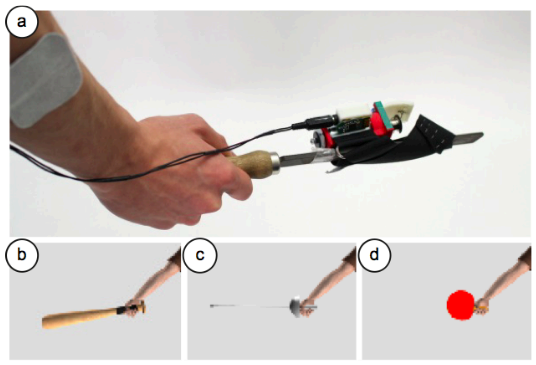


Figure 8: (a) Adjusting impacto's response and updating visuals turns the same prop into a range of different experiences, such as (b) a baseball bat, (c) a fencing weapon or (d) a ping-pong paddle

4 Related Work

The work presented in this paper builds on tactile stimulation, force feedback, virtual reality, and electrical muscle stimulation.

4.1 Tactile Stimulation

Tapping on the user's skin was, for example, used by Li et al. in order to convey messages [4]. Tapping is a special case of tactile feedback and it generally leads to a better tactile sensation than vibrotactile actuation because the tapping stimulates the SA1 receptors (Merkel cells) that sense pressure. Vibrotactile feedback, in contrast, is only sensed by the Pacinian corpuscles, which do not contribute to pressure sensing [3].

A common approach to recreate tapping is to emulate it using vibration. Lindeman et al. simulate impact in virtual reality using a suit that contains vibrotactile actuators [5]. In their virtual reality shooting application the suit communicates the spatial location of shots by activating the respective vibrotactile cell. However, there is no net force, thus no displacement of the user's limbs.

4.2 Force feedback

Impacto's way of simulating the transfer of impulse is a special case of force feedback.

Force feedback systems attach to the users' limbs using exoskeletons, such as the Utah Dextrous Hand Master [2] or pulley systems, such as SPIDAR [7]. Mechanical force feedback actuators of this kind tend to use an external apparatus mounted on the user, such as pulleys or an exoskeleton.

4.3 Force feedback using electrical muscle stimulation

More recently, researchers started administering force feedback using electrical muscle stimulation (EMS) to achieve force feedback in a compact form factor (e.g., Muscle Propelled Force Feedback [6]). Farbiz et al. used EMS on the wrist muscles to render the sensation of a ball hitting a racket in an augmented reality tennis game [1]. However, this system does not create a tactile component that supports the impact experience.

4.4 Transmitting impact through handheld props

Teck et al. simulate the impact of a virtual ball on a tennis racquet by attaching a solenoid to the prop [8]. They found that the output force generated by the high-power solenoid is two orders of magnitude below the force of a real ball hitting the prop [8]. This is why impacto is inspired by such approaches, but, additionally, uses EMS to render the strong force feedback sensation.

5 Implementation Details

To help readers replicate our design, we now provide the necessary technical details.

5.1 Impacto's Hardware

Figure 9 shows the circuitry inside the impacto bracelet. The bracelet uses three 7.4 V LiPo cells in series for a total of 22.2 V and 1050 mAh to drive the solenoid in the boxing simulation; for simulations that involve weaker impacts, such as football, we used half the voltage. The estimated power consumption is: EMS (0.1 A), solenoid (0.5 A to 0.7 A) and microntroller & bluetooth (0.2 A), allowing the unit to run for ≈ 2000 hits. The Arduino Pro Micro microcontroller (3.3 V, 8 MHz) receives commands from the virtual reality applications via a bluetooth module (RN42XVP).

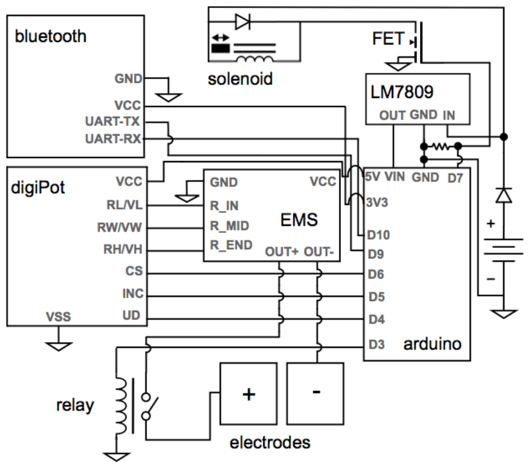


Figure 9: Schematic of impacto's circuitry

The microcontroller and EMS unit (TrueTens V3) are powered through a 9 V voltage regulator (LM7809). The solenoid receives power directly from the battery (22.2 V). Optionally, the solenoid power can be regulated down to 20 V via another adjustable voltage regulator (LM317).

The unit can control EMS and solenoid intensity separately. One non-volatile digital potentiometer (X9C103) controls the intensity of the electrical muscle stimulation; the microcontroller controls it via a 3-wire protocol. One relay (HFD4/3) switches the EMS channel on/off in 3 ms.

An N-Channel MOSFET (BUZ11) controls the intensity of the solenoid. It is sensitive enough to trigger at the low current output from the ATMEGA and can switch 30 A, which lies comfortably below the drain of our solenoid. The solenoid is bridged with a N4007 flyback diode to prevent the microcontroller from resetting due to the electromotive force that builds up when the solenoid is switched off. Modules that feature two solenoid outputs can switch between them using an additional relay (ommited from schematic).

5.2 VR Simulators and Tracking

We implemented all sports simulators in Unity 3D. All our applications use a Kinect to track the user's skeleton; it is connected to Unity 3D via the Microsoft's Kinect SDK Wrapper. The Unity3D system detects collisions using collider objects attached to the skeleton of the user as represented in the virtual world. When a collision is detected, the system sends a serial message over bluetooth to the impacto unit attached to that limb (each unit has its own bluetooth address). The message contains which EMS channel and solenoid to trigger as well as the desired intensity. Users experienced all applications through an Oculus V1 head mounted display.

The solenoid mechanics and wireless communication are inherently subject to 60 ms of lag. One way to make the system appear instantly responsive is to have Unity3D using colliders with bounding volumes 25 % larger than the actual limb, causing the collider to trigger ≈30 ms early, thereby compensating for the lag of the system. On the flipside, this technique does not work for targets spatially clustered together or if the user stops abruptly before the target, as it creates a false positive.

5.3 Measuring latency

We determined the device's lag using a series of measurements on the apparatus depicted in Figure 10. This apparatus drives impacto's solenoid, making it tap a load-cell (MSP6951-ND) sampled at 1 kHz by an ATMEGA328 microcontroller, and measures the time difference.

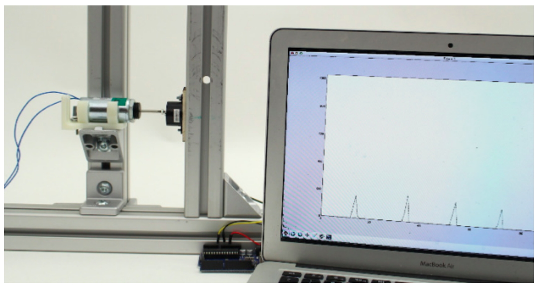


Figure 10: Apparatus for measuring force and latency

As a baseline we compare latency over bluetooth to direct serial connection, i.e., tethered over USB. Using a high-speed camera we measured 11 ms for the microcontroller to receive a single byte over USB and to turn on an LED in response. The HFD4/3 relays take a maximum of 3 ms to actuate (from datasheet). Using our apparatus, we measured that the solenoid takes 10 ms to 20 ms to extend fully and hit the load cell. Finally, our apparatus measured a latency of 50 ms to 60 ms for a tactile hit (bluetooth + solenoid mechanics), which lies within the haptic threshold of 50 ms to 200 ms, as set by psychophysics research.

5.4 Measuring loss at the 90° lever

To validate the mechanical design, in particular the deflection lever, which pulls the tip using a fishing string, we conducted a series of force measurements. The deflection lever redirects the solenoid's impulse by 90°, allowing the solenoid to be mounted parallel to the user's skin providing a much more compact form factor. We reused the apparatus shown in Figure 10 in two conditions, i.e., with and without the deflection lever.

Our measurements show that force exerted by the vertical hit (as in "User Study") is 26 N, while for the horizontally mounted solenoid, which hits through the 90° lever, we measured 21.1 N. Measurements are an average of 10 hits on the load-cell using the knuckle tip. These measurements clarify that both setups are comparable.

6 User Study

To validate the core idea behind impacto, i.e., the idea of decomposing an impact's haptic feedback into a tactile component (solenoid) and an impulse component (EMS), we conducted a user study. To do so, we immersed participants in a simplified study version of our boxing simulator in which they blocked punches by an avatar opponent. We varied the intensity of solenoid (no, low, high) and EMS (no, low, high) in a full-factorial design and asked participants to assess the realism of the punches. We hypothesized that the combination of both stimuli would lead to a more realistic experience.

6.1 Apparatus

Figure 11a shows our apparatus. Participants wore a head-mounted display (Oculus Rift V1). A single impacto unit was mounted to their right forearm, with the electrodes of the EMS component attached to the participant's biceps brachii muscle. We used an earlier design of impacto, however, it used the same EMS component and produced similar output force conditions as the bracelet (see previous section). For the tactile sensation we used the knuckle tip. To ensure a controlled experience, we used a scripted version of our boxing simulation, in which a video avatar repeatedly punched the participant (Figure 11b) on the dorsal side of their right forearm.

Participants were seated and held their arms in a guard position, so as to match the hands they saw in the video experience. Participants rested their elbows on the table between trials to reduce fatigue.

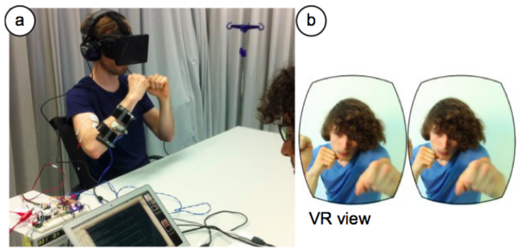


Figure 11: Experiment setup: (a) Participant wearing the head mounted display, electrodes on the biceps and the impact module on the right forearm. (b) The visual stimuli participants received through the head mounted display showed a first-person-view of a boxing experience.

6.2 Interface conditions

There were nine interface conditions, i.e., the full-factorial design of solenoid intensity (no, low, high) and EMS intensity (no, low, high).

In the high EMS conditions, the EMS component was calibrated to perform a full biceps curl, i.e., a 45 degrees movement from the default guard pose. In the low EMS conditions, the EMS component was calibrated so as to create the weakest visible contraction of the participant's biceps. In the no EMS conditions, the EMS component was off.

During setup, we made sure that participants felt comfortable with the setup and reached 45 degrees without any discomfort. This was the case for all participants.

In the high solenoid condition we overdrove the 12 V solenoid with 32 V for 200 ms, resulting in a strong (≈26 N) tap. In the low solenoid condition we operated the solenoid at its nominal voltage of 12 V for 200 ms resulting in a weaker (≈13 N) tap. In the no solenoid condition, the solenoid remained off.

6.3 Task and Procedure

For each trial, participants observed a 9 seconds video experience of being punched against their guard 3 times. This was accompanied by the respective haptic feedback created using the impacto unit. Participants then rated the realism of the punches on a 7-point Likert scale (1 = artificial, did not feel like being punched, 7 = realistic, like being punched).

Each participant performed a total of 27 trials: 3 force feedback settings (no EMS, low, or high EMS strength setting) × 3 tactile feedback settings (no solenoid, low, or high) × 3 repetitions. This yields a 3 × 3 within-subjects design.

The EMS calibration procedure took about 4 minutes during which the biceps contraction was repeated ten times to ensure that a similar contraction was found.

6.4 Participants

We recruited 12 participants (3 female), between 22 and 35 years old ($M = 26.9$ years) from a nearby organization. We excluded a thirteenth participant from the analysis who had stated that he/she had started with too high ratings, thereby producing a ceiling effect. One of the participants had boxing experience (sparring) and another was trained in martial arts. Two participants had never experienced a VR headset before and only one had experienced EMS before (in physiotherapy). With consent of the participants we videotaped the study sessions.

6.5 Results

Figure 12 shows the resulting data, i.e. participants' assessment of the realism of the punches as a result of the different haptic feedback conditions. We analyzed the data using a 3 (EMS) × 3 (solenoid) × 3 (repetition) repeated measures ANOVA ($\alpha = .05$). Since we found no learning effect as there was no main effect of repetition ($F2, 25 = 0.225, p = .800$), we used all three repetitions as data. As expected, we found main effects for force feedback (EMS, $F2, 14 = 89.726, p = .000$) and tactile feedback (solenoid, $F2, 14 = 56.840, p = .000$, Greenhouse-Geisser corrected for sphericity), i.e., higher solenoid intensity and higher EMS intensity both led to more realism. We did not find any interaction effect of EMS * solenoid ($F2, 14 = 1.524, p = .210$).

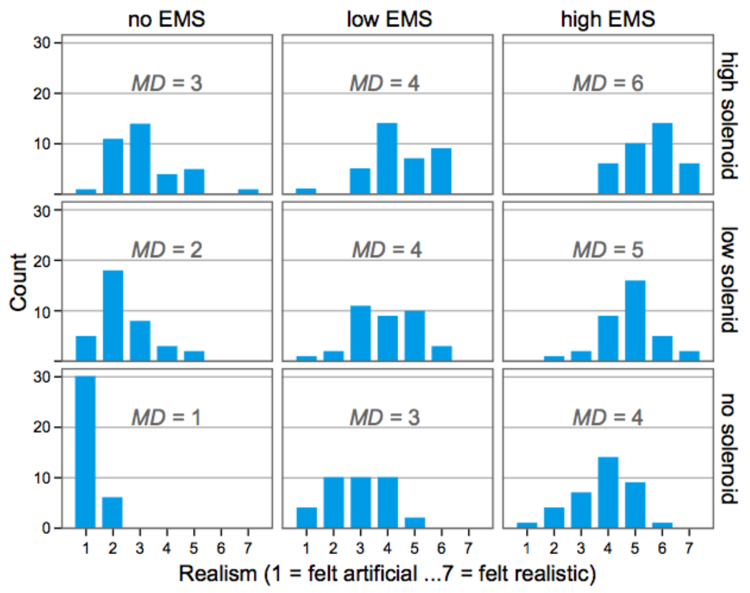


Figure 12: Realism ratings in dependence of force feedback (EMS) and tactile feedback (solenoid) conditions

Post hoc pair-wise comparisons using the test (Bonferroni corrected) confirmed the statistical differences across intensity levels for both, EMS (all pairwise comparisons, $p < .001$) and solenoid (all pairwise comparisons, $p < .001$).

6.6 Participants' feedback after the experience

After finishing all trials we interviewed participants about their experience.

Seven participants stated that they found the experience "immersive". Referring to the first time he/she had felt the combined effect P4 stated "it got immersive after a while, when I felt a stronger hit for the first time". P7 said "the first time I felt it, it was surprising, felt like a realistic force". P4 also added "I felt I needed to protect myself from the hits, it got real for me". P10 went further and stated: "this seems to really help VR, it is the most realistic VR experience I've ever had". P3, who was acquainted with boxing/sparring, stated "I know the feeling [impact] from sparing and this was really cool, could be even stronger [the solenoid hit]" and added "it is really impressive that this actually moves my arm". All participants stated that they liked the combined effect better than the individual effects, as suggested by their earlier assessments of "realism", P5 explained "the stimulation does not feel like a hit, but the combination really feels real because I suppose if you get hit your muscle moves back after the skin is hit". P8 said "I clearly felt that a hit [solenoid] and response [EMS] made it much more real". P7: "The solenoid feels like a punch and so its more important but then only with the EMS it felt real". Similarly, P9, who had 10 years of martial arts experience, said "solenoid is more important because it is like getting hit, but I prefer when both are on." P10 said "The EMS helps, but the primary thing is that it touched me." P12: "if you have solenoid, then the EMS really helps me to feel [that it is] real". P2 said "I was skeptical of the EMS during the calibration, but when I saw it in combination with the VR video and the solenoid, it was impressive".

Four participants stated that without the solenoid the experience feels unrealistic, such as "without the solenoid it was hard to understand when [the virtual boxer] hit me" (P3).

Three participants pointed out that the EMS tingling had slightly affected their sense of realism "I felt it vibrating, so that is a bit different from the pure movement" (P4).

When asked "what is missing for a fully realistic experience" participants answered: "resolution of the headset" (P10, P8), "remove the tingling caused by the EMS" (P12, P13), "it should also actuate my shoulder" (P4), and "the tactile part should be a larger surface, like a fist model" (P11, P9).

6.7 Discussion

Our study found main effects on both EMS and solenoid, suggesting that increasing the intensity of either of the haptic effects increases the perceived realism. The highest score, however, was achieved by combining both stimuli, supporting our hypothesis.

Participants' comments further support that hypothesis in that all participants stated that the combined effect had felt more realistic than either individual effects.

7 Conclusion and Outlook

We furthered our agenda on using muscles as I/O devices by introducing Impacto, a wearable device that allows users to experience impact in virtual reality. The key idea that allows the small and light impacto device to simulate a strong hit is that it decomposes the stimulus. It renders the tactile aspect of being hit by tapping the skin using a solenoid; it adds impulse to the hit by thrusting the user's arm backwards using electrical muscle stimulation. Both technologies are small enough for wearable use. We demonstrated a proof-of-concept module in three VR applications, each demonstrating that impacto enables a variety of haptic sensations, such as being hit or hitting back, by directly attaching it onto the user's body or even mediated through a passive prop. As future work, and to increase the fidelity of the force feedback, we plan to apply impacto to other locations such as the abdominal muscles or shoulders, as to generate a larger output motion.

As of the next steps we plan to: (1) expand upon the quality of EMS-based intearctive system to allow for spatial output; note that: all the EMS-systems I have developed or present in the related work, do not provide spatial output, such as, for instance, plotting a function; and, (2) investigate deep down what is the different between being actuated by electrical muscle stimulation and by a motor-based approach (e.g., an exoskeleton)?

7.1 Publications

Lopes, P., Ion, A., Baudisch, P.
Impacto: Simulating Physical Impact by Combining Tactile Stimulation with Electrical Muscle Stimulation.
at Proc. **UIST'15**, Charlotte, USA. Full Paper.

7.2 Leading a Workshop

Let your body move: electrical muscle stimuli as haptics
Pedro Lopes, Max Pfeiffer (Leibniz University Hannover), Michael Rohs (Leibniz University Hannover), Patrick Baudisch
Let your body move - a tutorial on electrical muscle stimuli as haptics, at **IEEE World Haptics**, Chicago, USA, 2015.

References

[1] F. Farbiz, Z. H. Yu, C. Manders, and W. Ahmad. "An electrical muscle stimulation haptic feedback for mixed reality tennis game." In: *Proc. SIGGRAPH (posters)*. 2007.

[2] H. J. and J. S. "Haptic Interfaces for Tel-eoperation and Virtual Environments." In: *Proc. Workshop on Simulation and Interaction in Virtual Environments*. 1995, pages 13–15.

[3] S. Kuroki, H. Kajimoto, H. Nii, N. Kawakami, and S. Tachi. "Proposal for tactile sense presentation that combines electrical and mechanical stimulus". In: *Proc. World Haptics*. 2007, pages 121, 126.

[4] K. Li, P. Baudisch, W. Griswold, and J. Hollan. "Tapping and Rubbing : Exploring New Dimensions of Tactile Feedback with Voice Coil Motors." In: *Proc. UIST*. 2008, pages 181–190.

[5] W. Lindeman, Y. Yanagida, H. Noma, and K. Hosaka. "Wearable vibrotactile systems for virtual contact and information display." In: *In Virtual Reality 9(2)*. 2006, pages 203–213.

[6] P. Lopes and P. Baudisch. "Muscle-propelled force feedback: bringing force feedback to mobile devices." In: *Proc. CHI*. 2013, pages 2577–2580.

[7] J. Murayama, L. Bougrila, Y. Luo, K. Akahane, S. Hasegawa, H. B., and M. Sato. "SPIDAR G&G: a two-handed haptic interface for bi-manual VR interaction." In: *Proc. EuroHaptics*. 2004, pages 138–146.

[8] F. Teck, C. Ling, F. Farbiz, and H. Zhiyong. "Un-grounded haptic rendering device for torque simulation in virtual tennis." In: *Proc. SIGGRAPH Emerging Technologies'12, Article 26*. 2012.

Integrating Complex Event Processing to Case Management

Sankalita Mandal

Business Process Technology
Hasso-Plattner-Institut
sankalita.mandal@hpi.uni-potsdam.de

In recent years, more and more systems are following event-driven approach. They produce events, consume events, react on events, take decisions based on events and also predict future paths from event-logs. On the other hand, case management is an approach where we don't consider the whole process at once. Instead, we break the process into smaller process fragments. Based on the case instance, a subset of these process fragments is executed. Here, an initiative is taken to connect the two worlds of complex event processing and case management. The approach emerges based on a scenario from logistic domain and evolves around the process fragments describing the scenario, events occurred during the execution of those fragments and the impacts created by the events on the fragments and associated data objects.

1 Introduction

Complex event processing (CEP) has been an emerging field in recent years and it is being more visible in different application domains every day. As truly said by D. Luckham, "Today, any kind of information system, from Internet to a cell phone, is driven by events" [5]. If we look around our everyday life, we can trace events everywhere. Starting from having a coffee in the morning, driving to office, submitting an assignment, withdrawing money from ATM or having an awesome retreat – everything can be interpreted as an event. Simple events can be aggregated to create higher level and more complex events. Using these events we can get relevant information, take decisions based on the information and react according to our decisions. Thus, complex event processing is considered to be a simple yet very powerful tool.

Now, business process management (BPM) is another field which has gained enormous popularity in the last decade. This is a key approach to organize work, and many companies represent their operations in business process models. But there are many domains where the processes often do not follow the routine or predictable path. Rather, depending on the situation, some variabilities of the intended process are executed. Therefore, the idea of case management (CM) is introduced [7]. For example, in many domains such as logistics, health care, agriculture; the processes cannot be always prescribed. They can vary extensively based on different situation. Some of the situations can be predicted based on historical data, but some may be completely unexpected. Therefore, the cases in these domains are treated separately, most of the time with the help of knowledge-workers. To avoid the high complexity and redundancy of a single process model describing all the variable situations, case management offers smaller process fragments which gives better understandability, more flexibility and efficient handling of changing situation.

Though complex event processing and case management appear to be two separate areas, there are strong connections between them. The variabilities in case management can often occur due to the occurrence of an event. The events can result in executing or aborting certain process fragments. If we can predict the occurrence of the events in a case and determine which set of fragments are to be executed in which order for this case, then we can have a far better control over the situation. We will be able to react to the changing situation in a faster and more efficient way. Undoubtedly, being able to react on time in today's changing situation adds a lot of business value to the organizations.

The structure of the report is as following. Section 2 introduces the fundamental concepts in the area of complex event processing as well as case management. Section 3 describes the motivating scenario based on which the connection between CEP and CM is explained in Section 4. The challenges and current approach of research are detailed in Section 5. Finally, Section 6 concludes the report.

2 Background

At this point, it is helpful to have a basic understanding of the two areas which are supposed to be connected through ongoing research, namely, complex event processing and case management. In this section, some fundamental definitions and concepts are introduced which will guide us through the rest of the paper.

2.1 Complex Event Processing

Before coming to the idea of complex event processing, let's make a few concepts clear. The following definitions will help creating a base for understanding event processing better [2].

Event An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain.

Event object The event object is an entity that represents such an occurrence in a computing system.

In the context of computing systems, we use the terms event and event object as synonyms.

Event processing Event processing is computing that performs operations on events. Common event processing operations include reading, creating, transforming and deleting events.

Event stream An event stream is a set of associated events; often a temporally totally ordered set.

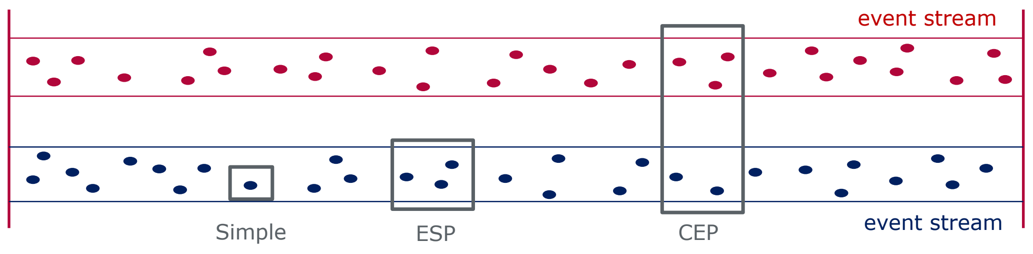


Figure 1: Event stream processing

Depending on the nature of input event stream, event processing can be of three different types. These are shown in Figure 1 and also described below.

Simple Event processing If the processing is done based on a single event then it is called simple event processing.

For example, a social network site wants to get notified whenever a user is logged in. Here, we have to act only on the individual login events.

Event Stream Processing (ESP): If the input source is a single event stream, but we need to act on multiple events in it, then it is named as event stream processing.

For example, a user wants a notification when the weekly withdrawal is more than 500 Euros. Now, we have an event stream with withdrawal events. For each week, these events are read. When the sum of withdrawn money exceeds 500 Euro, a notification is generated.

Complex Event Processing (CEP): Complex event processing signifies acting on multiple events from multiple event streams.

Let's think of a scenario where we need to notify a driver if there is an accident ahead of his path. Here, we need one event stream that informs us about any accident occurring on a specified route, also another event stream informing about the car's position at a regular interval. Only with these two event streams, we can determine if the accident has happened and it is ahead of the car or not. This is an example of complex event processing.

2.2 Case Management

Talking about case management, let's first look at the definition of a case.

Case: A Case is a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome [8].

The framework of case management represents a complex business process in smaller process fragments that are easier to understand. The fragments are interconnected by control and data flow. These fragments collectively describe the process behavior. Depending on each individual scenario, a subset of the fragments can be chosen for the specific case. During execution, it can be decided based on the path to follow [6].

The design phase for case management approach is very crucial. Here, the fragments composing predicted path are designed first. Then the exceptional behaviors are also taken into account and corresponding process fragments are modeled.

The aspects to be considered while designing the fragments can be listed as following:

Standard Procedures This covers the so called 'happy path' which should be followed if there is no exceptional situation during the process execution.

Variant Procedures In some cases, there can be additional activities to follow or certain activities to skip during execution. For example, if we consider a process of accepting a job application at HPI, for some cases, the research school can directly take the decision. But there can be cases where the applicant is more suitable for a particular group and the application is therefore forwarded to the chair.

Optional Procedures Sometimes, there can be some activities which may or may not be executed depending on the case. In above example of job application at HPI, if the applicant is from outside Germany, then the process may include an additional step of residence permit check. For the national applicants, this activity should be skipped.

3 Motivating Scenario

This section introduces the motivating scenario that runs throughout the paper. The source of the scenario is the EU project 'Green European Transportation' (GET Service) [3] in which our chair was one of the participants. GET service project came up with more efficient means for planning transportation in logistic domain thus, reducing CO_2 emission. The scenario presented here is a simpler version of Inland-Waterway Scenario from the GET project.

According to the current scenario, there are two orders to deliver. They start from the industrial park in Kechnec and should be delivered at Regensburg. The orders are transported from Kechnec to the port of Budapest via two different trucks and then they take the same vessel from Budapest to Regensburg. The order details along with the start time and delivery deadline are presented in the table below:

Order No.	Origin	Destination	Release Time	Delivery Time
1	Kechnec	Regensburg	Friday 09:00 am	Wednesday 09:00 am
2	Kechnec	Regensburg	Friday 12:00 pm	Wednesday 09:00 am

According to the order, an offline plan is made which includes the transportation details for each of the orders. The offline plan informs about the specific vehicles assigned to each order, the source and destination for each part of transportation and also the estimated time or arrival (ETA) scheduled for each destination. Figure 2 shows all these information and helps to understand the scenario better.

Now, the truck with order 1 starts from Kechnec on Friday at 9:00 am and drives towards Budapest. The truck with order 2 is ready to start at 12:00 pm. Both of them should take the vessel that leaves Budapest at 6:00 pm. At 11:00 am, a big accident takes place on the highway between Miskolc and Budapest. We call it event 'RoadTrafficDisruption'. The roads get closed due to the accident and it causes 2 hours and 15 min of delay. If we add this delay to the ETA of both of the trucks, then we see that the truck with order 1 can still reach Budapest before the vessel leaves. But in case of order 2, the new ETA is 6:15 pm and according to the current plan, it will miss the vessel. Definitely, we need to re-plan this transportation.

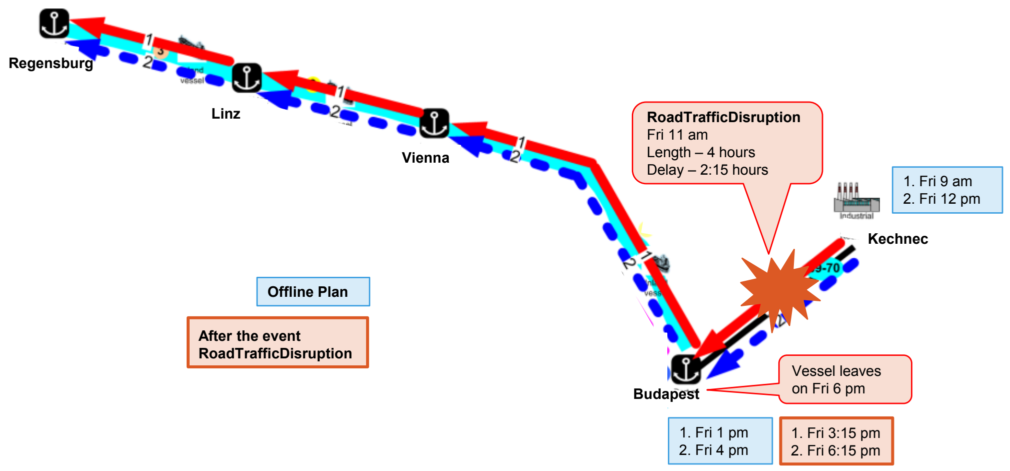


Figure 2: Event 'RoadTrafficDisruption' during transportation of orders

The planner gets the information about the event 'RoadTrafficDisruption' and re-plans so that order 2 now does not take the vessel from Budapest. Instead, the truck drives till port of Vienna and takes the vessel from there.

4 Interaction between CEP and CM

The scenario described above introduces us to the problem space of connecting event processing and case management. The event 'RoadTrafficDisruption' has important consequences on the transportation process execution.

Using BPMN 2.0 [4], we can model the whole scenario for order 2 as shown in Figure 3. As we can see, the process is quite long and several activities are being repeated. First, let's break down the whole process of transportation via truck and

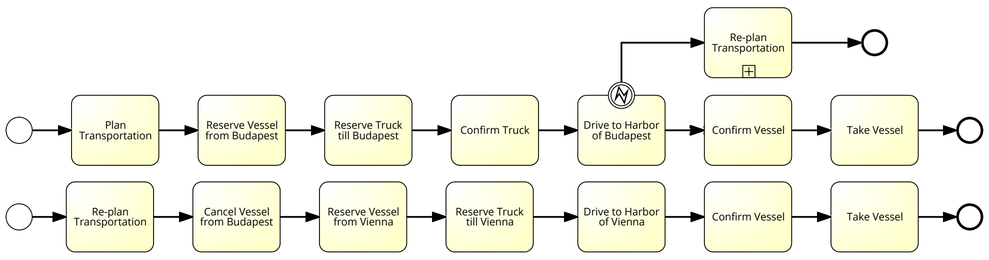

Re-plan Transportation
Plan Transportation
Reserve Vessel from Budapest
Reserve Truck till Budapest
Confirm Truck
Drive to Harbor of Budapest
Confirm Vessel
Take Vessel
Re-plan Transportation
Cancel Vessel from Budapest
Reserve Vessel from Vienna
Reserve Truck till Vienna
Drive to Harbor of Vienna
Confirm Vessel
Take Vessel

Figure 3: Process model for the motivating scenario

take into account the possible variants of that. If we split the process into smaller fragments, then we get the fragments described in Figure 4.

Based on the scenario, events can have three types of effects on process fragments. Each of them is described below:

4.1 Triggering Fragment

In the normal situation, we execute the fragments F1 and F2. But once we get notified about the event 'RoadTrafficDisruption', we need to execute F3 as well. Thus, the event triggers the execution of a certain process fragment.

4.2 Aborting Fragment

Due to the event 'RoadTrafficDisruption', we may need to stop executing some scheduled activities like driving to harbor, unloading from truck, loading to vessel etc. Here, an event can abort execution of one or more fragments.

4.3 Updating Data Object

In our scenario, after re-planning, the truck does not drive to the port of Budapest; instead it goes to the port of Vienna. Therefore, the fragment with activities 'Drive to Harbor', 'Confirm Vessel', 'and Take Vessel' is still executed, but the destination, route, estimated time of arrival etc. are updated according to the new plan. If we consider the data object 'Transportation Plan', then basically, the attribute values of this data object are updated due to the event occurred.

5 Challenges and Current Approach

Going by the described problem space, there are several questions to answer, several challenges to overcome. The most important ones are listed below:

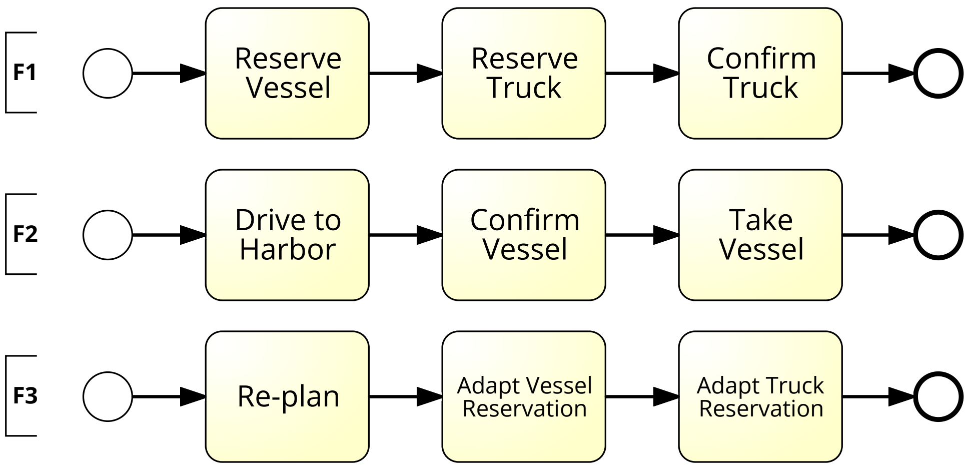


Figure 4: Process fragments for motivating scenario

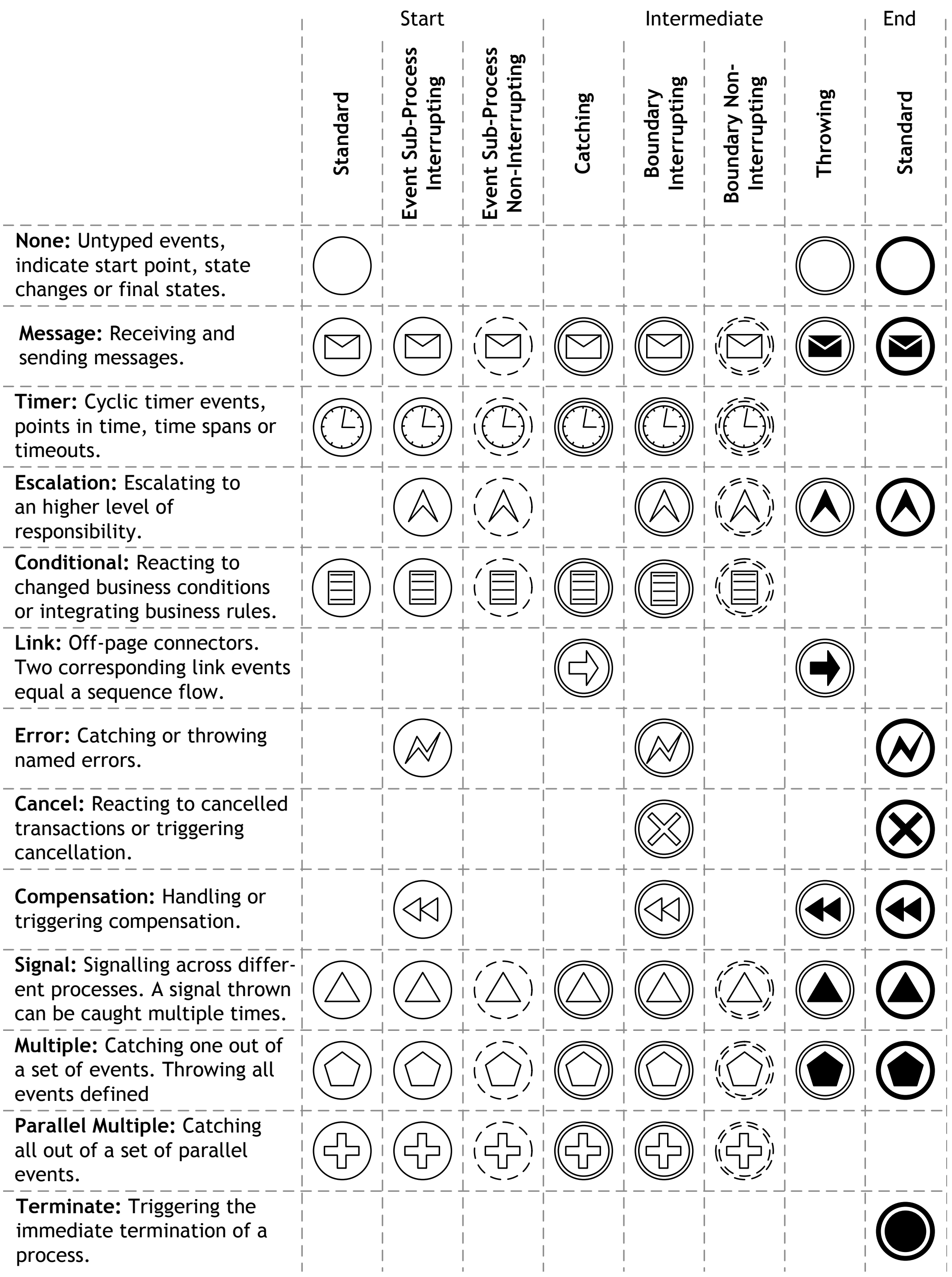
5.1 Types of events

The first question that we need to address is, *'What are the relevant events for the approach?'* According to BPMN 2.0 [4], there are start events, intermediate events, and end events which can occur at the beginning, during or end of a process, respectively. Those three event types can again be catching and/or throwing events.

Catching events are events with a specific trigger. We consider they take place when the trigger is fired. For example, a timer event is fired after a specified time is over. On the other hand, the throwing events trigger themselves instead of reacting to a trigger. A throwing message event can be an example of this. Now, there are various events for each of the categorization start, intermediate or end events and again sub-categorized by catching or throwing.

There can also be boundary events, which interrupt the execution of an activity or a sub-process. For example, during the activity 'Drive to Harbor' the exception event 'RoadTrafficDisruption' occurs, and it interrupts the ongoing activity.

Figure 5 gives an overview of the different event types included in BPMN 2.0. Now, it is required to find out the subset of relevant events that can be connected to case management approach from this vast pool of events. After classifying the BPMN events by relevance, the next step would be to sketch the exceptional events that can occur according to the use case at hand. Till now, we have considered only the 'RoadTrafficDisruption' event. But there can be, rather there will obviously be other exceptional events which can take place in the scope of the scenario. For example, the truck can be broken, the driver can fall sick or there can even be a storm. There can be events that occur only once, like the accident. But there can also be events that occur periodically and we need to take action based on that. An example can be the event notifying the driver taking a break. If we see that the driver is taking



	Start			Intermediate				End
	Standard	Event Sub-Process Interrupting	Event Sub-Process Non-Interrupting	Catching	Boundary Interrupting	Boundary Non-Interrupting	Throwing	Standard
None: Untyped events, indicate start point, state changes or final states.	○						○	●
Message: Receiving and sending messages.	○	○	○	○	○	○	●	●
Timer: Cyclic timer events, points in time, time spans or timeouts.	○	○	○	○	○	○		
Escalation: Escalating to an higher level of responsibility.		○	○		○	○	●	●
Conditional: Reacting to changed business conditions or integrating business rules.	○	○	○	○	○	○		
Link: Off-page connectors. Two corresponding link events equal a sequence flow.				○			●	
Error: Catching or throwing named errors.		○			○			●
Cancel: Reacting to cancelled transactions or triggering cancellation.					○			●
Compensation: Handling or triggering compensation.		○			○		●	●
Signal: Signalling across different processes. A signal thrown can be caught multiple times.	○	○	○	○	○	○	●	●
Multiple: Catching one out of a set of events. Throwing all events defined	○	○	○	○	○	○	●	●
Parallel Multiple: Catching all out of a set of parallel events.	○	○	○	○	○	○		
Terminate: Triggering the immediate termination of a process.								●

Figure 5: Event types according to BPMN 2.0

unusually frequent breaks, then we need to trigger a fragment with activities like checking if the driver is sick, replacing the driver and so on. Thus, the first step of the research would be to list and categorize all relevant events for the approach.

5.2 Realization of Fragments

The second challenge in the scope of current research approach would be to realize the process fragments. There are different aspects to consider in this regard. Some of them are described below.

Size First of all, we need to determine how to break down a process into efficient process fragments. There is no specification about the lower/upper bound of activities in a process fragment. There are unanswered questions like, *'Do we consider a single activity as a fragment?'* or *'What should be the ideal number of activities in one fragment?'*. We need to balance between reducing redundancy and grouping connected activities.

Mode If we think about the scenario stated earlier, the event 'RoadTrafficDisruption' happens when the truck with order 1 is already on its way to Budapest. So, fragment 2 has already started. Modeling this situation is a difficult task as the event interrupts the activity and as a result, the whole fragment is being aborted. But afterwards, the fragment is executed with the new plan. Now, the question is, *'What should be the mode of the fragment before and after the event has happened?'*. We need to think about the appropriate time to enable and disable a fragment. May be, we need to enable/disable few activities of the fragment instead of the whole fragment.

Order The next question in this series can be, *'How to determine the order of executing the fragments?'*. According to the current scenario, the fragments were supposed to be executed with the order $F_1 \rightarrow F_2$. But after the event occurs, the fragments are executed as following: $F_1 \rightarrow F_3 \rightarrow F_2$. We have to come up with a way to determine the scheduled order of fragments, change of order due to an event and the means to model both of them.

5.3 Role of data object

We still need to find out a lot more about the role of data object in the approach, both from modeling aspect and executing aspect. Generally, the data objects are modeled in BPMN without their attribute values, as attribute values are determined at instance level, not model level. In our approach, the specific values will be determined at instance level as well, but the intended change of certain attribute values may be generalized over the instances. For example, whenever there is an exceptional event of road accident, the plan is changed. Now, how the plan changes may be an instance specific decision by the planner, but the change in the data object 'Transportation Plan' is always there. The solution for modeling the above problem and connecting events with data object attribute values during execution would be a big challenge in this case.

5.4 Requirements for integration

After answering the conceptual questions, there will obviously be some technical challenges while implementing the approach. Included in this challenge will be the requirements for integrating the existing event processing platform Unicorn [9] and case management engine Chimera [1] – both developed in our chair, the architecture of the whole system, the technologies to be used, the input/output specification etc.

6 Conclusion

Case management is an approach where instead of the whole process; the focus is on smaller process fragments. The emerging trend of systems with event-driven approach calls for an integration of events with process fragments scoped by individual cases. Several aspects are to be considered while both the worlds of events and processes try to interact with each other, some of them being the selection of relevant events, their impact on the process fragments and the realization of efficient splitting of processes into fragments.

References

[1] *Chimera*. 2014. URL: https://bpt.hpi.uni-potsdam.de/Public/JEngineDoc (last accessed 2015-10-01).

[2] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., 2010.

[3] *GET Service: Efficient Transportation Planning and Execution*. 2015. URL: http://getservice-project.eu/ (last accessed 2015-10-01).

[4] ISO/IEC. *19510:2013: Information technology – Object Management Group Business Process Model and Notation*. International Organization for Standardization, Nov. 2013.

[5] D. C. Luckham. *Event Processing for Business: Organizing the Real-Time Enterprise*. John Wiley & Sons, 2011.

[6] A. Meyer, N. Herzberg, F. Puhlmann, and M. Weske. "Implementation Framework for Production Case Management: Modeling and Execution". In: *Enterprise Distributed Object Computing (EDOC)*. IEEE, 2014, pages 190–199.

[7] H. R. Motahari-Nezhad and K. D. Swenson. "Adaptive Case Management: Overview and Research Challenges". In: (2013). DOI: http://doi.ieeecomputersociety.org/10.1109/CBI.2013.44.

[8] OMG. *Case Management Model and Notation CMMN*. Object Management Group, May 2014.

[9] *UNICORN: A platform for event processing in business process management*. 2012. URL: https://bpt.hpi.uni-potsdam.de/UNICORN (last accessed 2015-10-01).

Exploring Latent Factors in Code Artifacts

Toni Mattis

Software Architecture Group
Hasso-Plattner-Institut
toni.mattis@hpi.uni-potsdam.de

Complex software poses a significant mental challenge when programmers need to understand which parts of the software interact in which way. Certain high-level concepts and connections are difficult to study because they might be scattered across many code artifacts, hidden in non-obvious places or implemented multiple times. We consider how probabilistic models, e.g. the widely used LDA topic model from natural language processing, can be applied to improve program comprehension. We discuss limitations, possible extensions and use cases for applying these models to source code artifacts.

1 Introduction

In order to understand complex software, tools can help to navigate and visualize code artifacts and their connections. They may also provide information retrieval and recommendation capabilities to satisfy the developer's information need. In order to be more useful than just showing a class diagram or enable full-text search, these tools should be based on a semantic model of the source code, which has to deal with problems like inconsistently named identifiers, missing types, meta-programming, or functionality which is scattered across numerous modules, so called *cross-cutting concerns*.

Natural language has been mined for structures and connections for decades. Among the most effective techniques to uncover similarities and links between documents are probabilistic latent factor models, such as the PLSA and LDA topic models. These will be investigated in the following sections, as they receive a growing attention from the software engineering community. They could eventually provide tools with models that can deal with inconsistencies and uncertainty, as in natural language, give new insights into large code bases and even generate new code.

2 Background

2.1 Probabilistic Models

Probabilistic models describe observable data X by approximating the underlying real distribution, from which X is a sample, with a function $P(X)$. Most of the time, these models are parametrized by a set of unobserved (*latent*) variables Θ, such that the model describes a formula $P(X|\Theta)$ or $P(X,\Theta)$. Depending on the structure of the model, the values given to Θ often allow to deduce high-level insights into the data.

Good Fit When discussing a probabilistic model, we consider the objective to choose parameters which maximize the probability $P(X, \Theta)$ or $P(X|\Theta)$ for any observable X. When a probabilistic model consists of multiple steps and layers, inferring good parameters is usually solved using step-wise approximations. As the inference is well understood, we focus only on the models.

2.2 Discriminative and Generative Models

In a probabilistic setting, models come in two flavors: discriminative and generative models.

Discriminative Models describe the probability distribution of "output" variables (Y) given the observations (X); they explicitly model $P(Y|X)$. A well-known discriminative model is *Logistic Regression*.

Generative Models model the joint probability $P(X, Y)$ of data and latent variables. They are suited for probability estimation and sampling in both directions, giving either $P(X|Y)$ or $P(Y|X)$ once fitted.

Especially with respect to our goal of assisting the developer, the ability to sample new data (e.g. example code) from our model is desirable. Therefore, we will focus on *generative models* in this work.

2.3 Topic Models

Topic models are the primary technique to detect high-level concepts in text artifacts based on the words they contain. The central idea is to model a topic as group of words which frequently occur alongside each other and subsequently assign topics to each text based on the words it contains.

In a probabilistic setting, topics are (multinomial) distributions over words, assigning each word a probability that it would occur in a text about exactly this topic:

$$\begin{aligned} \phi &= (\phi_1, \phi_2, ..., \phi_K) \\ \phi_k &= (p_{k,w_1}, p_{k,w_2}, ..., p_{k,w_V}) \end{aligned}$$

where K is the number of topics in the model, ϕ_k describes the word distribution in the k-th topic, p_{k,w_j} is the probability of word w_j occurring in topic k. Consequently, ϕ can be seen as a matrix having a row per topic and a column per word in a vocabulary of size V. Typically, K is chosen to be much smaller than V.

Text artifacts, also called *documents*, are also a (multinomial) distribution over words:

$$d_i = (q_{i,1}, q_{i,2}, ..., q_{i,V})$$

where d_i denotes the i-th document among M distinct documents and $q_{i,w}$ is the probability of word w occurring in the i-th document. Hence, the document-word-matrix d has V columns and M rows, making it a very large, sparse matrix.

Common topic models approximate document distributions d_i as mixture of topics, thus compressing the V-dimensional document vectors to K-dimensional topic vectors θ_i, each topic vector describing the weights (or probabilities) θ_{ik} of each topic k occurring in the document.

The PLSA Model A common model based on this decomposition of documents into topics is the *Probabilistic Latent Semantic Analysis* (PLSA) model [8]. Here, the topic-word-probabilities ϕ_k are interpreted as conditional $p_{k,w} = P(w|\phi_k)$. The document-topic-vectors θ_i store the conditional probabilities of a topic being selected given the document: $P(\phi_k|d_i)$.

For each word in a document, the probability of the word w occurring in the i-th document is modeled as the following equation, which represents our decomposition into k topics:

$$q_{i,w} = P(w|d_i) = \sum_k P(\phi_k|d_i)P(w|\phi_k)$$

This represents a process where for each word position in the text of d_i a topic k is first sampled from the multinomial distribution θ_i and subsequently the word w is sampled from the topic multinomial ϕ_k.

PLSA models are usually trained using the *Expectation Maximization* (EM) algorithm, also *Tempered Expectation Maximization* (TEM) has proven to generate good fits.

The LDA Model *Latent Dirichlet Allocation* (LDA) [2] extends PLSA by modeling how topic distributions are generated. The LDA model introduces two dirichlet priors, the K-dimensional α and the V-dimensional β vectors. The document-topic-mixtures θ_i are distributed according to $Dir(\alpha)$ and the topic-word-mixtures ϕ_k according to $Dir(\beta)$.

The following process describes document generation according to the LDA model:

1. For $i \in \{1,...,M\}$ choose $\theta_i \sim Dir(\alpha)$, M being the number of documents.
2. For $k \in \{1,...,K\}$ choose $\phi_k \sim Dir(\beta)$, K being the number of topics.
3. For each word at position $j \in \{1,...,N_i\}$ in document $i \in \{1,...,M\}$, N_i being the number of words in the i-th document:
   a) Choose topic $z_{ij} \sim Mult(\theta_i)$
   b) Choose word $w_{ij} \sim Mult(\phi_{z_{ij}})$

where Dir is the Dirichlet and $Mult$ the Multinomial distribution.

Algorithms for fitting the LDA model include variational Bayesian approximations [2], *Gibbs Sampling* and *Collapsed Gibbs Sampling* [20]. The latter is particulary easy to implement in software.

Link-LDA Especially in the context of hypertext, it might be beneficial to capture links to other documents in the model. One possible extension called *Link-LDA* [17] adds extra generation steps to LDA:

1. For each link l in the document i:
    a) Choose a link topic $z_{il} \sim Mult(\theta_i)$
    b) Choose target document $d_{il} \sim Mult(\Omega_{z_{il}})$

The newly introduced $K \times M$-matrix Ω defines the probabilities of a document being linked by a topic, which is not truly generative anymore as it serves as input to the model rather than being generated by topics.

Pairwise Citation LDA Pairwise Citation LDA [17] introduces a procedure for modeling linkage between documents in a more generative fashion. The topic affinity model η is a $K \times K$ matrix, such that $\eta_{z_\rightarrow, z_\leftarrow}$ denotes the probability that there is a link from topic $z_\rightarrow$ (source topic) to $z_\leftarrow$ (target topic).

The link-generating procedure adds these steps to the typical LDA procedure:

1. For each document pair (d_s, d_t):
    a) Choose topic $z_\rightarrow \sim Mult(\theta_s)$
    b) Choose topic $z_\leftarrow \sim Mult(\theta_t)$
    c) Generate link $d_s \rightarrow d_t \sim Bernoulli(\eta_{z_\rightarrow, z_\leftarrow})$

A serious weakness of this model is the high number of potential links (in $O(M^2)$) which makes randomized fitting nearly ineffective and non-randomized approximations computationally expensive.

Hierarchical Topic Models There are often latent hierarchies embedded in document graphs. Such hierarchy denotes a tree H, which is a subgraph of the full document-link-graph $G = (V, E)$, where one document node $d_{root} \in V$ is chosen as root node of the hierarchy and other nodes $d_{i \neq root} \in V$ select a single incoming vertex as their parent.

The *Hierarchical Document Topic Model* (HDTM) [23] models random walks from the root to a leaf and operates as follows. γ is the probability a random walk restarts at the root document, η and α are Dirichlet priors to the topic-word and topic-mixture distributions:

1. For each document $i \in G$ choose a topic $\beta_i \sim Dir(\eta)$
2. For each document $i \in G$:
    a) Choose path $c_i \sim RandomWalk(\gamma)$, let L be its length.
    b) Choose an L-dimensional topic mixture $\theta_i \sim Dir(\alpha)$
    c) For each word $j \in 1, ..., N_i$:
        i. Choose topic $z_{ij} \sim Mult(\theta_i)$

ii. Choose word $w_{ij} \sim Mult(\beta_{c_d,z_{ij}})$ where $\beta_{c_i,z_{ij}}$ denotes the topic at the z_{ij}-th position in c_i

By identifying documents with topics and sampling documents higher up the hierarchy more often, more general terms will be assigned to more general topics in the upper hierarchy levels, while more specific terms will end up at the leafs. The restart probability γ adjusts the tree structure, with high γ resulting in a very flat hierarchy and low γ resulting in a deep hierarchy.

2.4 Probabilistic Context-free Grammars

Probabilistic Context-free Grammars (PCFGs) [9] are models meant to capture the structure of textual information.

Normal context-free grammars consist of *production rules* (→) which map *non-terminals* to *terminals*, non-terminals, the empty string (ϵ), or any sequence (AB) or alternative ($A|B$) of them. PCFGs assign each alternative a probability vector indicating the chance of each choice being taken during a *parse* or while generating a string. A PCFG thus describes probability distributions over parse trees or strings.

As an example, consider the grammar of balanced parentheses (productions numbered for reference):

$$P \to^1 PP$$
$$P \to^2 (P)$$
$$P \to^3 ()$$

By putting high probability on $P(\to^1)$, one gets a flat, long sequence of parentheses. Increasing $P(\to^2)$ will increase the depth of nesting within randomly sampled strings, and increasing $P(\to^3)$ will shorten the length of the string.

Learning PCFGs PCFGs can be parametrized by methods such as *Markov-Chain Monte Carlo* (MCMC) [10] or spectral methods [6]. When no base grammar is at hand, a minimal context-free grammar can be constructed from examples using genetic algorithms or ant colony optimization [1, 12].

3 Related Work

There have been multiple approaches in exposing the high-level structure of source code to the developer.

3.1 Aspect Mining

Similar to topic models, an aspect mining algorithm tries to group code artifacts into *concerns* (with focus on *cross-cutting concerns* that may impede maintainability) [5, 11, 14]. Most aspect miners employ discriminative models, e.g. clustering of methods according to similarity measures based on call-graph similarity [18, 24], locality sensitive hashing (LSH) [18], and history [3]. Also execution traces [4] and formal concept analysis have been employed [22].

Aspect mining results can be consulted to refactor scattered concerns into modules using aspect-oriented programming (AOP) or context-oriented programming (COP) [7]. The features used in aspect mining can give insights in which direction (generative) probabilistic models can be adapted.

3.2 LDA on Code

Linstead *et al.* [13] have demonstrated the effectiveness of LDA on Java projects. A source file is treated as a document and words are extracted by tokenizing the source code into identifiers and filtering out common stop words.

Clean topics, such as database handling (with words "sql", "database", "jdbc") or file handling ("file", "path", "dir") were apparent. Other concerns related to server-client-architectures, thread pools, listener-event-structures, JSP templating and logging were clearly identifiable based on their words.

Tangling and Scattering as Entropy Topics provide a new approach to measure two important modularity metrics, *tangling* (amount of functionality in a module) and *scattering* (how distributed a concern is), as the information-theoretic *entropy* of a document's topic mixture and the entropy of how a topic is distributed across all documents [13] .

3.3 HDTM on Code

The Hierarchical Document Topic Model has been applied by McBurney *et al.* [15, 16] to facilitate automated source code summarization. In contrast to Linstead's approach using source files, this approach uses methods as documents. Links are represented by method invocations. The HDTM model fitted on the *JHotDraw* source code has been used to draw the five most important identifiers describing each method. In a preliminary study, three experienced developers rated the proposed words according to accuracy and contribution to a method's understanding, resulting in good overall ratings.

All in all, HDTM exhibits the potential to improve source code comprehension by tagging methods with the words that best describe the method's purpose.

4 Explaining Code

Motivated by the promising preliminary research in the area of employing latent factor models to code, the main questions are how to adapt the existing models to better fit the structure of code and how to employ them in tools.

4.1 Limitations and Possible Extensions of NLP Models

PLSA, LDA, its extensions, and HDTM originate from the analysis of natural language. They exhibit a range of limitations in their expressive power when applied to code, which suggests numerous possible improvements.

Structure Preservation In order to apply the models to source code, its structure must be flattened to match the notion of a document. Hence, none of the models is capable of explaining the structure of code. Also, it is unclear where document boundaries can be drawn: around a statement, a method, a class, or a source file?

An obvious choice might be the incorporation of PCFGs, which can encode the underlying *grammar* of the language and thus be able to reproduce syntactically valid code. Also, a topic model based on production probabilities rather than word distributions seems like an interesting choice to be explored.

Static Structure and Dynamic Behavior There is no concept yet which captures *behavior* seen at runtime alongside static source code. Especially models using random walks, e.g. HDTM, can easily be extended by replacing the random walk part with actual *execution traces*.

Code History and Authorship Source code artifacts often have multiple versions, which in turn are associated with developers. Extending models to consider the evolution of a code artifact, e.g. by modeling code updated at the same time as closely related, might be useful. Adding temporal factors to a topic model might offer additional use cases, e.g. grouping commits according to their topic, or determining which authors are specialized in which topic.

Auxiliary Artifacts Auxiliary artifacts, such as commit messages, related tickets in an issue tracker, hand-written documentation, or questions and answers on a Q&A platform, are often written in natural language and contain important information about the code related to these artifacts. Including these texts as side-information into probabilistic code models might not only improve topics, hierarchies and other latent factors, but also cluster and correlate all auxiliary artifacts in a way that can be useful to the developer.

4.2 Use Cases

A number of potential use cases that benefit from improved, code-aware models, can be imagined:

Navigation Navigation tools may, instead of showing the apparent structure of packages, classes and methods, also offer latent topical and hierarchical views on the system, allowing the developer to navigate code that is similar or related to certain topic, or to learn about some high-level connections inside a large code base. For an example of how such navigation can be presented to the user, see Figure 1. Auxiliary

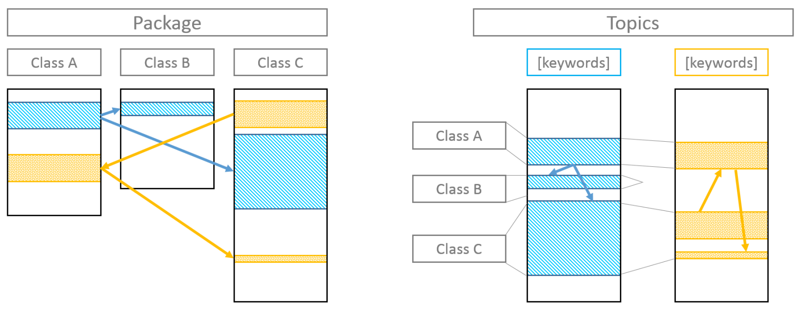


Figure 1: Proposed presentation of a code hierarchy (left) and a topic hierarchy (right) with topics and dependencies highlighted

information drawn from commit messages, tickets and authors may be displayed and support navigation and high-level system comprehension.

Code Artifact Recommendation Code recommendation can occur at multiple levels of immediacy, ranging from in-line *code completion*, over alerts popping up, to the use of external tools to explicitly obtain a recommendation [21]. Some of them may benefit from improved models, e.g. a context-aware code completion might rank proposed methods more prominently that are more probable in the current context.

Duplication prevention can recommend code which does a similar thing as the developer is currently trying to implement.

Aspect Recommendation can possibly be improved by probabilistic models to recommend code passages that may be refactored into layers, aspects or simply into a separate class.

Information Retrieval Related to code recommendation are use-cases where the developer explicitly formulates a question to the system. Questions that can be answered using latent probabilistic models are for example:

- How do I use this data type/method/API?
- How did the code that implements certain concern change over time?
- Which developers know most about certain code?
- How do unit tests look like for this kind of functionality?
- Has someone else already written code similar to this one?
- Which code artifacts might be related to a ticket?

Synonym Detection and Homonym Disambiguation *Synonyms* (different words having the same meaning) and *Homonyms* (same words having different meanings) make both navigation and understanding of source code difficult. Probabilistic models can detect which words probably occur interchangeably, e.g. if the user searches for usages of "offset", the search might also find "index" and "position". Also, if an overloaded term is searched, e.g."lines", the model may reveal whether this means "number of lines" or an actual collection of lines.

Inferring Unobserved Types Especially when test coverage is limited, tools for dynamic languages have difficulties inferring which types the programmer is dealing with in the current context. Execution traces, e.g. from test runs, can be used to harvest type information. Probabilistic models can subsequently help to infer most probable types and other dynamic information for unobserved code based on what has been seen during execution of other parts of the system.

Cross-Language Operations Topic models can be trained on multiple languages simultaneously, allowing cross-language recommendation and retrieval of code artifacts. Especially in the context of *domain-specific languages* (DSLs) a language-agnostic semantic code model can extend the capabilities of current DSL tools. Also, translation into other DSLs or natural language, like demonstrated by Oda *et al.* [19] can benefit from probabilistic models.

5 Conclusion

Probabilistic models originating from natural language processing can be augmented to give new insights into code bases and allow new and improved variants of development tools. Especially generative models are particularly interesting and start to gain traction in the domain of software engineering. We observe a wide range of yet unexplored extensions and usage scenarios of such models. Which of them actually improve the current state of the art in software engineering has yet to be explored.

References

[1] F. Benz and T. Kötzing. "An Effective Heuristic for the Smallest Grammar Problem". In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO '13. New York, NY, USA: ACM, 2013, pages 487–494. DOI: 10.1145/2463372.2463441.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* 3 (2003), pages 993–1022.

[3] S. Breu and T. Zimmermann. "Mining Aspects from Version History". In: *21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE '06*. Sept. 2006, pages 221–230. DOI: 10.1109/ASE.2006.50.

[4] S. Breu and J. Krinke. "Aspect Mining Using Event Traces". In: *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*. ASE '04.

Washington, DC, USA: IEEE Computer Society, 2004, pages 310–315. DOI: 10.1109/ASE.2004.12.

[5] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé. “Applying and combining three different aspect Mining Techniques”. In: *Software Quality Journal* 14.3 (Sept. 2006), pages 209–231. DOI: 10.1007/s11219-006-9217-3.

[6] S. B. Cohen, K. Stratos, M. Collins, D. P. Foster, and L. Ungar. “Spectral Learning of Latent-variable PCFGs: Algorithms and Sample Complexity”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pages 2399–2449.

[7] R. Hirschfeld, P. Costanza, and O. Nierstrasz. “Context-oriented programming”. In: *Journal of Object Technology* 7.3 (2008).

[8] T. Hofmann. “Probabilistic Latent Semantic Indexing”. In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’99. New York, NY, USA: ACM, 1999, pages 50–57. DOI: 10.1145/312624.312649.

[9] M. Johnson. “PCFGs, Topic Models, Adaptor Grammars and Learning Topical Collocations and the Structure of Proper Names”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. ACL ’10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pages 1148–1157.

[10] M. Johnson and T. L. Griffiths. “Bayesian inference for PCFGs via Markov chain Monte Carlo”. In: *In Proceedings of the North American Conference on Computational Linguistics (NAACL ’07)*. 2007.

[11] A. Kellens, K. Mens, and P. Tonella. “A Survey of Automated Code-Level Aspect Mining Techniques”. In: *Transactions on Aspect-Oriented Software Development IV*. Edited by A. Rashid and M. Aksit. Lecture Notes in Computer Science 4640. Springer Berlin Heidelberg, 2007, pages 143–162.

[12] B. Keller and R. Lutz. “Evolutionary induction of stochastic context free grammars”. In: *Pattern Recognition*. Grammatical Inference 38.9 (Sept. 2005), pages 1393–1406. DOI: 10.1016/j.patcog.2004.03.022.

[13] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. “Mining Concepts from Code with Probabilistic Topic Models”. In: *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. ASE ’07. New York, NY, USA: ACM, 2007, pages 461–464. DOI: 10.1145/1321631.1321709.

[14] M. Marin, L. Moonen, and A. van Deursen. “A common framework for aspect mining based on crosscutting concern sorts”. In: *13th Working Conference on Reverse Engineering, 2006. WCRE ’06*. Oct. 2006, pages 29–38. DOI: 10.1109/WCRE.2006.6.

[15] P. W. McBurney, C. Liu, C. McMillan, and T. Weninger. “Improving Topic Model Source Code Summarization”. In: *Proceedings of the 22Nd International Conference on Program Comprehension*. ICPC 2014. New York, NY, USA: ACM, 2014, pages 291–294. DOI: 10.1145/2597008.2597793.

[16] P. W. McBurney and C. McMillan. "Automatic Documentation Generation via Source Code Summarization of Method Context". In: *Proceedings of the 22nd International Conference on Program Comprehension*. ICPC 2014. New York, NY, USA: ACM, 2014, pages 279–290. DOI: 10.1145/2597008.2597149.

[17] R. M. Nallapati, A. Ahmed, E. P. Xing, and W. W. Cohen. "Joint latent topic models for text and citations". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pages 542–550.

[18] T. T. Nguyen, H. V. Nguyen, H. A. Nguyen, and T. N. Nguyen. "Aspect Recommendation for Evolving Software". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. New York, NY, USA: ACM, 2011, pages 361–370. DOI: 10.1145/1985793.1985843.

[19] Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura. "Learning to Generate Pseudo-code from Source Code using Statistical Machine Translation". In: *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Lincoln, Nebraska, USA, Nov. 2015.

[20] Z. Qiu, B. Wu, B. Wang, C. Shi, and L. Yu. "Collapsed Gibbs Sampling for Latent Dirichlet Allocation on Spark". In: *Journal Machine Learning Research* 36 (2014), pages 17–28.

[21] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors. *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[22] P. Tonella and M. Ceccato. "Aspect mining through the formal concept analysis of execution traces". In: *11th Working Conference on Reverse Engineering, 2004. Proceedings*. Nov. 2004, pages 112–121. DOI: 10.1109/WCRE.2004.13.

[23] T. Weninger, Y. Bisk, and J. Han. "Document-topic Hierarchies from Document Graphs". In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. CIKM '12. New York, NY, USA: ACM, 2012, pages 635–644. DOI: 10.1145/2396761.2396843.

[24] C. Zhang and H.-A. Jacobsen. "Efficiently Mining Crosscutting Concerns Through Random Walks". In: *Proceedings of the 6th International Conference on Aspect-oriented Software Development*. AOSD '07. New York, NY, USA: ACM, 2007, pages 226–238. DOI: 10.1145/1218563.1218588.

Adaptive Data Structure Optimization for Evolving Dynamic Programming Languages

Tobias Pape

Software Architecture Group
Hasso-Plattner-Institut
tobias.pape@hpi.uni-potsdam.de

Dynamic programming languages evolve over time. Using meta-programming is a common way of implementing new concepts that may ease the implementation and maintenance of large systems. However, these meta-level constructs can cause a performance overhead, as most execution environments are optimized for non-meta–level programs — with a potential impact on the programs using it. To alleviate this, a common approach is to adapt the execution environment, resorting to lower-level implementation means, handling special cases and optimizations for each new language element for better performance. On the other hand, programming language implementations are harder to maintain with every new special case for a certain feature. With adaptive data structure optimizations, that take ordinary, generic data structures and optimize them in the execution environment, no special cases in the execution environment nor in the programming language are necessary to obtain acceptable performance when using meta-level facilities to implement new programming concepts.

1 Introduction

Few programming languages remain unchanged over time. New concepts of programming, paradigms and methodologies, require adaption to languages and, hence, their implementations. The typical implementation process for programming languages, however, is complex, since a large number of programming languages are written in lower-level languages such as C or C++ mainly for reasons of performance. This holds especially for dynamic languages that typically run hosted on a Virtual Machine (VM). Sophisticated memory management, garbage collector (GC), multi-stage interpreters and just-in-time (JIT) compilers are complex tasks common to typical dynamic language VMs. These can make implementation complicated. Moreover, these implementations commonly have large dimensions. For example, the JDK8 implementation of Java amounts to half a million lines of code C/C++/Java and the V8 implementation of JavaScript to one million lines of C++, even excluding commentary. The sheer size is a challenge to the maintainability of programming language implementations and actually make the implementation of new concepts, language features, or changed semantics expensive, if not error prone.

On the other hand, dynamic programming languages typically have meta-level programming facilities that allow changing the language "from within". As such meta-level programs are written in the same high-level, dynamic language for which they are written, they can benefit from already present concepts from automatic memory management to extensive standard libraries, to name a few. Typically, these meta-level implementations of programming language elements and features have worse performance characteristics when compared with lower-level, in-VM implementations. There are several reasons for this. First, VM implementations often assume that programs do not change much during their execution, yet meta-level programs typically do exactly that, change the program. When meta-level programs are rare, this is typically no problem for program performance. However, implementing whole paradigms using meta-level facilities can lead to often-changing code. An example is the Squeak/Smalltalk implementation of context-oriented programming (COP) [14], ContextS, that uses meta-level handler of Squeak/Smalltalk to alter control flow based on contextual information; the performance impact is substantial [1]. The experience for other languages, such as JavaScript, Python, or Ruby has been similar. Second, beyond changing programs at execution time, meta-level programs for language concepts typically rely on layering behavior and data structures to alter or enhance existing language behavior or data structures, respectively. The application of Design by Contract (DBC) to the Racket language, a Scheme dialect, for example, has been done in a pure meta-level fashion. Yet a single contract for a procedure resulted in several hundred layered procedures [2]. Third, VM-level implementations allow introducing special cases new for language features and subsequently providing better performance. This has been done for context-oriented programming (COP) changes to a Java VM [13] and was proposed for DBC-support in Python.

We argue that fast generic VM optimizations, such as adaptive data structure optimizations [18] can alleviate the performance impact of meta-level implementations of programming language elements and support the evolution of dynamic programming language in a maintainable fashion.

2 Background

We provide some background information on selected programming language experiments as considered in this work and tracing JIT compilers

Design by Contract Design by contract [16] is a view on programming that augments typical declarations of state (for example, abstract data types) and behavior (for example, functions or method) with executable pre- and post-conditions. Originally part of the Eiffel language, several "native" implementations exist, for example in D, Fortess, or Clojure, but also as libraries, for example in Java, Common Lisp, or Racket, to name a few.

Context-oriented Programming Context-oriented Programming [14] provides a means for software modularity to cope with shortcomings of software decomposition dominated by one design factor. As such, COP focuses of the dynamic nature of software decomposition, taking in account dynamic stimuli to activate and deactivate specific control flow paths within one application while at the same time collocating the implementation of this context-dependent behavior.

Meta-tracing JITs Tracing just-in-time (JIT) compilers [3, 11, 17] optimize applications by observing the actual actions carried out by the program, recording, and optimizing these action for subsequent re-use. Meta-tracing JITs [6] differ in the application observed: rather than recording for the actual application, the programming language's interpreter is observed, making it possible to provide rather generic optimizations a broad range from applications and even programming languages can benefit from. For example, the meta-tracing JIT from the RPython toolchain powers implementations of Python (PyPy), Ruby (Topaz), PHP (HippyVM), Smalltalk (RSqueak), or statically typed languages such as Haskell (PyHaskell), to name a few.

3 Adaptive Data Structure Optimizations

Experience with language implementations mentioned above shows that new constructs or language elements are typically implemented with simple data structures provided by the runtime environment. This contrasts the "special-casing" approach that is typical when using the VM level for implementations Hence, providing a higher performance for generic data structures and faster execution for generic behavior should lower the impact of language element implementation on the meta-level.

Our approach of *adaptive data structure optimization* [18] shows that the overhead of more delegation-based data structures, such as trees and lists, can be reduced so much that other data structured such as vectors can be avoided.

3.1 Approach

Our optimization detects common patterns of how data structures objects reference each other. It then introduces short forms for these patterns, which we call *shapes*, that make it possible to represent these patterns more efficiently in memory.

A straightforward object representation would be a chunk of memory that stores pointers to all the fields. We call the contents *storage*. A descriptor contains information how many fields there are and how they are to be interpreted. This representation corresponds to the programmers' view.

In our approach, the storage area remains, but the descriptor is replaced by a shape. Like in the regular representation, the shape determines the meaning of its contents. There is a *default shape* that has no additional information compared to the straightforward representation. If the default shape was always used, the representation would be completely equivalent to the straightforward one. The difference to the straightforward representation is that a shape can additionally describe the shape of referenced objects, recursively. If a referenced object's shape is not specified in the referencing object's shape, it is stored as a reference in the storage. If the shape is specified, that object's content is inlined into the referencing object's storage. This process can be applied recursively.

To actually save memory, a shape has to be shard by as many objects as possible. Indeed, if every shape was used by only one object, the memory use is not improved.

Therefore, a new shape must only be introduced after runtime profiling makes sure that it occurs often enough.

3.1.1 Shapes and Recognition

A *shape* describes the abstract, structural representation of composite objects and is shared between all identically structured objects. Shapes are recursive; they consist of sub-shapes for each field in an object's storage. A special type of shapes denotes unaltered access to object content and termination of shape recursion. Objects with these shapes are treated as black boxes, for example, scalar data or unoptimized objects, they are stored directly in the storage. Storing the shape of objects may seem redundant given that the shape matches what it tries to describe. This only holds as long as no optimization has taken place. In this case, a object's shape is the *default shape* of its class and solely consist of *direct access* sub-shapes. Further, a mapping of replacement options for inlining (the *transformation rules*), and profiling data built up during object creation to aid the creation of new transformation rules (the *history*) are supplementary structures that we use to aid the inlining process.

The immutability of objects demands that all to-be-referenced objects that will constitute its content have already been constructed beforehand. Hence, their shapes will be available at construction time and we can count occurrences of *sub-shapes* at specific positions in the object. That way, we obtain a histogram of all possible shapes a referenced object can have. When a certain threshold of encounters has been reached, we generate a new transformation rule.

The transformation rules are mapping that drive the inlining process. When constructing a new object, they are consulted by the inlining algorithm. These mappings can be specified prior to program execution or inferred dynamically based on shape history.

Upon object creation, just after updating the shape history, we check whether the sub-shape counters hit a certain threshold, and if so, proceed to create a new shape that combines the object's current shape with the sub-shape that hit the threshold. In this new shape, we replace the *direct access* sub-shape at the position of the threshold hit with the sub-shape found in the history entry. The position of the hit, the sub-shape at that position, and the newly created shape are then recorded as new rule in the transformations table.

We call the process of recording the shape history and inferring transformation rules *shape recognition*.

3.1.2 Compaction

Since objects are immutable, compaction does only need to happen when creating new ones. With this premise, our optimization technique works by inlining the to-be-referenced objects into the to-be-created object upon its creation. The effect of this process is shown simplified with the example in Figure 1: creating a new node consisting of "1" and a rest list as in the figure. We start with a list of "1" and the rest list as initial content for the new object and a default shape. We iterate over the list and encounter "1" at position 0. For this example, we assume that the transformation table does not contain a mapping for "1" at position 0 and continue to the next position. At

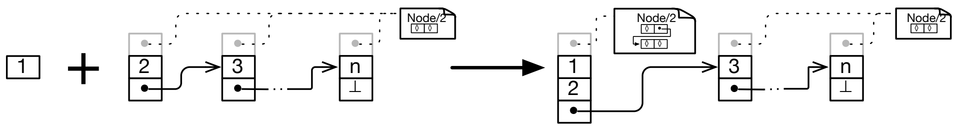


Figure 1: When creating a new node object that should contain "1" and the list as shown, a new object that merges the "1" with the "2" object and a different shape is created instead.

position 1, we find the rest list with the sub-shape we have a transformation rule for. Thus, we inline the current object's storage into the current content with now three elements. Crucially, the shape of the rest list remains unchanged while the shape of the to-be-created object is replaced.

The shape of thusly optimized objects are themselves subject to the shape recognition process and eventually, transition rules to more optimized shapes can be created in the default shapes for the classes. Thus, more specific shapes are directly available for the inlining process. Objects can be more directly transitioned into the most optimized shape compared to working off a long transition chain.

This inlining technique has two main advantages. First and foremost, inlined objects take up less space than individual, inter-referenced objects. But even more, the shape of a object provides structural information in a manner the meta-tracing JIT compiler can speculate on. This is crucial for optimizing the access to references of a object.

3.1.3 Transparent Field Access

While optimization of data structures takes place during construction, we have to apply the reverse during deconstruction, that is, when accessing a object referenced by another. This is no longer trivial, as several (formerly referenced) object may have been inlined into their referencing objects. Therefore, we construct new objects whenever a reference is navigated, essentially reifying it. We use the information a object's shape provides to identify which parts of the object's storage comprise the object to be reified. The structural information allows a direct mapping from the language view of the data structure to the actually stored elements.

3.2 Implementation and Results

We implemented our optimization approach in a simple execution model. It provides a λ-calculus with pattern matching as sole control structure and is implemented as a direct application of the control, environment, and continuation (CEK)-machine [9]. We used the RPython tool chain to incorporate its meta-tracing JIT compiler [4].

Our implementation has to interact with the JIT compiler. The first step is to treat the transformation tables as constant, which allows to compile object creation down to a series of type checks for the types of the referenced objects. Second, we have to avoid the otherwise necessary reification of referenced objects when it is being read

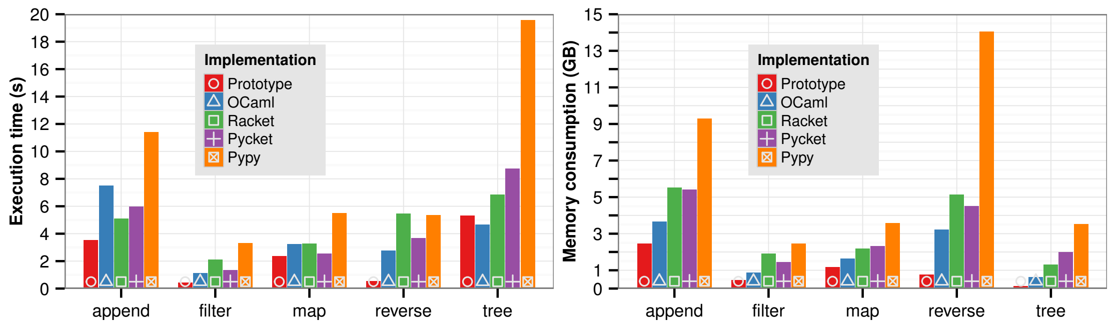


Figure 2: Benchmarking results. Each bar shows the arithmetic mean of ten runs for execution time (left) and memory consumption (right). Lower is better

from a object it has been inlined into. For that, the observation that most of these intermediate objects are actually short-lived is crucial; most object are created just to be either immediately discarded or consumed in another, typically larger data structure. This allows the tracing JIT compiler to optimize the reading of fields that need reification. Since the objects allocated when reifying a field are short-lived, the built-in escape analysis [5] will fully remove their allocation and thus remove the overhead of reification.

We tested five micro-benchmarks with our implementation and investigated their execution time and their maximal memory consumption (resident set size): *append*, *filter*, *map*, and *reverse* on very long linked lists and creation/complete traversal of a binary *tree*. In the left part of Figure 2, the execution time of all benchmarks is reported. Our implementation, labeled *prototype* 🟥, is significantly faster—from two to ten times faster—for all but the tree benchmark, where our implementation is second to just the ahead-of-time (AOT) compiled OCaml version. For memory consumption, shown in the right part of Figure 2, our implementation always uses significantly less memory than the other implementations.

Impact The results suggest that the otherwise typical overhead of deeply nested structures can be remedied. Pure delegation based systems, for example delegation-based multi-dimensional separation of concerns (delMDSOC) [19], create exactly such structures to facilitate the implementation of concepts like COP. It seems now possible to provide feasible and performant implementations of COP on the meta-level using this approach.

3.3 Fast Enough JITs for Meta-level

The meta-tracing JIT compiler—as employed to drive the implementation of our approach—has already proven itself to be useful to support meta-level language concept implementations in selected cases. This includes the implementation of *Pycket* [2], an implementation of the Racket [10] language. The careful optimization of certain basic structures (*loop finding*) of the Racket language reduced the overhead of the Racket-level DBC implementation with *Chaperones*, stand-in objects that check contracts upon procedure invocation. In Table 1 we present execution times for micro-benchmarks, testing the chaperone implementation, and macro-benchmarks that measure the impact of contracts on larger programs. While the impact of the new construct (chaperone/contract) tend to be substantial (up to 30 times slower), the implementation with meta-tracing JIT can mitigate this impact to a maximum of 13 times slower. For some benchmarks, the overhead is nearly completely eliminated.

Table 1: Benchmark times (in ms)

	Pycket	±	Racket	±
Bubble				
direct	564	5	1384	4
chaperone	656	6	6668	5
Church				
direct	656	10	1243	6
chaperone	5280	53	38497	66
Struct				
direct	114	1	527	0
chaperone	116	1	5664	68
ODE				
direct	2113	33	5476	91
contract	2564	31	12235	128
Binomial				
direct	1371	19	2931	24
contract	17563	112	52827	507

⁂

We now consider the meta-tracing JIT to be powerful enough to just eliminate the overhead of several often used yet simple data structures and behavioral patterns that tend to be used in today's programming language experiments. Especially the adaptive data structure approach suggest that commonly used proxy patterns for delegation could be used without the now typical overhead. Yet, the requirements on the language level to create a synergy with the VM level optimizations still require more in-depth investigation.

4 Related Work

Homogeneous Collection Specialization Storage strategies [7] have similar goals as adaptive data structure optimization and in fact benefit from one another. While they target different data structures, they are similar in spirit. The effect of storage strategies on meta-level implemented language concepts remains future work.

VM Frameworks and Language Evolution With the Klein VM [21], the Truffle VM framework and its Substrate VM approach [23], or the VMKit [12], there are efforts to on one hand ease the implementation of VMs, making fast, approachable programming language implementations feasible. While being similar in spirit, this work approaches the augmentation of programming languages and experimentation with new concepts on the level of the programming language itself rather than at the VM level.

The concept of meta-object protocols [15] approaches the augmentability of programming languages in terms of that very same language. However, typical implementations of meta-object protocols incur performance penalties when used heavily, as might be necessary for new programming language concepts and experiments.

Data Structure Optimization Wimmer has proposed object inlining [22] as a general data structure optimization for structured objects in Java. Superficially, this approach is similar to this work, yet object inlining is restricted to statically typed object oriented languages like Java, as the approach needs full knowledge of all class layouts.

The idea to improve data structures to gain execution speed was proposed especially to improve operations on linked lists in functional languages, for example by unrolling [20]. Typically, those optimizations are restricted to linked lists.

One of the key effects in our optimization is avoiding to allocate intermediate data structures. In that respect, *hash consing* [8], as used in functional languages for a long time, is related to this work. However, hash consing typically works at the language level using libraries, coding conventions, or source-to-source transformations. It is not adaptable at runtime.

5 Outlook

Generic VM optimizations are yet at an early stage of development, however, we consider the findings of previous work compelling enough to generalize the approaches taken by the adaptive data structure and Racket implementation examples and trying to transplant the findings from each to the other to find synergies and common patterns.

Following that, existing benchmarks from both approaches should give insight to what extent precisely other generic VM optimizations are feasible. Given they are at least as feasible as each approach on its own, we expect to apply the optimizations to other meta-tracing based implementations to then implement programming language concepts, such as COP, to assess specifically whether generic VM optimization meet their goal at facilitating programming-language level language experiments.

References

[1] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid. "A Comparison of Context-oriented Programming Languages". In: *COP '09: International Workshop on Context-Oriented Programming*. Genova, Italy: ACM, 2009, pages 1–6. DOI: 10.1145/1562112.1562118.

[2] S. Bauman, C. F. Bolz, R. Hirschfeld, V. Krilichev, T. Pape, J. Siek, and S. Tobin-Hochstadt. "Pycket: A Tracing JIT For a Functional Language". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*. ICFP '15. to appear. Vancouver, British Columbia, Canada: ACM, 2015.

[3] M. Bebenita, F. Brandner, M. Fahndrich, F. Logozzo, W. Schulte, N. Tillmann, and H. Venter. "SPUR: A Trace-based JIT Compiler for CIL". In: *SIGPLAN Notices* 45.10 (Oct. 2010), pages 708–725. DOI: 10.1145/1932682.1869517.

[4] C. F. Bolz. "Meta-tracing just-in-time compilation for RPython". PhD thesis. Mathematisch-Naturwissenschaftliche Fakultät, Heinrich Heine Universität Düsseldorf, 2012.

[5] C. F. Bolz, A. Cuni, M. Fijałkowski, M. Leuschel, S. Pedroni, and A. Rigo. "Allocation Removal by Partial Evaluation in a Tracing JIT". In: *Proceedings of the 20th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. PEPM '11. Austin, Texas, USA: ACM, 2011, pages 43–52. DOI: 10.1145/1929501.1929508.

[6] C. F. Bolz, A. Cuni, M. Fijałkowski, and A. Rigo. "Tracing the Meta-level: PyPy's Tracing JIT Compiler". In: *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*. ICOOOLPS '09. Genova, Italy: ACM, 2009, pages 18–25. DOI: 10.1145/1565824.1565827.

[7] C. F. Bolz, L. Diekmann, and L. Tratt. "Storage Strategies for Collections in Dynamically Typed Languages". In: *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*. OOPSLA '13. Indianapolis, Indiana, USA: ACM, 2013, pages 167–182. DOI: 10.1145/2509136.2509531.

[8] A. P. Ershov. "On Programming of Arithmetic Operations". In: *Communications of the ACM* 1.8 (Aug. 1958), pages 3–6. DOI: 10.1145/368892.368907.

[9] M. Felleisen and D. P. Friedman. "Control operators, the SECD-machine and the λ-calculus". In: *Proceedings of the 2nd Working Conference on Formal Description of Programming Concepts Pt. III*. Edited by M. Wirsing. Elsevier, 1987, pages 193–217.

[10] M. Flatt. *Reference: Racket*. Technical report PLT-TR-2010-1. PLT Design Inc., 2010.

[11] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. R. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. W. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Franz. "Trace-based Just-in-time Type Specialization for Dynamic Languages". In: *SIGPLAN Notices* 44.6 (June 2009), pages 465–478. DOI: 10.1145/1543135.1542528.

[12] N. Geoffray, G. Thomas, J. Lawall, G. Muller, and B. Folliot. "VMKit: A Substrate for Managed Runtime Environments". In: *SIGPLAN Notices* 45.7 (Mar. 2010), pages 51–62. DOI: 10.1145/1837854.1736006.

[13] M. Haupt. "Virtual Machine Support for Aspect-Oriented Programming Languages". PhD thesis. Software Technology Group, Darmstadt University of Technology, 2006.

[14] R. Hirschfeld, P. Costanza, and O. Nierstrasz. "Context-oriented Programming". In: *Journal of Object Technology* 7.3 (2008), pages 125–151.

[15] G. Kiczales, J. Des Rivieres, and D. G. Bobrow. *The art of the metaobject protocol*. MIT press, 1991.

[16] B. Meyer. "Design by Contract". In: *Advances in Object-Oriented Software Engineering*. Edited by D. Mandrioli and B. Meyer. Prentice Hall, 1991, pages 1–50.

[17] J. G. Mitchell. "The Design and Construction of Flexible and Efficient Interactive Programming Systems". PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University, 1970.

[18] T. Pape, C. F. Bolz, and R. Hirschfeld. "Adaptive Just-in-time Value Class Optimization: Transparent Data Structure Inlining for Fast Execution". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Volume 2. SAC '15. Salamanca, Spain: ACM, 2015. doi: 10.1145/2695664.2695837.

[19] H. Schippers, T. Molderez, and D. Janssens. "A graph-based operational semantics for context-oriented programming". In: *Proceedings of the 2nd International Workshop on Context-Oriented Programming*. ACM. 2010, page 6.

[20] Z. Shao, J. H. Reppy, and A. W. Appel. "Unrolling lists". In: *SIGPLAN Lisp Pointers* VII.3 (July 1994), pages 185–195. doi: 10.1145/182590.182453.

[21] D. Ungar, A. Spitz, and A. Ausch. "Constructing a metacircular Virtual machine in an exploratory programming environment". In: *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. OOPSLA '05. San Diego, CA, USA: ACM, 2005, pages 11–20. doi: 10.1145/1094855.1094865.

[22] C. Wimmer. "Automatic object inlining in a Java virtual machine". PhD thesis. Linz, Austria: Johannes Kepler Universität, 2008.

[23] T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. "One VM to Rule Them All". In: *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. Onward! '13. Indianapolis, Indiana, USA: ACM, 2013, pages 187–204. doi: 10.1145/2509578.2509581.

Trading Something In for an Increased Availability

Daniel Richter

Operating Systems and Middleware
Hasso-Plattner-Institut
daniel.richter@hpi.uni-potsdam.de

One way to increase the availability — the readiness for and continuity of correct service — of an IT system instead of rely on highly-available infrastructure and fault-free components is to question whether you can trade something in for it — like strong consistency guarantees, service quality, preciseness, or even correctness. The key idea is that in case of errors and failures one tries to ensure that the system does not turn off completely but degrades the provides set of functionality and ensures the efficient use of remaining intact resources.

To benefit from relaxed consistency, one has to identify different roles of components or code paths and their need for a specific level of consistency.

With flexible computation mandatory and optional parts are defined, where some optional parts can be deactivated. Imprecise computation means that depending on the runtime an algorithm is provided with, the accuracy of the result increases. Such algorithms usually are interruptible and can provide a result at any point in time. The most "extreme" approach is resilient computation, where the focus is to deliver a result under all circumstances — it is guaranteed to get a result, but that may is not correct.

1 Introduction

Besides reliability, availability — readiness for and continuity of correct service — is a dependability attribute [1] that is very important to modern IT-applications. Because of $Availability = {}^{Uptime}/_{Lifetime} = {}^{Uptime}/_{Uptime + Downtime}$, increasing an application's availability respectively guaranteeing a certain amount of availability means to reduce the downtime — the periods when a system cannot perform its primary function.

One mean to deal with faults, errors, and failures is to use highly-available infrastructure. But this solution usually is very expensive and cannot prevent the appearance of all classes of faults (e.g. design faults).

Another way to increase the availability is to question whether you can trade something in for it — for examples system properties like service quality, (strong) consistency, preciseness, or even correctness. Here, the key idea is that in case of errors and failures one tries to ensure that the system does not turn off completely but degrades the provides set of functionality and ensures the efficient use of remaining intact resources — error recovery and error mitigation.

In the following sections, we will focus on **relaxed consistency guarantees** and *relaxed computation guarantees* like **flexible, imprecise, and resilient computation**.

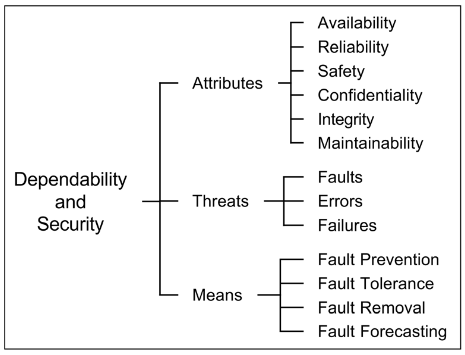


Figure 1: The Dependability and Security Tree shows the schema of the complete taxonomy of dependable and secure computing as outlined in [1]

2 Background

Our key idea to increase the availability of software systems without having highly-available infrastructure is to relax other system properties. This section will focus on consistency models, which targets at distributed applications with shared state or storage; and on service quality, preciseness, and correctness, which targets on the results of computing.

2.1 Consistency

According to the CAP theorem [3], one can only choose two out of the three CAP-properties: consistency (all nodes of a distributed system have the same view on memory), availability (every request receives a response), and partition tolerance (the system can deal with network partitioning). To have an available and partition tolerant (every distributed system has to be partition tolerant) system, one has to disclaim strong consistency.

While strong consistency requires that a read operation must return the value written by the most recent write operation, there are several other more relaxed consistency guarantees: [14]

Strong Consistency guarantees that a read operation returns the value that was last written. A read observes the effects of all previously completed writes.

Consistent Prefix guarantees that a read operation returns the value that was written at some time in the past. A read observes the effects of an ordered sequence of writes starting with the first write to a data object.

Bounded Staleness guarantees that a read operation returns a value that is not too old. A read observes any values written more than a specific time period ago or more recently written values.

Monotonic Reads guarantees that after a read operation a subsequent read operation returns the same value or the results of later writes. A read observes the effects of any previously completed writes.

Read my Writes guarantees that a read operation returns a value that was last written by the client or some other value that was later written by a different client. A read observes the effects of all previously completed writes performed by the same client.

Eventual Consistency means that a read can return any value for a data object that was written in the past. A read observes the effects of an arbitrary subset of previously completed writes.

Implementation methods While Strong Consistency and Eventual Consistency are the strongest respectively the weakest of the guarantees, none of Consistent Prefix, Bounded Staleness, Monotonic Reads, and Read My Writes is stronger than any other — they all could lead to read operation that returns a different value. It is also possible that a programmer wants to have multiple of these guarantees: for example both Monotonic Reads and Read my Writes, so that it observes a data store that is consistent with its own actions.

To benefit from relaxed consistency, one has to identify different roles of components or code paths and their need for a specific level of consistency. There may are components that need a global view on data and strong consistency, for other components it could sufficient to only know what they did, while some components can work with data that do not have to be up-to-date but should not be too old, and some other components may only need some value without having to know whether operations were performed in the correct order.

2.2 Flexible, Imprecise, and Resilient Computation

In context of trade in some properties related to computation and computed results, we roughly distinguish between the following categories:

Flexible computation Depending on the system state (appearance of errors, system loads) some optional components may are deactivated or replaced by alternative components (i.e. faster, smaller, more reliable).

Imprecise computation Depending on the runtime an algorithm is provided with, the accuracy of the result increases. Such algorithms usually are interruptible and can provide a result at any point in time.

Resilient computation The focus here is to deliver a result under all circumstances: it is guaranteed to get a result, but that may is not correct.

2.2.1 Flexible Computation

With flexible computation, an application is divided into parts that are mandatory and parts that are optional. Optional parts may have a priority. In case of faults inside optional parts the application can continue its operation without these parts. Likewise, in case of resource shortages (e.g. because of heavy load) optional parts could be (temporarily) deactivated to have more resources available for the required parts.

Implementation methods Imagine an image processing application: the mandatory part (service A) performs computations that are vital for reliable image interpretation. An optional part (service B) highlights several areas of the image that are convenient for the users. Another optional part (service C) post-processes logging data. Service C could use spare processing time to process logging data that is not important for the application, to save some time when the logging data is needed. Service B is also optional but has a higher priority than service C; in case of heavy load one could first dismiss instances of service C (because its purpose has nothing to do with service A) and later dismiss instances of service B (because its purpose is only for convenience but not mandatory).

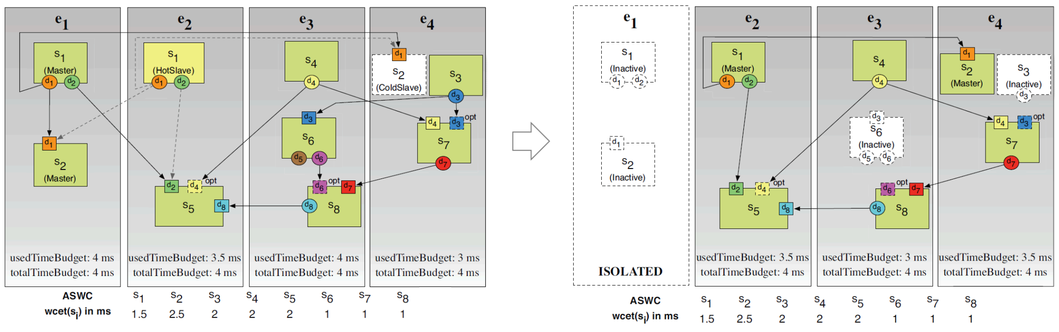


Figure 2: Deployment reconfiguration for isolated execution nodes after a fault [2]: the initial deployment (left) is reconfigured in case that execution node e_1 has been isolated after a fault (right).

The differentiation between mandatory and optional parts inside distributed environments produces additional challenges: a possible crash of an execution node itself has to be taken into account (so plenty of parts would become unavailable at same point in time) as well as the distribution of redundant parts that could take over the computation in case of errors (see 2, Figure 2).

We can divide the workload models into *flexible tasks* with 0/1 constraints, where the goal is to either execute a task to completion or discard the execution entirely, and *imprecise tasks*, where the result gets better the longer the computation runs.

2.2.2 Imprecise Computation

Imprecise computation means that a computation could be aborted at any point in time and also delivers a result; with increased run-time the quality of the result will improve (*IRIS* [8, 9], "Increasing Reward with Increasing Service"; "anytime algorithms" [15, 16]).

Implementation methods For imprecise computation, there are to basic implementation methods [9]:

- The *milestone method* is suitable for iterative algorithms, where the result converges to the precise value over time, e.g. numerical computations, statistical estimation and prediction, or heuristic search. The idea is to save the result produced by specific points in time and to return the last stored result in case of a cancellation of the computation.
- The *multiple version method* consists of a primary version and one or more alternatives for the computation. The primary version calculates the precise result but usually takes more time or needs more resources. The alternatives are smaller and/or faster, but imprecise. After an alternative has computed a result, the primary version possibly could be canceled.

2.2.3 Resilient Computation

Flexible and imprecise computation focus on tasks with a flexible resource demands but hard timing constraints—they degrade gracefully by reducing demands and result quality. But what happens when the application cannot meet its deadline? First, one could differentiate between firm deadlines (hard real-time) and the existence of a failure window (soft real-time). For many applications the result returned after exceeding the deadline is still useful to a certain degree (see 3, Figure 3).

Resilient computation guarantees that every request receives a response within a given deadline under all circumstances by executing a safe exit strategy. The goal is to hold or bring back the system in a stable state, or a state where it is possible for the system to operate. [11]

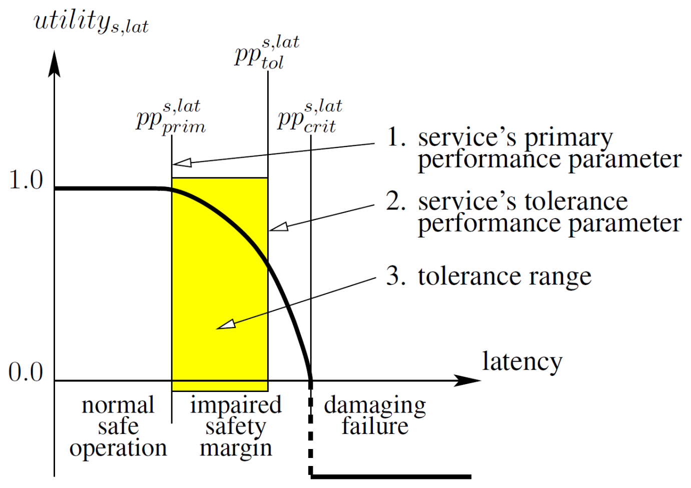


Figure 3: Latency utility of a real-time service with tolerance range [7]

Whereas the usage of fault tolerance patterns [6] can help to hold or bring back the system in a stable state, acceptability-oriented computing focuses on mechanisms how continue the execution and hold the system in a state where it is possible to operate — preciseness and even correctness are of secondary importance. [4, 12]

One way to produce a result within a specific time-window is to discard tasks that take too long or perforate loops — this could be similar to imprecise computation, when an iterative task gets canceled, but usually the cancellation point cannot be controlled by the developer.

Another way to get a result is to generate or estimate a value independently of the actual algorithm. This could be done by a fixed value, a randomly chosen one, or based on the history of previously returned values. History-based approaches could return the most recently returned value, the minimum/maximum/mean/average value of the returned values within a specific time-frame, or could use prediction algorithms to make a guess for the most probable value that could be returned.

In order to ensure a continued execution and output sanity, for all these methods one has to define acceptability properties. In case of acceptability violations further steps have to be performed.

2.3 Criteria of Optimality

The main challenge to introduce relaxed consistency guarantees; flexible, imprecise, and resilient computation is to define a criteria of optimality — there are plenty of options, but how to choose the best one? One objection is to execute as many optional tasks as possible (flexible computation), get the most precise result (imprecise and resilient computation), or get a strong consistent view on memory as fast as possible (relaxed consistency guarantees).

Criteria of optimality could be static error metrics (such as minimize total error, average error, or maximum error). For approaches such as imprecise computing usually domain-specific error measures are needed — e.g. confidence intervals or minimum mean square error).

These criteria then can be used for schedules for task execution (off-line or on-line) generated with the help of constraint optimization or prioritization. For time-bound tasks it is also important to distinguish between service time and response time (service time + waiting time).

3 Case Studies

The following section describes two case studies we implemented: a project with DB Systel (relaxed consistency guarantees) and Mobility-as-a-Service (flexible and imprecise computation).

3.1 DB Systel — Relaxed Consistency Guarantees

The objective of the project with DB Systel, the IT arm of the Deutsche Bahn AG, was to evaluate a given JEE-like reference architecture if it was possible to replace the current data storage component, that has strong consistency constraints, with one that supports weaker consistency guarantees.

A group of students adapted three example applications with different expectations on consistency and replaced the used relational database with NoSQL databases such as Cassandra, MongoDB, and NuoDB.

Feasibility evaluation showed that it is possible to use data stores with relaxed consistency guarantees. The hardest part is to use (or introduce) an application programmability interface that does not assume that the underlying database system is relational and strongly consistent.

3.2 Mobility-as-a-Service — Flexible & Imprecise Computation

Mobility-as-a-Service (MaaS) describes a class of applications where traditional real-time control systems are enhanced by remote (non real-time) backbone services accessed via the mobile Internet. Wide-distance communication channels cannot guarantee compliance with any real-time constraints. Using approaches such as analytic redundancy and safety controllers [5, 13], our architecture (see Figure 4) employs the concept of cascaded processing stages each with own safety controllers and allows for decoupling of hard real-time processing on embedded control units and soft real-time data acquisition on the outer layers. [10]

The idea of flexible computation is to have mandatory and optional parts; imprecise computing contains the multiple version method, where a primary version delivers the full functionality respectively a precise result, whereas an alternative version needs less resources but is imprecise. The Mobility-as-a-Service architecture is a combination of these two approaches: the application's logic is separated into multiple cascading stages that are connected via a communication channel and that have specific (different) timing constraints. In case any following cascade stage cannot fulfill its deadline, its result is ignored (= optional part) and instead the result of a safety controller (= alternative part with less functionality) is used.

We implemented and evaluated the Mobility-as-a-Service scenario with a customized Carrera slotcar racetrack with four stages (see Figure 6 and Figure 5). Stage 4 has the weakest timing-constraints, whereas stage 1 has to comply with strict deadlines.

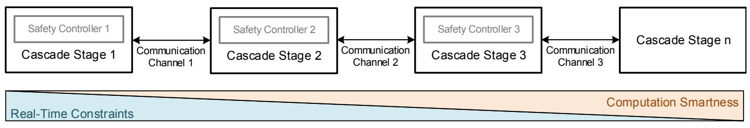


Figure 4: Our architecture blueprint for the cascaded interoperation of several layers of real-time functionality assisted by safety controllers

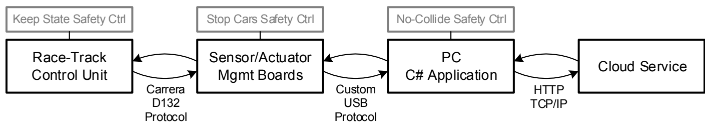


Figure 5: Our architecture blueprint applied to our concrete real-time simulation with Carrera slot-cars consisting of four cascade stages

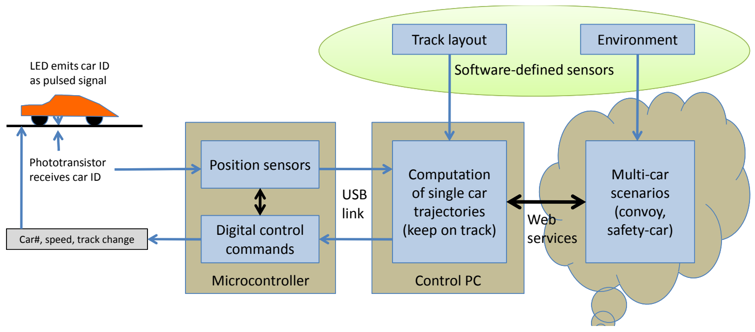

Track layout
Environment
Software-defined sensors
LED emits car ID as pulsed signal
Phototransistor receives car ID
Car#, speed, track change
Position sensors
Digital control commands
Microcontroller
USB link
Computation of single car trajectories (keep on track)
Control PC
Web services
Multi-car scenarios (convoy, safety-car)

Figure 6: Communication structure among real-time controllers and non-realtime services. We used an off-the-shelf Carrera racetrack construction kit and extended it with multiple position sensors.

Stage 4—Cloud Service Consists of a set of web services that implements a variety of multi-car driving strategies. The cloud manages driving strategies such as "convoy" or "safety car" where a global world view is needed.

Stage 3—PC Application A .NET application (written in C#) running on commodity PC hardware without any real-time guarantees. It computes the trajectory for a single car. *Communication Channel to stage 4*: HTTP/TCP/IP connections, where no guaranteed response times can be specified. *Safety Controller* (when stage 4 misses its deadline or the communication fails): Keeps the car moving at a steady, slow speed.

Stage 2—Sensor/Actuator Board Consists of a custom made circuit board and microcontroller firmware. It simulates handheld controllers (speed, brakes, turn switch points) and analyzes sensor signals. *Communication Channel to stage 4*: A custom USB format (neither real-time capable nor providing strict timing guarantees) *Safety Controller*: Keeps all cars within a safe state (sends commands to stop all cars)

Stage 1—Race-Track Control Unit An off-the-shelf Carrera Digital 132 race-track including a *Carrera control unit* extended with several additional position sensors. *Communication Channel to stage 2*: One RS232 interface and four RJ-12 interfaces are used to transmit standard Carrera D132 protocol commands. *Safety Controller*: Repeats the creation of command packets based the last signal received.

The outcome was that for many real-time applications compute and network capacity of today's commercial-of-the-shelf hardware allows to meet real-time deadlines even when operating on a best-effort model. The concept of software-defined sensors — in conjunction with the model of analytic redundancy — allows extensibility while still guaranteeing fail-safe behavior of the control system.

4 Conclusion and Outlook

To increase the availability of an application one first has to think about the following things: does the focus lie on reliability or on functionality? Is the goal perfection or can one accept that software is flawed? Is it still possible to deliver a service even when the software is partially faulty? Today's applications usually are distributed and rely on a set of many (third-party) components. Consequences of errors and failures are arbitrarily related in time, space, and severity to the cause.

By relaxing system properties like consistency guarantees and result quality, preciseness, or correctness, there are methods given so one does not have to rely on highly-available infrastructure or fault-free components. For an adequate implementation it is important to characterize the workload model and a role model for parts and components of the application.

Future work contains a more formal description how to identify and model parts of an application and algorithms that's properties can be relaxed, evaluate and prioritize which properties can or should be relaxed first or least, as well as providing patterns and blueprints for implementation.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing". In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (2004), pages 11–33.

[2] K. Becker and S. Voss. "Analyzing Graceful Degradation for Mixed Critical Fault-Tolerant Real-Time Systems". In: *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. IEEE, 2015, pages 110–118.

[3] E. A. Brewer. "Towards robust distributed systems". In: *PODC*. 2000, page 7.

[4] M. Carbin, S. Misailovic, and M. C. Rinard. "Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware". In: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications*. OOPSLA '13. New York, NY, USA: ACM, 2013, pages 33–52. DOI: 10.1145/2509136.2509546.

[5] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar. “The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures”. In: *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pages 400–412.

[6] R. Hanmer. *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.

[7] R. Kirner, S. Iacovelli, and M. Zolda. “Optimised Adaptation of Mixed-criticality Systems with Periodic Tasks on Uniform Multiprocessors in Case of Faults”. In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2015 IEEE International Symposium on*. IEEE, 2015, pages 17–25.

[8] K.-J. Lin, S. Natarajan, and J. W. Liu. *Imprecise results: Utilizing partial computations in real-time systems*. National Aeronautics and Space Administration, 1987.

[9] J. W. S. W. Liu. *Real-Time Systems*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[10] D. Richter, A. Grapentin, and A. Polze. “Mobility-as-a-Service: A Distributed Real-Time Simulation with Carrera Slot-Cars”. In: *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. IEEE, 2015, pages 276–279.

[11] M. Rinard. “Acceptability-oriented computing”. In: *ACM SIGPLAN Notices* 38.12 (2003), pages 57–75.

[12] M. C. Rinard, C. Cadar, D. Dumitran, D. M. Roy, T. Leu, and W. S. Beebee. “Enhancing Server Availability and Security Through Failure-Oblivious Computing.” In: *OSDI*. Volume 4. 2004, pages 21–21.

[13] L. Sha. “Using simplicity to control complexity”. In: *IEEE Software* 18.4 (2001), pages 20–28.

[14] D. Terry. “Replicated data consistency explained through baseball”. In: *Communications of the ACM* 56.12 (2013), pages 82–89.

[15] S. Zilberstein. “Operational rationality through compilation of anytime algorithms”. In: *AI Magazine* 16.2 (1995), page 79.

[16] S. Zilberstein. “Using anytime algorithms in intelligent systems”. In: *AI magazine* 17.3 (1996), page 73.

Analysis of Distributed Algorithms on Scale-free Networks

Ralf Rothenberger

Algorithm Engineering Group
Hasso-Plattner-Institut
ralf.rothenberger@hpi.uni-potsdam.de

Processor design has reached the so called "power wall": simply increasing computational power by increasing frequency and power intake of individual processors is not justifiable anymore since the imposed hardware problems outweigh their gain. Instead the trend goes to parallelizing IT systems to gain computational power. These changes in hardware and systems design were also accompanied by a change in algorithm design.

Although parallel algorithms and parallel computing have been intensively studied in the 80's and 90's, this work was mainly on the PRAM model and for homogenous networks. Our aim is the analysis of distributed processes and algorithms on heterogeneous networks, more precisely, on scale-free networks. For this purpose we proposed and analyzed load balancing on Chung-Lu random graphs as a first specific problem in the context of scale-free networks. At the moment of writing we are extending our results to satisfiability problems with non-uniform variable distributions.

1 Introduction

Large Peer-to-peer (P2P) systems like BitTorrent, PAST Storage, RetroShare or Skype are decentralized and unstructured. The fact that peers are connecting and disconnecting from the network has implications about the nature of the overlay topology. In practice, because peers tend to discover highly available and high-outdegree nodes, connections are typically formed preferentially. As shown by Barabási and Albert [5], this results in a *scale-free graph* in which nodes follow a power-law distribution [2, 29], meaning that the number of vertices with degree k is proportional to $k^{-\beta}$, where β is a constant intrinsic to the network. Another motivation to study scale-free graphs is the observation that many real-world networks are power-law networks, including Internet topologies [17], the Web [5, 26], social networks [1], industrial SAT instances [4], and literally hundreds of other domains [28].

Although Scale-free networks are omnipresent, surprisingly few rigorous insights are known about the performance of algorithms or processes on them. Some notable exceptions of such rigorous results are Rumor Spreading [18], Information Diffusion [23] and (amongst others) computing PageRank [11]. These results show that algorithms and processes on scale-free topologies are often able to outperform their counterparts on more homogeneous structures. For this reason we propose the design and analysis of distributed algorithms which exploit the unique structural properties of scale-free networks. As a first example of such algorithms we looked into distributed load balancing.

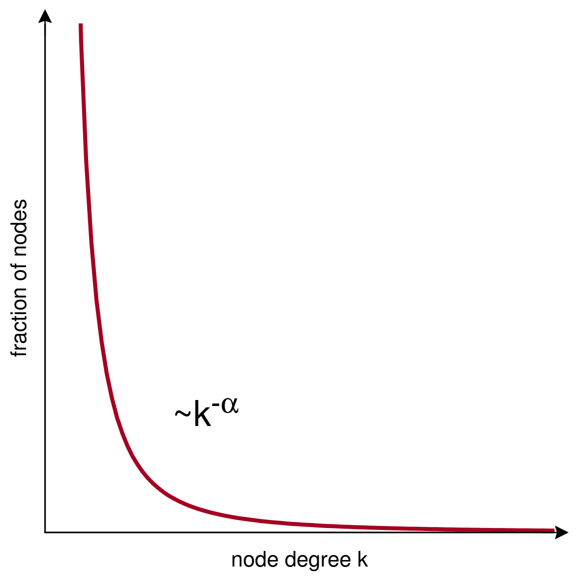


Figure 1: Power-Law distribution of node degrees

2 Background

2.1 Scale-Free Networks

A popular model for scale-free networks is the preferential attachment (PA) model introduced in [5], which is currently the fifth most cited article in *Science* according to ISI Web of Knowledge. In this model, the graphs are constructed in a random, 'rich-get-richer' fashion: a newly entering node connects to m existing ones chosen randomly, but gives preference to nodes that are already popular, that is, those nodes that have many neighbors. Note that the parameter m controls the density of the graph. For these graphs, the authors of [5] empirically discovered that the fraction of vertices with degree d is proportional to d^{-3}, which was later proven mathematically by Bollobás, Riordan, Spencer, and Tusnády [9]. There are a number of other models which lead to a power-law distribution [16, 21, 25]. An interesting alternative is the model of Aiello, Chung, and Lu [3] where the number of nodes of a given degree is fixed in advance according to some probability distribution. This model is related to the well-studied random graphs with given degree sequence [6, 8, 32] and tends to be easier to analyze than PA graphs as the edges are present independently.

One particular drawback of the previously mentioned models is that the resulting networks have either a small clustering coefficient, or a rather large average path length (typically of at least logarithmic order in the size of the network). Hence they still differ considerably from real networks. In the last few years it has been observed that complex scale-free network topologies with high clustering coefficients emerge naturally from hyperbolic metric spaces [13, 33]. There seems to be a close relationship between hyperbolic geometry and complex networks. This can be explained by

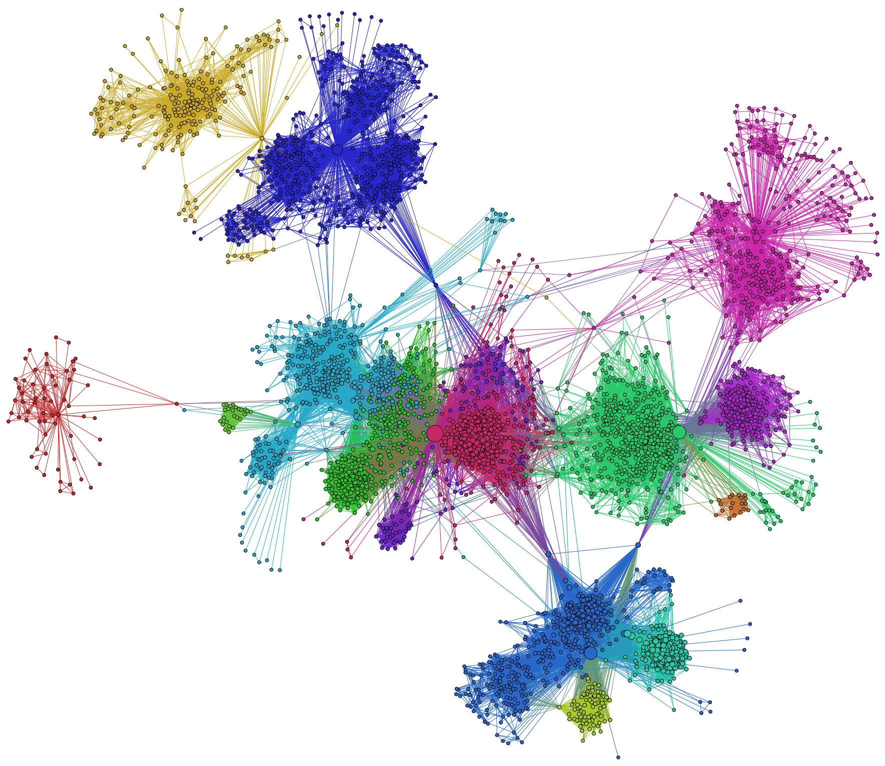


Figure 2: Snippet of the Facebook Graph created with Gephi (data from [27])

observing that the nodes of real-world networks can be often organized hierarchically, in an approximate tree-like fashion [19]. Based on this and other observations, *Hyperbolic Random Graphs* have been suggested in [33] and experimentally studied in [24]. They seem to combine all desired features of real networks in a natural model.

2.2 Load Balancing

An important prerequisite for the efficient usage of large compute networks is to balance their work efficiently. Load balancing is a ubiquitous problem which is also important for scheduling [35], routing [15], numerical computation such as solving partial differential equations [34, 36, 37], and finite element computations [22]. In the standard abstract formulation of load balancing, processors are represented by nodes of a graph, while links are represented by edges. The objective is to balance the load by allowing nodes to exchange loads with their neighbors via the incident edges. Particularly popular are decentralized, iterative algorithms where a processor knows only its current load and that of the neighboring processors and based on this, decides how many jobs should be sent (or received). Typically, the processors do not have any information about the structure of the graph, nor any "global" information about the load distribution such as the average or the total number of jobs.

A very popular load balancing strategy is diffusion, where the load sent along each edge in each step is proportional to the load difference.If we assume that the load can be divided arbitrarily often (continuous model), then the convergence rate is known to be characterized by the second largest eigenvalue of the diffusion matrix. The reason is that the iteration of the load vector is equivalent to the evolution of a (scaled) probability distribution of a simple Markov chain.

Besides diffusion, where a node is allowed to exchange load with *all* of its neighbors in each round, several theoretical studies consider the so-called matching model. In this model, one has to specify, for each round, a matching; load can then be exchanged only along the edges of the matching. This model reduces the communication in the network and moreover tends to behave in a more "monotone" fashion than its diffusive counterpart, since it avoids concurrent load exchanges which may increase the maximum load. However, the matching model requires the specification of a matching in each round, and this matching should be computable efficiently and locally. Fortunately, it was shown in [10, 31] that, if one generates the matchings by a simple greedy randomized algorithm, then the convergence of the load vector is asymptotically as fast as in the diffusion model, assuming that the load can be divided arbitrarily often.

To summarize, we can say that the continuous process is fairly well-understood, not only in the classical diffusion model, but also in the alternative matching model. This holds even for an asynchronous model [10, 30, 31], where processors communicate with each other according to a Poisson process with rate 1.

3 Related Work

Most results and developed techniques for theoretically studying load balancing only apply to regular (or almost-regular) graphs. To bridge this gap, we first observe that scale-free networks such as power-law graphs typically have a constant average degree and a polynomial maximum degree. Therefore, it is not obvious what load balancing should be aimed for. One possibility is that nodes with a larger degree also have greater computing power and thus should get a larger amount of load. On the other hand, it is also conceivable that all nodes have equal computational power and should get the same load even though their connectivity differs significantly.

We decided to use the following balancing model for networks with n nodes: at the beginning, each node i has some work load $x_i^{(0)}$. The goal is to obtain (a good approximation of) the balanced work load $\overline{x} := \sum_{i=1}^{n} x_i^{(0)}/n$ on all nodes. On heterogeneous graphs with largely varying node degrees it is also natural to consider a multiplicative quality measure: We want to find an algorithm which achieves $\max_i x_i^{(t)} = \mathcal{O}(\overline{x})$ at the earliest time t possible.

A first approach to reach this goal could be by using the diffusion model, which was first studied by Cybenko [15] and, independently, by Boillat [7]. The standard implementation is the *first order scheme* (FOS), where the load vector is multiplied with a diffusion matrix $\boldsymbol{P}$ in each step. For regular graphs with degree d, a common choice is $\boldsymbol{P}_{ij} = 1/(d+1)$ if $\{i,j\} \in E$. Already Cybenko [15] in 1989 shows for regular graphs a tight connection between the convergence rate of the diffusion algorithm and the absolute value of the second largest eigenvalue $\lambda_{\max}$ of the diffusion matrix $\boldsymbol{P}$. While FOS can be defined for non-regular graphs, its convergence is significantly affected by the loops which are induced by the degree discrepancies. We can show, that, if we define a first order diffusion scheme which converges to a uniform load distribution, the protocol will require $\Omega(\log n)$ rounds on a broad class of (only slightly) non-regular graphs. This especially includes graphs with power-law degree distributions.

Independently of the load balancing protocol, it is simple to show, that on a d-regular graph it takes at least $\Omega(\log n)$ steps to reach an asymptotically uniform load distribution. This means, that for graphs with constant average degree, like our scale-free networks, heterogeneity is needed to achieve faster load balancing. At the same time this heterogeneity forbids the use of a simple first order diffusion scheme.

4 Results

In a first paper [12] we model large scale-free networks by Chung-Lu random graphs and analyze a simple local algorithm for iterative load balancing. On n-node graphs our distributed algorithm balances the load within $\mathcal{O}((\log\log n)^2)$ steps, which we call *ultra-fast* (following the common use of the superlative "ultra" for double-logarithmic bounds [14, 18, 20]). This distributed algorithm requires only limited local knowledge on the degrees of the neighboring nodes and some estimates of global parameters. It does not need to know the exponent $\beta \in (2,3)$ of the power-law degree distribution or the weights w_i of the graph model. The algorithm assumes that the initial load is only distributed on nodes with degree $\Omega(\operatorname{polylog} n)$, which appears to be a natural assumption in typical load balancing applications. As the diameter of the graph is $\Theta(\log n)$, ultra-fast balancing is impossible if the initial load is allowed on arbitrary vertices. As standard FOS requires $\Omega(\log n)$ rounds, our algorithm uses a different, novel approach to overcome these restrictions.

The protocol we propose proceeds in waves, and each wave (roughly) proceeds as follows. First, the remaining load is balanced within a core of high-degree nodes. These nodes are known to compose a structure very similar to a dense Erdős-Rényi random graph and thereby allow very fast balancing. Afterwards, the load is disseminated into the network from high- to low-degree nodes. Each node absorbs some load and forwards the remaining to lower-degree neighbors. If there are no such neighbors, the excess load is routed back to nodes it was received from. In this way, the load moves like a wave over the graph in decreasing order of degree and then swaps back into the core. We show that each wave needs $\mathcal{O}(\log\log n)$ rounds. The algorithm keeps initiating waves until all load is absorbed, and we show that with high probability our random graphs have a structure, such that only $\mathcal{O}(\log\log n)$ waves are necessary.

There are a number of technical challenges in the analysis of this algorithm, mostly coming from the random graph model, and we have to develop new techniques to cope with them. For example, in scale-free random graphs there exist large sparse areas with many nodes of small degree that result in a high diameter. A challenge is

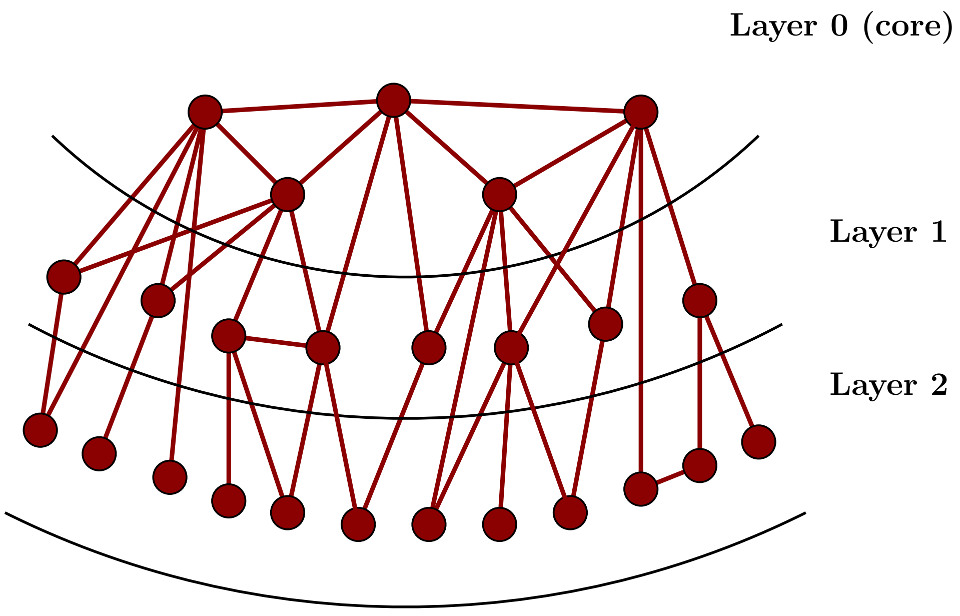


Figure 3: Layering of an example graph. Layer 0 contains all nodes of degree at least 6, Layer 1 all nodes of degrees between 3 and 5 and Layer 2 contains all nodes of degrees at most 2

to avoid that waves get lost by pushing too much load deep into these periphery areas. This is done by a partition of nodes into layers with significantly different degrees and waves that proceed only to neighboring layers. To derive the layer structure, we classify nodes based on their realized degrees, as can be seen in figure 3. However, this degree might be different from the expected degree corresponding to the weights w_i of the network model, which is unknown to the algorithm. This implies that nodes might not play their intended role in the graph and the analysis. This can lead to poor spread and the emersion of a few, large single loads during every wave. We show that several types of "wrong-degree" events causing this problem are sufficiently rare, or, more precisely, they tend to happen frequently only in parts of the graph that turn out not to be critical for the result. At the core, our analysis adjusts and applies fundamental probabilistic tools to derive concentration bounds, such as a variant of the method of bounded variances.

5 Conclusion and Outlook

Our first result shows that it is possible to do load balancing in double logarithmic time on scale-free Chung-Lu random graphs. The key to this result was designing and analyzing a novel algorithm which exploits specific structural graph properties and showing that these properties are present with high probability in the class of random graphs under consideration. One drawback of the algorithm is, that it still assumes too much knowledge of some global graph properties, like the number of nodes. Another drawback is that it only works on scale-free Chung-Lu random graphs, which do not model all properties of scale-free networks accurately. Our load balancing algorithm could therefore be improved in two ways: First, the assumptions on global knowledge could be weakened to make the algorithm applicable in realistic environments, where global properties of the network are unknown. Second, the necessary structural properties of the network could be stated explicitly. By having deterministically validatable graph properties, the algorithm might be applicable to real networks which fulfill these properties, independently of any random graph model. The same idea was successfully used by Brach, Cygan, Lacki, and Sankowski [11].

Another direction to extend our work is to look into the SATISFIABILITY problem. Ansótegui, Bonet, and Levy [4] found out that many industrial SAT instances exhibit power-law distributions in their clause lengths and variable frequencies. They also showed the efficiency of several heuristics on these instances. It would be interesting to try and use techniques from the analysis of scale-free networks to show rigorous results for the satisfiability or unsatisfiability of industrial sat instances depending on their clause and variable distributions. It might even be possible to design and analyze algorithms to solve these instances faster under certain circumstances.

References

[1] L. A. Adamic, O. Buyukkokten, and E. Adar. "A social network caught in the Web". In: *First Monday* 8.6 (2003).

[2] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. "Search in power-law networks". In: *Phys. Rev. E* 64.4 (2001), page 046135.

[3] W. Aiello, F. R. K. Chung, and L. Lu. "A random graph model for massive graphs". In: *32nd Symp. Theory of Computing (STOC)*. 2000, pages 171–180.

[4] C. Ansótegui, M. Bonet, and J. Levy. "On the Structure of Industrial SAT Instances". In: *Principles and Practice of Constraint Programming – CP 2009*. Edited by I. Gent. Volume 5732. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pages 127–141. DOI: 10.1007/978-3-642-04244-7_13.

[5] A.-L. Barabási and R. Albert. "Emergence of Scaling in Random Networks". In: *Science* 286 (1999), pages 509–512.

[6] E. A. Bender and E. R. Canfield. "The Asymptotic Number of Labeled Graphs with Given Degree Sequences". In: *J. Comb. Theory, Ser. A* 24.3 (1978), pages 296–307.

[7] J. E. Boillat. "Load balancing and Poisson equation in a graph". In: *Concurrency: Pract. Exper.* 2 (1990), pages 289–313.

[8] B. Bollobás. "A probabilistic proof of an asymptotic formula for the number of labelled regular graphs". In: *Europ. J. Combin.* 1 (1980), pages 311–316.

[9] B. Bollobás, O. Riordan, J. Spencer, and G. E. Tusnády. "The degree sequence of a scale-free random graph process". In: *Random Struct. Algorithms* 18.3 (2001), pages 279–290.

[10] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. "Randomized gossip algorithms". In: *IEEE/ACM Trans. Netw.* 14.6 (2006), pages 2508–2530.

[11] P. Brach, M. Cygan, J. Lacki, and P. Sankowski. "Algorithmic Complexity of Power Law Networks". In: *CoRR* abs/1507.02426 (2015).

[12] K. Bringmann, T. Friedrich, M. Hoefer, R. Rothenberger, and T. Sauerwald. "Ultra-Fast Load Balancing on Scale-Free Networks". In: *42nd Intl. Coll. Automata, Languages and Programming (ICALP)*. Volume 9135. Lecture Notes in Computer Science. Springer, 2015, pages 520–531. DOI: 10.1007/978-3-662-47666-6_45.

[13] A. Clauset, C. Moore, and M. E. J. Newman. "Hierarchical structure and the prediction of missing links in networks." In: *Nature* 453.7191 (2008), pages 98–101.

[14] R. Cohen and S. Havlin. "Scale-Free Networks Are Ultrasmall". In: *Phys. Rev. Lett.* 90 (5 Feb. 2003), page 058701. DOI: 10.1103/PhysRevLett.90.058701.

[15] G. Cybenko. "Load balancing for distributed memory multiprocessors". In: *J. Parallel and Distributed Comput.* 7 (1989), pages 279–301.

[16] E. Drinea, M. Enachescu, and M. Mitzenmacher. *Variations on Random Graph Models for the Web*. Technical report TR-06-01. Harvard Computer Science, 2001.

[17] M. Faloutsos, P. Faloutsos, and C. Faloutsos. "On Power-law Relationships of the Internet Topology". In: *Symp. Communications Architectures and Protocols (SIGCOMM)*. 1999, pages 251–262.

[18] N. Fountoulakis, K. Panagiotou, and T. Sauerwald. "Ultra-fast Rumor Spreading in Social Networks". In: *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '12. Kyoto, Japan: SIAM, 2012, pages 1642–1660.

[19] M. Gromov. *Metric structures for Riemannian and non-Riemannian spaces*. Birkhaeuser, 2007.

[20] R. van der Hofstad. "Random graphs and complex networks". 2011.

[21] B. A. Huberman and L. A. Adamic. "Growth dynamics of the World-Wide Web". In: *Nature* 401 (1999), page 131.

[22] K. H. Huebner, D. L. Dewhirst, D. E. Smith, and T. G. Byrom. *The Finite Element Methods for Engineers*. Wiley, 2001.

[23] A. Karbasi, J. Lengler, and A. Steger. "Normalization Phenomena in Asynchronous Networks". In: *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. 2015, pages 688–700. DOI: 10.1007/978-3-662-47666-6_55.

[24] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá. “Hyperbolic geometry of complex networks”. In: *Phys. Rev. E* 82.3 (2010), page 036106. DOI: 10.1103/PhysRevE.82.036106.

[25] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. “Stochastic models for the Web graph”. In: *41st Symp. Foundations of Computer Science (FOCS)*. 2000, pages 57–65.

[26] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. “Trawling the Web for Emerging Cyber-Communities”. In: *Computer Networks* 31.11-16 (1999), pages 1481–1493.

[27] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. June 2014. URL: http://snap.stanford.edu/data (last accessed 2015-10-01).

[28] M. E. J. Newman. “Random graphs as models of networks”. In: *Handbooks of Graphs and Networks*. Wiley-VCH, 2003, pages 35–68.

[29] R. Matei, A. Iamnitchi, and P. Foster. “Mapping the Gnutella network”. In: *IEEE Internet Computing* 6.1 (2002), pages 50–57.

[30] D. Mosk-Aoyama and D. Shah. “Computing separable functions via gossip”. In: *25th Symp. Principles of Distributed Computing (PODC)*. 2006, pages 113–122.

[31] S. Muthukrishnan and B. Ghosh. “Dynamic load balancing by random matchings”. In: *J. Comput. Syst. Sci.* 53 (1996), pages 357–370.

[32] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. “Random graphs with arbitrary degree distributions and their applications”. In: *Phys. Rev. E* 64.2 (2001), page 026118. DOI: 10.1103/PhysRevE.64.026118.

[33] F. Papadopoulos, D. V. Krioukov, M. Boguñá, and A. Vahdat. “Greedy Forwarding in Dynamic Scale-Free Networks Embedded in Hyperbolic Metric Spaces”. In: *29th IEEE Conf. Computer Communications (INFOCOM)*. 2010, pages 2973–2981.

[34] R. Subramanian and I. D. Scherson. “An analysis of diffusive load-balancing”. In: *6th Symp. Parallelism in Algorithms and Architectures (SPAA)*. 1994, pages 220–225.

[35] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica. “Load balancing in dynamic structured peer-to-peer systems”. In: *Performance Evaluation* 63.3 (2006), pages 217–240.

[36] R. D. Williams. “Performance of dynamic load balancing algorithms for unstructured mesh calculations”. In: *Concurrency: Practice and Experience* 3.5 (1991), pages 457–481.

[37] D. Zhanga, C. Jianga, and S. Li. “A fast adaptive load balancing method for parallel particle-based simulations”. In: *Simulation Modelling Practice and Theory* 17.6 (2009), pages 1032–1042.

Linespace: A Sensemaking Platform for the Blind

Thijs Roumen

Human Computer Interaction
Hasso-Plattner-Institut
thijs.roumen@hpi.uni-potsdam.de

For visually impaired users, making sense of spatial information is difficult as they have to scan and memorize contents before being able to analyze it. Even worse, any update to the displayed contents invalidates their spatial memory, which can force them to manually rescan the entire display. Making display contents persist, we argue, is thus the highest priority in designing a sensemaking system for the visually impaired. We present a tactile display system designed with this goal in mind. The foundation of our system is a very large tactile display (140 × 100 cm, 23× larger than Hyperbraille), which we achieve by using a 3D printer to print raised lines of filament. The system's software then trades in this space in order to minimize screen updates. Instead of panning and zooming, for example, our system creates additional views, leaving display contents intact and thus preserving users' spatial memory. We illustrate our system and its design principles at the example of four spatial applications. We evaluated our system with six blind users. Participants responded favorably to the system and expressed, for example, that having multiple views at the same time was helpful. They also judged the increased expressiveness of lines over the more traditional dots as useful for encoding information.

1 Introduction

For visually impaired users, making sense of spatial information is a challenge. While sighted users' ability to perceive many items in parallel allows certain similarities and structures to pop out, visually impaired users have to scan spatial information displays sequentially and slowly. Only after they have absorbed a relevant portion of the information can they start to find connections, recognize structure, and ultimately make sense of the data.

Since building up spatial memory is key, any update to displayed contents is potentially dangerous as it may invalidate users' spatial memory, in the worst case forcing them to manually rescan the entire display. Making display contents persist, we argue, is thus the highest priority in designing a sensemaking system for the visually impaired. Unfortunately, current systems designed to allow visually impaired users to browse spatial information (e.g., the Hyperbraille 120 × 60 Braille dot array) make it difficult to persist screen contents. Since they offer only a moderate amount of display space (30 × 15 cm), viewing larger data sets requires users to switch between views or to zoom and pan, all of which invalidate users' spatial memory.

In this paper, we present a tactile display system designed to minimize display updates in order to preserve users' spatial memory. We achieve this by making the

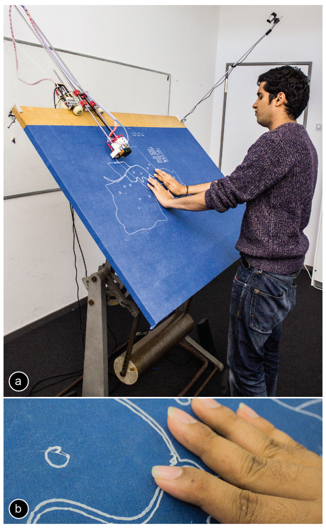


Figure 1: Linespace is a sensemaking platform for the blind. Its custom display hardware offers 140 × 100 cm display space and it draws lines as its main primitive. Here linespace runs the homefinder application that enables users to browse maps in search for a home.

display very large (140 × 100 cm) and by designing its software system to leverage this display space in order to persist displayed contents.

2 Design Principles

Linespace is an interactive system that consists of hardware and software and that allows visually impaired users to interact with spatial contents.

Linespace's hardware provides it with a large amount of display space and the ability to render lines, a primitive particularly well suited for the content types involved in spatial sensemaking tasks, such as graphs, diagrams, maps, and drawings. Based on this hardware, our objective in designing linespace's software system was to allow users to build up and maintain spatial memory.

Primary design rule: leave displayed contents intact In order to not destroy spatial memory linespace's primary design rule is: "leave printed display contents intact". We express this using four sub rules:

- p1. No panning and scrolling. Instead, extend contents.
- p2. No zooming. Instead, add overviews or detail views.
- p3. No animation. Instead, use static animation.
- p4. No pop-ups and dialogs. Instead, use auditory output.

Secondary design rule: spend display space carefully Within all solutions that satisfy these rules, our secondary design objective is to spent display space carefully, as it is the display space that allows the system to achieve its primary goal.

s1. No unnecessary scale. Render as small as readable.

s2. No chrome. Instead, structure contents with whitespace

s3. No display windows. Traditional windows are a way of reserving space oftentimes before it is really needed. While linespace allows apps to run in parallel, applications are supposed to start at display size zero and grow their space use over time as needed. Apps have whatever shape their content has, which will typically not be a rectangle.

s4. No displaying of text and no displaying of elaborate icons. Instead, use a small number of simple tactile icons that play back auditory output when touched.

Tertiary design rule: allow for speedy operation Within all solutions that satisfy these rules, our tertiary design rule is to allow for speedy operation, in particular by handling the limitations of linespace's print mechanism.

t1. No printing at app launch: all applications start with a blank display, allowing apps to start instantaneously.

t2. No printing at app switching: touching content of a different app moves the focus to that app instantaneously. Remove or relocate an application only when another application grows into its display space.

t3. Let users interact while system is printing in regions distant enough from the print arm.

t4. Let system print while user is interacting; prerender contents likely to become necessary soon.

t5. During printing sonify what is being printed: this allows for immediate feedback. Given that the speaker moves with the print head, users also build up spatial memory of what is printed where.

3 Demo Applications

We now go over our 4 demo applications and use them to explain how they implement our 3 sets of design rules.

3.1 Minesweeper

Minesweeper is an adapted version of the minesweeper number puzzle that used to ship with the Windows operating system. Players' objective is to clear a board containing hidden "mines", with help from clues about the number of neighboring mines in each field. While not a sensemaking application, minesweeper does involve a good amount of spatial reasoning, so we included it as our first example.

To launch minesweeper, users press the foot switch and say "launch minesweeper". The app launches with a blank screen (p1) and welcomes users with: "Minesweeper. Your entire screen now is a mine field. Touch anywhere and say "reveal" to see whether there is a bomb. Say "usage" to learn more." (p4).

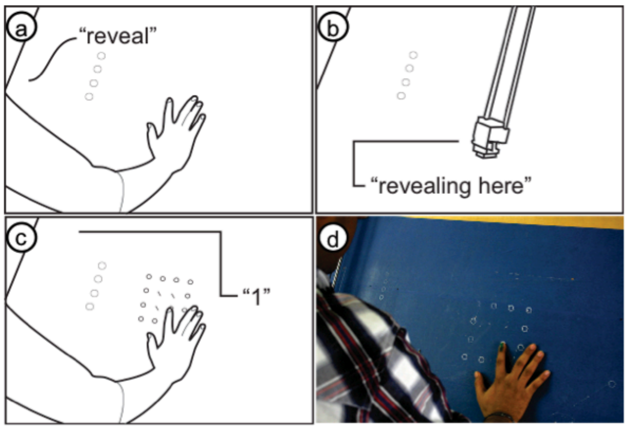


Figure 2: The Minesweeper app (a) reveals a cell, (b) here a free cell. (c) Users scan a local neighborhood of cells with their fingers to conclude the location of mines. (d) The prototype

As shown in Figure 2a users tap onto the board and say "reveal". Minesweeper responds by announcing the item that is located there (p5), i.e., either "free", "mine", or a number denoting the mines surrounding that cell. At the same time linespace persists this information by plotting an icon at the location. To maximize content density, minesweeper distinguishes only between a "free" cell (a slanted line icon) and cells that have an adjacent mine (a circle icon); instead, the actual number is read out loud every time the user touches the cell (p4). (b) In the shown case, the cell was "free" which causes the app to also reveal surrounding cells. Note how the app separates cells using whitespace rather than gridlines (p2).

Users spatial task is to locate mines without revealing them. Users scan an area of interest with their fingers, listen to the number and build up a mental model of the constraints. When they conclude where a mine must be located they touch that location and say "mine" (p4). The app responds "marking as mine" and draws a mine icon (a triangle). As the user continues to reveal more area of the board, the minesweeper application grows which extends the display space it occupies (p1). To explore the potential of the system, our version of minesweeper is intentionally designed to fill the entire display area by default (>9000 cells). If users solve the entire puzzle, the app plays a congratulatory message and terminates.

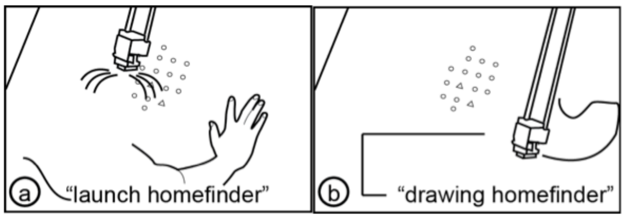


Figure 3: When an app terminates, linespace's app manager starts to clean up its screen space, until (a) the user requests a new application, which (b) causes linespace to interrupt its clean up immediately.

After a brief pause, the app manager starts to free up the app's display space by scraping off all its on-screen objects (Figure 3a). Users do not have to wait though. They can switch to a different app or (b) launch a new app (e.g., re-launch the game) in a fresh screen region any time. The system accommodates this by interrupting its clean up, allowing it to respond instantaneously (p3).

3.2 Homefinder

Homefinder is a simple app that allows users to search for real estate, such as a four bedroom in a city.

When users launch homefinder, the app launches with a blank screen (p1) and welcomes users with: "Welcome to homefinder. What city or neighborhood to plot where?" (p4). Users point to an empty screen region and name their city and neighborhood. Homefinder responds by saying, e.g. "63 homes" and plotting a few characteristic landmarks, such as an outline of the city (Figure 4a). The user says "filter four rooms or more" to reduce the set of houses. The system responds, e.g. with "12 homes found". (b) When users say "draw", homefinder plots the homes onto the map (Figure 4c), each one as a simple icon (a circle).

To learn more about a home, users scan the map with their fingers, pause over a circle icon and say "reveal". Homefinder responds with a brief verbal description of the place, in prioritized order starting with price, number of rooms, etc.

(c) When the query does not find enough homes in the neighborhood, users can point at a blank space and say "extend", causing homefinder to sketch an additional neighborhood and populate it with homes, in this case responding "7 additional homes found". Users can also adjust the filters using speech input, e.g. also allowing three rooms, which causes homefinder to fill in additional homes.

(d) To provide users with a sense of what has changed, the additional homes are plotted with a modified icon (a dash inside the circle icon). Similarly, users can reduce the number of homes with the filter, which (e) causes homefinder to scrape off the icons of the surplus homes and replace them with an icon indicating the absence of an item (a dash).

(f) To learn more about the relationship between price and number of places, users can also query a slider by saying "place price slider here", which causes homefinder to draw a slider at the specified location. Users can now slide their finger up and down the slider while homefinder is continuously announcing the numbers: "300 thousand—16 homes… 350 thousand—12 homes".

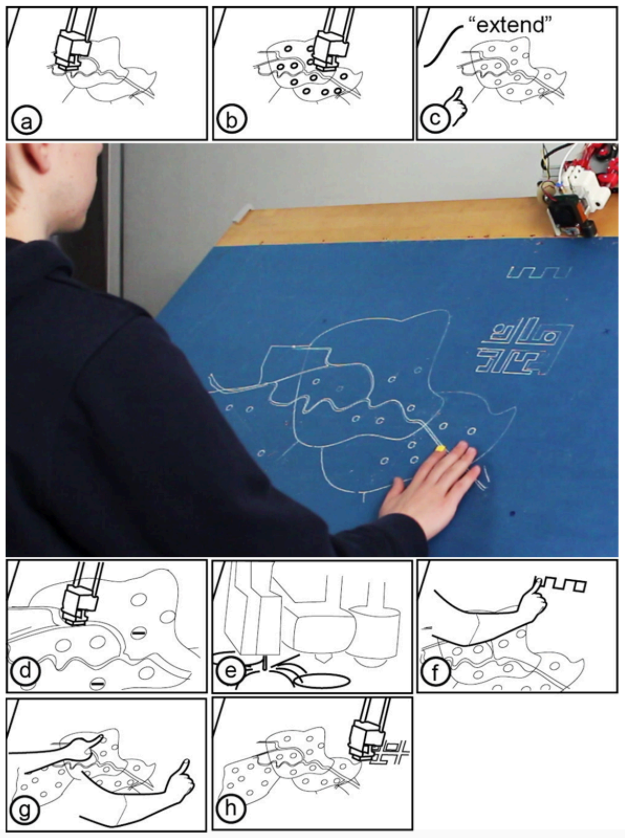


Figure 4: The homefinder application

Note how homefinder always provides an auditory summary first and only then refreshes the screen. This is very different from similar applications for sighted users, that tend to update the screen whenever possible, e.g., continuously while users drag a slider. Such tight coupling is only of limited use for visually impaired users, as users cannot take in the spatial display at a useful rate (independent of how fast or slow the system can render the changes).

(g) Finally, when users have found a home that sounds promising and would like to get a better understanding of its surroundings, they can display additional detail. For this, users point at the place with one hand and use their other hand to point at a patch of blank space. When they say "zoom here" linespace responds by (h) plotting a zoomed in map of the area (p4) in the blank space.

3.3 Drawing application

Since our first two applications are focused on allowing users to explore, we added a drawing app as a means for users to create. As an example drawing, we explain how to make a bicycle. To draw the front wheel, users place their fingers three inches apart and say "circle, draw", causing the drawing application to say "drawing circle" and drawing a three-inch circle in between. Users create the rear wheel by pointing at the front wheel and a location eight inches further right, then say "clone, draw".

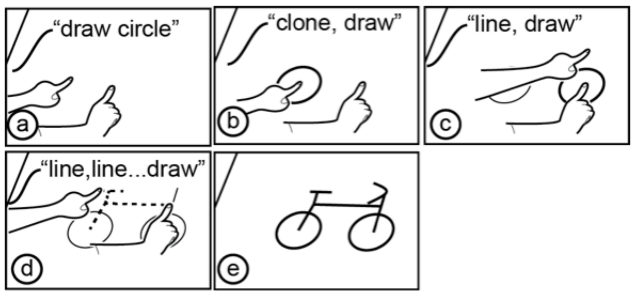


Figure 5: Drawing application

To draw the fork, users start by pointing to the center of the front wheel and where they want the upper end to go. After saying "line, draw", the app draws the line.

To allow for efficient drawing, users can create the frame by using the line tool in "polyline style", i.e. by specifying all five lines before updating the display. This also allows them to use their fingers as bookmarks as they can keep their fingers on the display. To save a line for later printing, users say "memorize line", which causes the system to respond with "line memorized". At the end, when users say "draw", they get the polyline.

Users can also add freehand drawings, such as the curved handles of the bike, by using the hand-held extruder pen.

3.4 App manager

The handling of individual applications and their canvases and sub-canvases is done by a program called app manager. App manager also allows users to launch and kill other apps, configure them, and switch between apps. App manager loads automatically whenever linespace starts up.

App manager launches with a blank screen display (p1) and does not occupy any screen space itself (s2, s3). Instead app manager runs in the background and listens in on speech input (p4), so that all interactions with app manager itself are based on speech.

Users, for example, launch an app by saying "start <app name>". Linespace responds by loading the respective app and confirms "<app name> loaded" and hands control over to the app, which follows up with a welcome message. The minesweeper app, for example, says "your entire screen now is a mine field. Touch anywhere and say "reveal" to see whether there is a bomb. Or say "usage" to learn more."

While app manager itself does not occupy display space, its apps do. Users consequently interact with the apps by pointing at them, then adding a verbal command, such as "kill" which causes app manager to terminate the app and remove its screen contents, or "relocate", which deletes display contents and redraws it to a new location pointed

4 Contribution, Benefits, Limitations

Our main contribution is a sensemaking platform for the blind. The key principle driving its design is to preserve users' spatial memory by leaving displayed contents intact. To allow for this strategy, we provide linespace with a very large display, i.e., 23× more display space than a Hyperbraille array. We achieve this by basing our mechanical design on a 3D printer that draws screen contents. Its ability to draw lines also makes our system particularly suited for the content types involved in spatial sensemaking tasks, such as graphs, diagrams, maps, and drawings. We also contribute a software framework that allows developers to quickly build applications for linespace. Finally, the approach of using a 3D printer allows us to fabricate the device inexpensively (400 dollar, about 1/200th of a Hyperbraille). The printing material incurs (insignificant) running costs. The main limitation of linespace is that plotting contents takes time; also the turn taking between user and device requires users to wait. We address these challenges in part using the design principles mentioned earlier in this paper.

5 Related Work

Linespace builds on research in accessibility and personal fabrication as a tool to fabricate tactile representations for the blind.

Blind technologies for interacting with spatial content While braille displays are normally used to sequentially display braille text, HyperBraille [10] is a large Braille display that can be used to explore spatial content. To make optimal use of the space, Prescher et al. [10] present a Braille-based windowing system.

Since scaling Braille arrays involves proportional cost, researchers have proposed to use alternative haptic cues, such as vibration, as a means to communicate spatial information to visually impaired users. For instance, TGuide [8] uses 8 vibrating elements to output directional information for navigation purposes. Beside vibration, researchers also suggested the use of force-feedback devices: Crossan et al. [4] designed a system that teaches shapes and trajectories using a force feedback arm. Similarly, Plimmer et al. [9] trained blind users to learn writing using a force feedback arm. Finally, researchers also suggested to add small braille displays onto force feedback arms and to update the display in accordance to its current location (PantoBraille [11]). To display 3D geometry with fine texture features, Colwell et al. [3] introduce a haptic device that provides feedback to the user by monitoring the position of hand and altering the force accordingly.

Finally, researchers have also examined how to combine non-computerized means for displaying spatial content (e.g. swell paper) with a touch screen. By overlaying swell paper onto the screen, users of TDRAW [7] can simultaneously draw and annotate their drawings using voice over. Users create drawings using a pen featuring a hot tip. The hot tip causes the swell paper to buckle, allowing users to feel strokes produced earlier. However, while the device provides users with a means to create tactile content, the device has no means of creating tactile output itself.

Personal Fabrication for Visually Impaired Originally, personal fabrication tools were developed as a means for rapid prototyping. However, the output created by personal fabrication machines, such as 3D printers and laser cutters is inherently tangible, giving it relevance to the visually impaired community. Physical visualizations of data that display multi-dimensional data [12], for example, result in a type of display that is accessible to blind users.

Recently, 3D printers have been proposed as a means to generate tactile output for blind users: VizTouch [1] for instance, generates 3D printed graphs and data plots by extracting contours from a 2D input image. ABC and 3D [2]. 3D print geometric objects that allow visually impaired students to tactilely interact with their math curriculum. Kane et al. [5] 3D print tactile representations of debugging output to make programming more accessible to the blind.

Tactile Picture Books [6] are books for blind children that contain 3D printed objects instead of 2D images.

6 Software Implementation

Linespace's software is written in Python 3. It uses the PrintrRun library for controlling the printer and several Inkscape extensions for simplifying path geometry.

We use an event driven architecture for sending and receiving events between different components of the system, such as when users select a printed part via touch or when users query information via a voice command. Events are sent to the app manager component and then propagated to the respective apps and their widgets.

For organizing the widgets in apps, we provide various layout containers, such as a stack container and a docking container. These enable system developers to specify how the widgets are distributed in an app.

Converting vector files to 3D printing g-code Linespace automatically imports and converts .svg files that specify the tactile paths for an application into a set of lines. An arc, for instance, is converted into lines using the cubic subdivide method of the Inkscape API. Linespace then stores these lines as internal geometric objects to enable geometry operations such as resizing the content based on the available space on the print bed.

When the tactile paths are queried for printing, the corresponding internal geometric objects are converted to 3D printer instructions in GCODE. We take the beginning and end point of each line for the print head travel commands (e.g. G1 X0 Y0, then G1 X10 Y0 draws a horizontal line). We use three different travel speeds: moving (F3600), printing (F1200), and erasing (F2400). Finally, linespace also computes how much material should be extruded while moving along a path. For this, linespace uses a fixed extrusion amount per unit, which is defined by the number of stepper steps to push the filament through the nozzle while moving along a certain distance (e.g. printing a length of 1cm requires the extruder stepper motor to make 5 steps, GCODE: E5).

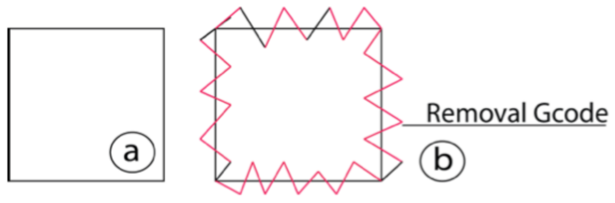


Figure 6: removal of g-code generation

Generating GCODE for removing outdated content To effectively remove lines, we move the scratching pin along the zigzag pattern shown in Figure 6b. To generate the GCODE for the print head movement, we first segment a shape into lines, then offset the start point of each line either towards the normal of the line or the reverse normal. Beside the horizontal pin movement, linespace also generates the GCODE for moving the pin up and down via the solenoid. We drive the solenoid directly from the PrintrBot microcontroller. For this, we connected the solenoid to the pin that is normally used to control the heated print bed. To activate and deactivate the solenoid, we set the voltage of the pin accordingly (M42 S255 P14 vs. M42 S0 P14).

Tracking user input via the camera To translate the camera coordinates to print bed coordinates, we perform a homography on all camera images. After this, we threshold the HSV values from the camera image to track the input color markers on users' fingers.

Audio output and speech input For both speech input recognition and speech output, linespace uses the Microsoft Speech Platform SDK 11.

7 Conclusion

We presented linespace, an interactive system that allows visually impaired users to interact with spatial contents. By basing our design on a 3D printer, we were able to extend the display area to 140 × 100 cm. The increased interaction space allowed us to eliminate the necessity for many types of display updates, such as panning and zooming, thus allowing blind users to always stay within their spatial reference system.

As future work, we plan to examine how linespace can be extended to help blind users with more complex sense making tasks. We are also planning on creating a mobile version.

References

[1] C. Brown and A. Hurst. "VizTouch: Automatically Generated Tactile Visualizations of Coordinate Spaces". In: *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. TEI '12. Kingston, Ontario, Canada: ACM, 2012, pages 131–138. DOI: 10.1145/2148131.2148160.

[2] E. Buehler, S. K. Kane, and A. Hurst. "ABC and 3D: Opportunities and Obstacles to 3D Printing in Special Education Environments". In: *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*. ASSETS '14. Rochester, New York, USA: ACM, 2014, pages 107–114. DOI: 10.1145/2661334.2661365.

[3] C. Colwell, H. Petrie, D. Kornbrot, A. Hardwick, and S. Furner. "Haptic Virtual Reality for Blind Computer Users". In: *Proceedings of the Third International ACM Conference on Assistive Technologies*. Assets '98. Marina del Rey, California, USA: ACM, 1998, pages 92–99. DOI: 10.1145/274497.274515.

[4] A. Crossan and S. Brewster. "Multimodal Trajectory Playback for Teaching Shape Information and Trajectories to Visually Impaired Computer Users". In: *ACM Trans. Access. Comput.* 1.2 (Oct. 2008), 12:1–12:34. DOI: 10.1145/1408760.1408766.

[5] S. K. Kane and J. P. Bigham. "Tracking @Stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and Twitter". In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, 2014, pages 247–252. DOI: 10.1145/2538862.2538975.

[6] J. Kim and T. Yeh. "Toward 3D-Printed Movable Tactile Pictures for Children with Visual Impairments". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: ACM, 2015, pages 2815–2824. DOI: 10.1145/2702123.2702144.

[7] M. Kurze. “TDraw: A Computer-based Tactile Drawing Tool for Blind People”. In: *Proceedings of the Second Annual ACM Conference on Assistive Technologies*. Assets ’96. Vancouver, British Columbia, Canada: ACM, 1996, pages 131–138. DOI: 10.1145/228347.228368.

[8] M. Kurze. “TGuide: A Guidance System for Tactile Image Exploration”. In: *Proceedings of the Third International ACM Conference on Assistive Technologies*. Assets ’98. Marina del Rey, California, USA: ACM, 1998, pages 85–91. DOI: 10.1145/274497.274514.

[9] B. Plimmer, A. Crossan, S. A. Brewster, and R. Blagojevic. “Multimodal Collaborative Handwriting Training for Visually-impaired People”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’08. Florence, Italy: ACM, 2008, pages 393–402. DOI: 10.1145/1357054.1357119.

[10] D. Prescher, G. Weber, and M. Spindler. “A Tactile Windowing System for Blind Users”. In: *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS ’10. Orlando, Florida, USA: ACM, 2010, pages 91–98. DOI: 10.1145/1878803.1878821.

[11] C. Ramstein. “Combining Haptic and Braille Technologies: Design Issues and Pilot Study”. In: *Proceedings of the Second Annual ACM Conference on Assistive Technologies*. Assets ’96. Vancouver, British Columbia, Canada: ACM, 1996, pages 37–44. DOI: 10.1145/228347.228355.

[12] S. Swaminathan, C. Shi, Y. Jansen, P. Dragicevic, L. A. Oehlberg, and J.-D. Fekete. “Supporting the Design and Fabrication of Physical Visualizations”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, Ontario, Canada: ACM, 2014, pages 3845–3854. DOI: 10.1145/2556288.2557310.

Cluster-based Sorted Neighborhoods for Deduplication

Ahmad Samiei

Information Systems Group
Hasso-Plattner-Institut
Ahmad.Samiei@hpi.uni-potsdam.de

Deduplication intends to detect multiple and different syntactical representations of the real-world entities in a dataset. The naïve way of deduplication entails a quadratic number of pair-comparisons using an expensive similarity measure to identify the duplicates. This number of comparisons even for an average size dataset might take hours to perform. As today's databases are growing very fast, different selection methods, such as Sorted Neighborhood, blocking, canopy clustering and their variations, address this problem by shrinking the comparison space. Volume of data and velocity of change, make us to find faster methods to tackle this problem. The current report describes my research on incremental deduplication during last year in the context of the HPI Research School on Service-oriented Systems Engineering and also outlines my future research directions.

1 Duplicate Detection

Databases play an important role in IT-based companies nowadays, and many industries, business' and government organizations rely on accuracy of data to perform their operations. Unfortunately, the data are not always clean: for instance, real-world entities have multiple different syntactical representations, which could be due to erroneous data entry, data evolution, data integration, etc. This in turn introduces so-called duplicate records into the databases. Deduplication intends to identify and ultimately eliminate these multiple representations of entities in a database. Entities, such as customers, publications, bibliographical citations, etc., are examples with high interest for deduplication. Another and recent application of deduplication with growing interest is to identify and compare duplicate products from different online shops.

Deduplication has been the topic of research for many years under different names such as purge and merge [10], record linkage [7, 14, 15], deduplication [17], etc. The naïve way of duplicate detection imposes quadratic number of record pair comparisons which is time expensive task even for an average size database (e.g., in a database of size 10^6 naïve way of duplicate detection has to execute 10^{12} pair comparisons). As a result, many different methods, such as the Sorted Neighborhood method and its variants [6, 9, 10], blocking [7, 18], canopy clustering [4, 11], has been proposed to tackle this problem by eliminating candidate pairs which are less likely to generate duplicates from this quadratic search space.

Wide spreading of internet and online devices have increased the speed of generating the digital data in an unprecedented rate. Majority of this data are produced by sensors or inserted into database from different sources and are inherently dirty.

However already existing deduplication methods perform good, these sheer amount of dirty and noisy data require ever faster methods of data cleansing and deduplications. In this report, we propose a novel deduplication approach based on traditional Sorted Neighborhood method. It uses a light-weight internal attribute similarity measure to cluster attribute values and leveraging the clustered attributes to reduce the number of candidate pair comparisons by disqualifying less likely candidate pairs.

The main contributions of this paper:

- We introduce a new variant for the Sorted Neighborhood method based on attribute value clustering.
- The proposed approach ingests input data incrementally that makes it favorable for many new applications with high velocity of change in the dataset.
- We evaluate performance, quality and runtime, of our approach over three different datasets and compare the result with existing methods.

2 Related Work

Duplicate detection algorithms are broadly categorized in three different categories, namely blocking, canopy clustering, and sorted neighborhood methods. The blocking is a traditional way of record deduplication. It basically partitions the dataset to some non-overlapping partitions, and does the record comparisons just for the records inside each partition. As a result, it reduces the number of the record comparisons.

The canopy clustering is a rather new category of the record deduplication algorithms. For instance, Monge and Elkan transform the problem of finding matching records to the problem of finding connected components of an undirected graph based on the assumption that record matching is a transitive operation [12]. They efficiently cluster the connected subgraphs using a union-find structure, and finally use only a representative of each subgraph to compare in future steps which results in reduction of number of the comparisons.

Mccallum et al. also utilize canopies to expedite the deduplication process [11]. In the first step, using a cheap comparison metric, they partition dataset to some overlapping partitions. Then they use a more expensive similarity metric for record comparison. Cohen et al. also use a Tf-idf similarity metric as a canopy similarity metric [4], Gravano et al. use the string length and number of q-grams of the two string as a canopy similarity metric [8].

The Sorted Neighborhood Method (SNM) was first presented by Hernandez et al. [9, 10]. Its basic idea is to sort the records according to a sorting key and move a window with a fixed length over the sorted list, while comparing all pairs of records within the windows. The main deficiency of this method is that it uses a fixed window size. That means if the window size is chosen too small it might not cover all sufficiently similar records in one window. Therefore it does not generate enough candidate pairs to cover all of the duplicates. On the other hand, choosing a too large window size generates many irrelevant candidate pairs and imposes a high runtime. Another drawback of this method is sensitivity to typographical errors in the sorting keys. To this end, Hernandez et al. have proposed the multi-pass approach, which simply executes SNM multiple times with different sorting keys and finally applies the transitive closure over all discovered duplicates.

Different variants of the SNM have been proposed to deal with the mentioned deficiencies. Christen introduces an extension to the traditional SNM in order to address the problem of fixed window size by the help of an inverted index [3]. In the proposed method he uses a list of unique values of sorting keys and stores the identifiers of the records with the same sorting key values in one row of the inverted index. While this approach is successful in resolving the problem of a fixed window size, it suffers from a new deficiency. The inverted index generates large blocks, which dominate the dataset and thus yield many irrelevant candidate pairs.

Another alternative method that attempts to deal with the problem of fixed window sizes is that of adaptive windows. Such methods enlarge or reduce the window size based on some dynamic criterion, such as the similarity of the records. For instance, Draisbach et al. propose a novel adaptive approach for the Sorted Neighborhood method called Duplicate Count Strategy. The idea behind their method is if any record in the window has a duplicate within the window, it is more likely to have further duplicates within adjacent windows. The algorithm starts with a fixed initial windows size. If the average number of duplicates in the window exceeds a specific threshold, the algorithm extends the current window size by $w - 1$. It was shown that that method is at least as efficient as SNM and typically reduces the number of comparisons significantly.

Progressive duplicate detection is another kind of extension for SNM. This category of algorithms are useful especially when execution time is limited. Progressive algorithms try to reduce the average time of discovering a duplicate by identifying most of the duplicate pairs in early stages [16].

Our method is a new extension to the Sorted Neighborhood method. In our approach before using a full similarity measure for a pair record comparison, we use a similarity measure for clustering a subset of attributes and utilizes the clustering attributes to cut down the number of potential candidate pairs. Apart from that our algorithm ingests the input records in an incremental manner, which makes it suitable for many use cases with incremental nature.

3 Attribute cluster based Incremental SNM

The naïve way of duplicate detection entails a quadratic number of pair comparisons. The main goal of all traditional de-duplication methods is to reduce number of redundant and unproductive pair comparisons. While these methods are successful in achieving their goal to some extent, there remains room for improvement. In this report, we introduce a novel de-duplication algorithm based on traditional sorted neighborhood method. Our approach named ciSNM is an attribute cluster-based extension of standard SNM. In contrast to the conventional SNM, it uses a window of size $2 \cdot w$ and is able to ingest records incrementally. It leverages a light-weight similarity measure for attribute similarity calculation and with the help of attribute clustering reduces the number of irrelevant candidate pair comparisons.

We consider the similarity measure for pair comparison as a black box, which returns a similarity value in the interval [0, 1] for every input record pair. Of course the similarity threshold of the algorithm to separate a duplicate from non-duplicate is influenced by the internal similarity measure in the black box; this aspect is not the topic of this paper – we assume the same settings for all approaches used in evaluation section.

Similar to the traditional SNM, ciSNM consists of three phases. The first phase generates sorting keys. As in SNM, the general idea behind sorting records is to bring similar records closer to each other so that they appear within a small window. Using attributes (or attribute combinations) that have many distinct values (or value combinations) perform better than those with few distinct values [13]. The second phase of the traditional SNM sorts the records based on the already generated sorting key in the first phase.

In the batch approaches, such as SNM, all of the data is available prior to executing de-duplication. As a result, sorting can be done in one run with the time complexity of $O(n \log n)$. In contrast to SNM, ciSNM does not assume to have access to complete dataset at once. Records arrive incrementally and must be placed in correct sort-order among those records that have arrived earlier. Thus, we need to use incremental sorting methods, which do not work well on the typical list-type data structures. In order to achieve the same time complexity as SNM during the sorting phase, ciSNM uses a *B-tree* data structure. Insertion into the (sorted) *B-tree* has a time complexity of $O(\log n)$. Considering n records in the dataset, the upper bound of time complexity of sorting is again $O(n \log n)$, which satisfies our need.

The third phase of ciSNM generates candidate pairs for similarity calculation. Considering the size of the sliding window w, the traditional SNM generates $n \cdot (w-1)$ candidate pairs. Taking into account the assumption of receiving the input records as a stream of data, and inserting records to the de-duplicated dataset incrementally, our ciSNM algorithm has to compare each newly arrived record with $2 \cdot (w-1)$ records (i.e., within two windows of size w). That is because we cannot know in advance, in which direction the record set will grow. In an extreme case, all further records are sorted to only one side of the record thus the more generous window. The first window contains records with a greater value in their sorting key, the second window those with smaller value. However, this number of candidate pairs and comparisons is almost twice as the number of the comparison in the SNM and is undesirable[1]. It guarantees at least the same *recall* of the result as SNM. To address this issue, ciSNM leverages a value-clustering scheme using an additional *light-weight* attribute similarity measure. Its third phase comprises of three following sub-steps

[1]In fact the number of comparison is less than $2 \cdot (w-1)$ if the new record is inserted to either ends of the sorted list or when the list size is still shorter than $2 \cdot (w-1)$.

that intend to further reduce the number of candidate pairs for which the expensive similarity measure is invoked:

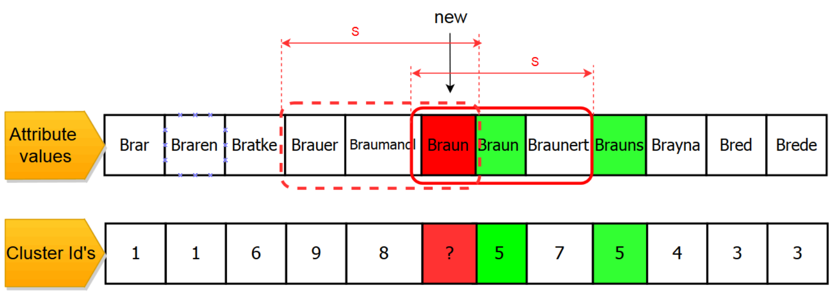


Figure 1: Attribute clustering

1. Attribute clustering
2. Candidate selection
3. Similarity calculation

The attribute clustering step, clusters attribute values of a preselected subset of attributes based on their similarities by utilizing a light-weight similarity measure. The clustering is done while inserting every newly arrived record into the proper order-position in the sorted list. Algorithm determines the most similar value to the attribute values of the inserted record. It compares each attribute value of the new record with the values of the corresponding attribute of the adjacent records in the ordered list. Two windows of size s, as depicted in the Figure 1, are considered for this purpose.

The first attribute of each sorting key is chosen for clustering. Due to this selection similar values are brought close to one another, therefore suitable cluster for the new attribute value can be found even without necessity of comparing it with all of the attribute values in the specified windows. In case of detecting an exact match, the algorithm assigns the new attribute value to the same cluster and stops, otherwise it continues to find an attribute value with the highest similarity value and at the same time greater value than the threshold. Finally, if it was not successful to find a suitable cluster it creates a new cluster and assigns the new attribute value to that.

The ciSNM algorithm exploits these attribute cluster results to filter out candidate pairs which are less likely to end up to a duplicate in its candidate selection step. This is done by applying a simple threshold on the number of matches among the cluster Id's of the two records. Applying this condition eliminates many irrelevant candidate pairs, and keeps those with higher probability of yielding a duplicate pair for the last step of the algorithm, similarity calculation.

4 Evaluation

We evaluate the performance of the proposed approach from two aspects, quality and runtime, and compare it with the traditional SNM and one more recent algorithm ASNM presented in [6]. we test the behavior of the new algorithm choosing datasets from different sizes and characteristics.

4.1 Datasets and experimental setup

To evaluate our algorithm, we use three datasets Cora, Febrl, and Person. The Cora dataset, citations of research papers, contains 1,879 records with a gold standard that contains 64,578 duplicate pairs. It has been widely used by research papers to evaluate duplicate detection algorithms [1, 5, 6]. The Febrl dataset is a synthetic dataset that contains personal data and is generated by the Febrl data generator [2]. It contains about 300k records and 100k duplicate pairs. The third dataset is Person, which contains over one million records. Every record consists of 12 attributes of personal data and address and is polluted with noise and duplicates artificially. This dataset is used by an industry partner as a duplicate detection benchmark. A summary of the datasets is in Table 1, showing that they have quite some differences in number of the cluster with size greater than two and maximum cluster size.

To perform our evaluation we use a PC with an i5 processor with 3.2 GHz, 8 GB of RAM, and running Windows 7. In addition to our algorithm ciSNM, we implemented standard SNM and adaptive window Sorted Neighborhood introduced by Draisbach et al. called ASNM [6], one of the most recent algorithms in order to compare our results. For the quality measure we used precision, recall and F-measure (the harmonic mean of precision and recall).

Table 1: Evaluation Datasets

Dataset	#Records	#Duplicate Pair	Clusters≥ 2	Records in Clusters≥ 2	Maximum Cluster Size
Cora	1,879	64,578	118	1,815	238
Febrl	300,009	101,153	7,301	37,301	10
Person	1,039,779	89,784	44,892	89,784	2

4.2 Experimental results — quality

We implemented our own similarity measure for the Febrl and Person datasets and used the similarity measure implemented by Draisbach et al. [6] for the Cora dataset. Meanwhile, we used the same similarity function, similarity threshold and three sorting keys for executing different algorithms over each dataset. We ran a multi pass version of all algorithms. For internal similarity measures of ciSNM, we chose a similarity threshold that produces at least the same accuracy as the better algorithm between other two algorithms, SNM and ASNM on the same dataset. However, choosing a very low value for this threshold yields higher accuracy imposes higher runtime. This contradicts our goal, reducing the number of the pointless comparisons.

The Figures 2, 3, and 4 depict the three evaluation measures for the three different algorithms on the three datasets, Cora, Febrl, and Person, respectively. In our experiments we varied the window size from the minimum 2 to the value after which F-measure no longer changes significantly.

As Figure 2 illustrates for the Cora dataset, precision starts from 0.95 for ciSNM, 0.94 for ASNM, and 0.92 for SNM, and in all cases slightly decreases with increasing size of the sliding window; more comparisons give more opportunity for incorrect decisions. Recall starts from 0.96 for SNM, 0.95 for ASNM and ciSNM. It reaches 0.99 for SNM and ASNM and 0.98 for ciSNM as we increase the window size to 30. All of the algorithms achieve very close F-measure values for all different window sizes, and ciSNM outperforms other two algorithm with a small margin between 1 % to 2 %. This improvement is due to selecting pairs from a bigger window size.

In case of the Febrl dataset, Figure 3 depicts a result very similar to the Cora dataset. Precision starts from 0.99 for SNM and ASNM, and 1 for ciSNM for the smallest window size. Precision remains almost constant for all different window sizes in ciSNM, it degrades only 1 %, but it shows a 4 % decrease in the other two algorithms. The SNM starts with the lowest value in recall, 0.74 for the smallest window size. ASNM and ciSNM both start with higher value in recall and these effects are reflected in their F-measure. Both of the algorithms, ASNM and ciSNM achieve higher F-measure than SNM.

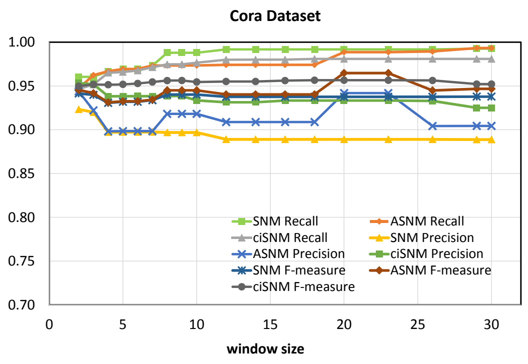


Figure 2: Quality on Cora dataset

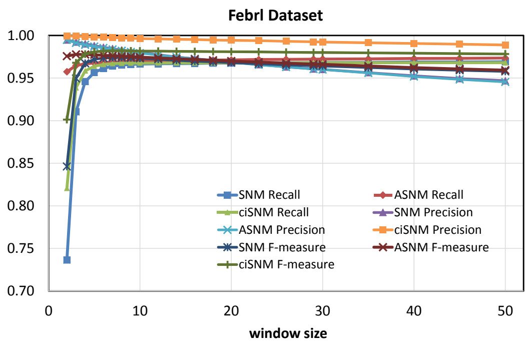


Figure 3: Quality on Febrl dataset

For the Persons dataset, which is our biggest dataset, we can make similar observations in Figure 4. All of the algorithms produce the same and almost steady precision. They start with 0.99 for the smallest window size and show one percent decrease for the largest window. Similar to the other two datasets, here SNM and ASNM start with the lower value in recall and F-measure and ciSNM achieves greater value

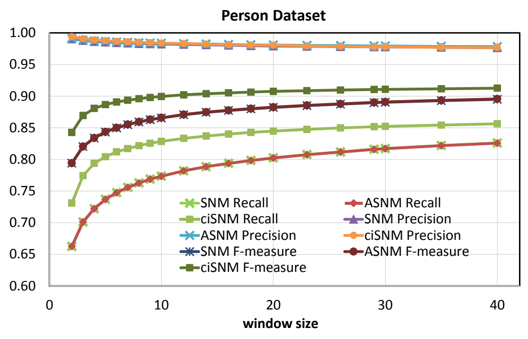


Figure 4: Quality on Person dataset

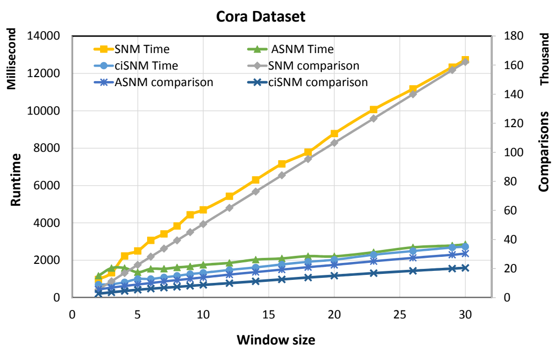


Figure 5: Runtime on Cora dataset

in F-measure for smaller window sizes and in general outperforms the other two algorithms.

4.3 Experimental results—runtime

Figures 5, 6, and 7 depict runtimes and the number of pair-comparisons of the algorithms for different sliding window sizes. As can be seen in the Figure 5, the ciSNM algorithm generates the fewest number of pair comparisons among all algorithms for the Cora dataset for each window size. It generates fewer pair comparisons (between 30 % to 50 % fewer than ASNM), and its average runtime improvement for the Cora dataset is 22 % in comparison with ASNM. Its runtime improvement varies from 4 % to 54 %. In contrast with the other two datasets this gap is bigger for the small window sizes which is due to heavy usage of transitive closure operator in ASNM algorithm and by increasing the size of the sliding window this gap becomes narrower as more duplicate discoveries happen in normal pair comparison. This results in less usage of transitive closure. While this gap becomes smaller by growing the size of the sliding window, it remains for all of the window sizes.

Figures 6 and 7 show an average reduction ratio of about 50 % over Febrl and 74 % over Person in the number of pair-comparisons for ciSNM in comparison with ASNM. As a result of this reduction, the runtime of ciSNM drops significantly, namely 25 % improvement for the Febrl dataset and 26 % for the Persons dataset. Interestingly by increasing the size of the sliding window, our runtime improvement also grows.

The ciSNM algorithm easily outperforms standard SNM in quality and runtime in all of our datasets, which have different sizes and different kinds of data (real-world or synthetic datasets). It also outperforms ASNM by reducing the number of the pair-comparisons and also improving quality and runtime. The improvement over Cora is not very large and constant as other algorithms due to its special characteristics. As it can be seen in Table 1, Cora dataset has the largest clusters, and the highest ratio of records belong to clusters of size greater than two. Although this special property makes this dataset favorable for ASNM, ciSNM performs well even here and slightly improves runtime.

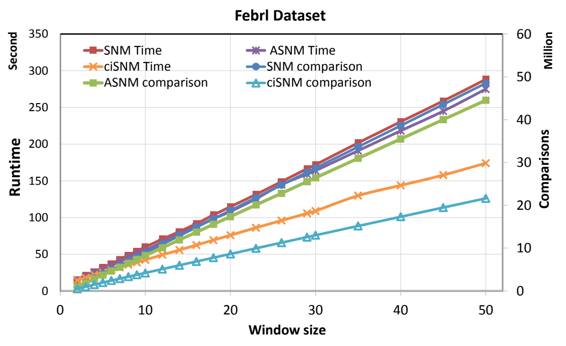


Figure 6: Runtime on Febrl dataset

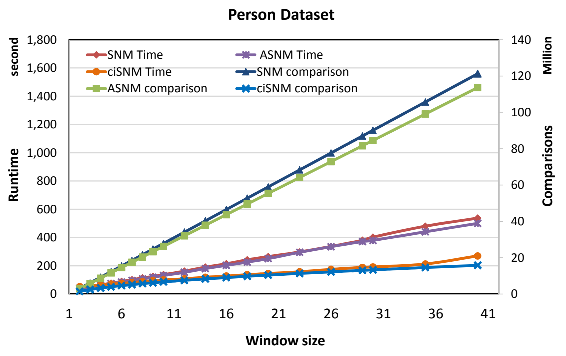


Figure 7: Runtime on Person dataset

Application of ciSNM on different sizes of datasets shows that it performs better on the larger datasets due to diminishing the effect of the clustering overhead. This in turn makes it more favorable for current trends of ever increasing database sizes.

5 Conclusion and future work

The proposed approach improves efficiency of deduplication on different kinds of datasets, real-world or synthetic datasets, by reducing number of pair comparisons in the search space. This reduction is result of filtering those pairs which are less likely to produce a duplicate pair. The ciSNM shows its strength especially for the larger datasets which is our need. The application of this approach is straight forward, because it does not need any extra parameter with a need to a complicated process to choose a value for. In addition to that it does not require any new similarity metric. It uses the same similarity metric used in the similarity function as an internal attribute similarity measure.
In the Big data era, volume of the data and velocity of it's changes make using distributed platform inevitable. In my future work, I would like to investigate on parallelization of the incremental deduplication algorithms in the platforms namely Flink and learn more about it and compare it with other existing platforms such as Hadoop and spark in the context of incremental record deduplication.

References

[1] M. Bilenko and R. J. Mooney. "Adaptive Duplicate Detection Using Learnable String Similarity Measures". In: *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. 2003.

[2] P. Christen. "Probabilistic Data Generation for Deduplication and Data Linkage". In: *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*. Springer-Verlag, 2005.

[3] P. Christen. *Towards parameter-free blocking for scalable record linkage*. Technical report. Faculty of Engineering and Information Technology The Australian National University Canberra, 2007.

[4] W. W. Cohen and J. Richman. "Learning to Match and Cluster Large High-dimensional Data Sets for Data Integration". In: *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. ACM, 2002.

[5] X. Dong, A. Halevy, and J. Madhavan. "Reference Reconciliation in Complex Information Spaces". In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 2005.

[6] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg. "Adaptive Windows for Duplicate Detection". In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2012.

[7] I. Fellegi and A. Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64 (1969).

[8] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. "Approximate String Joins in a Database (Almost) for Free". In: *Proceedings of the International Conference on Very Large Databases (VLDB)*. 2001.

[9] M. A. Hernández and S. J. Stolfo. "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem". In: *Data Mining and Knowledge Discovery* (1998).

[10] M. A. Hernández and S. J. Stolfo. "The Merge/Purge Problem for Large Databases". In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 1995.

[11] A. McCallum, K. Nigam, and L. H. Ungar. "Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching". In: *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. ACM, 2000.

[12] A. Monge and C. Elkan. "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records". In: *Data Mining and Knowledge Discovery* (1997).

[13] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan and Claypool Publishers, 2010.

[14] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. "Automatic Linkage of Vital Records". In: *Science* (1959).

[15] H. B. Newcombe and J. M. Kennedy. "Record Linkage: Making Maximum Use of the Discriminating Power of Identifying Information". In: *Communications of the ACM* (1962).

[16] T. Papenbrock, A. Heise, and F. Naumann. "Progressive Duplicate Detection". In: *IEEE Transactions on Knowledge and Data Engineering* (2015).

[17] S. Sarawagi and A. Bhamidipaty. "Interactive Deduplication Using Active Learning". In: *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. ACM, 2002.

[18] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. 1996.

Time-travel Queries for Omniscient Database Debuggers

Arian Treffer

Enterprise Platform and Integration Concepts
Hasso-Plattner-Institut
arian.treffer@hpi.de

Omniscient debuggers can greatly improve developer productivity. Not only do they allow for more efficient navigation in the execution of a program, they can be used as a foundation for dynamic analyses that further help the developer to identify relevant parts of code. Much work has been done on debugging and analyzing object-oriented code.

We present an approach of bringing omniscient debugging and advanced analysis algorithms to stored procedures. Our prototype allows omniscient debugging of SQLScript that handles large amounts of data, while creating only a small overhead by using an insert-only approach. Furthermore, we show an extension to SQL that allows the developer to express questions that cover a period of execution time.

1 Introduction

The debugger is one of the most important of a software developer. It allows to observe and inspect a program's execution and is useful for many purposes, such as bug detection and code comprehension. Studies found that developers spend up to 50 % of their time debugging.

The usage of a debugger usually follows the same pattern: First, the developer forms a hypothesis about the workings of a specific part of the program. Then, she sets a breakpoint inside or before the code of interest. If the control flow through the program is not certain, multiple breakpoints can be used. Once the debugger halts the execution, the program's state can be examined. The execution is continued in small or larger steps, e.g., using step instructions or more breakpoints. If unexpected values or behavior are observed, the hypothesis is adapted. This process is repeated until the developer's hypothesis is sufficiently confirmed.

Alas, with commonly used debuggers, this approach has several problems. To find good locations for setting breakpoints, extensive knowledge is often necessary. If the hypothesis is changed, other parts of the program may become of interest. If these parts have already been executed, the debug session has to be restarted. This is particularly common in bug hunting, where the infection chain has to be followed from the failure to the code defect, backwards in time. Furthermore, navigation errors such as stepping over a method call instead of into it often make restarting the debug session necessary.

The remainder of the report is structured as follows: The next section gives a brief introduction to omniscient debuggers and presents our previous work on debugging and slicing Java applications. Section 3 shows our ongoing work of adapting these concepts to SQLScript and describes an extension to SQL that allows to compare the database at different points in time. A history of omniscient debugging and other related work is presented in section 4, before we conclude in section 5.

2 Omniscient Debugging

A Backwards Debugger is a debugger that allows to step not only forwards, but also backwards in the execution. As an extension, an Omniscient Debugger is a debugger that knows every state of the program, in the past and future of the current point in time.

Working backwards debuggers have been implemented for several programming languages [7, 11, 12, 13]. Many of them internally work like omniscient debuggers, but do not reveal this to the user.

2.1 Slicing

According to Weiser [18], a (static) slice S is a subset of the statements of a program P on a slicing criterion C, so that for any input I, S and P produce state trajectories equivalent with respect to C. A typical example for a slicing criterion would be a variable in a given line. Then, all statements that can never impact the value of that variable can be removed from the slice. Dynamic slices are defined similarly, but have to produce the equivalent state trajectories only for specific inputs [10].

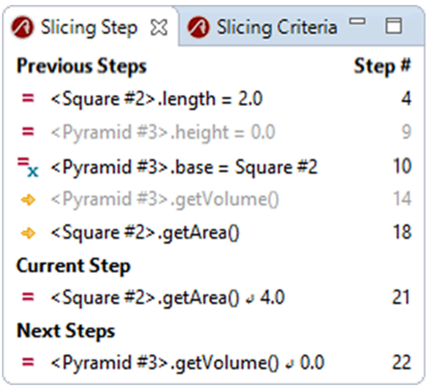
It has been shown that slicing represents how programmers naturally think about problems in programming [18] and has many applications, including, but not limited to, debugging [1] and program comprehension [4].

Typically, static and dynamic slicing algorithms focus on finding statements belonging to a slice. However, in many cases statements are executed multiple times in a single program run and not all executions are relevant for the slice. Therefore, our work focuses on state-changing events, i.e., actual executions of statements, instead of the statements themselves.

Augmenting our previous work on more efficient and customizable slicing algorithms, we developed a new view on the program that allows the developer to control both the debugger and the slicing back-end.

2.2 The Slice Navigator

The purpose of the slice navigator is to aid the developer's short-term memory. It provides a quick overview over previous and upcoming events, and how they relate to the current instruction. Figure 1 shows a screenshot of the slice navigator with the execution of the example test-case halted on the `return` instruction of `getArea()` in line 6. "Previous Steps" lists all past events that the curent or future events depend on. Likewise, "Next Steps" shows all events that depend on the current or previous events. Events that are not directly related to the current step are shown in gray.



Slicing Step | Slicing Criteria

Previous Steps	Step #
= <Square #2>.length = 2.0	4
= <Pyramid #3>.height = 0.0	9
=x <Pyramid #3>.base = Square #2	10
➪ <Pyramid #3>.getVolume()	14
➪ <Square #2>.getArea()	18
Current Step	
= <Square #2>.getArea() ↲ 4.0	21
Next Steps	
= <Pyramid #3>.getVolume() ↲ 0.0	22

Figure 1: The slice navigator view for a program that computes the volume of a pyramid

Using this event list, the developer can obtain different kinds of information about the current state of the execution, such as the immediate context of the current instruction, the relevant program state, and the dependencies to previous instructions.

Different dependency types are indicated by their icons. *Value dependencies* occur when the the value of an instruction is derived from another instruction's value. In the slice navigator, they are represented with a red equality sign. Instructions that determine if another instruction can be reached are *reachability dependencies*, indicated by a yellow arrow. Typically, these are method invocations and instruction in conditional statements. Sometimes, a value depends on only one of multiple candidate values. A *control dependency* determines which of these candidates is used. More formally, control dependencies are reachability dependencies of value candidates that are not also reachability dependencies of all other candidates. In the navigator, they are indicated by a blue "X".

The developer can now combine these different dependency types to adjust the slice for specific purposes. Clicking on an event's dependency symbol brings up a dialog that allows to choose which dependencies of that event to include. This way the developer can, for instance, put a focus on how a value was computed or how an instruction was reached. It is also possible to remove all dependencies of an event, for instance if it is known to be correct and its history is not of interest, thereby moving the focus of the slice to less well-understood parts of the program. Whenever a slicing criterion is modified, the slice is updated instantly, without locking the user interface or resetting the current debug session. the current instruction are added first.

3 Debugging Stored Procedures

Many large and complex applications use a database to persist large amounts of data. However, with the advance of in-memory databases and the decline of RAM cost, the database is no longer seen as a simple data provider [14]. For maximum performance, more and more business logic is moved away from the so-called application layer, which is typically coded in some high-level object-oriented language, into the database, where it has to be rewritten in SQL queries and stored procedures (mostly SQLScript). The increasing complexity of database routines brings an increased need for tool support for debugging.

Regular debuggers for stored procedures already exist. They allow to set breakpoints and to inspect variables and tables, as one would expect. However, often they can not be used as efficiently as debuggers for other languages. It is common for a stored procedure to run several seconds or even minutes, which increases the cost for restarting a debug session. Furthermore, the large amounts of data that can be processed in a single call can make it impossible for the developer to gain a complete understanding of the program state.

Both problems can be solved by testing with minimal example data. However, if the nature of a bug is not yet known, creating such an example may be impossible. In this section, we show how an omniscient debugger for stored procedures can be realized and discuss specific problems such a debugger has to face. A prototypical implementation is currently being developed.

3.1 Tracing and Omniscient Debugging

An omniscient debugger also suffers from the large amounts of data. Our Java debugger traces every field and variable access. Tracing every tuple of a table would create a dramatic overhead. Using an even-bigger database, just to manage a single debug session, is not feasible. Instead, we take advantage of the same that makes stored procedures so powerful in the first place: the declarative nature of SQL queries.

Unlike in object-oriented programs, where almost every behavior can be changed by virtual method calls, it is not possible to change the behavior of a where-clause. Furthermore, SQL queries are well defined so that it is not necessary to analyze the internal workings to allow an analysis of the overall behavior.

Instead, we only need to trace variable assignments to be able to reproduce the program execution. Queries don't have to be traced at all, although for some purposes it will be helpful to record some meta information, such as the execution time or the number of results. However, we need to be able to reproduce the query results, otherwise the debugger would be quite useless.

3.2 Reproducing Query Results

By tracing all variables, the debugger has enough information available to re-execute any query. However, the query will only yield the same results as long as the underlying data has not changed.

In general, one can expect that debugging will take place on a development machine where no other data manipulation occurs. However, in cases where this assumption doesn't hold, the debugger might end up showing wrong or misleading data the developer, which can make the tool outright harmful. Furthermore, the debugged stored procedure itself may change the data, which will cause a query to return different results at different points in time.

In systems that are relevant to accounting, such as ERP, finance, and CRM systems, data is never deleted. Such behavior often even is a legal requirement. If we require for all tables that data can never be changed or deleted and annotate all tuples with timestamps of when they have been created and invalidated, we can reconstruct the state of the database of any point in time.

Especially in in-memory databases, the overhead can be less than expected due to compression, and it may even improve the performance as inserts can be faster than updates or deletes. Finally, adding timestamp filters to select queries does not cause a significant slowdown. Our prototype was built using this approach.

3.3 Time-travel Queries

In our set-up, the debugger trying to recreate intermediate results of a stored procedure is just a special use case for the ability to submit arbitrary queries against the database of any previous point in time.

The query shown in listing 1 selects the total of open orders for previously selected projects. We will use it as an example to demonstrate how *time-traveling* queries are handled by our system.

Listing 1: Example for a time-travel query: select the current total of open orders for previously selected projects

```sql
SELECT pr.id, pr.name, pr.budget, SUM(po.total)
FROM :selected_projects pr
JOIN PurchaseOrders po ON po.project_id = pr.id
WHERE po.status = 'open'
GROUP BY pr.id, pr.name, pr.budget
AT STEP 1623
```

The last line shows an extension to SQL that can be used by the developer to explicitly query a point in time, with `1623` being an instruction ID that was obtained from the debugger UI. If omitted, the current step can be derived from the context from which the query is submitted, such as an SQL console that is associated with a specific point in time or the current debug step. The parameter `:selected_projects` refers to a variable from the current debug session and will be populated with its current value, independently of the value of the step-clause.

When submitted, our debugger applies two changes to the query before it can be submitted to the database. First, all parameters are replaced with corresponding views. When a stored procedure is debugged for the first time, the debugger automatically creates a view for each query in the procedure. These views are identified by the target variable name and the line number and expect a step identifier and all parameters that the actual query takes. In our example, `:selected_projects` might be replaced with `VAR_selected_projects_7(1055, 'Research')` when it was last set at step 1055 in code line 7 and called with the respective argument.

Second, a time-stamp filter is added for all tables that are referenced in the query. In our example,

```
po.createdOn < 1623 AND (pr.validTo IS NULL OR pr.validTo >
    ↳ 1623)
```

would be added to the Where-clause.

Now, the query can be submitted to the database and the result is subsequently presented to the user.

3.4 Time-diff Queries

To get a better overview about what happened in a piece of code, the developer might want to query multiple points in time at once and see the difference in the query result. For this example, she debugs a stored procedure that processes the payments for projects, but sometimes allows projects to go over budget. By stepping into the procedure, she has three defined points in time: *before*, at the beginning of the procedure; *now*, at the current instruction; and *after*, at the end of the execution.

Now she wants to compose a query that selects all projects that will go over budget and the orders that were processed. The query is shown in listing 2. Like before, the *AT STEP* clause does not have to be explicitly typed in the query, but can also be derived from the context. A language extension allows to add filter conditions that only apply to specific points in time. Table 1 shows a possible result for this query, with one project that goes over budget and two associated purchases, of which one was already processed.

To produce this result, the query has to be executed three times, once for each point in time, without the time-specific filter conditions. Then, to prepare the diffing of the results, they are outer-joined on the primary keys and the time-specific filters

Listing 2: Example of a time-diff query: "Select all projects that will go over budget and their respective purchase orders"

```
SELECT pr.id, pr.name, pr.budget, SUM(po.total), po2.id, po2.status,
    ↳ po2.total
FROM :selectedProjects pr
JOIN PurchaseOrders po ON po.project_id = pr.id
JOIN PurchaseOrders po2 ON po2.project_id = pr.id
WHERE po.status = 'open'
  AND now!pr.budget > 0 AND after!pr.budget < 0
  AND before!po2.status != after!po2.status
GROUP BY pr.id, pr.name, pr.budget, po2.id, po2.status, po2.total
AT STEP before=817, now=1623, after=2043
```

Table 1: Result of a time-diff query, with multiple values in some columns

pr.id	pr.name	pr.budget	total	po2.id	po2.status	po2.total
		1200	1500		open	
1	Project 1	200	500	1	paid	1000
		-300	0			
		1200	1500			
1	Project 1	200	500	2	open	500
		-300	0		paid	

are applied. For performance reasons, all of this happens inside a single SQL query, as shown in listing 3. The execution of the sub-queries is indicated in line 6, 7, and 10, the time-specific filters can be found in the Where-condition of line 15 and 16.

Listing 3: Parts of the time-diff query after transformation

```
1  SELECT COALESCE(__before."pr.id", ...) AS "pr.id",
          COALESCE(__before."po.id", ...) AS "po.id",
          __before.createdOn as _step_0,
          __before."pr_name" AS "pr_name_0", ...,
5         ...
   FROM (SELECT ... AT STEP before) __before
   FULL OUTER JOIN (SELECT ... AT STEP now) __now
       ON __before."pr.id" = __now."pr.id"
      AND __before."po.id" = __now."po.id"
10 FULL OUTER JOIN (SELECT ... AT STEP after) __after
       ON (__before."pr.id" = __after."pr.id"
           AND __before."po.id" = __after."po.id")
       OR (__now."pr.id" = __after."pr.id"
           AND __now."po.id" = __after."po.id")
15 WHERE __now."pr.budget" > 0 AND __after."pr.budget" < 0
     AND __before."po.status" != __after."po.status"
```

For the final result, the key attributes are coalesced while the other attributes are selected from each point in time. Furthermore, for each tuple its creation step is selected. This value is needed for two reasons: first, it is necessary to distinguish between tuples with `NULL` values and tuples completely missing from the result; second, it allows the debugger to know when the value was created or changed.

In the UI, the before and after values are only shown if they differ from the now value. Clicking on value allows the developer to jump to the `UPDATE` or `INSERT` statement that caused the change.

3.5 Limitations

Currently, our approach has two major limitations.

First, time-diff queries can only be executed on tables that have clearly defined primary keys, for key attributes are required to track a tuple's versions over time. For a query like "Sum budgets per project category", it has to be clear that categories are the entities that keep their identity over time. Here, an additional syntax extension could be used to convey this kind of information.

Second, it is currently not possible to use time qualifiers outside of the `WHERE` clause.

4 Related work

An omniscient debugger is a debugger that immediately knows about every event in the execution of a program [11]. While reversible execution for debugging purposes has been researched earlier [5], the first omniscient debugger was presented by Lewis [11]. The debugger supported several ways to jump through points-of-interest in the execution, but had no slicing capabilities. Subsequent work in the area focused mostly on memory aspects, for instance by developing a specialized event database [15] or allowing garbage-collection of unreachable past events [13].

The concept of slicing has first been introduced by Weiser, along with a first static slicing algorithm [18]. Korel and Laski, and Agrawal and Horgan later extended the idea to include runtime information to produce more precise slices [2, 10]. Furthermore, Agrawal et al. presented a debugger for C programs with dynamic slicing capabilities [1]. Since then, different slicing algorithms have been proposed and analyzed [3, 8, 19].

For Java, dynamic slicing has been implemented for byte-code traces [16, 17]. JSlice [17] and JavaSlicer [6] are available tools. Ko and Myers used a combination of techniques similar to static and dynamic slicing to automatically answer causality questions [9].

5 Conclusion and Future Work

We have shown how omniscient debuggers can increase developer productivity by allowing backwards navigation and providing fast advanced dynamic analyses. While these techniques can be applied to all imperative programming languages, a prototype for SQLScript has revealed that the database debuggers require additional tools to help understand the underlying data.

Future work will consist of developing omniscience-based algorithms for analyzing stored procedure, and evaluating them with regards to required tracing overhead and performance.

References

[1] H. Agrawal, R. A. Demillo, and E. H. Spafford. “Debugging with dynamic slicing and backtracking”. In: *Software: Practice and Experience* 23.6 (1993), pages 589–616. DOI: 10.1002/spe.4380230603.

[2] H. Agrawal and J. R. Horgan. “Dynamic Program Slicing”. In: *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*. PLDI ’90. New York, NY, USA: ACM, 1990, pages 246–256. DOI: 10.1145/93542.93576.

[3] D. Binkley, S. Danicic, T. Gyimóthy, M. Harman, Á. Kiss, and B. Korel. “Theoretical foundations of dynamic program slicing”. In: *Theoretical Computer Science* 360.1–3 (2006), pages 23–41. DOI: http://dx.doi.org/10.1016/j.tcs.2006.01.012.

[4] A. De Lucia. “Program slicing: methods and applications”. In: *First IEEE International Workshop on Source Code Analysis and Manipulation, 2001. Proceedings*. 2001, pages 142–149. DOI: 10.1109/SCAM.2001.972675.

[5] S. I. Feldman and C. B. Brown. “IGOR: a system for program debugging via reversible execution”. In: *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging*. PADD ’88. New York, NY, USA: ACM, 1988, pages 112–123. DOI: 10.1145/68210.69226.

[6] C. Hammacher. *Design and Implementation of an Efficient Dynamic Slicer for Java*. Published: Bachelor’s Thesis. Saarland University, Nov. 2008.

[7] C. Hofer, M. Denker, and S. Ducasse. “Design and implementation of a backward-in-time debugger”. In: *NODe 2006* (2006), pages 17–32.

[8] T. Hoffner. *Evaluation and comparison of program slicing tools*. Citeseer, 1995.

[9] A. J. Ko and B. A. Myers. “Debugging reinvented: asking and answering why and why not questions about program behavior”. In: *Proceedings of the 30th international conference on Software engineering*. ICSE ’08. New York, NY, USA: ACM, 2008, pages 301–310. DOI: 10.1145/1368088.1368130.

[10] B. Korel and J. Laski. “Dynamic slicing of computer programs”. In: *Journal of Systems and Software* 13.3 (Nov. 1990), pages 187–195. DOI: 10.1016/0164-1212(90)90094-3.

[11] B. Lewis. “Debugging backwards in time”. In: *Computing Research Repository* cs.SE/0310016 (2003).

[12] H. Lieberman. “Reversible Object-Oriented Interpreters”. In: *ECOOP’ 87 European Conference on Object-Oriented Programming*. Volume 276. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1987, pages 11–19.

[13] A. Lienhard, T. Gîrba, and O. Nierstrasz. “Practical Object-Oriented Back-in-Time Debugging”. In: *ECOOP 2008 – Object-Oriented Programming*. Edited by J. Vitek. Lecture Notes in Computer Science 5142. Springer Berlin Heidelberg, Jan. 2008, pages 592–615.

[14] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.

[15] G. Pothier, É. Tanter, and J. Piquer. "Scalable omniscient debugging". In: *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. OOPSLA '07. New York, NY, USA: ACM, 2007, pages 535–552. DOI: 10.1145/1297027.1297067.

[16] A. Szegedi and T. Gyimothy. "Dynamic slicing of Java bytecode programs". In: *Fifth IEEE International Workshop on Source Code Analysis and Manipulation, 2005*. Sept. 2005, pages 35–44. DOI: 10.1109/SCAM.2005.8.

[17] T. Wang and A. Roychoudhury. "Dynamic Slicing on Java Bytecode Traces". In: *ACM Trans. Program. Lang. Syst.* 30.2 (Mar. 2008), 10:1–10:49. DOI: 10.1145/1330017.1330021.

[18] M. Weiser. "Programmers use slices when debugging". In: *Commun. ACM* 25.7 (July 1982), pages 446–452. DOI: 10.1145/358557.358577.

[19] X. Zhang, R. Gupta, and Y. Zhang. "Precise dynamic slicing algorithms". In: *25th International Conference on Software Engineering, 2003. Proceedings*. May 2003, pages 319–329. DOI: 10.1109/ICSE.2003.1201211.

Video Classification with Convolutional Neural Network

Cheng Wang

Internet Technologies and Systems
Hasso-Plattner-Institut
Cheng.Wang@hpi.de

This report summaries my research activities of the past six months in the HPI Research School on Service Oriented Systems Engineer. In this report, I will introduce our work in video classification with deep learning. An automatic video classifier is trained based on the spatial and temporal information. We also proposed a fusion network for combining spatial and temporal information and further improved classification accuracy.

1 Introduction

With the rapid increasing Internet technology, tremendous amount of videos are uploaded to World Wide Web every day. Statistics[1] shows that 300 hours of video are uploaded to YouTube every minute. It is hard for a human to go through them all and find videos of interest. One possible way to narrow the choice is to look for video according to category or label information. But, most videos do not contain semantic meta data and the video platforms are left clueless about the contents. Thus, classifying video according to their content is important to video search and retrieval.

Video classification is a challenging task which attracts much attention recently. Inspired by the recent advance of deep learning, many efforts have been made to enhance the understanding of video, for example, video action recognition with convolutional neural network. One commonly used approach for video classification is based on classifying the key frames that extracted from videos. Recent work [17] proved that temporal clues can provide additional information for improving video classification performance. In this work, we firstly train spatial and temporal model with Convolutional Neural Network (CNN) separately. Since fusing multiple feature such as text-image fusion, audio-video fusion has provided a promising results for video classification, we also focus on the exploration the fusion approach for combining spatial and temporal.

Our system builds on the latest results in the video classification domain. Those latest results are listed and summarized in Section 2. To create our neural network we used the UCF101 [18] dataset. The data set and corresponding preprocessing is explained in detail in Section 3. Our resulting neural network architecture consists of a two-stream neural network architecture. The first stream is a *spatial* convolutional neural network, which is responsible for processing the individual key frame that

[1]https://www.youtube.com/yt/press/en-GB/statistics.html (last accessed 2015-10-01)

extracted from given video. It is explained in detail in Section 3.2. The second stream is *temporal* convolutional neural network that processing the optical flow of a video and is presented in Section 3.3 The two neural networks are merged into a third *fusion* network (Section 3.4). Then we report our experimental result in Section 4. Finally we conclude this work in Section 5.

2 Related work

Previous video recognition research has been focused on obtaining video descriptors which encode motion information and appearances in order to achieve state-of-art results. The research in this area has also been driven by advances in image recognition methods which can be adapted and reused for video processing tasks. Systems employing video descriptors rely on handcrafted features, intensive analysis and prolonged feature preparation. An example for an approach using motion information captured in local spatio-temporal features is called Histogram of Oriented Gradients [5]. Variations include the Histogram of oriented Optical Flow (HOOF) [4] or Motion Boundary Histograms (MBH) [6] around different kinds of trajectories [8]. Those features can be used to encode a video using a bag of words (BoW) approach [14] or Fisher vector based encodings [20]. In a later work [21] it was also shown that using local features instead of sparse interest points is beneficial. The creation of features can be further improved by using global camera motion reduction techniques [9, 13, 20].

There are also a number of attempts to develop a deep architecture for video recognition. Several approaches use 3D-convolution over short video clips to learn motion features from the raw image data [2, 11, 12]. This can be rather challenging as indicated by Karpathy et al. [12] and in [10] an HMAX architecture for video recognition with pre-defined spatial-temporal filters in the first layer is suggested. In [15, 19] convolutional restricted boltzmann machines (RBM) were used for unsupervised feature learning before getting plugged into discriminative models for action recognition.

Another approach by Simonyan et al. [17] directly incorporates motion information from optical flows. Although, this improves prediction quality compared to the naive approach of classifying individual frames, it still only uses information of 10 consecutive frames and can therefore still be considered a local classification approach. Building on those optical flow results Ng et al. [16] and Donahue et al. [7] introduce Long Short Term Memory layers into their networks. This results in an aggregation of strong CNN image features over long periods of a video. The work results in state of the art performance and is superior to the work of Baccouche et al. [1] because of the use of optical flow features.

3 Methodology

This section introduces our architecture, and describes how we process video data for spatial and temporal model training.

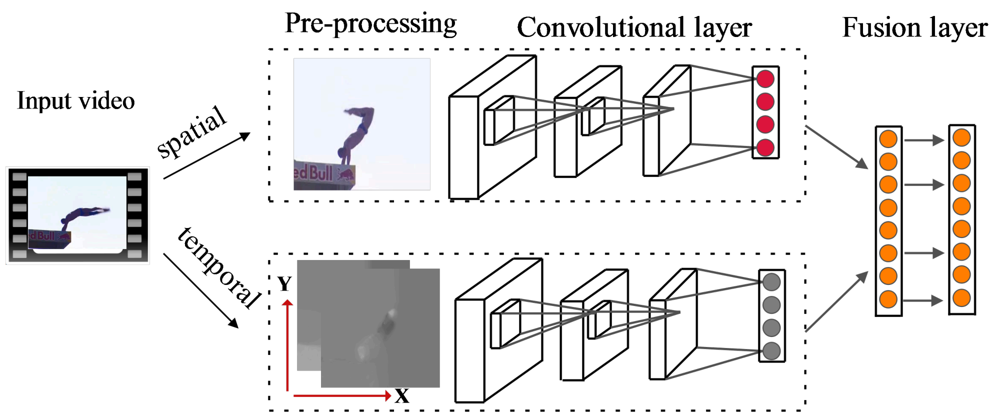


Figure 1: Architecture. Each input video is preprocessed to key frames and two-direction (X-direction and Y-direction) flow images. The input size for each image is 224×224. The features of spatial and temporal are extracted from the first fully connected layer.

3.1 Architecture

We adopted Caffe[2] framework for training spatial and temporal model. As shown in Figure 1, Our framework comprises of three components: the *spatial*, *flow* and *fusion* nets respectively. Based on pre-train models, we extracted the features of the first fully connected layer from each CNNs. And then, we impose a fusion layer which consists of two fully connected layers to combine the features from different streams.

The *spatial CNN* part is processing the single frames of a video in order to recognize objects and structures in frames. The *flow CNN* part captures the motion of actions by learning the optical flow images. To combine the information of both parts, *spatial* and *flow*. We propose to adopt additional fusion layer to fuse the features we extracted from different CNNs. The fusion feature then is applied to give the final predictions. Hence the final overall prediction is a combination of the two-streams.

[2]http://caffe.berkeleyvision.org/ (last accessed 2015-10-01)

3.2 Spatial CNN

The first preprocessing step needs to convert the given video files into key-frame images. We extracted the frame data from the videos with the *FFmpeg*[3] tool. As output, we chose JPEG files. Finding the correct frame rate for the frame extraction was challenging. High frame rates, such as 30 frames per second (FPS), often lead to two adjacent frames being exactly identical. This is a problem for the optical flow extraction, since there will be almost no measurable difference between the images and the optical flow is reported as empty. Especially for classes, where a lot of movement and characteristic optical flow is expected (such as *Archery* or *Juggling Balls*), this turned out to be problematic. The problem of identical images does not exist for lower frame rates, e.g. 5 frames per second. However in this case we create less overall training data and less details, especially for optical flow extraction. A variable frame rate extraction is not feasible, as the optical flow must be comparable between different video types, i.e. the time between two frames must be identical. In this work, we utilized two fps value for key-frame extraction: 30 FPS and 15 FPS.

3.3 Optical CNN

Optical flow is computed to capture the movement in a video sequence and is always based on two immediate consecutive grey-scale frames. We used the optical flow algorithm from Brox et al [3], an accepted standard in the research community. We were able to apply the out of the box OpenCV algorithm[4] and benefited from its GPU computation, leading to faster flow extraction. Optical flow can be computed both along X and Y axis, as shown in Figure 1. Therefore there are two optical flow images for each pair of consecutive frames, resulting in $2 * (N - 1)$ optical flows for N frames. To training optical flow CNN, we adopted CNN_M network [17]. We stack each group of multiple optical flow images as input in Caffe framework.

4 Experiments

This section reports the experimental results and discussion for three parts respectively.

Our spatial CNN is fine-tuned on VGG_19, which has 16 convolutional layers and 3 fully connected layers. From Table 1, we can see the number of fine-tuned layers and FPS are sightly effect the accuracy. By using 30 FPS for key frame extraction and fine tuning the last 5 year achieved the best result on spatial based classification. We also found that increasing the number of fine-tuning layers cannot consistently improve the classification accuracy.

[3]http://www.ffmpeg.org (last accessed 2015-10-01)

[4]http://docs.opencv.org/modules/gpu/doc/video.html#gpu-broxopticalflow (last accessed 2015-10-01)

Table 1: Spatial CNN fine tuning on VGG_19

layers involved	FPS	Accuracy
last 5 layers	30	75.3 %
last 6 layers	30	74.9 %
last 4 layers	15	74.8 %
last 7 layers	15	74.1 %
last 3 layers	30	73.1 %

Our optical flow CNN is trained based CNN_M architecture. We found that if we use the model for predicting directly, the accuracy is 44 %. We fine-tuned the last 3 layers and achieved the 66.4 % result. It did not show better performance when we fine-tuned all layers.

Table 2: Optical CNN fine tuning based on CNN_M

layers involved	Accuracy
last 3 layers	66.4 %
all layers	63.1 %
last 4 layers	62.5 %
none	44.3 %

As mentioned before, our fusion net is consists of two fully connected layer. In this work, adopted early fusion strategy. It means we fuse the two steams at feature level. We extracted the features (4096-D) from the first fully connected layer for each CNNs. And then combine the two feature vector by using linear combination into a long vector which is 8192-D. In our fusion experiments, we fixed batch size at 128, and based learning rate is set as 0.001. The results of our fusion training is described in Figure 2. We note that after 20000 iterations, the training tend to be stable and finally ended with 83.6 %. The summary of each model accuracy is shown in Table 3, our fusion layer improved from spatial by 8.3 % and flow by 17.2 %. It confirms the previous work that one modality can complement other modalities.

Table 3: Fusion result

Stream	Accuracy
Spatial	75.3 %
Flow	66.4 %
Fusion	83.6 %

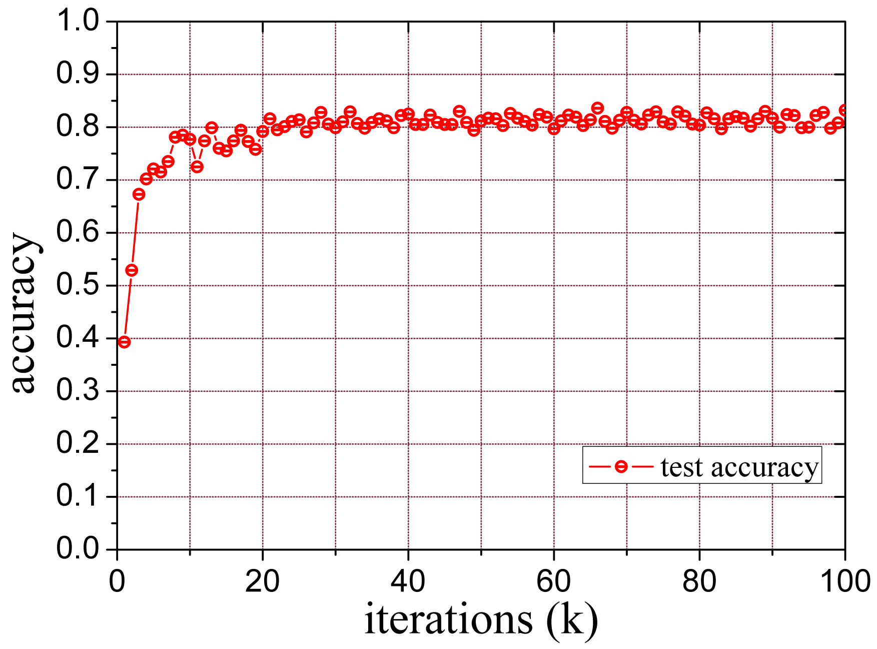


Figure 2: Fusion training

5 Conclusion and outlook

In this report, we proposed an unified framework for video classification. This framework consider the spatial and motion clues, and combine the feature we extracted from the two stream by using early fusion with neural network. Our experiments proved that the combination of different streams is beneficial for improving the final prediction results.

Our future work will focus on the exploration of more sophisticated fusion methods. On the other hand, we will also consider data augmentation methods to further spatial and flow model accuracy.

References

[1] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. “Action classification in soccer videos with long short-term memory recurrent neural networks”. In: *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pages 154–159.

[2] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. “Sequential deep learning for human action recognition”. In: *Human Behavior Understanding*. Springer, 2011, pages 29–39.

[3] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. "High accuracy optical flow estimation based on a theory for warping". In: *Computer Vision-ECCV 2004*. Springer, 2004, pages 25–36.

[4] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. "Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pages 1932–1939.

[5] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Volume 1. IEEE. 2005, pages 886–893.

[6] N. Dalal, B. Triggs, and C. Schmid. "Human detection using oriented histograms of flow and appearance". In: *Computer Vision–ECCV 2006*. Springer, 2006, pages 428–441.

[7] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. *Long-term recurrent convolutional networks for visual recognition and description*. Technical report UCB/EECS-2014-180. Electrical Engineering and Computer Sciences, University of California at Berkeley, 2014. arXiv: 1411.4389 [cs.CV].

[8] M. A. Goodale and A. D. Milner. "Separate visual pathways for perception and action". In: *Trends in neurosciences* 15.1 (1992), pages 20–25.

[9] M. Jain, H. Jégou, and P. Bouthemy. "Better exploiting motion for better action recognition". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pages 2555–2562.

[10] H. Jhuang, T. Serre, L. Wolf, and T. Poggio. "A biologically inspired system for action recognition". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. Ieee. 2007, pages 1–8.

[11] S. Ji, W. Xu, M. Yang, and K. Yu. "3D convolutional neural networks for human action recognition". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.1 (2013), pages 221–231.

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. "Large-scale video classification with convolutional neural networks". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE. 2014, pages 1725–1732.

[13] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. "HMDB: a large video database for human motion recognition". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pages 2556–2563.

[14] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld. "Learning realistic human actions from movies". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pages 1–8.

[15] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pages 3361–3368.

[16] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. "Beyond short snippets: Deep networks for video classification". In: *Computer Vision and Pattern Recognition*. 2015. arXiv: 1503.08909 [cs.CV].

[17] K. Simonyan and A. Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *Advances in Neural Information Processing Systems*. 2014, pages 568–576.

[18] K. Soomro, A. R. Zamir, and M. Shah. *Ucf101: A dataset of 101 human actions classes from videos in the wild*. Technical report CRCV-TR-12-01. Center for Research in Computer Vision, University of Central Florida, 2012. arXiv: 1212.0402 [cs.CV].

[19] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. "Convolutional learning of spatio-temporal features". In: *Computer Vision–ECCV 2010*. Springer, 2010, pages 140–153.

[20] H. Wang and C. Schmid. "Action recognition with improved trajectories". In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pages 3551–3558.

[21] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. "Evaluation of local spatio-temporal features for action recognition". In: *BMVC 2009-British Machine Vision Conference*. BMVA Press. 2009, pages 124.1–124.11. DOI: 10.5244/C.23.124.

ISBN 978-3-86956-345-9
ISSN 1613-5652