

Proceedings of the 7th 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 (Hrsg.)

Technische Berichte Nr. 83

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



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

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

**Proceedings of the 7th Ph.D. Retreat of the HPI
Research School on Service-oriented Systems
Engineering**

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 2014

<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

Das Manuskript ist urheberrechtlich geschützt.

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL <http://pub.ub.uni-potsdam.de/volltexte/2014/6349/>
URN [urn:nbn:de:kobv:517-opus-63490](http://nbn-resolving.org/urn:nbn:de:kobv:517-opus-63490)
<http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-63490>

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-273-5

Contents

Mining Association Rules on RDF Data	1
Ziawasch Abedjan	
On Enabling Context-aware Compliance Monitoring of Business Processes at Run-time in Distributed Systems	13
Anne Baumgrass	
Symbolic Representation and Constraint Reasoning in Invariant Checking	25
Johannes Dyck	
Solving Multidomain Constraints on Object Behavior	37
Tim Felgentreff	
Studying the Nature of MDE Evolution — Case Studies	49
Regina Hebig	
Modeling Interestingness and Serendipity in Relevance Search	59
Maximilian Jenders	
Describing and Comparing Datasets on the Web of Data	69
Anja Jentzsch	
Using Design Patterns to Manage the Productivity vs. Performance Tradeoff in Hybrid Parallel Computing	79
Fahad Khalid	
High-Quality Video Generation for Thin Clients — An Application for Image-Based 3D Portrayal Services	91
Jan Klimke	
Ultra Mobile Devices: using the users body as an interactive device	101
Pedro Lopes	

Adaptive Optimizations for Data Structures in Virtual Runtime Environments	109
Tobias Pape	
Challenges and Approaches of Interaction Techniques for Multi-Perspective Views	121
Sebastian Pasewaldt	
Architectures for Highly-Available Applications with Non-HA Infrastructure	133
Daniel Richter	
Towards a Secure Multi-tenant SaaS Environment	139
Eyad Saleh	
Visualization of Varying Hierarchical Data with Treemaps	149
Sebastian Schmechel	
No Tools But Objects: Towards Direct Manipulation Programming Environments	161
Marcel Taeumel	
Communication-Aware and Memory-Aware VMs Consolidation	173
Ibrahim Takouna	
Using Omniscient Debuggers	185
Arian Treffer	
Enabling Adaptation in Cyber-Physical Systems	195
Sebastian Wätzoldt	

Mining Association Rules on RDF Data

Ziawasch Abedjan

Information Systems Group

Hasso-Plattner-Institut

ziawasch.abedjan@hpi.uni-potsdam.de

Linked Open Data brings new challenges and opportunities for the data mining community. Its underlying data model RDF is heterogeneous and contains machine readable semantic relations. The amount of available open data requires profiling and integration for desired applications. One of the promising underlying techniques is association rule mining. This report presents an overview of elaborated solutions to improve RDF data. In particular, we revisit the concept of mining configurations. Based on the mining configuration methodology, we describe several use cases: ontology engineering, auto-completion, data imputation and synonym discovery.

1 Introduction

Linked Open Data (LOD) is often represented in the Resource Description Framework (RDF) data model: a triple structure consisting of a subject, a predicate, and an object (SPO). Each triple represents a statement or fact.

When processing RDF data, meta information, such as ontological structures and exact range definitions of predicates, are desirable and ideally provided by a knowledge base. However in the context of LOD, knowledge bases are often incomplete or simply not available. Even when a knowledge base is available, we often observe triples that violate its axioms. This inconsistency and lack of metadata impedes the utilization of LOD. Thus, it is useful to automatically generate meta information, such as ontological dependencies, range definitions, and topical associations of resources.

As resources can be connected through multiple predicates, co-occurring in multiple relations, frequencies and co-occurrences of statement elements become an interesting object of investigation for pattern analysis methods, such as association rule mining [6]. To mine RDF data, several questions must be answered: What should be mined in which context, and what are the application fields for each approach. Previous work concentrates on inductive logic programming and graph mining, or is restricted to scenarios where domain knowledge and complete ontology structures are available.

To this end, we introduced the concept of *mining configurations* [2]. A mining configuration specifies one element of the SPO construct as the context of rule mining (the transaction identifiers) and another as the target of rule mining (the items and transactions). For each of the possible six configuration we describe the corresponding application fields. In particular we contribute algorithms to four applications that benefit the usability and machine-readability of RDF data¹:

¹A longer version of this report has been published in the journal "Datenbankspektrum" [3].

- Predicate suggestion.
- Enrichment with missing facts.
- Data-driven ontology re-engineering.
- Query relaxation through predicate expansion.

As a proof of concept we build a profiling tool that integrates all the above mentioned functionalities based on the previously created tool ProLOD [8].

2 Preliminaries

Our approach is based on association rule mining that is enabled by our concept of mining configurations [2]. First, we give a brief introduction to the concept of association rule mining. Next, we introduce our approach of mining configurations for RDF data and outline the characteristics of each configuration.

2.1 Association rule mining

The concept of association rules has been widely studied in the context of market basket analysis [5], however the formal definition is not restricted to any domain: Given a set of items $I = \{i_1, i_2, \dots, i_m\}$, an association rule is an implication $X \rightarrow Y$ consisting of the *itemsets* $X, Y \subset I$ with $X \cap Y = \emptyset$. Given a set of transactions $T = \{t \mid t \subseteq I\}$, association rule mining aims at discovering rules holding two thresholds: minimum support and minimum confidence.

Support s of a rule $X \rightarrow Y$ denotes the percentage of transactions in T that include the union of the *antecedent* (left-hand-side itemset X) and *consequent* (right-hand-side itemset Y) of the rule, i.e., $s\%$ of the transactions in T contain $X \cup Y$. The confidence c of a rule denotes the statistical dependency of the *consequent* of a rule from the *antecedent*. The rule $X \rightarrow Y$ has confidence c if $c\%$ of the transactions T that contain X also contain Y . Algorithms to generate association rules decompose the problem into two separate steps: (1) Discover all frequent itemsets, i.e., itemsets that hold minimum support. (2) For each frequent itemset a generate rules of the form $l \rightarrow a - l$ with $l \subset a$ that hold minimum confidence.

2.2 Mining configurations

To apply association rule mining to RDF data, it is necessary to identify the respective item set I as well as the transaction base T and its transactions. Our mining approach is based on the subject-predicate-object (SPO) view of RDF data.

Any part of the SPO statement can be regarded as a *context*, which is used for grouping one of the two remaining parts of the statement as the *target* for mining. So, a transaction is a set of target elements associated with one context element that represents the transaction id (TID). We call each of those *context* and *target* combinations

a *configuration*. Table 1 shows an overview of the six possible configurations and their preliminarily identified use-cases. Each can be further constrained to derive more refined configurations. For instance, the subjects may be restricted to be of type Person. In the following we further elaborate the meaning of each configuration with regard to the according target of mining.

Conf.	Context	Target	Use case
1	Subject	Predicate	Schema discovery
2	Subject	Object	Basket analysis
3	Predicate	Subject	Clustering
4	Predicate	Object	Range discovery
5	Object	Subject	Topical clustering
6	Object	Predicate	Schema matching

Table 1: Six configurations of context and target

Mining Subjects. In the RDF model, all statements with same subject represent one entity. Subjects with many common predicates can be considered as similar subjects. Thus, mining subjects in the context of predicates (Conf. 3) results in rules that express clustering or ontological affiliation of entities. For instance, in the DBpedia set we retrieved rules between subjects that can be classified as presidents, musicians, or athletes, such as *George Washington* \rightarrow *Lyndon B. Johnson* with 92% confidence. As rules come up when subjects share a minimum number of properties, it can be expected that varying the support leads to clusterings and ontological concepts that differ in granularity.

Mining subjects in the context of objects (Conf. 5), i.e., discovering subjects that share a minimum number of object values, results in rules between entities that are topically related. Objects are values that might be associated with subjects in different relations. E.g., several persons may share the object *Berlin* in different roles like *birth* or *death_place* or *home_town*. In fact, up to 50 distinct predicates in the DBpedia ontology infoboxes data set version 3.7 involve the city *Berlin* as object value. Therefore, organizations as well as persons and instances of other types might share the same objects, and are consequently topically related.

Mining Predicates. While subjects represent entities in RDF data, predicates represent the schema for those entities. So, mining predicates in the context of subjects (Conf. 1) results in patterns and rules that show dependencies of schema elements among entities and can be used for schema discovery and analysis.

Mining predicates in the context of objects (Conf. 6) aims at discovering predicates that have a strong overlap in their value ranges. As predicates define the schema of entities, rules within this configuration can be used for schema matching or synonym discovery. For instance, we discovered rules between the predicates *associatedBand* and *associatedMusicalArtist* that have a confidence of 100% in both directions.

Mining Objects. In accordance to our view of entities and schemata, objects represent the actual values that describe an entity. Thus, mining in the context of subjects (Conf. 2) means to discover patterns between values that are associated to each other by co-occurring for many entities. For example, the rule *Buenos Aires* \rightarrow *Argentina* with 85% confidence shows that entities associated with a capital town are probably also associated with the corresponding country.

Rules in the context of predicates (Conf. 4) imply range discovery of predicates as they connect values, such as numbers, countries, or cities. Exemplary rules include $1 \rightarrow \{2, 3\}$ or $Albania \rightarrow Italy$. In fact, the mining results of this configuration is very similar to the configuration for mining subjects in the context of predicates. Regarding the fact that subjects and objects have semantically different roles in a statement, it is worth reasoning about the actual difference of both configurations.

In the following we exemplify the application of two configurations for mining predicates. Table 2 illustrates some SPO facts extracted from DBpedia. For legibility, we omit the complete URI representations of the resources and just give the human-readable values. The application of Configuration 1 from Tab. 1 to our example data set would transform the facts into three transactions, one for each distinct subject as illustrated in Tab. 3a. In this example, the itemset $\{birthPlace, party, orderInOffice\}$ is a frequent itemset (support 66.7%), implying rules such as $birthPlace \rightarrow orderInOffice, party$ and $orderInOffice \rightarrow birthPlace, party$ with 66.7% and 100% confidence, respectively. Furthermore, we can infer negative rules, such as $birthPlace \rightarrow \neg born$.

Subject	Predicate	Object
Obama	birthPlace	Hawaii
Obama	party	Democrats
Obama	orderInOffice	President
Merkel	birthPlace	Hamburg
Merkel	orderInOffice	Chancellor
Merkel	party	CDU
Brahms	born	Hamburg
Brahms	type	Musician

Table 2: Facts in SPO structure from DBpedia

TID	transaction
Obama	$\{birthPlace, party, orderInOffice\}$
Merkel	$\{birthPlace, party, orderInOffice\}$
Brahms	$\{born, type\}$

(a) Context: Subject, Target: Predicate

TID	transaction
Musician	$\{type\}$
Hamburg	$\{born, birthPlace\}$
Hawaii	$\{birthPlace\}$
President	$\{orderInOffice\}$

(b) Context: Object, Target: Predicate

Table 3: Configuration examples

Configuration 6 in the context of objects would create the transactions presented in Tab. 3b. The frequent itemsets here contain predicates that are similar in their ranges, e.g., $\{born, birthPlace\}$. Given the negative rule in Conf. 1 and the pattern in Conf. 6, one could conclude that both predicates *born* and *birthPlace* have synonymous meanings and can be used for predicate expansion.

3 Predicate Suggestion

Suggestion of predicates or objects aims at two goals. First, the user who is creating new facts for a certain subject might be grateful for reasonable hints. Second, system

feedback might prevent the user from using inappropriate synonyms for predicates as well as objects.

Suggestion Workflow. For suggesting predicates or objects for a user that is creating facts for a certain subject we directly apply the Configurations 1 or 2, respectively. The suggestion workflow for predicates requires two preprocessing steps: (1) Generate all association rules between predicates. (2) Create an index to facilitate the retrieval of all relevant rules for a specific suggestion situation.

When a user is inserting or editing the facts related to a specific subject, the system is aware of all predicates that have already been inserted for the current subject. For generating a list of suggestions, all rules that incorporate the previously inserted predicates as their antecedents are retrieved. The suggestions then are all those predicates that occurred as consequences of the retrieved rules. The ranking of the suggestions is based on scores that are computed for each suggestion by aggregating all confidence values of the retrieved rules that have the specific predicate suggestion as their consequence. Based on the next chosen predicate the suggestion list changes again, because the rules that contain the new predicate as their antecedent are also taken into account.

Tables 4a and 4b show the performance of our algorithm on several datasets from DBpedia. Here we randomly removed predicates and objects from each entity and tried to suggest it with our algorithm. The tables display precision at x ($p@x$) and mean reciprocal rank (MRR) scores, showing that indeed association rule mining is much more suited for suggesting predicates than objects.

Type	p@5	p@10	MRR at 10
Thing	0.420	0.639	0.20888
Person	0.510	0.714	0.26199
Place	0.507	0.771	0.21717
Work	0.275	0.555	0.12450
Species	0.648	0.909	0.27802

(a) Predicate suggestions

Type	p@5	p@10	MRR at 10
Thing	0.050	0.055	0.03179
Person	0.027	0.028	0.02315
Place	0.069	0.069	0.06058
Work	0.015	0.015	0.01276
Species	0.440	0.541	0.22191

(b) Object suggestions

Table 4: Evaluations for 10,000 predicate/object suggestions per data set

4 Auto-amendment of Triples

We propose two different approaches to amend a dataset with new facts: *user-driven auto-amendment* and *data-driven auto-amendment*. When creating new statements where the user decides which subject has to be amended with new triples we speak of *user-driven auto-amendment*. This approach follows our suggestion scenario from Sec. 3. Based on a subject that is being edited, the algorithm could try to generate new facts by guessing predicate and object combinations. Our *data-driven auto-amendment* approach lets the system itself choose the subjects that should be amended with new triples. Our data-driven approach is based on the following intuitions:

1. For object rules $o_1 \rightarrow o_2$ with high confidence (above 90%) the subjects occurring

with the object o_1 are also likely to occur with the object o_2 . However 10% of the subjects that occur with o_1 *violate* the rule by not occurring with o_2 in any fact. Our assumption is that those facts are absent, because of missing thoroughness during data creation. For example a user that adds Honolulu as the `birthPlace` of a person assumes that the country where Honolulu lies (namely the USA) is implicitly given.

2. A subject should not be enriched with a fact containing object o_2 if on the basis of the rules involving schema predicates, no predicate can be chosen for the connection with o_2 . This intuition allows a softening of the earlier intuition that expects all subjects that violate $o_1 \rightarrow o_2$ should be extended with a triple containing o_2 .

For data-driven auto-amendment we need to generate all predicate rules, corresponding to Conf. 1 from Tab. 1, and store them within a predicate-predicate rule matrix. Then we generate high-confidence object rules $o_i \rightarrow o_j$ in the manner of Conf. 2. For each object rule $o_i \rightarrow o_j$, all subjects that occur with the antecedent of a high-confidence rule but not with its consequent are retrieved. These subjects may be amended with new facts having the current object rule consequent o_j as their value. The choice for 90% as the high confidence threshold is arbitrary. The higher this threshold is, the fewer new statements can be generated but higher precision is achieved. Then the algorithm proceeds with retrieving the candidate predicates that have o_j in their range. The score for each candidate predicate is then computed in the same manner as described for predicate suggestions based on given rules with schema predicates as antecedents.

Note the number of new facts depends on the number of existent high-confidence rules and their corresponding set of rule violating subjects. Using this approach on DBpedia v3.6 we were able to generate 26,646 new facts out of which 31% were actually included in the later version 3.7. Most of the inclusions correspond to new facts on entities of types `Artist` or `Animal`, where the ratio was above 50%.

5 Reconciling Ontologies and Data

A common case of inconsistency is the mismatch of ontology definitions and the underlying data. In particular, divergences between ontology specification and instance data may occur in two scenarios: On the one hand, the ontology might have been developed independently and before actual data using it was published, e.g., in the case of the “Friend of a Friend”² ontology (FOAF). On the other hand, the ontology might have been tailored for existing data, e.g., in the case of the DBpedia project, which evolved extensively since its first specification, while revising existing class definitions was sometimes neglected during this evolutionary process.

Based on an existing ontology, we identify two typical cases where the specification differs from usage patterns: *overspecification* and *underspecification*. We refer to a certain class as being overspecified, if one or more properties are declared for this class by the ontology, but are rarely (if ever) used for real-world data, e.g., `scottishName`

²<http://xmlns.com/foaf/spec/>

for `Settlement`. There are several reasons, why overspecification occurs: For example, data providers cannot set proper values for the defined properties, e.g., a `scottishName` for a non-Scottish `Settlement`.

A class is underspecified, when in real-world data certain properties are used frequently even though they are not specified by the vocabulary. Underspecification may occur when the class definition lacks certain properties that are commonplace in instance data, e.g., `genre` for `Band`. Note that a class can simultaneously be overspecified and underspecified (with regard to different properties).

Given a dataset with typed instances and a corresponding ontology, we apply frequency and association rule analysis by applying Conf. 1 to identify and remedy over- and underspecification [1]. We identified 503 removal suggestions in the DBpedia 3.6 ontology and 622 removal suggestions in the DBpedia 3.7 ontology, all with support $\leq 1\%$. Table 5 shows sample results of overspecification in DBpedia 3.7. Some of the removal suggestions can be moved to a more suitable subclass as suggested in [1].

Property	Class	Support
<code>scottishName</code>	<code>Settlement</code>	0.000%
<code>distanceToEdinburgh</code>	<code>Settlement</code>	0.021%
<code>philosophicalSchool</code>	<code>Person</code>	0.202%
<code>countySeat</code>	<code>PopulatedPlace</code>	0.831%
<code>anthem</code>	<code>PopulatedPlace</code>	0.147%
<code>depth</code>	<code>Place</code>	0.723%
<code>numberOfGraduateStudents</code>	<code>EducationalInstitution</code>	0.300%

Table 5: Overspecified properties for DBpedia 3.7

Table 6 illustrates the amount and quality of class property suggestions for DBpedia 3.6 and 3.7 (*minSupp*: 1%, *minConf*: 70%). Overall, the majority of the suggestions have been labeled as useful. Suggestions marked as undecided are those for which we could not decide whether they enhance the class definition or not. This was often the case, when a similar or synonymous property had already been defined for a class in the ontology (e.g., for `Person`, `Person/weight` is specified, `weight` is suggested). Synonym discrepancy constitutes a major problem for data consumers, as it happens that either properties are inconsistently used or the expectation of of a user towards a property and the ontology designer may diverge. In the next section we pick up on the synonym discrepancy and present a solution for discovering such synonyms.

DBpedia	Total	Useful	Not Useful	Undecided
3.6	283	234 (83%)	15	34
3.7	317	268 (85%)	31	18

Table 6: Suggestion quality for DBpedia 3.6 and 3.7

6 Predicate Expansion

We already showed that some discrepancies between property usage and ontology definitions emerge when instead of defined properties synonymous predicates are used in the data. Some examples that we encountered during our evaluations on the DBpedia data set where for instance `city` or `location` instead of `locationCity`.

Of course two synonymous predicates may have been defined deliberately for two disjoint purposes, but because they have been used in substitution of each other, the data consumer has to deal with the inconsistency. We developed an approach for automatically discovering predicates that have been used in substitution of each other in the data, i.e., they have some synonymous meaning. The discovery of such dependencies is relevant for query expansion. A user that looks for actors of a movie and intuitively chooses the predicate `starring` will miss all actors where a synonymous predicate like `artist` has been used. Note, we explicitly talk about synonymously used predicates instead of synonym predicates. For example, predicates with more general or specific meaning often substitute each other in the data. E.g., `artist` is often used as a substitute for `starring` even though `artist` is more general than `starring`.

We apply Configurations 1 and 6 in the same manner as exemplified in Sec. 2. With Configuration 1 we perform schema analysis in the context of subjects. Configuration 6 enables us to mine similar predicates in the context of objects. We also looked into the range structure of predicates by looking at value type distributions. Despite the fact that type definitions might not always be available we could not identify any benefit to the range analysis approach in our experiments.

Configuration 1 enables us to do frequency analysis and rule discovery per entity. We used this configuration already for suggesting new predicates for data creators and generating inclusion suggestions for the ontology. Here we have to look at a different intuition: Expansion candidates for a predicate should not co-occur with it for any entity. It is more likely for entities to include only one representative of a synonymous predicate group within their schema, e.g., either `starring` or `artist`. That is why we look for negative correlations in Configuration 1. Negative schema correlations might also lead to false positives, such as `recordLabel` and `author` as both occur for different entities. While songs have the predicate `recordLabel`, books have the predicate `author`. So a negative correlation is not a sufficient condition for a predicate to be expanded by another. Therefore we also take the range content of predicates into account.

Our second intuition is that as synonym predicates have a similar meaning they also share a similar range of object values. Normally when trying to compute the value overlap between two predicates one would look at the ratio of overlaps depending on the total number of values of such a predicate. We apply a more efficient range content filtering approach (RCF) based on Conf. 6 that constitutes a mining scenario where each transaction is defined by a distinct object value. So each transaction consists of all predicates containing the distinct object value in their range. Frequent patterns in this configuration are sets of predicates that share a significant number of object values in their range. Experiments showed that this approach is by magnitudes faster than the pairwise overlap recognition approach.

Our approach works in the following way: (1) first retrieve all predicate pairs through range content filtering, then (2) analyze their schema co-occurrences.

We performed multiple experiments on many real world data sets. Our experiments showed that our combined approach generates less false positives the more homogeneous the entities in the data set are. For example, on the Magnatune dataset that contained only music data we achieved precision values of 100% on a 0.1% support threshold for RCF, while on the DBpedia data set the precision was around 40%. Here

the algorithm generated false positives like `foundingPlace` and `birthPlace`, because the subjects of these predicates are from very different domains while the range of both very similar. The evidences by Configurations 1 and 6 are not enough. Table 7 shows our top 5 results on the DBpedia Work and Organisation data set [4].

	DBpedia Work	DBpedia Organisation
1.	artist, starring	city, location
2.	artist, musicComposer	city, hometown
3.	author, writer	location, hometown
4.	creator, writer	city, ground
5.	composer, musicComposer	city, locationCity

Table 7: Discovered top 5 synonym pairs on DBpedia subsets

7 Related Work

We apply existing data mining algorithms to the new domain of LOD and proposed four different use cases on this basis. Therefore, we show an overview of related work with regard to data mining in the semantic web as well as the most related approaches to our presented use cases.

Mining the Semantic Web. Most research on mining the semantic web is so far in the fields of inductive logic programming and approaches that make use of the description logic of a knowledge base [14]. Those approaches concentrate on mining answer-sets of queries towards a knowledge base. Based on a general reference concept, additional logical relations are considered for refining the entries in an answer-set. A statistical approach for mining the semantic web is proposed by Nebot et al. [18], where a SPARQL endpoint allows the user to define targets of mining in any desired graph context.

Looking at RDF data as graph where resources are connected via predicates as edges, another related field of research is mining frequent subgraphs or subtrees [15].

ALEPH [17], WARMR [10], and Sherlock [19] are known systems to mine horn rules for generating new statements. ALEPH is an ILP system based on Muggleton’s Inverse Entailment Algorithm [17]. WARMR uses a declarative language to mine association rules on small sets of conjunctive queries. Sherlock uses a probabilistic graphical model to infer first order clauses from a set of facts for a given relation [19]. A recent system for fact generation in RDF data is AMIE [13]. In a number of experiments AMIE showed to be the most efficient and effective approach to generate new facts compared to ALEPH [17] and WARMR [13]. Therefore, we compare our system to AMIE.

Ontology Engineering The most related work in this field is the schema induction approach by Völker et al. [20]. The authors describe how association rules can be used to recreate axioms of the DBpedia ontology. Fleischhacker et. al. extend this approach to discover also characteristics that are not predefined by an ontology, such as predicate symmetry, asymmetry, and disjointness [12]. Our work on improving ontologies differs as we create specific suggestions for changing the ontology of a data set by removing or adding properties. Several works in the field on ontology engineering aim

at establishing and enriching ontology specifications by using machine learning techniques [9]. The authors of [16] present a semi-automatic approach for cross-domain ontology learning. Similarly, in [21] machine learning methods are employed to refine the definition of the Wikipedia infobox-class ontology.

Query expansion and synonym discovery Research on query expansion includes stemming techniques, relevance feedback, and other dictionary based approaches [7]. On their technical level the approaches do not apply to our SPARQL scenario as we do not retrieve documents but structured entities. Elbassuoni et al. have already presented a query expansion approach based on language models [11]. Our approach is based on association rules and a more simplistic model and we were able to process large datasets, such as DBpedia, in a few minutes.

8 Conclusion

This report gave an overview of the main contributions of my thesis. We showed our mining configuration methodology and applied it to several use cases that detect and can prevent inconsistency in RDF data. We showed how one configuration can be used for predicate suggestion and ontology re-engineering. Furthermore, we introduced approaches for triple amendment and predicate expansion, based on combining two configurations. Currently we work on a tool that integrates all the presented uses cases along with other basic profiling tasks.

References

- [1] Ziawasch Abedjan, Johannes Lorey, and Felix Naumann. Reconciling ontologies and the web of data. In *CIKM*, pages 1532–1536, 2012.
- [2] Ziawasch Abedjan and Felix Naumann. Context and target configurations for mining RDF data. In *SMER*, 2011.
- [3] Ziawasch Abedjan and Felix Naumann. Improving rdf data through association rule mining. *Datenbank-Spektrum*, 13(2):111–120, 2013.
- [4] Ziawasch Abedjan and Felix Naumann. Synonym analysis for predicate expansion. In *ESWC*, 2013.
- [5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.
- [6] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*, pages 487–499, 1994.
- [7] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. 1999.

-
- [8] Christoph Böhm, Felix Naumann, Ziawasch, Dandy Fenz, Toni Grütze, Daniel Hefenbrock, Matthias Pohl, and David Sonnabend. Profiling linked open data with ProLOD. In *NTII*, pages 175–178, 2010.
- [9] Paul Buitelaar and Philipp Cimiano, editors. *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, volume 167 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.
- [10] Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
- [11] Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. RDF Xpress: a flexible expressive RDF search engine. In *SIGIR*, 2012.
- [12] Daniel Fleischhacker, Johanna Völker, and Heiner Stuckenschmidt. In *OTM*, volume 7566, pages 718–735. 2012.
- [13] Luis Galàrraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.
- [14] Joanna Józefowska, Agnieszka Lawrynowicz, and Tomasz Lukaszewski. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *Theory Pract. Log. Program.*, 10:251–289, 2010.
- [15] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.
- [16] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16:72–79, 2001.
- [17] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
- [18] Victoria Nebot and Rafael Berlanga. Mining association rules from semantic web data. In *IEA/AIE*, volume 2, pages 504–513, 2010.
- [19] Stefan Schoenmackers, Oren Etzioni, Daniel S. Weld, and Jesse Davis. Learning first-order horn clauses from web text. pages 1088–1098, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- [20] Johanna Völker and Mathias Niepert. Statistical schema induction. In *ESWC*, pages 124–138, 2011.
- [21] Fei Wu and Daniel S. Weld. Automatically refining the Wikipedia infobox ontology. In *WWW*, pages 635–644, 2008.

On Enabling Context-aware Compliance Monitoring of Business Processes at Run-time in Distributed Systems

Anne Baumgrass

Business Process Technologies

Hasso Plattner Institute

anne.baumgrass@hpi.uni-potsdam.de

This report summarizes my activities in the HPI Research School on Service Oriented Systems Engineering of the last six months. I have worked in the area of Complex Event Processing (CEP) and how this area can be of use for Business Process Management (BPM). Specifically, I investigated approaches and techniques on how events can be used to ensure the compliant execution of business processes in an organization.

1 Introduction

Organizations demand flexible business processes for their competitiveness. At the same time they also demand that their business processes as well as their executions are compliant with numerous laws, regulations and business policies in order to avoid penalties, scandals, and loss of business reputation. As stated in [1], due to the constant evolution of business processes and compliance rules, automated approaches to reason about the adherence of process models to compliance rules, become necessary. The evolution of business processes is reflected in the business process lifecycle [15]. Specifically, as described in [9], compliance must be ideally considered throughout all the phases of such lifecycle, giving rise to a full-coverage integrated Business Process Management System (BPMS). Thus, adequate mechanisms for supporting and ensuring compliance in each phase are required.

As derived from the studies presented in [5], most of the compliance approaches focus on defining rules and checking compliance at design-time based on process models, but only a few proposals face run-time compliance monitoring. In this regard, current approaches present some shortcomings that should be addressed. Firstly, business process compliance is not addressed from a distributed environment perspective. However, nowadays there is an increasing trend to integrate business processes with others that exist outside the boundaries of an organization [8]. This brings additional complexity to managing processes across multiple enterprises and, thus, makes it harder to ensure the compliance. Secondly, in distributed and typically service-oriented scenarios, as well as in intra-organizational collaborations, the communication among the parties involved is usually performed by means of events that may influence the

business operations. Utilizing, transforming, and processing these events for different purposes would be in the responsibility of a CEP engine [8] and is disregarded in most of the existing BPMSs. There exist some run-time compliance checking approaches based on event handling [12, 14]. However, compliance checking in such approaches is either focusing on monitoring events and mechanisms to use these events to act in an organization, or they are limited to the business process perspectives, both disregarding interesting aspects in flexible and distributed process-oriented environments such as context and mechanisms to react to certain happenings. Thirdly, and derived from the previous two points, current approaches hardly consider context when deciding which compliance rules must be checked at every stage in business process execution. Contrary to this, the scope is often bound to one or two business process perspectives, typically control flow and/or time. However, a single compliance rule commonly involves taking into account several perspectives, e.g., dependencies between activities observed in the control flow, or the presence of documents to enable the execution of activities.

This report sketches a novel approach that processes events in order to ensure context-aware compliance of business processes with specific rules at run-time. We have worked with real complex distributed scenarios in the logistics domain, discovered in the context of the FP7 EU GET Service project¹. The aim of the GET Service project is to develop a European Wide Service Platform for Green European Transportation (GET Service) that provides transportation planners and operators with the means to plan, re-plan, and control transportation routes efficiently and in a manner that reduces greenhouse gas emissions. For this purpose, we identified three major challenges in the monitoring of process-oriented complex logistics chains and described the required features that such a monitoring system should provide, as well as related literature referring to these challenges, see P1 in Table 4. Based on these results, we define a framework for the monitoring of complex distributed systems and the detection of compliance violations considering all the aspects based on the use of CEP functionality and rule anti-patterns. This framework can meet and already fulfills parts of the requirements desirable in a compliance management framework according to [9].

2 Running example

In BPM, several approaches have shown how to check the compliance of the control flow, associated data objects, or resources at design-time [9]. Our approach complements these approaches by the enabling of context-aware compliance at run-time using events. Therefore, although all other perspectives also apply for the described process below, we focus on considering contextual information required for processing events related to a business process that enables context-aware compliance at run-time in distributed systems.

A simplified transportation process of a Logistics Service Provider (LSP) is shown in Figure 1. Here, the driver gets notified to pick-up a container, to drive to a factory where the container is loaded, and to bring the container to the harbor for shipping offshore. For all executions of this process, the compliance rules in Table 1 have to

¹<http://getservice-project.eu/>

hold, although some of them are not applicable for all executions or may differ in their details for each execution. On the one hand, the checking of compliance rule CR1 is not required for goods that are not classified as dangerous goods. On the other hand, working hours must be arranged and calculated separately for each driver, and, thus, each business process execution. Therefore, the details to check compliance rule CR3 differ for each execution.

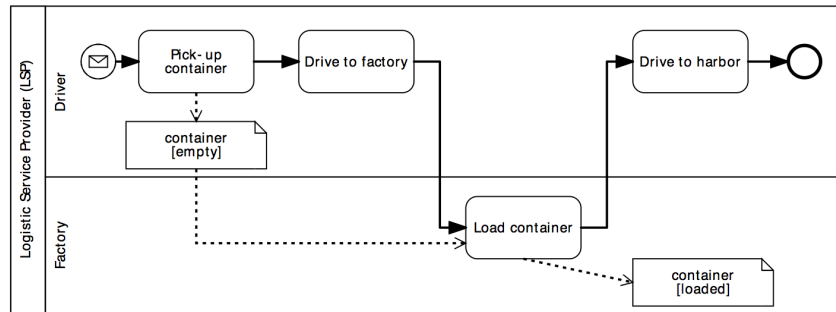


Figure 1: BPMN diagram of a general transportation process

Table 1: Examples of compliance rules for transportation processes

CR1	Only drivers owning security certificates can transport dangerous goods.
CR2	For cooling goods, the cold chain must not be interrupted.
CR3	Drivers shall only drive within their working hours.

The compliance rules in Table 1 are context-aware in the sense that the required information for checking these compliance rules is not included in the business process model itself but must be available via events and their context correlated to the business process. The context of this example is given in Tables 2 and 3. For instance, an event occurred that includes the loading of 'Goods1' for 'Driver1'. Since the goods are of type 'dangerous' the driver needs to have a security class of 'Class2' or higher (resp. 'Class1'). In this case no alert is thrown for the compliance rule CR1. In contrast, an alert will be forwarded in case 'Driver3' would try to carry 'Goods1' or 'Goods3'.

	Working hours	Security
Driver1	9-17	Class1
Driver2	15-24	Class2
Driver3	9-15	-
...
DriverN	9-17	-

Table 2: Driver information

	Type	Security	Cooling range
Goods1	dangerous	Class2	-
Goods2	normal	-	-
Goods3	explosive	Class1	1°C to 15°C
...
GoodsN	cooling	-	-10°C to -5°C

Table 3: Goods information

the compliance rule CR2 defines that the cooling chain must not be disrupted, however, from this rule it is not obvious under which degrees which goods must be transported. While frozen meat must be cooled at -18°C the fresh meat must be transported at 4°C .

4. The compliance rules enriched with knowledge are converted to anti-patterns for checking the event cloud for violations of the process model and corresponding compliance rules. In CEP, patterns define the sequence and the properties that events must follow to process them, in our case for compliance monitoring. Since we are interested in the violations of compliance rules we do not check all events if they hold for the defined compliance rules but we check those that do not hold. Therefore, the CME component must be able to transform compliance rules to anti-patterns in order to check for violations. As soon as the compliance rules (in form of anti-patterns) are transformed into a format that can be used to parse the event cloud our CME component can check for event occurrences that match these anti-patterns.
5. Events that match an anti-pattern represent a violation of a compliance rule and must be forwarded by the CME component to the process manager.
6. A violation of a compliance rule must be presented in a human-readable format to the user. This can be as alerts, notifications, and in the ideal case as instructions to prevent non-compliant behavior and to meet compliance. The latter is considered as an very relevant part of the research initiative with Bosch Software Innovations GmbH in which we examine mechanisms for dynamically changing process executions based on events, see P6 in Table 4. In the logistics example, the temperature in a container may falls under a certain degree (violating CR2), then a corresponding compliance instruction could be to transship the goods into another cooling container (as additional tasks) without disrupting the cooling chain. In addition, notifications and instructions could also be enriched with knowledge, e.g., to signalize where to find a working cooling container close to the current position of the driver.
7. In case of violations against process analyst's compliance rules she can now decide to reconfigure the process or just the single process instance.

For implementation, we propose an event monitoring framework for business process compliance including the following components, see Figure 3:

A **process model repository** contains the *process models* under consideration while the **event type repository** contains *event types*. The correlation of event types to process models is defined via process monitoring bindings [7]. This correlation is required to process *events* available in an **event cloud** and then to identify the right events for process monitoring. The **knowledge base** contains all related *knowledge* of a process or an event, e.g. traffic information, container requirements, or driver's qualifications. For compliance checking, the **query processor** receives *compliance rules* from an **analyst** and transforms them to machine-readable *anti-patterns* for the **pattern matcher**. This transformation is also done based on a process model. Afterwards, *events* (correlated to a business process over their event types) are checked

against defined anti-patterns. The system raises *alerts* in case events and the corresponding context match the specific anti-pattern.

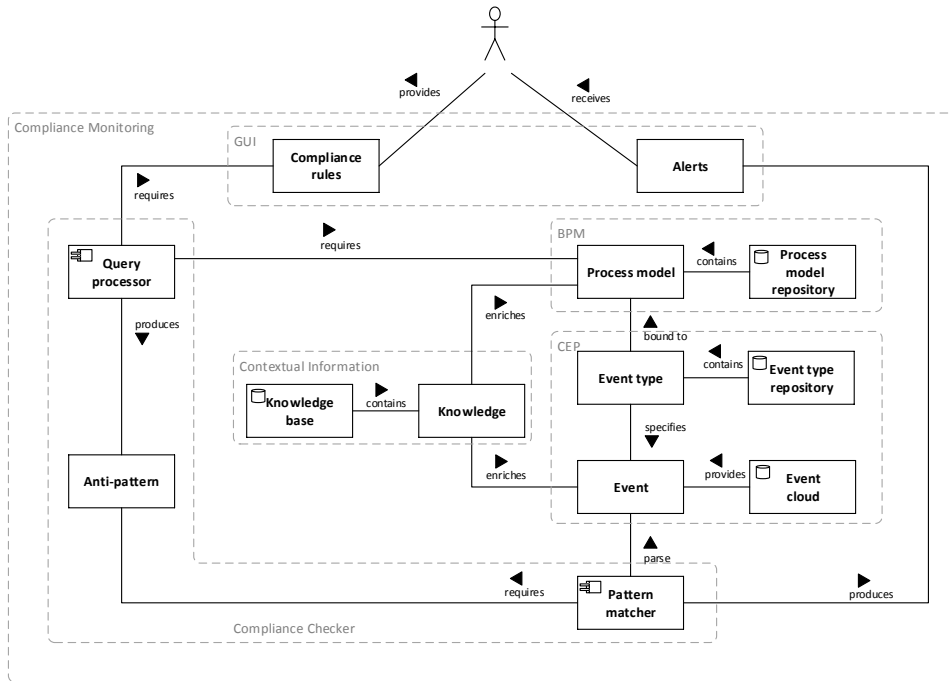


Figure 3: Copmlance Monitoring Components

For the example in Section 2, the analyst is the LSP watching the compliance rule CR1, see Table 1. For this purpose, the query processor produces an anti-pattern for the given compliance rule – based on the process model, a set of process monitoring bindings, and context information on driver’s certificates and carried goods (in the container). The pattern matcher checks the events arriving in the event cloud and alerts the LSP whenever the compiled compliance rule is not satisfied. This can be the case whenever a driving event occurred for a container containing dangerous goods for which the driver does not have the specific certificate to carry these kinds of goods.

4 Evaluation

Our implementation on enabling context-aware business process compliance at runtime using events follows the eight fundamental requirements of [9]). We assume an BPMS exists that integrates CEP functionality to detect, enrich, correlate, and process events. As pre-processing to validate compliance, we reuse and combine existing approaches that are able to detect events in event cloud (see, e.g., [3, 7]), to enrich events with context (see, e.g., [16]), and to correlate these to process instances (see, e.g., [11]). Below, each requirement is discussed for our implementation by listing its name, a brief description, if existent its prototypical implementations using the example in Section 2, and the challenges that still have to be tackled.

Req. 1 – A formal language for context-aware compliance rule specification: *An compliance specification language should provide an appropriate balance of expressiveness, formal foundation, and efficient analysis.*

Our approach enables this requirement by formalizing context-aware compliance rules using an Event Pattern Language (EPL), thus relying on CEP that provides continuous and incremental processing, a low latency, meaning near real-time results, as well as high data rates. Assuming that our context is stored in an ontology (the knowledge base), we can even define queries to identify events according to CR1 using Semantic Complex Event Processing (SCEP) queries (to show the applicability we used this EPL, but others are also applicable):

Listing 1: An example for a SCEP query that selects drivers who do not have the required security class to transport dangerous goods.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX trans: <http://example.org/transport/>
SELECT * FROM DrivingStream as $event
{$event dbo:person ?driver.
$event trans:transportunit ?goods.
?driver trans:securityclass ?classdriver.
?goods trans:securityclass ?classgoods.
filter(?classdriver > ?classgoods)}
```

To detect events that correspond to such queries incoming events to the implementation are attached with knowledge, so event's attributes are matched to semantic entities and linked to external knowledge bases in order to enrich the information of the event [16]. In the example, events that occur for drivers and goods are associated with a security class stored in an ontology. Thus, events referring to a security class "2" or "1" (i.e., two or lower) can be interpreted as dangerous goods and are therefore relevant for processing and evaluating the compliance rule CR1 (cf. Listing 1).

Req. 2 – Compliance rule organization: *Compliance rules can be relevant for a single or multiple processes as well as choreographies (across enterprise boundaries in distributed systems).*

To recap, we assume that all relevant events are published in an event cloud and are consumable by our implementation. The events in an event cloud represent the execution of business processes. Currently, we integrated a storage for compliance rules that are formalized as CEP pattern or as SCEP and described in P5 shown in Table 4. Furthermore, these stored rules can be changed easily, however the mechanisms to version and propagate changes to relevant processes is still an open issue we will work on in future work.

Req. 3 – Views on compliance rules at different abstraction levels: *Both a high-level (i.e., conceptual) view on compliance rules focusing on their semantics and an implementation level view for compliance rule evaluation are essential.*

Compliance measures are usually implemented using procedures, policies, and controls; hard-coded compliance measures are the source of high costs for compliance-aware organizations. As the example in Section 2 has shown, different compliance rules can apply at different levels of abstraction. For instance, it is irrelevant how the

implementation determines driver's working hours but they have to be provided each time for compliance checking of rule CR3. This is especially important for the reuse of compliance rules and requires the mapping from one level to the other; e.g., if we change the implementation underneath.

For example, to serve the mapping between different abstraction levels, we developed a model-driven approach to support the automated derivation of CEP queries from business process models for process monitoring, details are to be found in P4 shown in Table 4. We decompose a process model that includes monitoring information into its structural components which are then transformed to CEP queries to allow the monitoring of business process execution based on events.

Req. 4 – Support for compliance validation: *It is desirable to support the error-prone and time-consuming validation of compliance at the process modeling level, at run-time, in case of process changes and for change propagation to process instances.*

A huge number of existing approach deal with validating the compliance with process models and check non-compliant of executed process instances, see e.g., [13]. This report focuses on the consideration of run-time compliance that enables to avoid non-compliant behavior during execution. To enable this function, events can be processed to identify those events that are violating compliance rules.

The platform that is used for the compliance monitoring framework [3] is based on CEP and may be used to validate compliant behavior of process instances while they are running. For this purpose, we convert compliance rules to anti-patterns – to check for violations. Thus, we assume that a compliance rule holds if we do not find any violations. Similar to Req. 1, anti-patterns are translated to an EPL to check in the event cloud for counterexamples. For example, a violation is immediately communicated if based on a defined event pattern the platform detects that a driver wants to transport dangerous goods without having the required security class, see Section 2.

In this way, EPL-based anti-patterns provide the ability to subscribe to violations of desired executions. The influence of changes in process models on its execution and a decision support to deal with exceptional situations are research directions of our future work.

Req. 5 – Support of process-spanning scenarios: *The scope of compliance rules may reach across multiple processes.*

According to [9], there exists a demand for mechanisms which allow to validate and ensure compliant behavior for business processes independently and depending on their semantic interrelation across process boundaries. As shown in Figure 3, the framework detects non-compliant behavior based on events that are associated by their event type to one or more process models. This allows the validation and ensuring of compliant behavior in and between multiple business processes.

Req. 6 – Providing intelligible feedback: *Helpful feedback proving an error diagnosis and assisting the user in applying adequate conflict avoidance is of high importance for user acceptance.*

The main benefit of the compliance framework is that it can provide a perspective on several value chains, rather than a single process instance. For the GET Service

project this, in turn, will enable important practical applications that are currently not possible, or at least very hard to achieve. Till the end of the project these practical applications will be added to the current implementations and will include (1) effortless use of real-time information about the current status of transportation, (2) monitoring each transportation as it passes through the entire value chain, enabling end-to-end tracking and tracing of the transportation, and (3) real-time replacement of one or more transportation tasks in the value chain by others.

Req. 7 – Support of flexible compliance rule handling: *Compliance rules are often not stringent, must not conflict with the need for flexible processes and therefore they should be overwritable.*

Compliance violations need not necessarily be an error but could also be intended. For example, the driver cannot stop directly on the highway in case her working hours are reached, she better drives to a resting station and rests there. Also possible would be the situation in which the system notifies the driver to make a rest before the working time is over. The latter part will be investigated in the context of predictive maintenance in cooperation with Bosch SI (see P6 in Table 4).

Currently, the CEP platform used for implementing the compliance framework is able to enforce two different reactions to non-compliant behavior. The notifications about events signaling non-compliance can be forwarded as email or solely displayed in the user interface depending on the security level the user defines for an event pattern (or anti-pattern). In future work, we plan to directly integrate one or more process engines to also drive business process execution based on events and, thus, to not only display this behavior but also provide possibilities to prevent non-compliant behavior.

Req. 8 – Support of traceability: *The results of compliance checks have to be documented.*

To reconstruct past compliance checks and corresponding results they have to be documented. This is especially valuable for violations and decisions made in the past as they record who (over)wrote rules and for what reason. This can be the basis to reason about future decisions that have to be made in case of violations. The open event processing platform [3] used to implement the compliance framework is extendable and can, therefore, be extended to store the results and decisions of compliance checks.

5 Conclusion and Future Work

This report presented a context-aware compliance framework for business processes by considering events occurring in distributed systems. Outcome of the research presented in the compliance framework result in one presented and three submitted papers as well as the participation in a EU project called GET Service and the cooperation with Bosch Software Innovations GmbH (Bosch SI), see Table 4. Based on the papers, project results and eight common requirements for checking, monitoring, and enforcing

compliant behavior in organizations, we discussed the framework and its implementations already supported and those challenges that have to be investigated in future work.

Table 4: Current activities and project participation

P1	Towards the Enhancement of Business Process Monitoring for Complex Logistics Chains	Presented PALS13 [4]
P2	Towards Automating the Detection of Event Sources	Accepted WESOA13 [6]
P3	Enabling Semantic Complex Event Processing in the Domain of Logistics	Accepted PASCEB13 [10]
P4	Model-driven Event Query Generation for Business Process Monitoring	Accepted PASCEB13 [2]
P5	Aggregation services for Green European Transportation (GET)	Lead work package
P6	RESEARCH INITIATIVE BOSCH SI AND HPI: Internet Application Platform am Beispiel von Industrie 4.0.	Research & supervision Bachelorproject

References

- [1] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Visually specifying compliance rules and explaining their violations for business processes. *Journal of Visual Languages & Computing*, 22(1):30–55, 2011.
- [2] Michael Backmann, Anne Baumgrass, Nico Herzberg, Andreas Meyer, and Mathias Weske. Model-driven Event Query Generation for Business Process Monitoring. In *1st Workshop on Pervasive Analytical Service Clouds for the Enterprise and Beyond*, 2013. (accepted for publication).
- [3] Susanne Bülow, Michael Backmann, Nico Herzberg, Thomas Hille, Andreas Meyer, Benjamin Ulm, Tsun Yin Wong, and Mathias Weske. Monitoring of Business Processes with Complex Event Processing. In *BPM Workshops*. Springer, 2013.
- [4] Cristina Cabanillas, Anne Baumgrass, Jan Mendling, Patricia Rogetzer, and Bruno Bellovoda. Towards the Enhancement of Business Process Monitoring for Complex Logistics Chains. In *11th International Conference on Business Process Management Workshop on "Process-Aware Logistics Systems"*. Springer, 2013.
- [5] Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés. Hints on how to face business process compliance. In *III Taller de Procesos de Negocio e Ingeniería de Servicios (PNIS'10) in JISBD'10*, volume 4, pages 26–32, 2010.

-
- [6] Nico Herzberg, Oleh Khovalko, Anne Baumgrass, and Mathias Weske. Towards Automating the Detection of Event Sources. In *8th International Workshop on Engineering Service-Oriented Applications*, 2013. (accepted for publication).
- [7] Nico Herzberg, Andreas Meyer, and Mathias Weske. An Event Processing Platform for Business Process Management. In *EDOC*, pages 107–116. IEEE, 2013.
- [8] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.
- [9] Linh Thao Ly, Stefanie Rinderle-Ma, Kevin Göser, and Peter Dadam. On enabling integrated process compliance with semantic constraints in process management systems. *Information Systems Frontiers*, 14(2):195–219, April 2012.
- [10] Tobias Metzke, Andreas Rogge-Solti, Anne Baumgrass, Jan Mendling, and Mathias Weske. Enabling Semantic Complex Event Processing in the Domain of Logistics. In *1st Workshop on Pervasive Analytical Service Clouds for the Enterprise and Beyond*, 2013. (accepted for publication).
- [11] Szabolcs Rozsnyai, Aleksander Slominski, and Geetika T. Lakshmanan. Discovering event correlation rules for semi-structured business processes. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, pages 75–86. ACM, 2011.
- [12] Robert Thullner, Szabolcs Rozsnyai, Josef Schiefer, Hannes Obwegger, and Martin Suntinger. Proactive Business Process Compliance Monitoring with Event-Based Systems. *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*, pages 429–437, August 2011.
- [13] Wil M. P. van der Aalst, Huub T. de Beer, and Boudewijn F. van Dongen. Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In *OTM Confederated International Conferences: CoopIS, DOA and ODBASE*, volume 3760, pages 130–147. Springer-Verlag, October 2005.
- [14] Matthias Weidlich, Holger Ziekow, Jan Mendling, Oliver Günther, Mathias Weske, and Nimit Desai. Event-based monitoring of process execution violations. In *Business Process Management*, volume 6896 of *LNCS*, pages 182–198. Springer Berlin Heidelberg, 2011.
- [15] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag Berlin Heidelberg, 2nd edition, 2012.
- [16] Qunzhi Zhou, Yogesh Simmhan, and Viktor Prasanna. Towards an Inexact Semantic Complex Event Processing Framework. In *Proc. of the 5th ACM International Conference on Distributed Event-based Systems (DEBS)*, pages 401–402, 2011.

Symbolic Representation and Constraint Reasoning in Invariant Checking

Johannes Dyck

System Analysis and Modeling Group
Hasso Plattner Institute
johannes.dyck@hpi.uni-potsdam.de

Tools for formal verification of complex systems often encounter problems of infinite size, making solutions with explicit representations impossible. Even for finite systems, formal verification is often infeasible when expecting an answer in reasonable time due to exponential complexity of the underlying algorithms. This report demonstrates the occurrence of such complexity challenges in our verification tool, which is based on graph transformation systems and inductive invariants. As two orthogonal approaches to such challenges, the report explains symbolic representation for negative application conditions and constraint reasoning. Both techniques have already been seen to significantly reduce the algorithm's runtime for specific examples.

1 Introduction

With systems, whether software or otherwise, growing in size and complexity, automated analysis and verification become more desirable. Invariant checking, which is described extensively in [1] and [5], is such a technique for formal verification. Based on the specification of a system's behavior, it can be used to verify the validity of certain properties important for the system's safety or operability, for example safety or liveness properties.

While classical model checking approaches achieve this goal by exhaustively generating and analyzing the system's state space, invariant checking is only concerned with the analysis of the system's behavior. Instead of generating all reachable states, the technique verifies inductive invariants. An inductive invariant is a property whose validity before a change of the system's state implies its validity after the change. The verification technique is concerned with the transitions between system states rather than the actual—and attainable—states. Since initial states of the system are not considered, a change of the system's initial state does not invalidate the result of the verification algorithm, once obtained.

The approach has been applied in a number of areas, including:

Model transformations and behavior preservation [8].

Consistency-preserving refactorings, where refactorings are described by transformation rules and consistency is based on well-formedness constraints of the respective programming language [4].

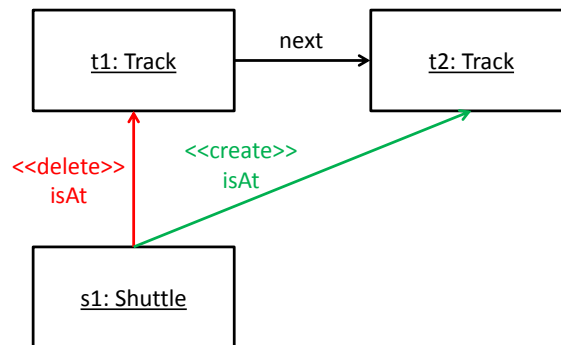


Figure 1: Graph rule

Self-adaptive systems, which (due to their nature) may conduct structural changes, but are also required to ensure certain properties [2].

Service-oriented approaches to coordination of autonomous entities, where collaboration is modeled using graphs and graph transformation systems [3].

Correctness of safety-critical systems with real-time behavior, using graph transformation systems with attributes and timing extensions [3].

While invariant checking with inductive invariants is capable of handling infinite systems in finite time and is not prone to state space explosion as many model checking tools, the algorithm still contains elements with exponential complexity. This has been known as a problem for the verification of large systems, for example extensive model transformations as described in [8]. It is also a challenge to be addressed as part of the intended extension of inductive invariants to k -inductive invariants (similar to the technique described in [13, 14]), taking a path of transitions into account for verification.

2 Foundations and Complexity Challenges

The invariant checking tool in question (which is explained in depth in [1, 5]) is based on graph transformation systems, with graph transformations describing the system's behavior, meaning transitions between states, and graphs describing system states. The underlying theory of graphs and graph transformation systems can be found in [6] and [12]. Figure 1 displays a possible graph rule in an example system of shuttles and tracks (from [3]), with nodes representing shuttles and tracks and edges representing connections between shuttles or tracks, respectively. Red elements will be deleted upon rule application, green elements will be created while all other elements will be preserved. The rule causes a shuttle to move from one track to an adjacent track.

To represent multiple (and possibly infinitely many) graphs, graph patterns are used. A simple graph pattern represents all graphs containing the pattern as a subgraph. For example, Figure 2 shows a graph pattern, representing two shuttle being located on the same track (cf. [1, 3]). The graph in Figure 3(a) contains—or fulfills—the pattern while the graph in Figure 3(b) does not. Such graph patterns are used to describe forbidden

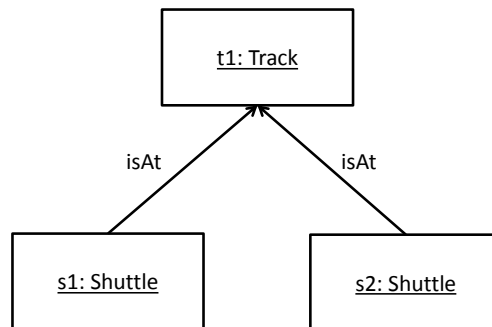
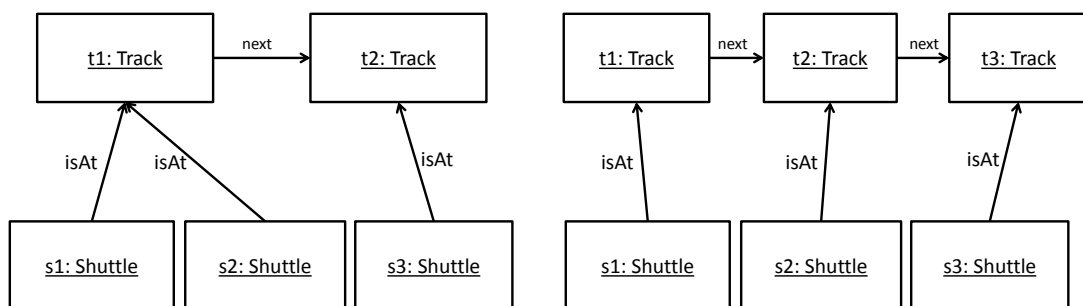


Figure 2: Graph pattern



(a) Graph fulfilling the pattern

(b) Graph not fulfilling the pattern

Figure 3: Patterns and graphs

properties which must not be encountered during system execution. They are also a means to display violations of these properties found by the verification algorithm. Since one graph pattern can represent infinitely many graphs, they offer a means to analyze infinite systems in finite time.

To increase the expressive power of the specification language, graph patterns can be enhanced by negative application conditions (see [9] or [7]). In contrast to the common elements of graph patterns, which demand the presence of nodes and edges of the respective types, negative application conditions require the absence of certain nodes or edges for the pattern to be fulfilled by graphs. Figure 4 shows a graph pattern describing two subsequent tracks, where the first track must not be occupied by a shuttle. Since there are no other restrictions, graphs with shuttles on the second track or with additional tracks still fulfill the graph pattern. For example, the graph in Figure 5(a) fulfills the pattern while the graph in Figure 5(b) does not.

The verification algorithm (see [1, 5] for details) of the invariant checking tool includes the combination of the graph transformation rules, specifying the system's behavior, with the forbidden graph patterns, specifying states that should not occur in the system. All pairs of rules and properties are considered and each such pair may suffer from exponential complexity in two cases:

1. Both rule and forbidden property are merged along all possible common subgraphs (overappings) between each other. With n being the number of nodes and e being the number of edges, the number of subgraphs of a graph is at least

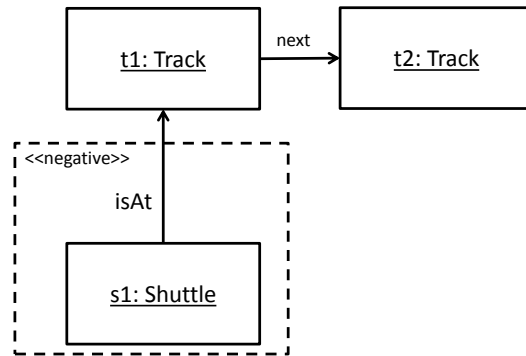


Figure 4: Pattern with negative application condition

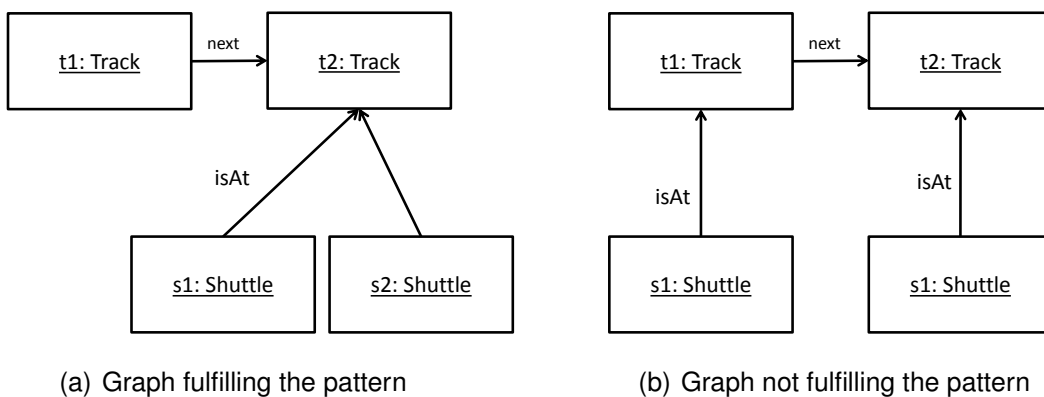


Figure 5: Conditions and graphs

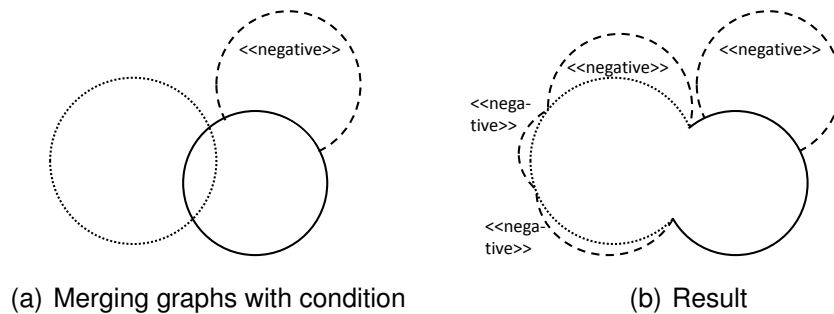


Figure 6: Transformation of conditions

2^n and at most 2^{n+e} . Thus, creating all possible overlappings is a problem of exponential complexity.

2. For a specific merge between rule and property, negative application conditions have to be transferred from the property to the merged graph. This includes the creation of subgraphs of the negative application conditions, which results in the same complexity as above.

The following sections explain ways to circumvent these problems.

3 Symbolic Representation for Application Conditions

When merging graph patterns—such as forbidden properties—with graph transformation rules to analyze the rule’s capability to violate a forbidden property, negative application conditions from the graph pattern will be transferred to the new context, namely the merged graph. Since the pattern’s negative application conditions must still be valid after merging, it must be ensured that the elements required to be absent will not be added in the process of merging the graphs. However, since all possible completions of the negative application condition must be considered, the correct transformation of a negative application condition involves the analysis of all its subgraphs.

Figure 6(a) shows an abstraction of such a transformation, with graphs represented by the circles. To ensure that the condition is valid after the merge, all elements from the dotted graph that may lead to the presence of elements in the negative application condition need to be considered. Depending on the number of nodes and edges of the negative application condition and the target graph, this transformation process can quickly become infeasible for large conditions and will often be the major factor determining the runtime of the invariant checking process. Figure 6(b) shows possible completions of the condition that need to be included as new negative application conditions in the merged graph.

While it is possible to remove negative application conditions entirely, this leads to loss of information often required in later steps of the verification process. This procedure may then lead to false negatives, meaning that actually safe systems may be classified as unsafe. However, it should be noted that this will never lead to an unsafe system being labeled as safe.

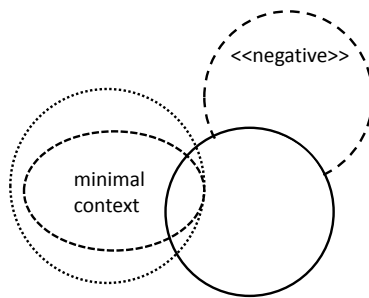


Figure 7: Partial transformation of conditions

The merging of patterns and graphs already creates a symbolic representation for a number of graphs—possibly infinite—thus enabling the technique to handle infinite systems. A similar procedure can be applied to the handling of negative application conditions. Instead of transferring application conditions completely to the new context of the merged graph, they are merely transformed to the minimal context necessary for verification. This minimal context depends on the number of items (nodes or edges) actually changed by the application of the graph rule in question.

The abstract example in Figure 7 illustrates this approach. While exponential complexity cannot be avoided, the number of nodes and edges to be considered for sub-graph generation will be reduced, as there are usually fewer matching counterparts (and fewer elements altogether) in the minimal context.

More formally, negative application conditions usually rely on total graph morphisms, which are functions describing a mapping between two graphs (see [6] and [7]). In the case of graphs merged along a common subgraph as described above, the use of total morphisms would require the extension of the negative application condition from one graph pattern to all elements of the newly created graph. Instead, the symbolic representation shown above makes use of partial morphisms (see, for example, [11]), which do not require the extension of the condition to the complete graph.

Unfortunately, the graph patterns thusly created must often be compared with other patterns at a later stage in the verification process. In some cases, such a comparison still requires a transformation of negative application conditions to a greater context as avoided by the approach described above. However, these extensions seldom take the dimension of the transformations required without the symbolic representation of negative application conditions. More precisely, these comparisons require the extension of conditions to the union of the graphs' minimal contexts to be compared.

4 Constraint Reasoning

Large graph patterns are often impractical to verify, as the merge process involves the creation of subgraphs, whose number grows exponentially with the number of nodes in a graph. In addition, the time needed for the transformation of negative application conditions can still be problematic if the symbolic representation as demonstrated above requires a large part of the condition to be translated (which depends on the rule in

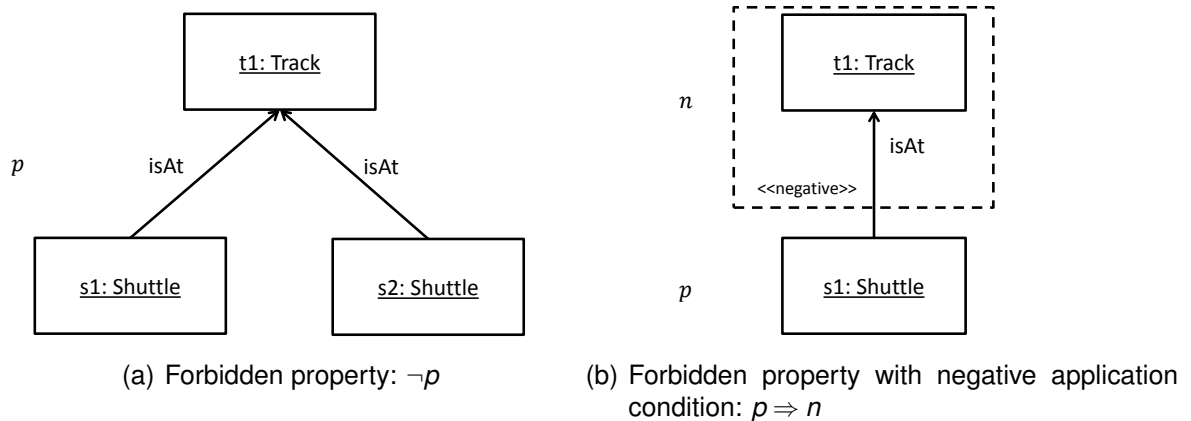


Figure 8: Forbidden properties

question). Consequently, it is desirable to avoid directly analyzing all possible overlaps along common subgraphs between large patterns and rules. This is especially important if the pattern and rule in question are similar, as this will drastically increase the number of common subgraphs.

Usually, when checking a property's validity for a graph transformation system, there are other properties already verified as inductive invariants. In other cases, there exist properties whose validity is guaranteed by the topology of the system or by other additional information available to the user or the tool. This knowledge and such intermediary results can then be used to conclude the validity of other properties for the system in the form of a logical implication: If $P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow P$ can be shown for verified (or guaranteed) properties P_1 to P_n and the property P , the invariant checking algorithm does not need to separately verify P by combining it with the system's transformation rules. Reasoning that the validity of a property or constraint follows from other properties in this way avoids the possible combinatorial explosion following the combination of large or similar patterns and rules.

In the context of the invariant checking tool, there are two types of constraints available to describe system properties: Forbidden properties and forbidden properties with negative application conditions. While the former type only states the absence of a certain subgraph ($\neg p$), the latter kind demands the absence of a subgraph (p) unless the elements specified by one of the negative application conditions are also present (n). This is logically equivalent to an implication $p \Rightarrow n$. In addition, the constraint to be proven by the procedure also needs to have the form of a forbidden property with negative application conditions.

Figure 8(a) shows a forbidden property without a negative application condition. Here, the existence of two shuttles on the same track is unconditionally forbidden. With p being the graph consisting of the track and both shuttles, this is logically described as $\neg p$. Conversely, Figure 8(b) shows a forbidden property with a negative application condition. There, the existence of a shuttle is forbidden unless it is positioned on a track (as a shuttle without a track to stand on does not make sense). Logically, this corresponds to: p (the shuttle) implies n (the track). The original shuttle system example is explained in more detail in [3] and [1].

More expressive (and more general) algorithms to deal with constraint reasoning and theorem proving can be found in [10] and [11]. The approach presented herein is restricted to the subset of graph transformation systems and application conditions used in the invariant checking tool. While not as expressive as other techniques, this approach allows a number of optimizations based on certain restrictions on the formalism's expressive power.

This technique has already been successfully applied to a more complex case of the verification of model transformations, similar to [8]. A simplified version of the constraint reasoning process is shown in algorithm 1.

```
Data: Verified constraints:  $p'_i \Rightarrow n'_i$   
Data: Constraint to be proven:  $p \Rightarrow n$   
Result: true, if the constraint can be proven; false otherwise  
 $c = p$ ;  
while verified constraints remaining do  
   $p' \Rightarrow n' =$  next verified constraint;  
  for all possible subsets  $p' \subseteq c$  do  
    add  $n'$  to  $c$ ;  
    if  $n \subseteq c$  then  
      return true;  
    end  
  end  
end  
return false;
```

Algorithm 1: Constraint reasoning

Unfortunately, termination of this algorithm cannot be guaranteed. The addition of elements to the constraint to be proven can, in theory, lead to the applicability of other constraints, or even the same constraint. More precisely, if a constraint $p_i \Rightarrow n_i$ adds information n_i such that $p_i \subseteq n_i$, this constraint can be used to generate context infinitely. The identification of properties of constraints that guarantee termination is the subject of ongoing work.

5 Conclusion and Outlook

While the use of symbolic representation for negative application conditions and constraint reasoning for large graph patterns has enabled the verification of problems that have been infeasible before, their introduction opens a number of questions to be considered.

5.1 Symbolic Representation

With respect to the representation of negative application conditions by partial graph morphisms, which avoids the necessity of completely expanding conditions, the following aspects still need to be investigated and discussed.

- Aside from specific examples, is it possible to estimate the technique's impact given the number and size of rules, patterns and their negative application conditions?
- Is it possible to further improve performance by deferring the transformation of negative application conditions as long as possible, for example until comparison of graph patterns is required?
- Is it possible to avoid the transformation of negative application conditions to a minimal context altogether when comparing graph patterns?
- If the expressiveness of the specification language for rules and properties is increased—for example to include nested conditions (see [7])—what changes have to be done to adjust the current technique?
- Is it possible to apply the concept of symbolic representation for negative application conditions to symbolic representation for the application of transformation rules and could this be exploited to handle complexity in k -induction?

Furthermore, formal justification and proof of the approach will be provided in further publications to ensure formal correctness of the invariant checking algorithm.

5.2 Constraint Reasoning

As with symbolic representation, the introduction of constraint reasoning raises the questions of impact analysis, changes required upon increasing expressive power and its capabilities in the long-term goal of extending the approach to invariant checking with k -inductive invariants. Also, future work will include a detailed formal proof. In addition, termination of the constraint reasoning algorithm will be analyzed. While the underlying problem is, in general, undecidable (cf. [11]), termination can probably be guaranteed for graph patterns with specific properties. The identification and analysis of these properties will be part of future work.

Lastly, it may be possible to combine constraint reasoning with the existing algorithm for verification to exploit parallel execution of both algorithms. If the constraint reasoning algorithm yields a positive result, the processing of the invariant checking algorithm can be aborted. Conversely, if the constraint reasoning does not return a result in acceptable time or does return a negative result, the invariant checking algorithm will continue to analyze the respective pattern.

References

- [1] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proc. of the 28th International Conference on Software Engineering (ICSE)*, Shanghai, China. ACM Press, 0 2006.

- [2] Basil Becker and Holger Giese. Modeling of Correct Self-Adaptive Systems: A Graph Transformation System Based Approach. In *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology, CSTST '08*, pages 508–516, New York, NY, USA, 2008. ACM.
- [3] Basil Becker and Holger Giese. On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles. In *In Proc. of 11th International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 203–210. IEEE Computer Society Press, 5 2008.
- [4] Basil Becker, Leen Lambers, Johannes Dyck, Stefanie Birth, and Holger Giese. Iterative Development of Consistency-Preserving Rule-Based Refactorings. In Jordi Cabot and Eelco Visser, editors, *Theory and Practice of Model Transformations, Fourth International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*, volume 6707 of *Lecture Notes in Computer Science*, pages 123–137. Springer / Heidelberg, 0 2011.
- [5] Johannes Dyck. Increasing expressive power of graph rules and conditions and automatic verification with inductive invariants. Master's thesis, Hasso Plattner Institute, University of Potsdam, 2012.
- [6] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [7] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. M-Adhesive Transformation Systems with Nested Application Conditions, Part 1: Parallelism, Concurrency and Amalgamation. *Mathematical Structures in Computer Science*, 0 2012. to appear.
- [8] Holger Giese and Leen Lambers. Towards Automatic Verification of Behavior Preservation for Model Transformation via Invariant Checking. In *Proceedings of International Conference on Graph Transformation (ICGT'12)*, volume 7562 of *LNCS*, pages 249–263. Springer, 2012.
- [9] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26:287–313, 1995.
- [10] Karl-Heinz Pennemann. Resolution-like theorem proving for high-level conditions. In *Graph Transformations (ICGT'08)*, volume 5214 of *Lecture Notes in Computer Science*, pages 289–304. Springer-Verlag, 2008.
- [11] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg, 2009.
- [12] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

- [13] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In Warren Hunt and Steven Johnson, editors, *Formal Methods in Computer-Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin / Heidelberg, 2000.
- [14] Max Thalmaier, Minh D. Nguyen, Markus Wedler, Dominik Stoffel, Jörg Bormann, and Wolfgang Kunz. Analyzing k-step induction to compute invariants for sat-based property checking. In *Proceedings of the 47th Design Automation Conference, DAC '10*, pages 176–181, New York, NY, USA, 2010. ACM.

Solving Multidomain Constraints on Object Behavior

Tim Felgentreff

Software Architecture Group
Hasso-Plattner-Institut
tim.felgentreff@hpi.uni-potsdam.de

Constraints allow developers to specify desired properties of a system. These properties are then maintained automatically rather than with scattered checks that imperatively re-satisfy the constraints. This results in compact, localized code to specify system invariants. However, despite expectations that this improves comprehensibility and maintainability, as well as possible performance advantages, constraint programming is not yet widespread, with standard imperative programming still the norm.

We propose an extension to existing imperative, object-oriented languages that unifies the methods of encapsulation and abstraction for both the declarative and the imperative paradigm. Our current research addresses integration of these different paradigms into one another and how this integration can be useful for in a variety of application domains.

1 Introduction

Constraint programming has a long history of research interest starting with the Sketch-Pad [30] system. As a paradigm for general purpose programming, the CLP(\mathcal{R}) [16] and Kaleidoscope [19] systems provided integration of constraints with logic programming and objects, respectively. A *constraint* here is a relation that should hold, for example: that there be a minimum of 10 pixels horizontal space between two buttons on a screen, that a resistor in an electrical circuit simulation obey Ohm's Law, that a maximum of 10 parts per hour can be produced by a machine in a factory. Constraints are declarative: they specify *what* should be the case rather than *how* to achieve it.

Recently, there has been renewed interest in using constraints through libraries [25] or DSLs such as the in the Mac OS X layout system [1]. Constraint programming allows developers to precisely express desired properties of a system and have those properties be automatically maintained by the runtime. This enables concise code, while avoiding scattered code that trigger checks and re-satisfaction of those properties.

However, existing approaches retain a number of key issues that need to be addressed if constraints in imperative programs should become more widespread:

- a) The declarative and imperative paradigms should not be separate, with parallel methods of encapsulation and abstraction, which cannot be freely mixed. This impedes re-use and comprehensibility of modules, as clients can only use modules in the paradigm for which the implementers provide the appropriate abstractions and implementers have to be fluent in both paradigms.

- b) Performance for imperative code should be as good as in a purely imperative runtime if the declarative features are not used.
- c) Constraint solving is undecidable in the general case [24]. To provide good performance, most solvers only work in a limited number of type domains (such as reals, booleans, integers) and under certain restrictions (such as linearity). Without interactions between the solvers, problems that span type domains cannot be solved, limiting the generality of the approach.

In this work, we present a language – BABELSBERG – that builds on the ideas from the Kaleidoscope system and other constraint-imperative programming (CIP) languages. We attempt to address the above issues in the following ways:

- a) BABELSBERG unifies the methods of encapsulation and abstraction for both the declarative and the imperative paradigm. In BABELSBERG, constraints restrict object behavior by constraining the results of messages send to them. A common syntax and a declarative semantics for imperative constructs allow programmers to add constraints to existing object-oriented (OO) programs in incremental steps, without having to learn a completely new paradigm.
- b) The performance and semantics of purely OO code is unaffected in BABELSBERG, as we show with our prototype implementations in state-of-the-art Ruby virtual machine (VM) [10] and on top of the Lively Kernel [18].
- c) We implement an architecture for interactions between imperative execution and different constraint solvers for different domains that allows constraints to be declared and solved for multidomain problems.

Thus, our current research focuses on the following questions:

- What imperative, OO constructs (such as message sends, system calls, destructive assignments, or explicit parallelization) are useful in constraint expressions and to find a declarative semantics for them.
- Which subset of constraint solver features (such as constraint hierarchies or edit constraints for interactive use) are required to solve a variety of problems and what interactions between solvers are required to enable these features.

The rest of this report is structured as follows: section 2 explains the related work using a running example, section 3 presents our work on BABELSBERG and its prototypes BABELSBERG/R and BABELSBERG/JS, and section 4 presents our next steps and concludes.

2 State of the Art

Consider a rectangle implemented as a pair of points as in Listing 1. This rectangle is displayed in an application window which the user can resize. Suppose this rectangle encompasses some information that we want to make sure remains visible. We want

to make sure the area of the rectangle is never less than 100 square pixels and that its origin is always within display bounds, i.e. positive.

```
class Rectangle
  attr_accessor :origin, :extent

  def visible?
    origin.x >= 0 and origin.y >= 0
  end

  def area
    extent.x * extent.y
  end
end
```

Listing 1: A rectangle implemented as a pair of points, with a predicate to test it starts within display bounds and a method to calculate the area

Imperatively, we can use, for example, aspects to satisfy these constraints explicitly whenever the rectangle changes:

```
class RectAspect < Aspect
  def ensure_constraints(method, rect, status, *args)
    rect.origin.x = 0 if rect.origin.x <= 0
    rect.origin.y = 0 if rect.origin.y <= 0
    rect.extent.x = 100.0 / rect.extent.y if rect.area <= 100
  end
end
```

Notice that we had to transform the constraints into conditional branches and assignments. There are multiple solutions to these constraints, but which one is selected is only implicit in the code. Since there is no declarative specification of an optimal solution, it is not trivial to tell the rate the solution. If we want to add more constraints that may possibly conflict, i.e., multiobjective optimizations, this becomes even harder.

Thus, it is usually clearer to express and satisfy the constraints explicitly. In the remainder of this section, we compare how state of the art approaches solve constraints in imperative programs. We present the approaches four groups: 1. constraint solver libraries, 2. constraint solver domain specific languages (DSLs), 3. *DataFlow* and functional-reactive programming (FRP) languages, and 4. constraint-imperative programming that combines constraints and imperative statements in one language.

2.1 Constraint Solver Libraries

There is a large number of solvers available as libraries that can be called directly from imperative code. A few solvers of particular note in the programming language community are Z3 [7], an SMT solver designed for theorem proving (e.g., for program verification), and kodkod [31] for constraints over finite domains. Solvers for use in interactive graphics systems include Cassowary [2], an incremental solver for linear equality and inequality constraints that supports soft constraints [6] as well as hard ones, the Auckland Layout Editor [23], which includes support for a GUI builder using constraints, and DeltaBlue [13], a multi-way local propagation solver that also supports hard and soft constraints.

The following code adapts the aspect-oriented solution to use the Z3 constraint solver to solve our constraints:

```
class RectAspect < Aspect
  def ensure_constraints(method, rect, status, *args)
    ctx = Z3::Context.new
    ctx << Z3::Variable.new("extent_x", rect.extent.x)
    ctx << Z3::Variable.new("extent_y", rect.extent.y)
    ctx << Z3::Constraint.new("extent_x * extent_y >= 100")
    # ... same for origin constraint
    ctx.solve
    rect.extent.x = ctx["extent_x"]
    rect.extent.y = ctx["extent_y"]
  end
end
```

The constraint code can be clearly written in a domain the solver understands (i.e., reals). However, the programmer is left to write the boilerplate code to decompose objects (breaking object encapsulation) and transform OO values into constraint variables.

2.2 Domain-specific Languages for Constraints

For specialized domains such as user interface layout, constraints are sometimes available as separate DSLs that describe relations between visible objects that can be automatically maintained by the runtime. Examples of such DSLs are CSS [17], the *Mac OS X* [1] layout specification language, and the Python GUI framework *Enaml* [9]. These constraints are automatically re-satisfied by the runtime when imperative code changes the user interface.

The following listing shows the *Enaml* specification for our problem:

```
enameldef Main(Window):
  Container:
    constraints = [
      # a rectangle's area is exposed as 'content' in Enaml
      content_left >= 0, content_top >= 0,
      (content_right - content_left) *
      (content_bottom - content_top) >= 100
    ]
```

This approach allows programmers to specify constraints and avoid boilerplate code to trigger constraint solving and has found widespread adoption, particularly through the Mac OS X layout system. However, to use these constraints programmers use a separate language that works only for predetermined types (i.e., graphical objects).

2.3 DataFlow Constraints and FRP

Some languages have built-in support for data flow, which allows programmers to express unidirectional constraints between objects and their parts. Examples of such systems are Scratch [27], LivelyKernel/Webwerkstatt [18], and KScript [26].

The following code uses LivelyKernel connections to react to changes in the `origin` and `extent` of a `Rectangle`.


```

connect(rect, "origin", rect, "origin",
  function(origin, prevOrigin) {
    if (this.isVisible()) return prevOrigin;
    else return origin;
  })
connect(rect, "extent", rect, "extent",
  function(extent, prevExtent) {
    if (this.area() <= 100) return prevExtent;
    else return extent;
  })

```

Although these systems are not constraint solvers, programmers can use constraint solvers (in the hook function passed to `connect`) to calculate new values. KScript already integrates a constraint solver to use in the connection. FRP approaches provide one answer to the question of when to trigger constraint solving and provide a convenient imperative API, but still require the programmers and convert between OO values and constraint variables.

2.4 Constraint-Imperative Programming

Our goal is to support a more standard imperative, OO programming style and syntactic integration of constraint and imperative programming. With these goals, BABELSBERG follows the work on CIP [12, 20–22] and the Kaleidoscope language. Systems related to Kaleidoscope include Siri [15], Turtle [14], and SOUL [8]. BackTalk [29] is another system that aims to integrate a rich set of constraint solvers with imperative languages, but without syntactic integration.

Kaleidoscope supported standard classes and instances, and in addition, integrated constraints with the language itself. To support this, it included built-in constraints over primitive objects (such as floats) and constraints over user-defined objects, which were provided by *constraint constructors*. For example, the `+` constraint for Points could be defined using a constraint constructor `a+b=c` that then expanded this into constraints on the `x` and `y` instance variables. Separately, the language also provided methods.

This is our example in CIP:

```

class Rectangle
  constructor area = (n: Integer)
    always: extent.x * extent.y = n
  end

  constructor visible?
    always: origin.x >= 0
    always: origin.y >= 0
  end
end

rect = Rectangle.new
always: rect.area = 100
always: rect.visible?

```

The above code uses *constraint constructors* to encapsulate the calculated properties visibility and area so they can be used in constraints. However, to test visibility both in constraints and in imperative code, developers have to duplicate definitions to provide both methods and constraint constructors.

3 BABELSBERG

BABELSBERG is an object-constraint programming (OCP) language; the term *object-constraint programming* is chosen to emphasize the integration with standard object-oriented programming ideas, in particular methods, messages, and object encapsulation.

Our first prototype is an extension of the Ruby programming language [11] and is consequently called BABELSBERG/R. To verify the applicability of our approach to other object-oriented languages, we have additionally created a hosted implementation of this concept on top of the JavaScript environment Lively Kernel [18].

The Ruby VM we used as a basis for BABELSBERG/R is *Topaz* [10], an experimental VM built using the *PyPy/RPython* toolchain [28]. This has allowed us to extend the interpreter and use RPython's VM-generation toolchain to create a VM including a fast just-in-time (JIT) and garbage collector.

In BABELSBERG/R, we can express the above problem using the methods already defined for the Rectangle class:

```
rect = Rectangle.new
always { rect.area == 100 }
always { rect.visible? }
```

The first constraint says that the result returned from calling the `area` method should always be greater than or equal to 100, and if, for example, another part of the program assigns to the height of the rectangle, if necessary the width will be adjusted automatically to keep the constraint satisfied. Similarly, if a negative location is assigned to the origin, it will be moved back to keep the rectangle visible.¹

By placing the constraint on the result of sending messages rather than on fields, the system respects object encapsulation. The values returned from the message sends in the rectangle example are both primitive types (float and boolean), but they can also be arbitrary objects. For example, we could add a constraint on the rectangle's center (a computed rather than a stored value, and a point rather than a primitive type):

```
always { rect.center == Point.new(100,100) }
```

OCP keeps desirable properties from other approaches and is, for the most part, a continuation of CIP. Table 1 shows these properties and compares OCP to the approaches presented in section section 2.

Unified Language Constructs Programs in BABELSBERG appear as ordinary OO programs if no constraints are used, but can be easily adapted to use constraints where it makes sense. If constraints are used, they respect encapsulation and re-use the object-oriented method definitions. Furthermore, techniques such as inheritance and dynamic typing operate correctly with constraints.

¹There are multiple possible locations that satisfy the `rect.visible?` constraint; here the system will move the origin as little as possible from the assigned location but so that the constraint is satisfied. The same holds for the area constraint. This behavior is a result of soft “stay” constraints that specify that, if it is necessary to change the value of a variable to satisfy other constraints, it should be changed as little as possible. These are left implicit in this example, but can also be stated explicitly if desired.

	Libraries	DSLs	FRP	CIP	OCF
Unified Language Constructs	○	○	○	◐	●
Automatic Solving	○	●	●	●	●
Linguistic Symbiosis	○	○	●	●	●
Exchangeable Solvers	●	●	◐	◐	●
Suitably Expressive Constraints	●	◐	○	●	●
Performant Pure-OO code	●	●	●	○	●
Interactions between Solvers	◐	○	○	○	●

Table 1: Comparison of our OCF approach implemented in BABELSBERG with related work

In contrast, library and DSL based approaches separate constraints from imperative code through a different syntax and semantics. For example, FRP and CIP languages use *propagation hooks* and *constraint constructors* respectively to support constraints.

Automatic Solving Using libraries for constraint satisfaction allows programmers to write code that (intentionally or unintentionally) circumvents previously asserted constraints. Approaches that integrate constraints at a language level do not allow such circumvention, and attempt to re-satisfy constraints whenever they are violated during program execution.

Linguistic Symbiosis D’Hondt et al. [8] argue that linguistic symbiosis between different programming paradigms is required to support the evolution of programs from the object-oriented paradigm to a constraint-oriented solution and vice versa. DSL and library based approaches do not support such incremental refactoring between paradigms as well as approaches in which constraints are written in the host language.

Exchangeable Solvers Libraries provide the most flexibility for choosing different solvers depending on programmer needs. FRP languages can, to some extent, be combined with solver libraries to achieve a comparable flexibility. CIP languages also provide a more controlled way for developers to use different solvers by writing constraint constructors that reformulate constraints using a different solver.

In BABELSBERG, all solvers use the same interface to communicate with the VM so developers can add new solvers and replace existing ones to support new type domains, or to use solvers that give better results or performance for a particular problem.

Suitably Expressive Constraints To take advantage of the constraint paradigm, the language should allow a rich set of constraints to be written and solved. BABELSBERG provides a number of features in this regard:

Read-only Variables A read-only variable can only be changed by other solvers upstream of the constraint with the read-only variable, or imperatively, but not to satisfy the constraint in which it occurs [5].

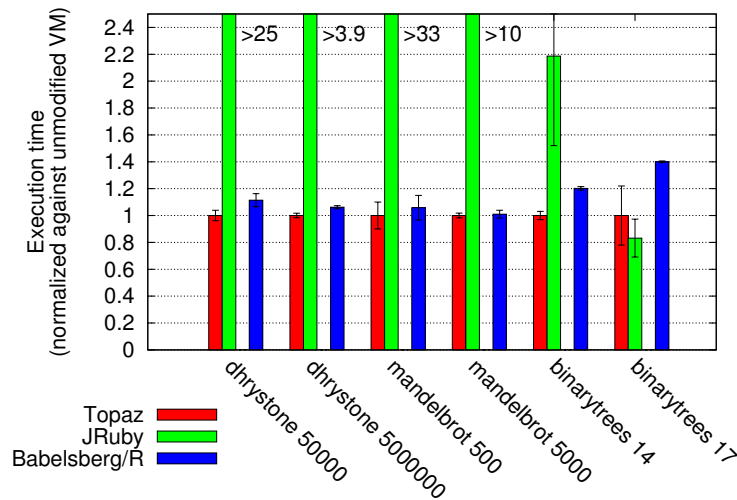


Figure 1: Metatracing VM benchmark run on an Intel i7 Quad-Core CPU with 3.4 GHz

Incremental Solving Interactive graphical applications involve repeatedly re-satisfying the same constraints with differing inputs. A number of solvers support this efficiently using so-called *edit constraints*.

Constraint Hierarchies Constraint hierarchies allow trading off multiple constraints to optimize for multiple objectives [5].

Stay Constraints A desire that value remain the same if possible is represented as a *stay constraint*.

Identity Constraints We can also write constraints on properties of objects such as their identity, class, and the messages that it responds to.

Constraint Durations Constraints have *durations* during which they are active. Besides *always*, which declares that its constraint remains active indefinitely, a *once* constraint is activated, satisfied, and then retracted, whereas an *assert-during* constraint is active for the duration of the evaluation of its associated block.

However, more experience is needed to test whether these features suffice, and to adjust it as needed; and as noted in section 4, an important direction will be adding better support for debugging, explanation, and benchmarking.

Performant Pure OO Code Kaleidoscope provided a declarative semantics for assignment, type declaration, and subclassing. However, this declarative semantics was also used if no actual constraints are in the program. Our implementation approach in BABELSBERG/R uses different execution contexts for constraint construction/solving and imperative code.

To measure OO code performance, we ran a number of tests from the metatracing VMs experiment [3] against the unmodified *Topaz* Ruby VM and the *JRuby* VM (Figure 1). Due to its state of the art JIT, BABELSBERG/R is generally around 10% slower than Topaz (or less than 20% including standard deviation). The only benchmark where

we are doing significantly worse than Topaz is *Binarytrees*. *Binarytrees* is a strongly recursive benchmark, which our JIT is bad at optimizing.

Interactions between Solvers We implement an architecture for cooperating solvers that allows constraints to be declared and solved for multidomain problems [4]. All presented approaches can solve problems in multiple type domains using multiple solvers, but interactions between those solvers are required for data flow between type domains. The flexibility of using constraint libraries directly allows programmers to apply this architecture, too, with additional boilerplate code.

4 Conclusions and Next Steps

We have presented BABELSBERG, an object constraint language that extends a standard object-oriented language to support constraints, along with an implementation as an extension to Ruby using a state of the art virtual machine.

In contrast to other approaches, BABELSBERG unifies the constructs for encapsulation and abstraction for both the declarative constraint parts of the language and the traditional imperative parts by using only object-oriented method definitions for both declarative and imperative code. Our implementation is integrated with an existing object-oriented virtual machine and provides full performance for imperative evaluation. It offers a selection of features from multiple constraint solvers to solve a useful variety of problems. Although our initial implementation extended a Ruby VM, the ideas are applicable to other dynamic object-oriented languages, as our implementation in JavaScript without VM support shows.

This work serves as a basis for a number of questions we want to answer in future work. One is to show that our approach is general and usable by applying the system in a wide variety of application domains, and also to work on improving the performance of the constraint evaluation and satisfaction. Another direction is to find which additional solvers should be included in the library, for example a finite domain solver or solvers that support constraints on other primitive storage types such as arrays, strings, and hashes. Furthermore, we want to continue to implement a design for cooperating solvers, to not only solve constraints in a variety of domains, but also have data flow and features such as incremental solving between domains. Yet another direction regarding solvers is to introduce an oracle that can automatically select one or more applicable solvers for a given set of constraints. This could be a “meta-solver” that trades off features of different solvers, such as performance or stability. Finally, another important question is what support for debugging, explanation, and benchmarking is required and how to provide it. Currently, if the constraint solver is unable to satisfy the constraints, there is no indication of why this is. If the solver produces an unexpected answer, it is unclear how this answer was arrived at. Or if one solver is slow, another may be more appropriate for the problem.

Despite these open questions, BABELSBERG/R and BABELSBERG/JS are already useful in existing applications.

References

- [1] Apple Inc. *Cocoa Auto Layout Guide*, September 2012.
- [2] Greg J. Badros, Alan Borning, and Peter J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4):267–306, 2001.
- [3] Carl Friedrich Bolz and Laurence Tratt. The impact of meta-tracing on VM design and implementation. *Science of Computer Programming*, 2013.
- [4] Alan Borning. Architectures for cooperating constraint solvers. Technical report, VPRI, 2012.
- [5] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, September 1992.
- [6] Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In *Proceedings of the Sixth International Conference on Logic Programming*, pages 149–164, Lisbon, June 1989.
- [7] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
- [8] M. D’Hondt, K. Gybels, and V. Jonckers. Seamless integration of rule-based knowledge and object-oriented functionality with linguistic symbiosis. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1328–1335. ACM, 2004.
- [9] Enthought Inc. Enaml 0.6.3 documentation, 2013.
- [10] Tim Felgentreff. Ruby Topaz. Presented at wroc_love.rb 2013, Wrocław, Poland, March 2013.
- [11] David Flanagan and Yukihiro Matsumoto. *The ruby programming language*. O’Reilly, 2008.
- [12] Bjorn Freeman-Benson and Alan Borning. Integrating constraints with an object-oriented language. In *Proceedings of the 1992 European Conference on Object-Oriented Programming*, pages 268–286, June 1992.
- [13] Bjorn Freeman-Benson and John Maloney. The DeltaBlue algorithm: An incremental constraint hierarchy solver. In *Proceedings of the Eighth Annual IEEE Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, March 1989. IEEE.
- [14] Martin Grabmüller and Petra Hofstedt. Turtle: A constraint imperative programming language. In *Research and Development in Intelligent Systems XX*, pages 185–198. Springer, 2004.

-
- [15] Bruce Horn. Constraint patterns as a basis for object-oriented constraint programming. In *Proceedings of the 1992 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 218–233, Vancouver, British Columbia, October 1992.
- [16] Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap. The CLP(\mathcal{R}) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.
- [17] Håkon Wium Lie and Bert Bos. *Cascading style sheets: Designing for the web*, 1997.
- [18] Jens Lincke, Robert Krahn, Dan Ingalls, Marko Roder, and Robert Hirschfeld. The lively partsbin—a cloud-based repository for collaborative development of active web content. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 693–701. IEEE, 2012.
- [19] Gus Lopez. *The Design and Implementation of Kaleidoscope, A Constraint Imperative Programming Language*. PhD thesis, University of Washington, Department of Computer Science and Engineering, April 1997. Published as Department of Computer Science and Engineering Technical Report 97-04-08.
- [20] Gus Lopez, Bjorn Freeman-Benson, and Alan Borning. Constraints and object identity. In *Proceedings of the 1994 European Conference on Object-Oriented Programming*, pages 260–279, July 1994.
- [21] Gus Lopez, Bjorn Freeman-Benson, and Alan Borning. Implementing constraint imperative programming languages: The Kaleidoscope’93 virtual machine. In *Proceedings of the 1994 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 259–271, October 1994.
- [22] Gus Lopez, Bjorn Freeman-Benson, and Alan Borning. Kaleidoscope: A constraint imperative programming language. In Brian Mayoh, Enn Tyugu, and Jaan Penjam, editors, *Constraint Programming*. Springer-Verlag, 1994. NATO Advanced Science Institute Series, Series F: Computer and System Sciences, Vol. 131. Also published as UW CSE Technical Report 93-09-04.
- [23] Christof Lutteroth and Gerald Weber. End-user GUI customization. In *Proceedings of the 9th ACM SIGCHI New Zealand Chapter’s International Conference on Human-Computer Interaction: Design Centered HCI*, pages 1–8. ACM, 2008.
- [24] Kim Marriott and Peter Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.
- [25] Aleksandar Milicevic, Derek Rayside, Kuat Yessenov, and Daniel Jackson. Unifying execution of imperative and declarative code. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 511–520. ACM, 2011.
- [26] Yoshiki Ohshima, Bert Freudenberg, Aran Lunzer, and Ted Kaehler. A report on KScript and KSWorld. *VPRI Research Note 2012-008*, 2012.

- [27] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
- [28] Armin Rigo and Samuele Pedroni. Pypy’s approach to virtual machine construction. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 944–953. ACM, 2006.
- [29] P. Roy and F. Pachtet. Reifying constraint satisfaction in Smalltalk. *JOOP*, 10(4):43–51, 1997.
- [30] Ivan Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, pages 329–346. IFIPS, 1963.
- [31] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 632–647. Springer, 2007.

Studying the Nature of MDE Evolution – Case Studies

Regina Hebig

System Analysis and Modeling Group
Hasso-Plattner-Institut
regina.hebig@hpi.uni-potsdam.de

In the previous retreat report it was shown that structural changes and substantial structural changes occur commonly in practice. However, there is still a lack of knowledge about structural evolution and motivations that drive structural evolution. To address these issues, we extended our descriptive and exploratory field study to also capture information about the evolution history of MDE settings. This report is an extended short version of [1].

1 Study Design

Since the third study has a similar character and partly similar goals as the first study, the design of this first study was mainly adopted. Following, it is described how the design of the third study was adapted to fit the new needs.

1.1 Conceptual Context

The conceptual context remained mainly the same. At the start it seemed that the topic of the evolution of MDE settings might become to an additional challenge for the communication. Fortunately, it turned out that it is relatively easy to inquire the evolution history by asking question like: „Did you always use `tool X`?” or „How did you develop `artifact Y` before `transformation Z` was introduced?”.

1.2 Choice of Cases

Since it was a goal to collect MDE settings from different companies, the process of choosing case studies had to be adapted completely. First, it was a challenge to overcome the obstacle that most companies are very cautious when it is about providing detailed data to external researchers (as it is necessary for case studies). This problem was approached in this study by contacting alumni students of the Hasso Plattner Institute and personal contacts in different companies. The request was accompanied with a short description of the project and of the data that will be captured. Eventually the contacted persons passed the request on to other projects. When the persons that are in charge for such a project where interested in the study they answered the request.

All in all this led to responses from six projects (from five companies). Unfortunately, in two companies the management did not agree with the participation due to confidentiality reasons. However, three companies agreed to participate in the study:

Capgemini (2 projects with 3 MDE settings), VCat, and Carmeq. In addition, one of the contact persons of the first study at SAP agreed to resume the participation in the study and helped to document the evolution history to one of the already captured MDE settings. Finally, it was possible to document a single evolution step of an MDE setting from Ableton. However, in this case the corresponding MDE setting was not captured in detail.

1.3 Research method

As in the first study it was the goal to collect detailed case studies. Similarly interviews were used.

To address the new issues the method of eliciting the MDE settings was changed. Thereby the rounds of feedback that were performed per email were substituting with a third interview. Further, the MDE settings were directly modeled using Software Manufacture Models. In addition, new questions on how the MDE settings evolved over time were included. This includes questions for motivations and triggers for the captured evolution steps. All captured evolution steps were planned and/or performed before they were captured for the study. As result models from different historical versions of the MDE settings together with records from the interviews are captured.

The more direct form of communication in this adapted research method, together with the direct use of Software Manufacture Models, and the experience the interviewer gained during the first study, reduced the effort for elicitation of an MDE setting to five days of work per MDE setting.

1.4 Analysis of Data

The records from the interviews were systematically sighted and coded following the constant comparison method described in [2]. At the start a set of preformed codes was used. These codes referred to the motivation for an evolution step, the institution or role that triggered the evolution step, and the institution or role that implemented the evolution step. During the sighting of the records codes were added when necessary (e.g. for external influences on the evolution). Based on these codes it was possible to derived several observations.

The collected quantitative data (i.e. the models) were analyzed similar to the models from the first study (except that the models already were Software Manufacture Models). Further, the collected information about the evolution history was analyzed. Thereby it was extracted what evolution steps happened. These evolution steps were further categorized according to the change types presented in [1].

1.5 Threats to Validity

It is always difficult to draw general conclusions from a few case studies. Thus, a broader set of data that captures more domains of software engineering and different companies would be helpful to further substantiate the outcomes of this study. Despite the small number of case studies, we are lucky that indeed different companies are under study. All observations presented here are based on at least two of the case

studies, which is adequate for this initial stage of research on structural evolution in practice. The data was not only captured to study evolution but also other aspects of MDE settings in practice. It cannot be excluded that this leads to a selection bias. However, all parts of the study were observational. Therefore, we do not expect that capturing the MDE settings as explicit models influences our results on the captured evolution histories.

2 Overview on Cases Studies

In context of this third study five new MDE settings were captured together with their evolution history. In addition, the evolution history of the case study BO as well as an evolution step of a case study from Ableton were documented. This third study has a focus on structural evolution and which reflects in the questions that were used. Non-structural evolution steps (e.g. language evolution) were in most cases not captured. Consequently, this data cannot be used to make quantitative statements about distribution of structural compared to non-structural evolution steps.

In the following, an overview about the captured case studies is given and summarized in tables 1 and 2.

Case Study	Company	Full Name
Cap1	Capgemini	Capgemini case study 1
Cap2a	Capgemini	Capgemini case study 2
Cap2b	Capgemini	Capgemini case study 3
VCat	VCat Consulting GmbH	Development of TYPO3 based websites
Carneq	Carneq GmbH	Development of AUTOSAR standard documents Carneq
Ableton	Ableton AG	Development of sound libraries for users of the software Live

Table 1: Summary on captured case studies from third study

2.1 Capgemini first case study (Cap1)

The first case study (Cap1) was captured in cooperation with Capgemini¹ and is used in a project that runs since four years. In this project Capgemini builds software for a customer. There are two interacting MDE settings involved. The first MDE setting is used by the customer to collect requirements and create or prepare parts of the specification. The second MDE setting is applied within Capgemini to create prototypes, generate the specification, and to implement the software. This second MDE setting and its history were captured. The MDE setting is specific for the project, which holds especially for the used generator. Initially, a Capgemini internal standard generator was

¹<http://www.capgemini.com/>

in use, which was soon substituted by the project specific generator. In consequence, the generator can be flexibly changes or extended.

The case study was captured in summer 2012. Thereby, 16 activities as well as eight historic versions of this MDE setting were documented (including the MDE setting that was in use at the time of the interviews. Thereby, seven structural evolution steps were identified.

2.2 Capgemini second and third case studies (Cap2a and Cap2b)

Also the second and third case studies (Cap2a and Cap2b) were captured in summer 2012 in cooperation with Capgemini. The two MDE settings are parts of the same project. This project aims at providing two MDE settings that are used by a customer of Capgemini. Both settings can be applied in the same customer projects. They aim at reaching similar goals for different use cases.

MDE setting Cap2a is in use since three years. The team that developed the MDE setting Cap2a consisted initially of one person and had the size of three to four persons later on. For Cap2a 18 activities as well as six historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews) were documented. Thereby, five evolution steps were identified. Three of these evolution steps are structural.

MDE setting Cap2b is in use since five years. The team that developed the MDE setting Cap2b consisted initially of one person and grew for a short phase of ca. an half year to the size of four to five persons. For Cap2b 27 activities as well as seven historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews) were documented. Thereby, six evolution steps were identified. Five of these evolution steps are structural.

2.3 Development of TYPO3 based websites (VCat)

The fourth case study is the case study was collected in cooperation with VCat Consulting GmbH². The documented MDE setting supports development of websites that rely on TYPO3 as underlying content management systems (CMS). Motivation for that MDE setting was to improve productivity through automation and standardization. The case study was captured in winter 2012/2013. At VCat the MDE setting is developed and used since seven years (one year in its current version). For this case study 10 activities as well as three historic versions of the MDE setting were documented: the current version, a historic version, and a version that is planned to be applied in future. Thereby, two evolution steps were identified.

2.4 Development of AUTOSAR standard documents (Carmeq)

The fifth case study was captured in cooperation with Carmeq GmbH³. The captured MDE setting is used to create documents of the AUTOSAR standard⁴, including models

²<http://www.vcat.de/>

³<http://www.carmeq.de/>

⁴<http://www.autosar.org/>

Table 2: Key information on captured case studies from third study

Case study	Cap1	Cap2a	Cap2b	VCat	Carneq	Ableton	BO
Number of modeled activities	16	18	27	10	25	–	19
Years in use	4	3	5	7	8	2	>2
Number of captured evolution steps	7	5	6	2	5	1	7

and tables. The case study was captured at the start of 2013. Different versions of the MDE setting are in use since 2004. For this case study 25 activities as well as six historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews) were documented. Thereby, five evolution steps were identified.

2.5 Development of sound libraries for users of the software Live (Ableton)

The major product of Ableton AG⁵ is a software called Live, which provides artists and musicians with an environment for musical compositions and productions. An important part of the business of Ableton is the development of libraries, which provide users with a collection of presets for instruments included in Live.

The changes that are currently applied to this development of libraries is the case study that was captured. Thereby, the actual MDE setting of this development could not be captured in detail. The captured evolution step was documented at the start of 2012.

2.6 Development of Business Objects for the feature package 2.0 (BO)

First, there is the MDE setting that is used to develop business objects (an SAP specific type of components that captures functionality specific to a certain business need) for the feature package 2.0. This object of study (referred to as *BO* in the following) describes an old development approach, which was used from 2004 on ca. 100 times in nine teams. Today *BO* is substituted by a new development approach, through substitution of tools. Motivation for the initial introduction of *BO* was to enhance quality, but also to reach transparency and a unified procedure for business object development. For this object of study 19 activities were captured. Seven evolution steps were collected.

2.7 Summary

As summarized in Table 2, the captured MDE settings were between two and eight years in use. Each captured model includes between 10 and 30 activities. For each of the different cases studies between one and seven evolution steps were captured (overall 33 evolution steps).

⁵<https://www.ableton.com/>

3 Data on Evolution

All in all, the seven case studies span 33 evolution steps. The observed structural changes are summarized in the Tables 3, 4, and 5. In the following an overview on the collected evolution steps is given.

The case study BO was already subject to seven substantial structural evolution steps in a period of around six years. The MDE setting started as almost classical code centric approach with some activities to ensure tracing between code and data model. Later on a modeling tool was introduced followed by further tools that supported partial generation of the code and eventually, one of these generation tools was adopted company wide as standard (evolution step *S1*). To improve quality of behavioral implementation an interpretable modeling language was introduced in addition to the code (evolution step *S2*). The introduction of an additional modeling tool was motivated by the aim to introduce further quality assurance (evolution step *S3*). A next improvement was reached by introducing a semi-automated support for data migration between the modeling tools (evolution step *S4*). In order to reduce the number of tools the three modeling tools were integrated into a single new modeling tool (evolution step *S5*). A variant of the MDE setting was created to enable simple use at the cost of a reduced set of supported features (evolution step *S6*). Finally, the generation functionality was moved to this new modeling tool (evolution step *S7*).

For the case study Ableton, one substantial structural evolution step was captured. In this case study a fully automated generation process is subject to refactoring (evolution step *S1*). As a side effect of this refactoring parts of the automated quality assurance are separated from the generation process.

Table 3: Identified structural changes in evolution steps of the case studies BO, Ableton, and VCat (• = documented change)

	<i>SAP (BO)</i>							<i>Ableton</i>	<i>VCat</i>	
	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>	<i>S6</i>	<i>S7</i>	<i>S1</i>	<i>S1</i>	<i>S2</i>
<i>Structural Changes</i>										
[C3] change number of artifacts	•	•			•	•	•	•	•	•
[C4] change number of languages	•	•	•		•	•	•	•		
[C5] change number of manual activities	•	•	•	•	•	•	•		•	
[C6] change number of tools	•	•	•	•	•	•	•		•	•
[C7] change number of automated activities	•	•	•	•	•	•	•	•	•	•
[C8] change order of manual / automated activities	•		•		•	•		•	•	

For the case study Cap1 seven substantial structural evolution steps from a period of around four years are captured. The project started with a standard code generator, which was soon substituted by a project specific generator. In addition, a semi-automated support for the export of the data between two modeling tools was added (evolution step *S1*). To improve merge of the old version of the model and the version that is result of the semi-automated export, a first version of a diff-tool was introduced (evolution step *S2*). Due to changes the MDE setting of the customer the support for the export as well as the diff-tool were taken out of operation (evolution step *S3*). Further the automation of the export between the main modeling tool and the code generator was improved (evolution step *S4*). Later on automated support for the implementation of a user interface based on mock-ups was introduced (evolution step *S5*). A change in the development process led to a situation that modified versions of the model are created by different teams and need to be merged. To support this merge a second version of the diff-tool was reintroduced (evolution step *S6*). Finally, to address a new need on additional documentation an additional generator was introduced (evolution step *S7*).

For the case study Cap2a five evolution steps from a period of around three years are captured. Three of the five evolution steps are structural evolution steps (and two of them are substantial structural evolution steps). The project started with a first generator that was substituted later on by a more flexible version (evolution step *S1*). This substitution was planned from the beginning and was motivated by the need to rapidly provide a working MDE setting to the customer. The underlying meta model was permanently changed over the time. To create new output artifacts the generator implementation was extended (evolution step *S2*). Later on a part of the generator was reimplemented, such that independence of the formerly used implementation technology is reached (evolution step *S3*). Finally, checks have been optimized over time, such that they can be applied automatically and regular (evolution steps *S4* and *S5*).

For the case study Cap2a six evolution steps from a period of around five years are captured. Five of the six evolution steps are substantial structural evolution steps. The first version of the MDE setting included a generator for the creation of a documentation. To improve the usability, the generator was integrated to a modeling tool and adapted the meta model that was already used in case study Cap2a (evolution step *S1*). In order to support creation of additional resulting documents three further generation activities were embedded into the existing generator over the time (evolution steps *S2*, *S4*, and *S6*). Addressing quality assurance, basic consistency checks for the models were introduced (evolution step *S3*) and the introduction of further consistency checks (following the example of case study Cap2a) is planned. Finally, the output format had to be changed at least one time (evolution step *S5*).

For the case study VCat two substantial structural evolution steps from a period of around seven years are captured. The first version of the MDE setting included a manual task to copy and clean TYPO3 instances from old projects, such that configurations could be reused. To improve this process and decrease the probability that content from old projects is preserved in the copied TYPO3 instance without notice, an automated instantiation and configuration of new TYPO3 instances was introduced (evolution step *S1*). Further the use of open source TYPO3 extensions should be improved in future. For several reasons, extensions need to be adapted before they are

Table 4: Identified structural changes in evolution steps of the case studies Carmeq and Cap1 (● = documented change)

	<i>Carmeq</i>					<i>Cap1</i>						
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5	S6	S7
<i>Structural Changes</i>												
[C3] change number of artifacts	●		●				●	●		●	●	
[C4] change number of languages	●		●				●	●		●		
[C5] change number of manual activities		●							●	●		
[C6] change number of tools	●	●	●	●	●	●	●	●		●	●	●
[C7] change number of automated activities	●	●	●			●	●	●		●	●	●
[C8] change order of manual / automated activities	●	●	●			●	●	●	●	●	●	●

used. To support reuse of these adaption among the different projects at VCat, there are concrete plans to introduce an internal extension repository within VCat (evolution step S2).

For the case study Carmeq five structural evolution steps from a period of around eight years are captured. Three of the five evolution steps are substantial structural evolution steps. Initially, the AUTOSAR documentation was created without explicit modeling. To deal with inconsistencies between documents, a central model of the standardized software as well as an UML profile for AUTOSAR were introduced (evolution step S1). Thereby, an automated generation of figures and tables on the basis of the central model was introduced. Later on macros were implemented to support the integration of figures and tables into the standard documents (evolution step S2). To support quality assurance for the generated figures and tables the modelers started to use diff-tools for comparison between old and new versions of the artifacts (evolution step S3). Further, a CI server was introduced, such that the generator is executed centrally (evolution step S4). Finally, an alternative implementation for some parts of the generator (e.g., the automated import between two of the modeling tools) was introduced (evolution step S5).

Table 5: Identified structural changes in evolution steps of the case studies Cap2a and Cap2b (● = documented change)

	Cap2a					Cap2b					
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5	S6
<i>Non-Structural Changes</i>											
[C1] exchange automated activity	●	●	●	●						●	
[C2] exchange language										●	
<i>Structural Changes</i>											
[C3] change number of artifacts		●			●	●	●	●	●		●
[C4] change number of languages	●	●			●	●	●	●	●		●
[C5] change number of manual activities	●					●					●
[C6] change number of tools	●					●					
[C7] change number of automated activities					●		●	●	●		●
[C8] change order of manual / automated activities					●			●			

4 Observations

The case studies provide us with some observations on the occurrence and combination of evolution steps based on the documented data (*O1 – O3*). In addition, we derived observations on motivations and drivers for structural evolution from the coded records of the interviews (*O4 – O8*).

O1: Structural evolution steps are not necessarily exceptions, but can occur in sequence several times (e.g., SAP and Capgemini case studies).

O2: Structural evolution steps are most often combinations of multiple different structural changes (see Table 3).

O3: Substantial structural changes occur in a major part of observed structural evolution steps. We observed change type *C8* in 10 of the 15 evolution steps of our case studies. A minor observation in that context is that just one of the occurrences of *C8* was caused by improving an existing automated activity such that a manual activity was no longer necessary (*C5*). In most cases *C8* was caused by the introduction of additional automated activities (*C7*).

O4: Structural changes are often trade-offs, e.g. w.r.t costs and manageability. For example, implementing a smaller new generation step is easier to manage than apply-

ing a change to an existing automated activity. A further factor in such a trade-off is the weight that is given to the different productivity dimensions. In many of the observed cases it was decided to increase the *degree of automation* or tool support by adding new automated activities instead of adapting existing automated activities like transformation steps. Thus, a substantial structural change that might lead to drawbacks for the changeability is accepted in favor of costs and manageability of the structural evolution step.

O5: The factors involved in such trade-offs change over time. For example, costs that can be invested in a change can differ strongly. We even captured cases where developers implemented evolution steps in their leisure time. The weight that is given to different productivity dimensions can also change. For example, while evolution step *S1* in the SAP case study was mainly driven by the desire to increase the *degree of automation*, a priority that led to evolution step *S5* was the desire to decrease *cost of ownership* and *complexity* by decreasing the number of involved tools.

O6: Changes in an MDE setting can be driven by the need to take other MDE settings into account (e.g., the evolution in [3] or evolution steps *S3* and *S5* in the Capgemini case study). This can happen, when models or other artifacts in software development are supplied by one company and used the other. Then changes in the MDE setting of one company can lead to new opportunities for integration of both settings.

O7: Some evolution steps are motivated by preceding evolution steps. They reduce the complexity of MDE settings, which can be considered as ‘refactoring’, after several preceding evolution steps were applied. An example of this is the introduction of the new repository in business object development (evolution step *S5* in the SAP case study).

O8: Some evolution steps are not planned centrally, but are caused by developers who add automation steps to ease their daily work. Examples are evolution steps *S4* and *S6* in the Capgemini case study as well as the solutions added for code generation in evolution step *S1* in the SAP case study.

References

- [1] Regina Hebig, Holger Giese, Florian Stallmann, Andreas Seibel. On the Complex Nature of MDE Evolution. In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems, MODELS 2013*, Miami, USA, 2013.
- [2] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. In *IEEE Transactions on Software Engineering*, Vol. 25(4), pages 557–572, IEEE Press, 1999.
- [3] Franck Fleurey, Erwan Breton, Benoît Baudry, Alain Nicolas, and Jean-Marc Jézéquel. Model-Driven Engineering for Software Migration in a Large Industrial Context. In Gregor Engels, Bill Opdyke, Douglas Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 482–497. Springer Berlin / Heidelberg, 2007.

Modeling Interestingness and Serendipity in Relevance Search

Maximilian Jenders

Information Systems Group

Hasso-Plattner-Institut

maximilian.jenders@hpi.uni-potsdam.de

Popular recommendation algorithms such as Collaborative Filtering, Latent Semantic Analysis, or Locality-Sensitive Hashing often operate under the assumption that similar users like similar items and that user who like a specific item will also like similar items (i.e., items with similar features). Therefore, these algorithms determine the most similar users / items and then recommend the *top-k*, i.e., the k most similar users or items.

While these recommender systems are hugely popular and successful, their recommendations are very similar to each other. Especially in the case of text recommendation scenarios, this can lead to decreased user satisfaction, since a user will not be interested in reading very similar texts. We therefore aim to improve traditional recommender systems by providing an information-theoretic measure of interestingness and serendipity, i.e., the discovery of relevant but unexpected content.

1 Introduction

In the modern world, the variety of products we can consume or buy is steadily increasing. To make sense of all the information and help us make choices, we rely on recommendations made by others—from personal friends and professional reviewers to reviews written by persons unknown to us. In the digital world, recommendation algorithms provide recommendations tailored to individual users, thereby increasingly guiding our decisions and helping us find interesting and relevant items in a vast information space. These personalized recommendations encompass a short list of ranked (and therefore prioritized) items and are typically based on information of the user's past behavior (e.g., a purchase history) as well as information about the choices of similar users (e.g., purchase decisions) and / or the features of similar items.

Optimizing recommender systems therefore helps suggest better items to users, therefore increasing user satisfaction. For optimizing recommender systems in general, but especially in text recommendation scenarios, we propose re-examining the role of similarity metrics and accuracy-based evaluation metrics.

- *Similarity Metrics*: As recommendation algorithms usually predict user behavior based on similar users or similarities between items, these similarity metrics are applied for the generation of *top-k* recommendation lists. As a result, the recommendations are usually very similar to the users' past purchase or consumption

behavior. A user that listens to rock music will receive recommendations for rock artists with similar music styles, a user reading news about just one specific topic will be recommended articles about the same or a very similar topic.

While these kinds of recommendations certainly recommend items that a user will most probably favor (e.g., in the case of music recommenders, rock musicians), they are very limited in their capabilities to help discover new, diverse things that the user might also like (in this example, the user might also like electronic music).

By recommending more diverse items instead of relying on the most similar items, user satisfaction may be improved significantly.

- *Evaluation Metrics:* Traditionally, evaluations have been focused on accuracy-based measures (e.g., precision, recall [5], or mean average precision [3]), typically by measuring the overlap of recommended items with the items the user then consumed. These evaluation measures allow for a quantitative appraisal of recommendation algorithms, yet they completely neglect the qualitative side.

Recommending similar items is relatively safe, as users tend to like things similar to their (original) taste. A user that likes an artist's album is very likely to like other music from the artist as well. Finding items from a completely different topic that a user will like is much harder, yet it allows the user to discover and acquire new tastes, thereby increasing user satisfaction. Such riskier recommendations can therefore add value, yet that risk is not reflected in accuracy-based evaluation metrics, where each recommendation is worth as much as the other.

We therefore propose to focus research on algorithms that aim to find relevant, interesting and serendipitous items. As much of the research so far has been conducted in the realms of music or article recommendations, we direct our attention on text recommendation scenarios, namely newspaper articles. Improving recommender systems in this field can help users discover interesting facts yet unknown to them and change their perception on current and important events. Providing users with a variety of unexpected yet interesting articles will also increase their satisfaction.

Please note that in this report we are focusing solely on information encoded in a document corpus and are not considering any user information, such as expressed preferences, user interactions, and past user behavior.

2 Related Work

Related work can be classified into three categories: Discussing pitfalls and alternatives to accuracy-based metrics, improving diversity and novelty, and improving serendipity in recommender systems.

2.1 Accuracy

A popularity bias of traditional recommender systems, i.e., a tendency to excessively recommend very popular items, has been shown in [1]. The authors introduce item-

and user-centric methods that make more novel recommendations, yet an evaluation shows that users perceive the new methods to have lower quality than traditional Collaborative Filtering techniques.

Other work finds that most research has focused on increasing the accuracy of recommender systems [6]. It is contended that this focus proves hurtful to recommender systems, as the most accurate recommendations are not always the most useful to users. The authors argue for the focus on other evaluation measures and propose new directions, including the introduction of serendipity as an evaluation metric.

2.2 Diversity and Novelty

Due to the use of similarity metrics, the items in top- k recommendations tend to be too similar to each other [9]. As a mitigation technique, the authors improve the diversity of the recommendations by measuring and improving the dissimilarity of the items in the set of recommended items that is to be shown to the users.

Other related work presents a formal framework for the definition of novelty and diversity metrics [8]. Novelty can be defined by how different an item is from past experience, while diversity is expressed as a rank-sensitive average intra-list distance between items.

A special problem arises when trying to accommodate users trying to search for different viewpoints on a controversial topic [4]. For such search queries, the results need to be diverse while minimizing any bias against specific sentiments. The authors propose an opinion diversification model that relies on the relevance of documents, uses semantic diversification to capture different arguments and also employs sentiment diversification to retrieve positive, negative and neutral arguments. In experiments, this model always significantly outperformed the native ranking of opinionated web pages.

2.3 Serendipity

Serendipity can be defined as a measure of the degree to which recommendations are both attractive and surprising to users [3]. An evaluation of recommender systems not by accuracy but by coverage and serendipity has been carried out in [2]. The authors find that very few experimental studies on serendipity have been carried out. To capture serendipity, they focus on two aspects: unexpectedness and usefulness. All items that are not generated by a traditional recommendation algorithm are considered unexpected, while the usefulness of candidate items is being judged by the user. This approach is also employed by [7].

Finally, the *Auralist* recommendation framework, which takes into consideration accuracy, diversity, novelty, and serendipity, has been introduced in [10]. Serendipity is measured as the surprise of recommendations and imagined as the distance between recommended items and their expected contents. It is measured through a new *Unserendipity* metric, which uses the cosine similarity to measure the average similarity between items in a user's history and new recommendations. The authors evaluate their framework with an user study on a music recommendation dataset, improving

user satisfaction. The definition of serendipity the authors use only captures the unexpectedness of the recommendation, yet does not assess whether the recommendation is useful for the user, making the discovery a “happy coincidence”.

In contrast to prior work, we provide an information-theoretic approach to the task of finding serendipitous documents without relying on traditional recommender systems to quantify the unexpectedness of results.

3 Modeling Interestingness and Serendipity

In order to improve existing recommender systems, we want to rank documents according to their interestingness and surprise. Interestingness itself is hard to define, since users can find an article interesting for a variety of reasons. It may re-affirm their point of view, expose them to arguments opposing their point of view, introduce new arguments, or presents intriguing facts. Serendipity on the other hand means a happy accident, or a fortunate mistake. By using serendipity in the realm of recommender systems, we refer to users stumbling over a recommendation that they did not expect, yet are pleased to see. Hence, we model the serendipity of documents, assuming that a document that is found to be serendipitous by a user will also be of interest to them.

For the purpose of text article recommendations, we therefore proceed as follows: For a given article, we first determine a set of *core documents* which are very similar to each other and will therefore be excluded from our recommendations. Secondly, we try to determine documents which show certain characteristics of core documents (thereby ensuring they are relevant), yet also different enough to be surprising.

3.1 Finding Core Documents

First, we assume that there is a multinomial distribution θ from which documents can be generated. The goal is to find to approximate this distribution as closely as possible and calculate the probability of a document being generated by this distribution.

The process of a document being generated by such a distribution can be pictured as rolling a weighted dice with l sides a number of n times, whereas each side corresponds to a unique word. Each time the dice is thrown, the side that comes up determine a word being generated by the distribution. Since the dice is weighted, different words can have different probabilities of being used. Through this process, we can simulate a document with a total of n words being generated from a multinomial distribution θ over l different (unique) words.

Once we can calculate the probability of a document being generated by θ , we can determine the most probable documents, which add up to the core documents.

3.1.1 Simple algorithm

As stated above, we assume a multinomial distribution θ , from which documents are generated. The algorithm in pseudo-code is as follows:

Data: Documents $d_1 \dots d_m$

Result: Set of core documents $D = \{d_1, \dots, d_x\}$ with $x \leq m$,
multinomial distribution θ

Input: Search query

Set $D = \{d_1\}$, d_1 : document that best matches the query;

repeat

 Estimate θ based on word frequencies of all $d \in D$;

 Iterate over all $d_i \notin D$ to determine $\operatorname{argmax}_{d_i} P(d_i|\theta)$;

 Add d_i to D ;

until *Change in entropy $\Delta H(D)$ is only marginal*;

Output: D_1, θ

Algorithm 1: Simple algorithm to determine core documents

For a given search query, we determine the document d which best fits that query and for which we want to find serendipitous documents. In the first step, we want to find all other core documents for d , i.e., all other documents that are very similar to d , or

$$\operatorname{argmax}_D P(D|\theta) \text{ with } \{d_1, \dots, d_x\} \in D$$

A document d is made up of l different words w_1, \dots, w_l , whereby each word w_i occurs k_i times. d can therefore be expressed as

$$d = w_1^{k_1}, \dots, w_l^{k_l}$$

For each word w_i , there exists a probability p_i of that word being generated by the multinomial distribution θ . We can therefore calculate the probability of a document d being generated from θ :

$$P(d|\theta) = \binom{n}{k_1 \dots k_l} \times p_1^{k_1} \times \dots \times p_l^{k_l} \text{ with } \sum_{i=1}^l k_i = n$$

Since we start with one document $d_1 (D = \{d_1\})$, we can use the word distributions of d to approximate θ . We then calculate the document d_i with the highest probability of being generated by θ :

$$\operatorname{argmax}_{d_i} P(d_i|\theta) \text{ with } d_i \notin D$$

Note that the new document d_i may contain words that are not contained within any document in D . Since the probability of the new, unknown word being generated would be 0, the whole term would reach 0. Therefore, some kind of smoothing (e.g., Laplacian smoothing) has to be used when calculating the probabilities.

Additionally, since a lot of small probabilities are being multiplied for each other, the end result may become 0 due to the limited range of possible float values in a computer. A practical approach is to calculate the logarithm of the probabilities (log probabilities), which are asymptotic to the real probabilities. As probabilities $p \in [0, 1]$ the log probabilities are calculated as $-\log(p)$, resulting in a non-negative real value. The advantages are that multiplications of probabilities now become addition of log probabilities, which are faster to calculate, and an increased numerical stability.

After $\operatorname{argmax}_{d_i} P(d_i|\theta)$ has been found, it can be added to $D = d_1, \dots, d_i$ and θ can be updated using the word distributions from all $d \in D$. To determine when D includes all core documents, we calculate the entropy H in all documents in D :

$$H(D) = \sum_{j \in d_i} P(w_j) I(w_j) = - \sum_{j \in d_i} P(w_j) \log_2 P(w_j) \text{ with } d_i \in D$$

In each iteration, the entropy $H(D)$ will continue to decrease because words will be less and less surprising. As soon as the change in entropy $\Delta H(D)$ becomes negligible, we assume that D encompasses all core documents and terminate the first step of our algorithm.

3.1.2 Advanced Algorithm

The previous algorithm only takes into account a single distribution θ from which all core documents can be generated. For an advanced algorithm, we model a second multinomial distribution θ_2 that captures a distribution which generates non-core documents. For each document, we can then determine whether it was more likely to be generated from a core topic distribution or a non-core topic distribution.

Data: Documents $d_1 \dots d_n$ (documents that are relevant to a query)
Result: Set of core documents D and parameters of multinomials θ_1, θ_2 , with priors π_1, π_2
Input: Search query
 Initialize $\pi_1 = \pi_2 = 0.5$;
 Set $D_1 = \{d^*\}$, d^* : document that best matches the query;
 Set $D_2 = \{\}$;
 Calculate θ_1 from D_1 ;
repeat
 for each document $d_i, i \in \{1, \dots, n\}$ **with** $d_i \neq d^*$ **do**
 $D'_1 := D_1$;
 $D'_2 := D_2$;
 $D'_1 \leftarrow D'_1 \cup \{d_i\}$;
 $D'_2 \leftarrow D'_2 \cup \{d_i\}$;
 Compute smoothed θ'_1 based on D'_1 ;
 Compute smoothed θ'_2 based on D'_2 ;
 Calculate $\operatorname{argmax}_j (\pi_j \times P(d_i|\theta'_j))$;
 Add d_i to D_j ;
 Update θ_j based on D_j ;
 end
 Compute $P(\theta_1) = \pi_1 = \frac{1}{n} \sum_{i=1}^n P(\theta_1|d_i)$;
 Compute $P(\theta_2) = \pi_2 = \frac{1}{n} \sum_{i=1}^n P(\theta_2|d_i)$;
until $\log P(d_1, \dots, d_n|\theta_1, \theta_2, \pi_1, \pi_2)$ *does not change much*;
Output: $D_1, \theta_1, \theta_2, \pi_1, \pi_2$

Algorithm 2: Advanced algorithm to determine core documents

In the above algorithm, π_1 and π_2 are priors that determine the respective probabilities of the distributions θ_1 and θ_2 , of being used to generate a document. At the start,

both are set to 0.5 and the initial document d^* that matches the query best is being used to initialize θ_1 .

The algorithm then iterates over all other documents $d_i, i \in \{1, \dots, n\}$ with $d_i \neq d^*$. A given document d_i may contain words for which no probabilities are yet determined in θ_1 and θ_2 based on D_1, D_2 . Therefore, temporary Document sets D'_1 and D'_2 are constructed with $D'_j \leftarrow D_j \cup \{d_i\}$ (with $j \in \{1, 2\}$), thereby assuring that no unknown words exist in D'_j . Subsequently, smoothed multinomial distributions θ_1^i and θ_2^i can be approximated.

Based on these smoothed distributions, we can now analyze d_i and determine the distribution θ_j for $\operatorname{argmax}_j P(d_i|\theta_j)$. Accordingly, d_i is then added to the D_j whose multinomial distribution is more likely to have generated d_i . As D_j changes, θ_j is updated based on the word distributions of all documents $d \in D_j$.

After all documents have been assigned to a θ_j , we recalculate the prior probabilities π_j for both distributions given the documents and the old distribution priors:

$$\pi_j = P(\theta_j) = \frac{1}{n} \sum_{i=1}^n P(\theta_j|d_i)$$

$$P(\theta_j|d_i) = \frac{P(d_i|\theta_j) \times P(\theta_j)}{P(d_i|\theta_1) \times P(\theta_1) + P(d_i|\theta_2) \times P(\theta_2)} = \frac{P(d_i|\theta_j) \times \pi_j}{P(d_i|\theta_1) \times \pi_1 + P(d_i|\theta_2) \times \pi_2}$$

Note that π_1 and π_2 are normalize so that $\pi_1 + \pi_2 = 1$. The above algorithm is iteratively executed until $\log P(d_1, \dots, d_n|\theta_1, \theta_2, \pi_1, \pi_2)$ is relatively stable, i.e., does not change much after each iteration. The calculation is as follows:

$$\log P(d_1, \dots, d_n|\theta_1, \theta_2, \pi_1, \pi_2) = \sum_i \log(P(d_i|\theta_1) \times P(\theta_1) + P(d_i|\theta_2) \times P(\theta_2))$$

Keep in mind that $P(\theta_j) = \pi_j$. All documents that are in D_1 are now the core documents that have been generated by θ_1 , whereas all documents in D_2 are non-core documents generated by θ_2 . In the next step, we can therefore look for surprising documents in D_2 .

3.2 Finding Serendipitous Documents

After having determined the core documents, we now want to find documents that are serendipitous to a user, i.e., that are surprising yet relevant and interesting. We therefore try to find documents that have two different, contradicting characteristics:

1. On the one hand, they should be very different from core documents, thereby increasing the surpriseness of the document and eliminating “obvious” recommendations.
2. On the other hand, they should have some characteristics of core documents, thereby affirming their relevance to the topic and therefore their interestingness to the user.

The first condition can easily be measured using the already calculated multinomial distribution θ for the set of core documents D . Given any candidate document d , $P(d|\theta)$ can easily be calculated given the formulas stated above. The lower the probability of the d being generated through θ , the greater the difference to core documents. To measure the condition 2, we use the original θ to generate a second multinomial distribution θ^{red} : Given the word probabilities $p_1, \dots, p_{k-1}, p_k, p_{k+1}, \dots, p_l$ from θ , we create a new θ^{red} with only the highest k word probabilities from θ ($k \ll l$). We then can calculate $P(d|\theta^{red})$, which gives us information about how close the document d is to a certain portion of the document core.

To find the document $d \in D_2$ that best satisfies both conditions 1 and 2, we can now calculate

$$\operatorname{argmax}_d \frac{P(d|\theta_1^{red})}{P(d|\theta_1)}.$$

The result of this calculation is $\in (0, \infty]$. The document d maximizing the formula satisfies conditions 1 and 2 best: It is different from the core documents, yet still maintains a relevance to the core documents.

4 Ongoing Work

This report so far has presented an information-theoretic model for a recommender system that suggests serendipitous items. In order to realize this goal, the following steps are ongoing at the moment:

4.1 Evaluation

These algorithms for finding the most serendipitous documents can unfortunately not be evaluated without human judgement. All available data sets are optimized for accuracy; while one is able to prove that a new algorithm recommends the items that a user has consumed in a real-world scenario, we aim to help users discover completely new and surprising content. Therefore, we will have to conduct experiments where humans judge whether a recommendation is both surprising and relevant or interesting to them. This is also the only way to show whether the detection of core documents is superior with the advanced algorithm than with the simple one. Other algorithms from related work can serve as a baseline as well as a traditional recommender systems.

At the time of this writing, we have not yet been able to conduct such an experiment. We aim to provide users with a framework which lets them query a corpus and evaluate the recommendations with respect to their serendipity in order to compare the usefulness of our algorithm with the state of the art.

4.2 Data Set

To properly evaluate the algorithms, a document corpus is needed. A very interesting use case for text recommendations is in the realm of newspaper articles: When a user reads a document or enters a search query, it would prove beneficial to not only

recommend newspaper articles with similar words as the original one, but supply them with novel, serendipitous articles which can broaden the user's horizon.

From past experiences we have learned that automatically crawled newspaper articles, e.g. from RSS feeds, still contain a substantial degree of clutter—for example advertisements, user comments, links to other stories. We are therefore manually harvesting newspaper articles for a few different, specific news topics. Additionally, we are looking for and evaluating clean newspaper corpora which can serve as a dataset.

4.3 Parameter Optimization

Furthermore, through experiments and user feedback, parameters can be optimized. In the simple algorithm for finding core documents the $\Delta H(D)$ at which the iterative algorithm is stopped may be improved, as well as the prior calculation π_1 and π_2 in the advanced algorithm. When determining the most serendipitous documents, weights for $P(d|\theta_1^{red})$ and $P(d|\theta_1)$ may help achieve better results.

5 Conclusion And Future Work

In this report, we presented an information-theoretic algorithm that calculates serendipity in text documents. Given a start document that best fits a search query, it first determines core documents which are very similar to a user and then retrieves the most serendipitous documents which are not covered by the core documents. Including serendipitous results in recommender systems can be a key factor in increasing the quality of the recommendations and therefore improving user satisfaction.

In addition to the ongoing work that has been outlined in the previous section, future work encompasses the inclusion of user-based information. So far, the presented algorithm makes no assumptions about the user and calculates serendipity purely based on a start document and the available document corpus. While this would be the solution for websites that do not know their users, typically the past user behavior is available, i.e., the documents that have already been read by a user. User preferences can also be constructed by the terms a user has searched for, the time a user has spent on reading articles or the number of times the user has shared an article with friends. Incorporating such user behavior can significantly improve the discovery of serendipitous documents and will be addressed in future work.

References

- [1] Òscar Celma and Perfecto Herrera. A New Approach to Evaluating Novel Recommendations. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 179–186, New York, NY, USA, 2008. ACM Press.
- [2] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings*

- of the fourth ACM conference on Recommender systems, RecSys '10*, pages 257–260, New York, NY, USA, 2010. ACM.
- [3] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.
- [4] Mouna Kacimi and Johann Gamper. Diversifying search results of controversial queries. In *Proceedings of the 20th ACM international conference on Information and knowledge management, CIKM '11*, pages 93–98. ACM, 2011.
- [5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Sch a tze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [6] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being Accurate is Not Enough: How Accuracy Metrics have hurt Recommender Systems. In *Computer Human Interaction, CHI '06*, pages 1097–1101, New York, NY, USA, 2006. ACM.
- [7] Neha Pruthi and Chhavi Rana. Surprising discoveries by recommender systems. In *International Journal of IT, Engineering and Applied Sciences Research*, volume 2, 2013.
- [8] Sa ul Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, volume 107 of *RecSys '11*, pages 109–116, New York, NY, USA, 2011. ACM.
- [9] Mi Zhang and Neil Hurley. Avoiding Monotony: Improving the Diversity of Recommendation Lists. In *Proceedings of the 2008 ACM conference on Recommender systems, RecSys '08*, pages 123–130, New York, NY, USA, 2008. ACM.
- [10] Yuan Cao Zhang, Diarmuid   S eaghda, Daniele Quercia, and Tamas Jambor. Auralist: Introducing serendipity into music recommendation. In *Proceedings of the fifth ACM international conference on Web search and data mining, WSDM '12*, pages 13–22, New York, NY, USA, 2012. ACM.

Describing and Comparing Datasets on the Web of Data

Anja Jentzsch

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

Over the last years, an increasing number of web sites have started to embed structured data into HTML documents as well as to publish structured data in addition to HTML documents directly on the Web. This trend has led to the extension of the Web with a global data space – the Web of Data. As the classic document Web, the Web of Data covers a wide variety of topics ranging from data describing people, organizations and events, over products and reviews to statistical data provided by governments as well as research data from various scientific disciplines. In order to benefit from this massive amount of open data, e.g. to add value to an organizations internal data, such external data sets must be analyzed and understood already at the basic level of data types, constraints, value patterns etc.

We present an approach for finding minimal descriptive sets of properties and analyzing the uniqueness and relevance of properties in Linked Data sets.

1 Introduction

With the growing number of Linked Open Data (LOD) sets on the Web of Data its heterogeneity increases. Simultaneously the number of existing schemas as well as the content and granularity of data increases and thus also diverges.

The Web of Data already comprised already roughly 900 data sources including prominent examples, such as DBpedia, YAGO, and DBLP. A LOD dataset is usually represented in the Resource Description Framework (RDF) embodying an entity-relationship-graph or a set of facts in triple format (subject, predicate, and object). Algorithms and tools are needed that profile the dataset to retrieve relevant and interesting meta-data analyzing the entire dataset [12]. Indeed, there are many commercial tools, data sets are openly available and connected amongst each other via identity (`owl:sameAs`) or other links between representations of same real-world entities. Most of the RDF data sets are listed in registries as for instance at <http://datahub.io>.

However, consuming LOD is not easy, because the sources are heterogeneous, often inconsistent, and lack often even basic metadata. One of the main reasons for this problem is that many of the data sources, such as DBpedia [11] or YAGO [10], have been extracted from unstructured data sets. Furthermore, knowledge bases and their schemata usually evolve over time. Hence it is vital to thoroughly examine and understand each data set, its structure, and its properties before usage.

There are many commercial tools, such as IBM's Information Analyzer, Microsoft's SQL Server Integration Services (SSIS), or others for profiling relational data sets. However all of these tool were designed to profile relational data. LOD which is represented in RDF data has a very different nature and calls for specific profiling and mining techniques.

Profiling Linked Data sets allows for finding, interlinking and integrating them by analyzing semantic and structural heterogeneity.

Finding information data sets is an open issue on the constantly growing Web of Data. While there are data set registries, these often have to be curated manually. Furthermore, existing ways for describing data sets are often limited in their depth of information. VoiD [2] and Semantic Sitemaps [4] cover basic details of the data set but do not cover detailed information on the data set's content like topics or distribution of topics covered.

The Web of Linked Data is built upon the idea that data items on the Web are connected by RDF links. Sadly, the reality on the Web shows that Linked Data sources set some RDF links pointing at data items in related data sources, but they clearly do not set RDF links to all data sources that provide related data¹. Writing linkage rules for unknown and large data sets is a time-consuming task. This includes finding main classes occurring in the data set as well as finding relevant sets of properties that define entities unambiguously.

Describing entities in an unambiguous way is also a crucial task for integrating Linked Data sets with other data sets for e.g. the use in applications or websites. News sites that incorporate info boxes for named entities can use a comprehensive set of property values to describe a real-world entity to the reader.

In comparison to other data models (i.e., the relational model), RDF lacks to provide explicit schema information that precisely defines the types of entities and their attributes. Therefore, many data sets provide ontologies that categorize entities and define data types and semantics of properties. However, ontology information for a Linked Data set is not always available or may be incomplete.

Our approach generates basic statistics to understand the uniqueness and relevance of properties, as well as minimal unique property combinations. Having these at hand, the user can determine possible keys for Linked Data sets or subsets.

Our approach has been implemented in ProLOD++ [1]. ProLOD++ is a web-based tool for profiling and mining Linked Data sets. ProLOD++ comprises various traditional data profiling tasks, adapted to the RDF data model. In addition, it features many specific profiling results for open data, such as schema discovery for user-generated attributes, association rule discovery to uncover synonymous properties, and uniqueness discovery along ontology hierarchies. It is highly efficient, allowing interactive profiling for users interested in exploring the properties and structure of yet unknown datasets.

¹<http://lod-cloud.net/state/>, retrieved October 2013.

2 Related Work

Equality of entities in Linked Data sets is generally defined for all entities sharing the same URI, also across data sets.

Due to the requirement of a good URI being defined in an HTTP namespace under ones control to make them actually dereferenceable [13], entity links across data sets need to be established by using the *owl:sameAs* statement.

Many languages have a so-called unique names assumption: different names refer to different things in the world. On the Web, such an assumption is not possible. For example, the same person could be referred to in many different ways (i.e. with different URI references), allowing each publisher to state their view on a person. For this reason OWL does not make this assumption. Unless an explicit statement is being made that two URI references refer to the same or to different individuals, OWL tools should in principle assume either situation is possible.

OWL provides three constructs for stating facts about the identity of individuals:

- *owl:sameAs* is used to state that two URI references refer to the same individual.
- *owl:differentFrom* is used to state that two URI references refer to different individuals
- *owl:AllDifferent* provides an idiom for stating that a list of individuals are all different.

This has two important consequences. First of all following such an approach in a highly dynamic and extremely rapidly growing environment such as the Web requires the permanent search, analysis and alignment of new data. Additionally, reasoning over *owl:sameAs* relations in distributed ontologies may be a complex task: the creation of *owl:sameAs* statements among the URIs identifying the same entity implies the computation of the transitive closure, potentially across the whole of the Web of Data. The transitive closure computation is known to belong to the NL computational complexity class.

OWL 1 does not provide a means to define keys. However, keys are clearly of vital importance to many applications in order to uniquely identify individuals of a given class by values of (a set of) key properties. In OWL 2 [8] a collection of properties can be assigned as a key to a class using the *owl:hasKey* statement. This means that each named instance of the class is uniquely identified by the set of values. While in OWL 2 key properties are not required to be functional or total properties, it is always possible to separately state that a key property is functional, if desired. OWL also the *functionalProperty* statement to express that a property can have at most one value.

Though OWL allows the definition of key properties, it has not yet fully arrived on the Web of Data. Glimm et.al. [5] found that only one data set uses the *hasKey* property, while properties like *owl:sameAs* are already widely used on the Web of Data.

Thus, actually analyzing and profiling Linked Data sets requires manual, time consuming inspection or the help of tools.

There are many commercial tools, such as IBM's Information Analyzer, Microsoft's SQL Server Integration Services (SSIS), or Informatica's Data Explorer, and some research prototypes, such as [9], for profiling relational data sets. However all of these tool were designed to profile relational data. LOD which is represented in RDF data has a very different nature and calls for specific profiling and mining techniques. Current tools to work on RDF data are limited to graph visualization and editing: LODlive² is a browser-based tool to browse and search in RDF data sets. RDF Pro³ is a suite for visual editing RDF data. LODStats [3] is a stream-based approach for gathering comprehensive statistics about RDF data sets. Finally, RelFinder [6] is a web-based tool to interactively discover relationships between entities on the Web of Data.

Our approach allows for finding properties that are key candidates, retrieves data set insights and also a class hierarchy analysis without requiring the user to inspect the data set content manually or with the help of tools.

3 Approach

Finding keys for Linked Data sets consists of finding minimal unique property combinations. Furthermore, additional statistics to understand the uniqueness and relevance of properties are helpful. These might be the number of NULL and non-NULL values per property, the uniqueness of all the property values per property, and the number of unique values per property. Having these values at hand, a user can determine possible keys based on the unique property combinations.

We define the keys of Linked Data (sub-)sets as the set of properties that uniquely identify an entity. This means that each entity of the data (sub-)set is uniquely identified by the set of property values.

As Linked Data sets are usually sparsely populated, more statistics on the uniqueness and thus relevance of a property are needed.

We define the *uniqueness* of a property as the number of unique values per number of total values for a given property.

We define the *density* of a property as the ratio of non-NULL values to the number of entities.

We call a property *full key candidate* if its density is 1.

We call a property *key candidate* if its density and uniqueness are 1.

Properties on the Web of Data can have multiple property values. E.g. the birth place for Albert Einstein can be given as Baden-Württemberg, Ulm, and the German Empire. We combine these property values and define the density as the sum of the separate value densities divided by the number of property values, thus the average of all densities.

²<http://en.lodlive.it/>, retrieved October 2013.

³<http://www.linkeddatatools.com/rdf-pro-semantic-web>, retrieved October 2013.

3.1 Uniqueness in Hierarchies

A common practice in the Linked Data community is to reuse terms from widely deployed vocabularies whenever possible, in order to increase homogeneity of descriptions and, consequently, easing the understanding of these descriptions. As Linked Data sources cover a wide variety of topics, widely deployed vocabularies that cover all aspects of these topics may not exist yet. Therefore publishers commonly also define proprietary terms and mix these terms with the widely used ones in order to cover the more specific aspects and to publish the complete content of a data set on the Web. There are at least 369 different vocabularies to be found on the Web of Data⁴. Vocabularies define classes and their relationships. Ontology classes usually are arranged in a taxonomic (subclass–superclass) hierarchy.

Analyzing the uniqueness of properties as well as the unique property combinations across the class hierarchy can bring insights into the data distribution inside the the data set. It allows for finding relevant properties for each level in the class hierarchy and see if the relevance is increasing or decreasing along the path down the hierarchy.

4 Implementation

Our approach has been implemented in ProLOD++. ProLOD++ uses our new unique discovery technique, DUC [7], to identify unique property combinations on clusters. DUC provides a scalable and efficient method for finding all unique *and* non-unique column combinations in big data sets by using a novel hybrid graph traversal technique, which traverses the lattice in combination of depth-first and random walk.

Besides the minimal unique property combinations, ProLOD++ offers a range of statistics to understand the uniqueness and relevance of properties. The statistics cover the number of NULL and non-NULL values per property, the uniqueness of all the property values per property, the density of the property, as well as the number of unique values per property. Having these values at hand, the user can determine possible keys based on the unique property combinations retrieved by DUC taking into consideration the usage of this property based on the number of values (non-NULL values). For example out of the 185,081 athletes in DBpedia, only 36 have a `dbpedia:espnId` value, yet all of these values are unique. This defines `dbpedia:espnId` as a unique property combination for athletes in DBpedia.

Furthermore, ProLOD++ can cluster entities based on a data set's underlying ontology. This allows the user to see the evolution of key features through hierarchical levels, thus determining class-specific properties in unique property combinations. I.e., the degree of uniqueness or density of `dbpedia:espnId` can be observed to for things, persons, athletes, and finally football players.

⁴<http://lov.okfn.org/dataset/lov/>, retrieved October 2013.

5 Evaluation

Our approach has been evaluated using the DBpedia data set and within that the Person class subcluster (see Table 1).

Class	Number of Instances	Number of Properties
Person	720,811	255
Athlete	185,081	127
Baseball Player	17,487	37
Basketball Player	2,605	42
Rugby Player	9,206	38
Soccer Player	81,438	33
Scientist	12,841	41

Table 1: Part of the DBpedia class taxonomy for persons.

Like other data sets, DBpedia is often sparsely populated. Figure 1 shows the distribution of values and unique values for properties in the Person class cluster. 86 % of the properties do only have up to 5 % values available. This stresses the need for taking into account further details like the density value of a property when choosing key properties.

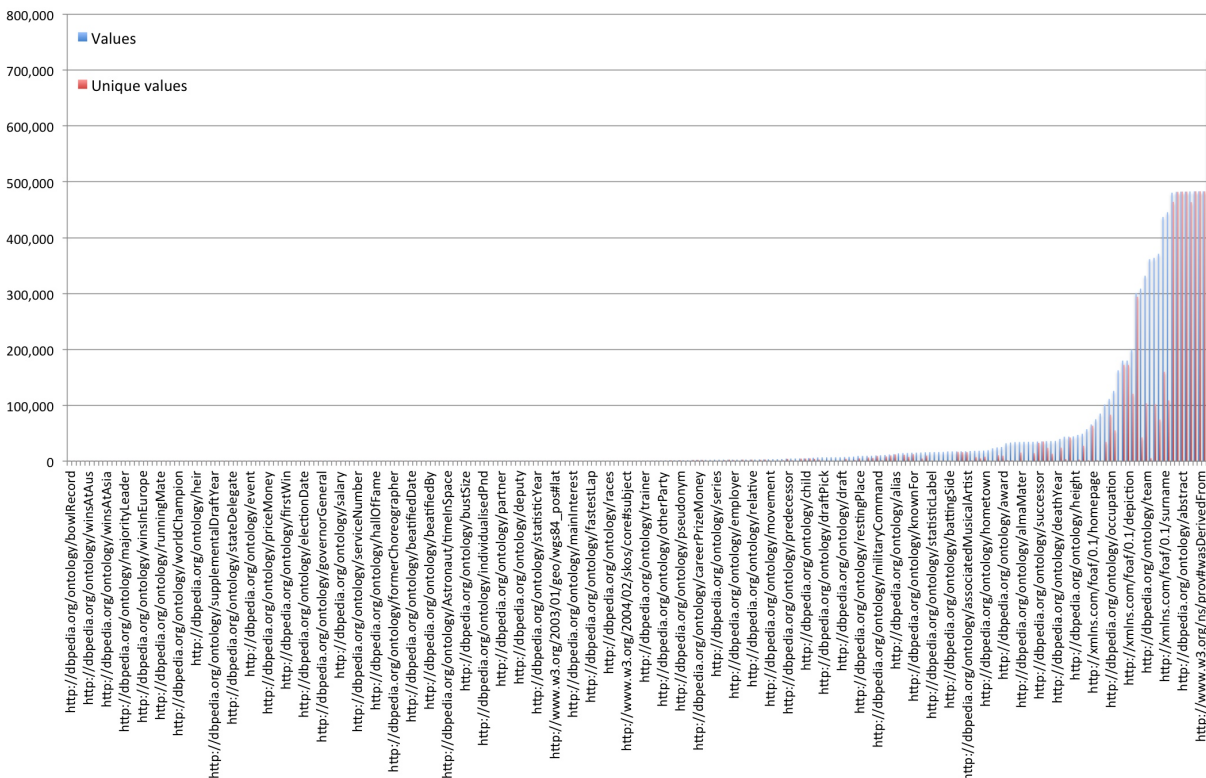


Figure 1: Value and unique value distribution for properties of the DBpedia Person class cluster.

Property	Values	Unique values	Uniqueness	Density
dbpedia: bowlRecord	2	2	1	0.01
dbpedia: espnId	36	36	1.00	0.01
dbpedia: individualisedPnd	587	584	0.99	0.01
dbpedia: birthDate	371,127	74,457	0.20	0.51
dbpedia: abstract	482,511	482,478	0.99	0.67
rdfs: label	483,004	482996	0.99	0.67
rdf: type	720,812	306,804	0.43	1.00

Table 2: Value and unique value distribution, uniqueness and density for DBpedia Person cluster (excerpt).

Table 2 shows an excerpt of the 256 properties for the Person cluster (covering 720,812 entities) along with the value and unique value distribution, the uniqueness and density.

What is visible already from this excerpt are some typical characteristics of Linked Data sets. They often contain properties with few values but a high uniqueness (nearly 1), e.g. `dbpedia: individualisedPnd`. A lot of the properties have a high uniqueness but below 100 % values, e.g. `rdfs: label`, and `dbpedia: abstract`.

The density rarely reaches 0.5 and only in one of 256 properties (`rdfs: label`) reaches 1.

Property value sets are of specific interest, e.g.

```
dbpedia:Einstein dbpedia: birthPlace dbpedia: Baden-Württemberg,
dbpedia: birthPlace dbpedia: Ulm,
dbpedia: birthPlace dbpedia: German_Empire.
```

We treat those as lists of elements.

By applying the adjusted version of DUCG to the data set, we can generate (minimal) unique property combinations for the different class clusters in DBpedia. This allows for analyzing key candidates.

Table 3 shows the minimal unique property combinations for persons, scientists and athletes. For choosing key properties from these possible key candidates we also need to take the density of a property into consideration.

The unique property combinations can also highlight characteristics of specific properties. E.g. while `dbpedia: spouse` is not unique for all persons, it is unique for athletes and scientists.

Furthermore, we evaluated the uniqueness in class hierarchies. Table 4 shows the uniqueness for two DBpedia properties along the person class hierarchy.

We found that there are properties that get more specific per class level, thus their uniqueness gets higher for subclasses (e.g. `dbpedia: team`). We also found properties that are generic, thus their uniqueness stays constant throughout the class hierarchy (e.g. `dbpedia: birthDate`).

Having these profiling results at hand helps users in finding minimal descriptive sets of properties and analyzing the uniqueness and relevance of properties in Linked Data sets.

Person	Scientist	Athlete
dbpedia:careerPoints	dbpedia:abstract	dbpedia:birthName
dbpedia:dateOfBurial	dbpedia:spouse	dbpedia:espnId
dbpedia:espnId	georss:point	dbpedia:firstWin
dbpedia:heir	rdfs:label	dbpedia:individualisedPnd
dbpedia:overallRecord		dbpedia:lastWin
dbpedia:restingPlacePosition		dbpedia:spouse
dbpedia:runningMate		georss:point
dbpedia:winsAtMajors		
dbpedia:winsAtProTournaments		
dbpedia:worldChampion		

Table 3: Unique property combinations (minimal) for some DBpedia person class clusters.

Ontology Class	Uniqueness for dbpedia:team	Uniqueness for dbpedia:birthDate
Person	0.31	0.20
Athlete	0.70	0.25
Soccer Player	0.91	0.31
Scientist	-	0.5

Table 4: Uniqueness for DBpedia properties dbpedia:team and dbpedia:birthDate along the class hierarchy.

6 Ongoing and Future Work

While there are a number of other interesting profiling tasks, the discovery of uniqueness in Linked Data sets is only a specific one.

In my PhD thesis I am compiling a set of profiling tasks for Linked Data sets. These profiling tasks are implemented using Apache Pig ⁵

Further ongoing work in my PhD thesis is the optimization of Linked Data profiling tasks. The process of computing profiling metrics for large data sets can take hours to days depending on the complexity of profiling tasks used and the size of the respective data sets.

A number of different approaches can be chosen when trying to optimize the execution time of algorithms dealing with RDF data in general and data profiling tasks in particular.

Algorithmic optimization: Profiling tasks that have high computational complexity cannot be computed naively, e.g. it is infeasible to detect property co-occurrence by considering all possible combinations of properties. Such metrics require innovative algorithms for computing the targeted result. If such an algorithm cannot be found, then approximation techniques (e.g. sampling) may be required. Because these algorithms are often highly specialized for a specific (profiling) task, they usually do not benefit other tasks. Algorithmic optimization of individual profiling tasks is *not* a goal; instead

⁵<http://pig.apache.org>, retrieved October 2013.

the focus will be on multi-measure optimization (see below).

Parallelization: When dealing with large data sets, a good approach for improving performance is to perform calculations in parallel when possible [12]. This can be done on different levels: Firstly, when multiple data sets are profiled, each data set can be profiled in parallel. Secondly, data profiling runs with multiple profiling tasks can make use of parallelism by executing individual tasks in parallel. Thirdly, if a data profiling algorithm allows the aggregation of profiling results for a subset of data into the overall result, then triples can be partitioned and processed in chunks. Finally, intermediate tasks required by a profiling algorithm, such as sorting or hashing, can be parallelized as well. Cluster-based parallelization based on MapReduce is a reasonable choice when working with Linked Data and thus considered essential.

Multi-Measure optimization: A data profiling run usually consists of a number of different tasks, which all have to be computed on the same data set. Depending on the set of data profiling tasks, different tasks may require the same preprocessing steps, or perform similar computation steps in order to reach their respective output. If this is the case, overall execution time can be reduced by avoiding duplicate computations. For example, if different tasks perform similar computation steps, it may be possible to interweave their execution and thereby reduce runtime and I/O costs. If different tasks require similar intermediate results, these can be stored in materialized views. Challenges herein include the identification and specification of the view structure, the optimization of algorithms to use existing views, and the maintenance of existing views.

Based on an ongoing master's thesis which investigates the problem of multi-measure optimization for data profiling tasks on Linked Data, optimization strategies will be discussed and summarized in a user guide. The optimization approaches will be implemented in ProLOD++. Furthermore, they are being evaluated using large Linked Data sets such as DBpedia. The evaluation compares the effects of different optimization strategies against each other and against the unoptimized version.

An analysis of data set characteristics and the implication for the choice of suitable approaches will be provided.

References

- [1] Ziawasch Abedjan, Toni Grütze, Anja Jentzsch, and Felix Naumann. Profiling and Mining RDF Data with ProLOD++. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE), Demo*, Chicago, IL, 2014.
- [2] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets - On the Design and Usage of void, the 'Vocabulary of Interlinked Datasets'. In *WWW 2009 Workshop: Linked Data on the Web (LDOW2009)*, Madrid, Spain, 2009.
- [3] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. Lodstats – an extensible framework for high-performance dataset analytics. In Annette Teije, Johanna Völker, Siegfried Handschuh, Heiner Stuckenschmidt, Mathieu d'Acquin, Andriy Nikolov, Nathalie Aussenac-Gilles, and Nathalie Hernandez, editors, *Knowledge*

- Engineering and Knowledge Management*, volume 7603 of *Lecture Notes in Computer Science*, pages 353–362. Springer Berlin Heidelberg, 2012.
- [4] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In *In Proceedings of the 5th European Semantic Web Conference*, pages 690–704, 2008.
- [5] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. OWL: Yet to arrive on the Web of Data? In *WWW2012 Workshop on Linked Data on the Web (LDOW)*, 2012.
- [6] Philipp Heim, Steffen Lohmann, and Timo Stegemann. Interactive relationship discovery via the semantic web. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, volume 6088 of *LNCS*, pages 303–317, Berlin/Heidelberg, 2010. Springer.
- [7] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. Scalable Discovery of Unique Column Combinations. In *PVLDB*, 2013. Available upon request.
- [8] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at <http://www.w3.org/TR/owl2-primer/>.
- [9] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Advanced Visual Interfaces*, 2012.
- [10] Gjergji Kasneci, Maya Ramanath, Fabian Suchanek, and Gerhard Weikum. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Record*, 37(4):41–47, 2009.
- [11] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 2013. Under review.
- [12] Felix Naumann. Data profiling revisited. *SIGMOD Record*, 42(4), 2013. to appear.
- [13] Leo Sauermann, Richard Cyganiak, Danny Ayers, and Max Völkel. Cool URIs for the Semantic Web. W3C Interest Group Note, W3C, December 2008. Available online at <http://www.w3.org/TR/cooluris>.

Using Design Patterns to Manage the Productivity vs. Performance Tradeoff in Hybrid Parallel Computing

Fahad Khalid

Operating Systems and Middleware Group
Hasso-Plattner-Institute
fahad.khalid@hpi.uni-potsdam.de

In the domain of High Performance Computing, the use of accelerator architectures has become common place. However, the cost of developing and maintaining codes for such architectures is very high, and implies low productivity. Therefore, employing hybrid architectures for high performance computing inherently leads one to the performance vs. productivity tradeoff.

This report presents a novel approach for improving the performance vs. productivity tradeoff for hybrid architectures. The proposed solution is based on the premise that not all algorithms are suitable for accelerator architectures. These algorithms, however, can be easily implemented for CPUs. Therefore, for a given computational kernel, an effective implementation should be distributed across the accelerator and the CPU in such a way, that the accelerator performs only those computations that are feasible for it, while the rest are offloaded to the CPU.

In previous work, the validity of this approach was shown for a specific problem from the domain of Computational Biology. This report proposes the road map for generalization of the approach. It is argued that the suitability of a computational kernel to a specific device architecture can be analyzed in terms of a combination of design patterns for parallel programming and other constraints such as arithmetic intensity.

1 Introduction

Hybrid computing architectures can be defined as those that in addition to the conventional CPU processors, employ accelerators for performance improvement. Employing accelerators can yield significant performance gains [11]; and consequently, hybrid architectures are now commonly used in HPC. According to the Top500 [15] list of the fastest super computers (published in June 2013), the top two positions are held by super computers based on hybrid architectures.

Graphical Processing Unit (GPU) is the most commonly used general purpose accelerator. Over the past few years, a multitude of computational kernels have been ported to GPUs with successful performance results. These results, however, come at a cost. GPUs are built on a massively parallel architecture with a large number of compute cores. However, as compared to the CPU, each GPU core supports a rather

simplistic feature set. Moreover, GPU caches are much smaller than that of the CPU. Differences like these limit the types of kernels that can gain significant performance improvement by a straight forward port to the GPU. For a large number of computational problems, a straight forward port to the GPU does not result in substantial performance improvement.

Even though productivity tools for GPU programming have improved markedly over time, the range of features provided by these tools is far narrower than those available for programming CPUs. Much of the code optimizations are still left to the programmer, which increases both the development and maintenance effort. This leads to the fact that even though it is possible to accelerate computational kernels using GPUs, it results in a considerable reduction in productivity.

1.1 Research Question

The arguments presented above lead to the following research question:

- *How can we adapt an algorithm to make effective use of the underlying hybrid architecture, given the following constraints:*
 - *There is a significant improvement in performance as compared to an optimized CPU-only implementation*
 - *The complexity of the implementation does not increase considerably as compared to that of a CPU-only optimized implementation*

The research question essentially points to the need for methods that make it possible to improve the *performance vs. productivity* tradeoff, i.e., we need methods that make it possible to leverage the performance potential of hybrid architectures while simplifying the development process.

Throughout the rest of the document, the term *effectiveness* will be treated as a two-dimensional artifact comprising of *efficiency* and *productivity*.

2 Existing Solutions

A major improvement in the programmability of NVIDIA GPUs was made possible by the development of the CUDA [12] programming model. Later on, the OpenCL [6] standard was developed in order to provide CUDA like capabilities for vendor neutral accelerator programming. These advancements further led to the following two major approaches for improving the effective use of hybrid architectures:

2.1 Accelerated Domain Specific Libraries

The most commonly used computational kernels are based on Linear Algebra, i.e., matrix and vector operations. Realizing this significance, the scientific computing and high performance computing communities started investing into libraries for linear algebra computation routines during the very early days of these fields. This resulted in

the development of BLAS [1] for basic linear algebra problem, LINPACK [8] for equation solvers and EISPACK [14] for solving Eigen problems. Initially, these libraries were available for distributed memory architectures only. However, once shared memory computing became mainstream, packages like LAPACK [2] were developed that implement previously available interfaces with data access patterns optimized for shared memory architectures.

Following this trend, linear algebra libraries have been developed that implement the same standard interfaces as provided by the above mentioned libraries, however, the computations are performed on accelerator architectures. E.g., CUBLAS [13] is built using the CUDA programming model and implements the BLAS interface for execution on GPUs. The CULA [7] and MAGMA [16] libraries go one step further and implement the computations that can (at least to some extent) take advantage of both the CPU and GPU.

The basic purpose of these libraries is to provide accelerated routines for linear algebra based computations. This hides the cumbersome implementation level details from the application developers, and provides them with a standard interface with which they are already familiar. Therefore, the application developers do not have to deal with the complications that arise from programming GPUs.

2.2 Compiler based code generation

The most commonly used parallelization framework up until the 1990s was the Message Passing Interface (MPI). MPI is used for communication between the nodes in a distributed memory architecture. Since the most powerful computers even today are clusters, MPI is still as popular and has become a de facto standard for high performance computing. Today, however, each of the nodes within the cluster is a shared memory parallel system. In order to fully utilize the resources within each of the nodes, the code executing on an individual node utilizes some form of multi-threading. OpenMP [3] is a standard that provides pragma based parallelization of code for shared memory architectures. All major C/C++ compilers provide OpenMP support, and it is the most popular programming model for shared memory architectures.

The pragma based parallelization approach used in OpenMP has recently been adapted for generating code that executes on a GPU. The OpenACC [17] standard defines a set of pragmas and associated routines. At the moment, the standard is supported by only a few major C/C++ compiler vendors.

This approach makes it possible for the programmer to concentrate on the application logic, without having to deal with the low level GPU specific code. Much of the more complex code is generated by the compiler, which uses the pragma directives as hints for parallelization. Even though the standard is not widely used in the mainstream HPC community, with time, the use of OpenACC is gaining popularity. The major drawback is that the compiler generates code for the GPU only, and is not capable of effectively utilizing the entire hybrid architecture. Further research is needed to see how the code generation approach can be used for effective utilization of hybrid architectures.

2.3 Research Gap

In terms of effective utilization of the underlying hybrid architecture, each of the above mentioned solution approaches has associated drawbacks. Following is a brief summary:

- **Accelerated Libraries:** The domain specific libraries mentioned above are very useful for scientific computing. However, these serve a very specific purpose and cannot be used as building blocks for problems in other domains. There is room for the development of libraries that provide mechanisms for coordination between GPUs and CPUs, and simplify the process of implementing a much larger set of problems on hybrid architectures.
- **Compiler Directives:** Not all kinds of computations are suitable for GPU architectures. Therefore, simply automating the process of code generation does not guarantee a significant performance gain. For certain computational kernels, even optimized GPU code result in wasted clock cycles. Therefore, it is important that code generation is complemented with libraries that make it possible to write *cross-device* optimized code.

3 Foundations

Note: From this point onward in the report, the term *Host* will refer to the CPU and the term *Device* will refer to the accelerator. The scope of this document is limited to the use of GPUs as accelerators. In future work, however, the methods developed will be applicable to other accelerators such as Intel's Xeon Phi co-processor.

In order to answer the research question presented in Section 1.1, it is argued here that a viable solution would have to satisfy the following conditions:

1. The application must be able to utilize all available processing resources across both the *Device* and the *Host*.
2. In order to do so effectively, it should be possible to structure the program flow in such a way that different parts of the kernel are executed by either the *Device* or the *Host* depending on which architecture is more suitable for that part of the kernel.

This raises the following question:

- *What characteristics of a computational kernel make it more suitable for one architecture rather than the other?*

The following Subsection will explore these characteristics, which will be used later to compose the proposed solution.

3.1 Important Characteristics of Computational Kernels

The *Host* architectures (i.e., CPUs such as Intel Core i7, Intel Xeon series etc.) are equipped with a feature rich Instruction Set Architecture (ISA), Vector processing units, and advanced features like branch prediction. This makes it possible for the *Host* to efficiently perform all kinds of computations. The major disadvantage as compared to *Device* architectures is the lack of massive parallelism.

Device architectures have the following major limitations:

- A high *Degree of parallelism* in a kernel is a fundamental requirement. If the degree of parallelism is not enough, compute cycles are not fully utilized.
- Cache size is much smaller on the *Device*. Therefore, kernels with low *arithmetic intensity* are not particularly suitable.
- *Device* architectures do not support branch prediction, and are bounded by a minimum number of threads that must execute the same instruction in a single clock cycle. Therefore, excessive *control divergence* in the kernel can lead to a significant amount of wasted cycles.

It is argued in this report that *Device* kernels for which the processing is dominated by one or more of the above mentioned limitations, can gain performance improvement if the appropriate parts of the kernel are executed on the suitable architecture.

3.2 Algorithm Decomposition

An algorithm can be decomposed based on architectural features if the following conditions are satisfied:

- At least two design patterns can be identified, where one pattern is suitable for the *Device* architecture, and the other one is suitable for the *Host* architecture. Each pattern can then be implemented as a phase of computation.
- The *pipeline* pattern is applicable across the two phases of computation.

4 Design Patterns and Algorithm Decomposition

The design patterns introduced by the Gang of Four [5] are considered as building blocks for a robust object oriented design. These patterns provide abstractions from the domain, and make it possible for design ideas to be used irrespective of the domain. These however are not sufficient for parallel programming. In order to fill this gap, *Design Patterns for Parallel Programming* [9] were explicitly proposed, and have gained considerable popularity over the years.

The design patterns for parallel programming can be used to identify the data and control flow in a computational kernel. This makes it possible to see whether a certain kernel would be suitable for a given architecture. The following subsections describe the patterns that are suitable for *Device* architectures, as well as those that are not suitable for *Device* architectures.

4.1 Patterns Suitable for *Device* Architectures

Some of the important patterns suitable for execution on the *Device* are:

- **Map:** Perhaps the simplest and most commonly used pattern that is ideal for *Device* architectures. The Map pattern can be seen as a fully unrolled loop, where each iteration of the loop is mapped to different processing unit of a massively parallel processor. There are no dependencies between the loop iterations, which implies that each processing unit reads input values and writes output values that are completely independent of all other processing units. The degree of parallelism is proportional to the number of loop iterations, which means that the hardware utilization is maximum for a large number of iterations.
- **Stencil:** Similar to Map, there are no output related dependencies amongst the processing units. This pattern is also similar to a loop that has no dependencies. In the Stencil pattern, however, a processing unit needs access to input values from the neighboring processing units as well. In fact, Map can be seen as a special case of Stencil with the restriction that each processing unit only reads its own corresponding input values.

Both these patterns map very well onto the massively parallel *Device* architecture. This is because each processing unit can perform computations completely independent of all other units. There is no need for elaborate locking and synchronization primitives, which could potentially slow down the computation.

4.2 Patterns Suitable for *Host* Architectures

As mentioned in Section 3.1, the *Host* architecture is suitable for all computational kernels. The only limitation is that massive parallelism is not available on these architectures. Therefore, it is not that the following are the only patterns suitable for the *Host*; these patterns are not particularly suitable for execution on the *Device* because the effective utilization of the massively parallel architecture is not possible. Therefore, it is argued that these patterns are better suited to execution on the *Host*.

- **Reduce:** This pattern is employed when a large number of values have to be reduced to a single value. E.g., a vector of 10 values is reduced to a scalar by summing all the elements of the vector. The pattern is more general in the sense that it can be applied to any data type on which an associative binary operator is defined. The most efficient method for parallel reduction follows computation in a tree structure. For a sum reduction, it starts with all the values at the leaves, and proceeds upwards to the root by applying the operator at each level, and halving the number of operands. Eventually, only the final scalar value is left at the root, which is the desired result. Figure 1 shows an example of a reduction tree with sum as the operator.
- **Scan:** An example of the Scan pattern is the prefix-sum operation. In this case, the output vector is the same size as the input vector. Therefore, the tree structure

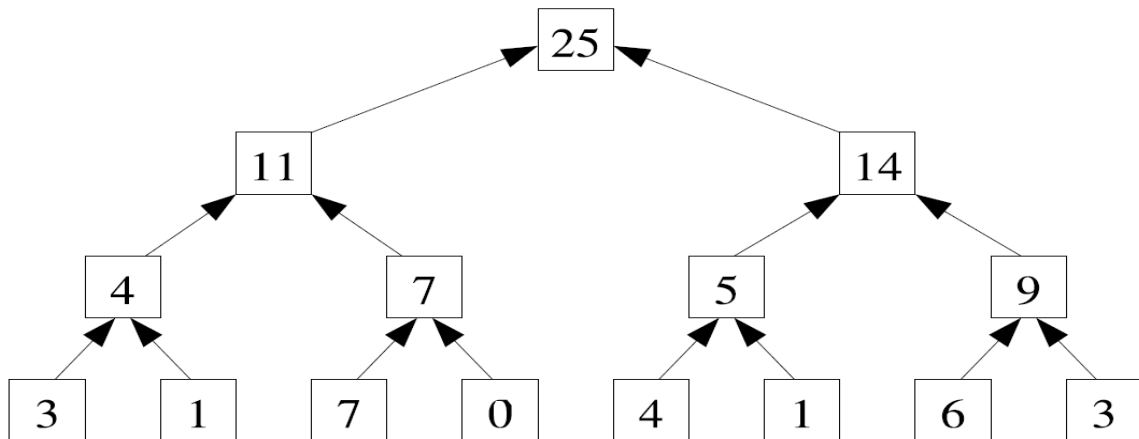


Figure 1: A tree structured sum reduction [4].

consists of two sweeps: 1) *Up-sweep* from the leaves to the root to compute partial reductions, and 2) *Down-sweep* from the root back to the leaves to complete the scan.

Both the Reduce and Scan patterns share the same problem when it comes to efficient resource utilization. The degree of parallelism varies at each step of computation. In Reduce, e.g., the number of processing units that can be used at the first step is equal to the number of elements in the vector. For a large vector, this amounts to maximum resource utilization. However, at each successive step, the number of processing units required is halved; eventually leading to the point where most of the compute resources are idle.

5 Hybrid Pipelining

If we have a computational kernel which comprises of patterns such that the kernel can be divided into two parts where one is a pattern suitable for the *Device*, while the other is a pattern suitable for the *Host*, each pattern can be implemented as a sub-kernel for the corresponding architecture. However, this distribution of code across the two architectures complicates the program control and data flow. A serial flow across the two devices would constitute the following steps:

1. Input data is transferred from the *Host* to the *Device*
2. The *Device* performs computation for a suitable pattern such as Map and produces results
3. The results are transferred from the *Device* to the *Host*
4. The *Host* applies the post-processing pattern such as Reduce

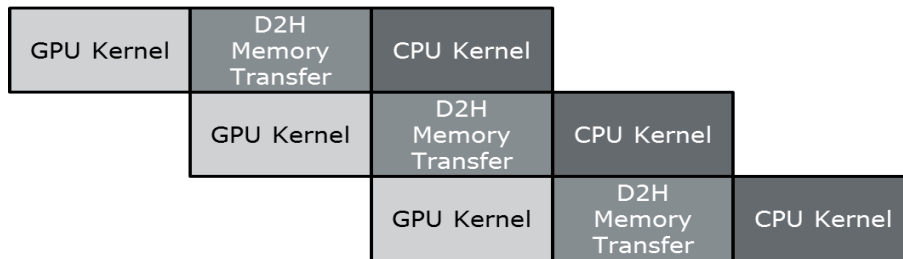


Figure 2: A 3-stage hybrid pipeline.

5.1 Big Data and the Pipeline Pattern

In business as well as science, the necessity to process large volumes of data is becoming more and more relevant with the passage of time. The above mentioned sequence of steps has to occur in a loop if big data processing is the objective. In order to achieve that efficiently, the above steps are executed as the following three stages of a software pipeline:

1. The *Device* kernel
2. The transfer of results from the *Device* to the *Host*
3. The post-processing *Host* kernel

The term *Hybrid Pipeline* stems from the fact that the pipeline spans two different processors across a hybrid architecture.

The hybrid pipeline approach was initially developed to solve a particular problem in the domain of Computational Biology. The details of the method as well as performance results can be found in detail in [10]. The method will not be discussed in further detail in this report.

5.2 Pipelining and Effective Utilization of Hybrid Resources

Consider the scenario where the *Device* stage of the pipeline implements the Map pattern and the *Host* stage implements the Reduce pattern. In this case, the *Device* code is very simple, since the kernel does not need to implement any special logic except the index algebra. Also, all the *Device* resources are utilized efficiently since the pattern is suitable for the architecture. Had the Reduce pattern been implemented on the *Device* instead of the *Host*, the code would have gotten considerably complex, and resource utilization would have been much lower than optimal.

The *Host* code is always much simpler to implement because of the various high level library and compiler based tools that hide much of the complexity of the algorithm and parallel programming details. E.g., the Reduce stage can easily be implemented using the Intel Threading Building Blocks Parallel Reduce function.

6 Ongoing Work

This report has so far presented a road map for generalizing the hybrid pipeline approach in the language of patterns for parallel programming. In order to realize the method, the following activities are in progress:

6.1 Generic Framework for Hybrid Pipelining

The available GPU programming models such as CUDA provide only primitive support for message passing between the *Device* and the *Host*. Moreover, there is no high level framework available to hide the complex concurrency mechanisms required to implement a hybrid pipeline.

A high level hybrid pipelining framework is being developed that would hide the *Device-Host* communication and complex concurrency mechanisms. This would make it possible for the application developer to implement a hybrid pipeline with minimal development effort; thereby, significantly increasing productivity.

6.2 Empirical Evaluation of Thresholds for Computation Intensity on the *Device*

The hybrid pipeline approach would yield performance only if the intensity of computation is balanced across the different stages of the pipeline. The amount of computation in the *Device* kernel must be large enough so that it can balance both the time it takes to transfer the result data from the *Device* to the *Host*, as well the computation time for the *Host* kernel. If the computation time for the *Host* is low enough, the algorithm distribution can result in a free computation phase, i.e., as compared to an equivalent *Device* implementation, the *Host* processing would be completely hidden behind the *Device* computation. The *Host* computation would only have a discernible impact on the overall execution time during the last iteration of the pipeline. This is because the last iteration executes in serial. Nevertheless, the total impact of the *Host* execution time would be negligible.

In order to thoroughly investigate and establish the thresholds that define balanced pipeline execution, experiments are being designed where kernels with different amounts of computation are tested for the impact on the overall pipeline efficiency. Optimization of several other parameters is also being considered, e.g., partition size for input data to be processed during one iteration of the pipeline, the amount of page-locked [12] memory to use, etc.

7 Summary and Future Work

It has been shown that looking at the problem from the point of view of design patterns in parallel computing can lead to effective utilization of hybrid architectures. Not only

does it highlight possibilities for performance improvement, it also improves programmer productivity. The ongoing work will result in the possibility to apply the method to a wide array of problem domains.

For new codes, the patterns and associated constraints can be used to decide how the algorithm can be effectively distributed across the architecture. This fits into the existing design strategy for parallel programs. The method presented in this report is just an added step to an established process.

For existing codes, an automated process will be introduced. A tool will be developed for the analysis of existing sources. This tool will provide recommendations on how the algorithm should be decomposed. An automated tool will require the representation of patterns and constraints in a formal language. Also, an efficient implementation of the hybrid pipeline model will require optimization of the control flow graph. Therefore, much mathematical formalism will be used in future work.

References

- [1] Basic linear algebra subprograms (BLAS). In David A. Padua, editor, *Encyclopedia of Parallel Computing*, page 120. Springer, 2011.
- [2] Edward Anderson et al. *LAPACK Users' guide*, volume 9. Siam, 1999.
- [3] Eduard Ayguade and Barbara Chapman. Introduction: Special issue: OpenMP. *Scientific Programming*, 11(2):79–80, 2003.
- [4] Guy E Blelloch. Prefix sums and their applications. In John H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 35–60. Morgan Kaufmann, 1993.
- [5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.
- [6] Khronos OpenCL Working Group. The OpenCL specification. Standard specification, December 2011.
- [7] John R. Humphrey, Daniel K. Price, Kyle E. Spagnoli, Aaron L. Paolini, and Eric J Kelmelis. Cula: hybrid gpu accelerated linear algebra routines. In *SPIE Defense, Security, and Sensing*, pages 770502–770502. International Society for Optics and Photonics, 2010.
- [8] Jack Dongarra et al. *LINPACK "User's Guide"*. SIAM Publications, 1979.
- [9] Kurt Keutzer, Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders. A design pattern language for engineering (parallel) software: merging the PLPP and OPL projects, 2010.
- [10] Fahad Khalid, Zoran Nikoloski, Peter Tröger, and Andreas Polze. Heterogeneous combinatorial candidate generation. In Felix Wolf, Bernd Mohr, and Dieter Mey, editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 751–762. Springer Berlin Heidelberg, 2013.

- [11] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 451–460. ACM, 2010.
- [12] NVIDIA. CUDA C programming guide. Design Guide PG-02829-001_v5.0, October 2012.
- [13] NVIDIA. CUBLAS Library. User Guide DU-06702-001_v5.5, July 2013.
- [14] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines- EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*, Editors: G. Goos and J. Hartmanis. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., second edition, 1976.
- [15] Erich Strohmaier. TOP500 - TOP500 supercomputer. In *SC*, page 18. ACM Press, 2006.
- [16] Stanimire Tomov and Jack Dongarra. Matrix algebra on gpu and multicore architectures. In *Proc. Workshop Electronic Structure Calculation Methods Accelerators*, pages 5–8, 2010.
- [17] Michael Wolfe. Introduction to GPU computing with openACC. In *SC'12 Tutorials CD-ROM: Conference on High Performance Computing Networking, Storage and Analysis*, Salt Lake City, UT, USA, November 2012. ACM SIGARCH/IEEE Computer Society.

High-Quality Video Generation for Thin Clients - An Application for Image-Based 3D Portrayal Services

Jan Klimke

Computer Graphics Systems
Hasso-Plattner-Institut
jan.klimke@hpi.uni-potsdam.de

1 Introduction

3D geodata, such as virtual 3D city models, is currently the basis for a broad range of applications in different areas, such as city marketing or urban planning. The increasing availability, volume, and quality of 3D geodata raises the complexity of 3D visualization applications using such data. Service-based 3D portrayal approaches, either image-based or based on geometry and texture streaming services, provide solutions for interactive exploration of large 3D data sets. But, even if the challenge of provisioning visualizations of large scale 3D models to heterogeneous hardware and software platforms was addressed in recent years, the complexity of navigation remains challenging for individual users. The communication of specific intentions (e.g., a positive impression of certain planning projects in an urban context) in connection with 3D geoinformation demands for specially designed camera paths, rendering effects (e.g., effects for highlighting, focus and context visualization [13], or real time ambient occlusion [1]), and transitions between specific sets of camera and scene configurations. Due to this complexity in operating a visualization system for end users, production of video presentations of large 3D models are still a usual way of communicating spatial information. Still images are not well suited to communicate sufficient amounts of spatial information since “the acquisition of spatial knowledge, essential for wayfinding, is primarily based on direct environmental experience, which is usually gained via movement” [5]. The advantage of preproduced video presentations is its compatibility (virtually any computing platform used by end users is able to play common video formats), the simplicity in playback (in terms of necessary hardware, software, and network resources for their transmission and playback), and the assured appearance of the 3D scene contained in the video presentation.

In recent years, several approaches have been presented for interactive presentation of 3D geodata in several scenarios, in stationary (e.g., on desktop PCs or notebooks) and also in mobile scenarios (interactive 3D visualization on hand-held devices, such as tablet PCs or mobile phones). Nevertheless, producing distributable artifacts for communication of 3D geodata, specifically video presentations, are currently mainly

a manual, and therefore, resource intensive task. Generation of video presentations usually includes involves two major steps: a) creation of planned flythrough sequences and b) post production and cutting of the assembled video presentation. An overview over the usual process can be found in Figure 1.

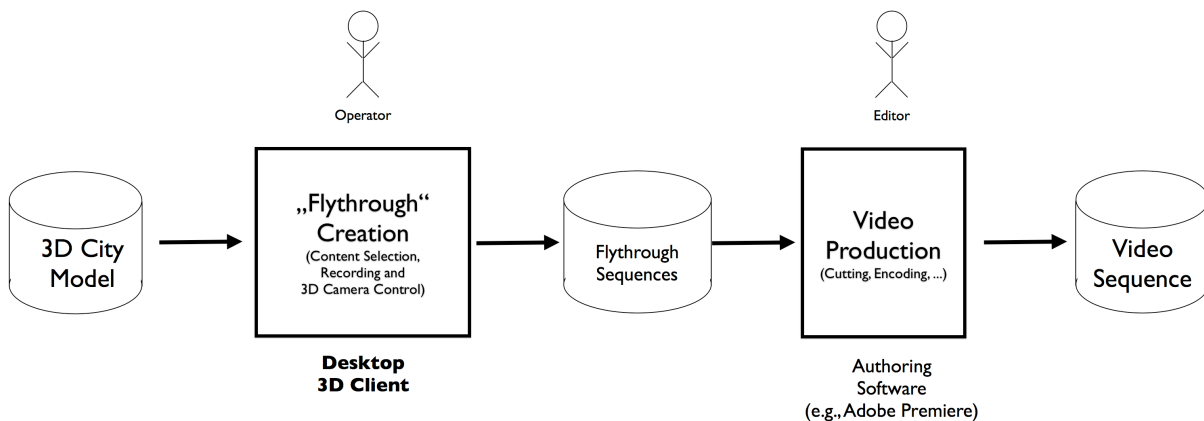


Figure 1: Overview over a conventional process for generating video presentations of 3D data. There is a lot of manual work involved. The virtual camera is controlled manually during video capturing for flythroughs. This makes the whole generation process costly and hard to repeat even if there are just minor changes in settings or scene content.

Since both of the two production steps involve high degrees of manual work, a repeated production with only minor changes in the 3D scene (e.g., an updated city model or a change in planning models) is an expensive task. We propose to automate the overall generation process by providing a web service that implements this process. The service takes a video description through its interface and automatically creates video presentations using image-based portrayal services, namely Web View Services (WVS) [6]. The video descriptions can be stored and reused, which facilitates to recreate videos, e.g., with equal cutting and camera path but different versions of the underlying data. Further, if underlying services are improved (e.g., new rendering techniques that provide an improved visual quality for 3D renderings) it is very easy to recreate video presentations with little or no costs, since no additional manual work for video clip recording or post production is necessary. In this way, we provide a high-level service that is able to provide additional value for owners of 3D city model data and proof that these Web View Services provide an important part of 3D geodata infrastructures [2].

The remainder of this report is organized as follows. Section 2 provides an overview over the work related work. Section 3 describes the designed system, its components, and the current implementation status. Finally, an outlook of planned future work concludes this report in Section 4.

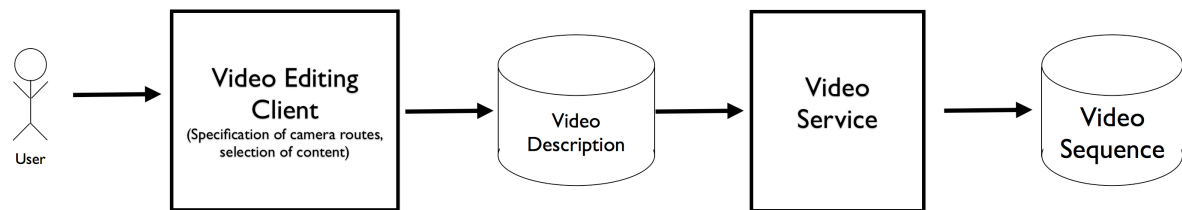


Figure 2: Overview over an optimized process for generating video presentations of 3D data. Video descriptions are created using a dedicated client application that also triggers the video rendering process. Descriptions can be stored and reused to automatically create video presentations using, e.g., extended or updated data or improved image-based portrayal services.

2 Related Work

Previously, we presented solutions to distribute 3D geoinformation to different devices and platforms (focused especially on web browsers and mobile devices) in homogeneous quality [3, 10]. These systems are based on a Web View Service [6] as image-based 3D portrayal service encapsulating processing, management, and 3D rendering of massive amounts of 3D geodata. Thin clients query interactive these services and provide partial reconstructions of the 3D scene on the client side. This way, users are able to interactively explore 3D geovirtual environments, which is important to gain understanding of the 3D information. Nevertheless, the systems introduced so far are missing a possibility to create artifacts that can be distributed easily. Currently, the only artifacts that can be created using these client applications are static images. For the work presented in this paper, we utilize the technology that has been developed before to build a video editing client that can be used to define video presentation descriptions using a variety of client platforms.

Currently the topic of automatic, distributed generation of video presentations from 3D data has primarily been addressed in the area of medical visualization. Iserhardt-Bauer et al. introduce a system that encapsulates rendering of datasets acquired from CT scanners [9]. They encapsulate hardware rendering behind a service interface in order to enable users to analyze the scan data without having to have specialized, powerful graphics hardware available in end user devices. Unless our approach, they do not support composite video presentations that are assembled from different, specifically user designed camera path sequences and auxiliary sequences containing text or images. The approach supports a standardized camera path only that cannot be customized. Further the approach is limited to the very specific data generated by CT scans. In contrast, our approach provides an abstraction over the underlying data by using service-based rendering for image generation. Roßler et al. extended this approach by using a conventional GPU-based PC cluster [12]. As like our architecture, they use a kind of manager process to perform dispatching and other supporting tasks.

Further, there are several approaches that are primarily targeted at bringing high-end graphics to low-end devices. E.g., Lamberti et al. perform remote rendering of complex 3D objects, e.g., containing several millions of voxels or textured polygons, and stream the result encoded as video to clients [11]. This approach leaves the configuration of camera parameters to clients, which are able to adjust the camera settings interactively. It is designed to support interactive rendering of video streams and does not provide a service interface for definition of complex video presentations as our approach does.

3 A Service-Based System for Automated Generation of Video Presentations

Generating video presentations of large scale 3D datasets is a resource intensive task. We system described in this section allows to decompose the task of video generation into reusable service components for definition of video presentations, image generation from 3D data, and video clip generation. The system is based on Web View Services (WVS) for image generation introduced earlier [7] that is currently in a standardization process within the Open Geospatial Consortium (OGC). Since we want the video service to be as responsive also during the generation process for single video presentations, we created an asynchronous process to generate video presentations. We separated the generation process into two services components: a) A frontend evaluating and validating service requests and b) a video rendering component that implements the more computationally expensive task of video clip generation. In this way, clients do not receive a video presentation as response to their service request, but the generation status is signaled using a callback mechanism (either via email messages or calls to a remote URL).

3.1 Service Components

There are five main components within the system (see Figure 3 for an overview of the framework architecture):

Video Editing Client The client application provides the user interface to specify video sequences including scene contents, camera paths, and the transitions between them. This way, we focus the user interaction on the one tasks of defining the complete video presentation instead of distributing it into a clip recording and a postproduction phase. Since the representation of video presentations in terms of the service interface (see Section 3.2 for details) does not necessarily correlate with the process of defining such presentations, the client application is built upon a cross platform library providing utility functions, e.g., for creating camera paths or defining transition screens. The client is able to store the video projects in a database allowing for reproduction and adaption of videos that have been created once.

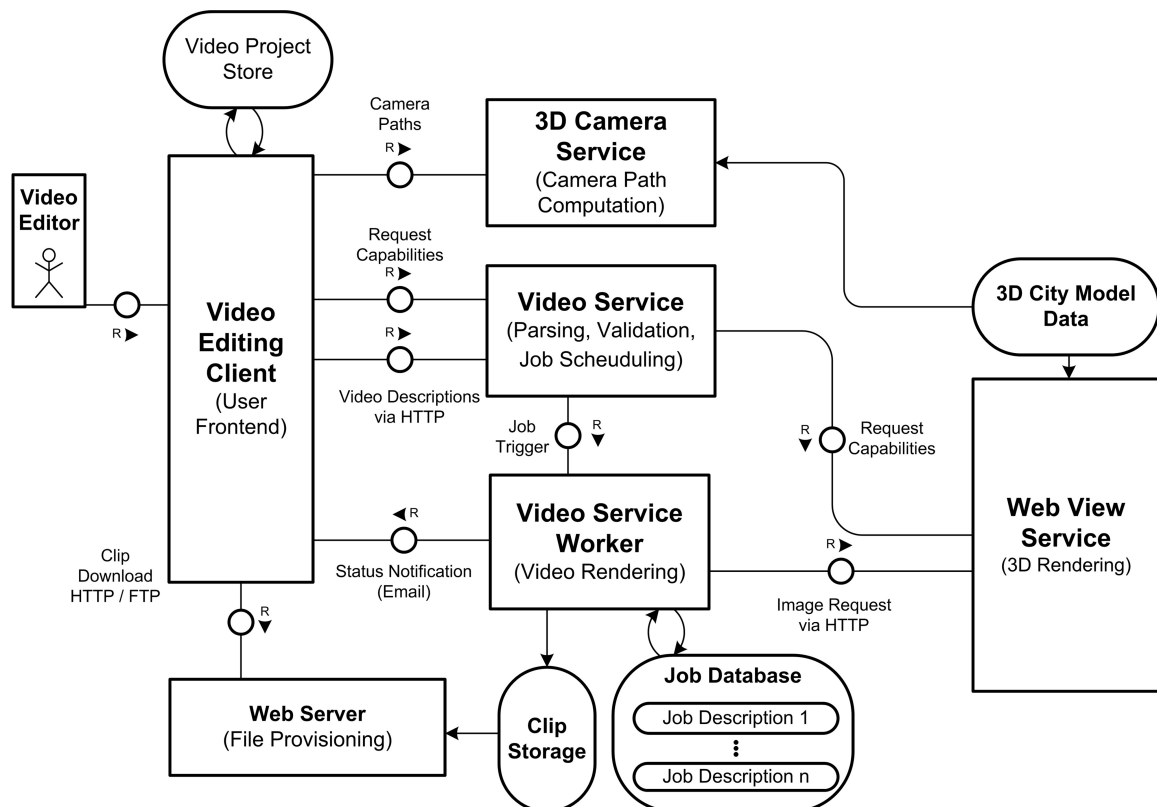


Figure 3: Components of the service-based system for video clip generation from 3D city model data. A video editing client is used to determine single clips, their camera paths (supported by 3D Camera Services) and scene content, and their interconnection. Project descriptions are stored in a database for easy altering and reproduction of video presentations. A Video Service instance utilizes Web View Services to retrieve single video frames and combines them to video sequences.

3D Camera Service 3D camera services are utility services in 3D geodata infrastructures encapsulating the complexity of 3D camera path computation in complex 3D environments such as virtual 3D city models [10]. The Video Editing Client can utilize such services in order to support users in defining visually appealing camera transitions. A 3D camera service is able to deliver proposals for such camera paths, which can be customized by video editors in order to fulfill their navigation intentions.

Web View Service Web View Services encapsulate image generation from 3D data. Each service provides portrayal capabilities for a specific set of data or for a specific spatial region. By utilizing this service-based approach for rendering of 3D geodata, the video service can be implemented without having to deal with the complexity of geodata visualization. Several WVS instances can be used in one video presentation, enabling the combination of data providers, each providing its own WVS instance with its own content. This allows to use 3D geodata for visualization purposes, without transferring geodata, which might be confidential or their access might be subject to a charge.

Video Service The Video Service serves as a frontend providing the service interface described in Section 3.2 to service consumers. Requests are parsed and validated by this component. If a valid request was issued, a job for the Video Service Worker is created and a confirmation is sent back to the service consumer. The video service can also implement further application logic for authentication of service consumers as well as functionality for billing.

Since video frame generation, especially for longer running video presentations, generates a large amount of load to underlying portrayal services. Here, a Video Service has the possibility to schedule video generation jobs to specific low traffic hours. Depending on the priority of the video service request, a generation task can be scheduled immediately or can be postponed to be processed later when enough service capacity is available.

Video Service Worker The worker performs the video rendering from single frames that have been previously fetched from a WVS (in case of images of a 3D scene) or generated by the service itself (e.g., overlay screens containing text or images). For camera paths, the camera position and orientation is interpolated for every frame from given camera paths. If two adjacent camera path sequences exist, the service is able to connect the two paths so that no visible jumps in camera paths are visible. Videos are often designed to communicate certain intentions. The worker service supports this by providing possibilities to show intermediate screen containing text and images as well as screen overlays that are shown over 3D flythrough sequences. Such overlays usually contain logos (e.g., for project specific logos or contractor logos). The frames or the necessary parts of them are rendered locally by the Video Service Worker. Since all 3D rendering techniques are implemented as web-based portrayal service, a worker implementation does not need hardware acceleration or any special hardware setup. This way, a Video Service Worker can be deployed easily, also as cloud-based service.

3.2 Service Interface

We define a model for describing video presentations that is used for generating through a `GetVideo` request allows to specify complex composite video presentations (see Figure 4 for an object model). A `Video` has certain settings that are valid for the overall presentation: The resolution of the video and a string defining the encoding (codec) to be used. Further there are default settings for a video presentation that can be overwritten by single sub elements of the document (e.g., specific point definitions or camera path sequences). The spatial reference system (SRS) and the default scene contents (defined as layer in WVS interface terms) are examples for such properties. A video presentations consists of one or more video `sequences`, that are single parts of the video presentation with an assigned duration. We distinguish two types of video sequences: `CameraPath` sequences are flythrough video clips that are based on a camera path, that can be defined either explicitly by providing path geometry or implicitly, e.g, by defining camera tasks like rotation around a scene object. The other kind of sequences are `Screens` that either show textual content or images for a certain duration. `Sequence` objects are connected by `SequenceTransition`

definitions. Here, the service needs to expose the types of sequence transitions it supports. There are two types of sequence transitions: a) Image based transitions, such as different blending techniques that can connect all types of video sequences and b) `CameraPathTransitions` that interpolate camera paths in order to connect them without having non continuous camera paths. The transitions are assigned implicitly to sequences using correlating indices.

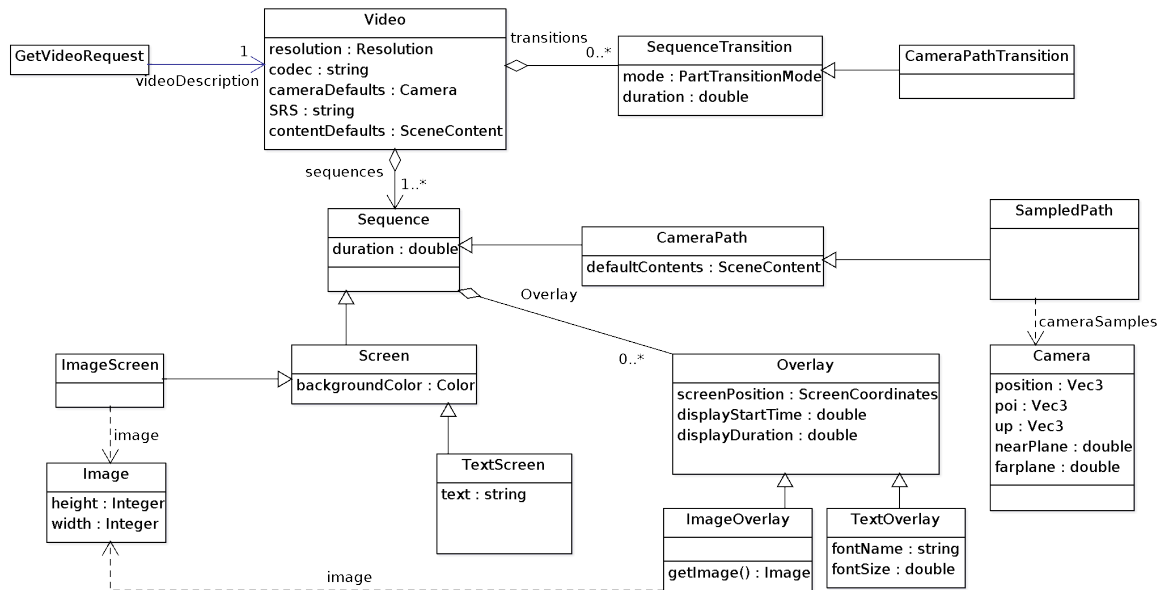


Figure 4: Excerpt of the object model for describing video presentations in a service request. A video contains one or more sequences, which may either be a camera path through a 3D scene or a screen (either an image or text). Two sequences are connected through sequence transitions that configure the blending properties and the interpolation between two adjacent camera paths.

3.3 State of the Work

Currently the implementation of the video generation system is work in progress. We implemented a first version of the video generation process including a basic Video Service and a Video Service Worker.

The next steps we are currently working on is to implement a platform independent client library that defines a process for defining video service descriptions. Our client is currently build for tablet PCs based on the iOS operating system. Its visualization component is following an image based approach introduced earlier based on the WVS we are running [3]. The base data we are currently working with is the virtual 3D city model of the city of Berlin, which is one of the largest, highly detailed, and fully textured 3D city models that are available world wide. At the moment, the `GetCapabilities` operation of the video service is not yet implemented, so the service is not yet self descriptive. The supported content layers are always dependent

4 Conclusions and Future Work

The implementation of a video production process that supports nearly arbitrary WVS image sources massively facilitates the generation of video presentations in the context of virtual 3D city model. The amount of manual effort, especially for regeneration of video presentations due to updated data, is reduced significantly through the automatic, repeatable generation of video presentations. Using image-based portrayal services to encapsulate complex 3D geodata management and 3D rendering provides a major advantage since a video service can be implemented in a generic manner using the well defined WVS interface specification, which is going to be an OGC standard for image-based portrayal of 3D geodata.

Currently image-based styling and post processing of frames is performed separately for each WVS that is queried for images. This can cause an inhomogeneous appearance, since there is no standard way of implementing and configuring image-based rendering effects, such as global illumination [4]. Therefore, as proposed by Hildebrandt [8], image-based styling could be externalized to a separate styling service that processes single frames using the same implementations for image-based styling for all sources.

The client for specification of 3D video presentation descriptions offers further possibilities to explore 3D interaction techniques that are specifically designed to assist users in specifying scene contents, 3D camera paths, and sequence transitions. Further, the client implementations as well as the video service implementation can be further generalized to configure themselves according to `GetCapabilities` documents delivered by WVS instances and the other services involved in the process.

References

- [1] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 3rd edition, 2008.
- [2] J. Basanow, P. Neis, S. Neubauer, A. Schilling, and A. Zipf. Towards 3D Spatial Data Infrastructures (3D-SDI) based on open standards - experiences, results and future issues. In *Advances in 3D Geoinformation Systems*, Lecture Notes in Geoinformation and Cartography, pages 65–86, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [3] J. Doellner, B. Hagedorn, and J. Klimke. Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps. In *Proceedings of the 17th International Conference on 3D Web Technology, Web3D '12*, pages 97–100, New York, NY, USA, 2012. ACM.
- [4] W. Engel, editor. *GPU Pro 4: Advanced Rendering Techniques*. Routledge Chapman & Hall, 2013.

-
- [5] N. Gale, R. G. Golledge, J. W. Pellegrino, and S. Doherty. The acquisition and integration of route knowledge in an unfamiliar neighborhood. *Journal of Environmental Psychology*, 10(1):3–25, March 1990.
- [6] B. Hagedorn. Web view service discussion paper, Version 0.6. 0. *Open Geospatial Consortium Inc*, 2010.
- [7] B. Hagedorn, D. Hildebrandt, and J. Döllner. Towards Advanced and Interactive Web Perspective View Services. In *Developments in 3D Geo-Information Sciences*, pages 33–51, Berlin/Heidelberg, 2009. Springer.
- [8] D. Hildebrandt. Towards Service-Oriented, Standards- and Image-Based Styling of 3D Geovirtual Environments. In *Proceedings of the 5th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering*, pages 133–147, 2010.
- [9] S. Iserhardt-Bauer, P. Hastreiter, T. Ertl, K. Eberhardt, and B. Tomandl. Case study: Medical Web Service for the Automatic 3D Documentation for Neuroradiological Diagnosis. In *VIS '01 Proceedings of the conference on Visualization '01*, pages 425–428, Washington, DC, 2001. IEEE Computer Society.
- [10] J. Klimke, B. Hagedorn, and J. Doellner. A Service-Oriented Platform for Interactive 3D Web Mapping. In M. Jobst, editor, *Service-Oriented Mapping*, pages 127–139, Vienna, 2012.
- [11] F. Lamberti and A. Sanna. A streaming-based solution for remote visualization of 3D graphics on mobile devices. *IEEE transactions on visualization and computer graphics*, 13(2):247–60, 2007.
- [12] F. Rößler, T. Wolff, S. Iserhardt-Bauer, B. Tomandl, P. Hastreiter, and T. Ertl. Distributed Video Generation on a GPU-Cluster for the Web-Based Analysis of Medical Image Data. In Kevin R. Cleary and Michael I. Miga, editors, *Medical Imaging 2007: Visualization and Image-Guided Procedures*, pages 650903–650903–9, March 2007.
- [13] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 3rd revised edition, 2012.

Ultra Mobile Devices: using the user's body as an interactive device

Pedro Lopes

Human Computer Interaction Group, Prof. Patrick Baudisch
Hasso-Plattner-Institut
pedro.lopes@hpi.uni-potsdam.de

We present a new type of wearable eyes-free interactive devices that use the user's body itself for input and output. Going beyond recent output-only muscle interfaces, we have created a system that reads the state of the user's muscles using electromyography (EMG) and at the same time actuate the same muscle using electrical muscle stimulation (EMS). We achieve this using time-division multiplexing with a sample hold circuit. Based on this technology, we are creating a series of simple sub-wearable input/output devices. The benefits of embodied devices include (1) the wearable form factor without visible input or output component allows users to interact with a computer system in an inconspicuous way, (2) unless used, users' hands are free, allowing for micro-interactions; always available, (3) Exploiting the flexibility of the user's arm, we can embody several specialized devices, without the need to carry multiple devices, and (4) good affordance resulting from input and output taking place in the same unified space, allowing us to use the same gesture language for input and output.

1 Introduction: why haptics matter?

For a long time, the key to immersion in interactive experience and games was sought in photorealistic graphics [1]. More recently, game makers made games more immersive by requiring players to physically enact the game such as with Wii and Kinect. With graphics and user interaction now part of many games, many researchers argue that haptics and motion are the next step towards increasing immersion and realism, i.e., applying the forces triggered by the game onto the player's body during the experience.

While some game events can be realistically rendered using one or more vibrotactile actuators (e.g., driving over gravel in a racing game [3]), a much larger number of gaming events result in directional forces, such as centrifugal forces pulling at a steering wheel or a car bumping into the railing. Such events have been simulated using motion platforms actuated by motors and mechanics for stationary installations (such as found in theme parks and research labs) or, for mobile situations, are simulated using robotic exoskeleton or other mechanical contraptions [7], as found in rehabilitation and gaming exoskeletons.

Unfortunately, the size and weight of these devices tends to be proportional to what they actuate. As a result, motion platforms not only tend to be prohibitively expensive, large, heavy and thus stationary, limiting their use to arcades and lab environments.

Further, exoskeletons draw similar disadvantages because are also based on motors and mechanical contraptions, thus are heavy and need to be installed on the user, covering large portions the body. My research explores how to achieve similar effects using mobile hardware, with smaller footprints.

2 Towards ultra-mobile interactive devices

We present a new type of wearable eyes-free interactive devices that use the user's body itself for input and output. Going beyond recent output-only muscle interfaces, such as the possessed hand [6], we have created a system that reads the state of the user's muscles using electromyography (EMG) and at the same time actuate the same muscle using electrical muscle stimulation (EMS). We achieve this using time-division multiplexing with a sample hold circuit. Based on this technology, we are creating a series of simple sub-wearable input/output devices. We envision these to be implemented in the form of a wristband-shaped device that users wear invisibly at the forearm close to the elbow that communicates with the user's arm muscles using a set of electrodes. Users invoke the device by performing an activation gesture, such as overextending the hand or even posing the hand directly as the intended device. Our embodied video player device, for example, responds by actuating the user's muscles so as to form a fader that moves automatically as the video plays. Users operate scrub the video back and forward by moving the exact same muscle, and pushing it back and forth to adjust the video play position. Users dismiss the device by performing a deactivation gesture.

The benefits of embodied devices include (1) the wearable form factor without visible input or output component allows users to interact with a computer system in an inconspicuous way, (2) unless used, users' hands are free, allowing for micro-interactions; always available, (3) Exploiting the flexibility of the user's arm, we can embody several specialized devices, without the need to carry multiple devices, and (4) good affordance resulting from input and output taking place in the same unified space, allowing us to use the same gesture language for input and output.

2.1 Related Work: eyes-free mobile input is not-hands-free

To eyes-free interact with mobile technology, researchers have explored different approaches: from spatial touch exploration of the palm [2] to muscular contractions [5]; thus, accounting for hands-free situations. However, the response they get back from the system comes through a rather unfortunate output channels, such as the visual display on a smartphone, auditory, or through vibrotactile actuators. These channels are not always available in mobile situations, such as crowded spaces and while walking. This means that, currently, there is a disconnection between input and output modalities when it comes to mobile interactions.

Roudaut et. al showed how to unify the input and output channels for a touch-enabled mobile device [4]. Their prototype, Gesture Output, uses a unistroke alphabet for input and output, the latter rendered through an actuated touchscreen that moves the user's finger. Unfortunately, such approach is not hands-free, since, to get an answer from

the device, users need to be in contact with the mobile phone, through the actuated surface.

In our research, we push the boundary of hands-free mobile interaction, by purposely removing any external device, such as the smartphone, and instead appropriating the users' muscles as truly ubiquitous input and output devices. Therefore, the same modality—poses—is employed for interacting with the system, or getting an answer from the system.

2.2 Embodied Devices: on-body input and output

We present what we call embodied devices, i.e., interactive input/output devices that communicate with the user by reading from and writing to the same muscle group. In Figure 1: (a) the user controls presentation software. To rewinds the video (b) the user invokes an em-bodied scrubbing device by performing an invocation gesture (here over-extending the palm). The faded device is loaded and it responds by posing his hand so as to reflect the current position in the video (and updates this pose in real-time as the video advances). (c-d-e) the user now rewinds the video by actively posing his hand so as to reflect the desired video position, here bending it backwards. In the end, satisfied with the result, the user dis-misses the scrubbing device by performing a dismiss gesture, here over-extending his palm again. Our device that implements the embodied devices is typically worn invisibly under a sleeve.

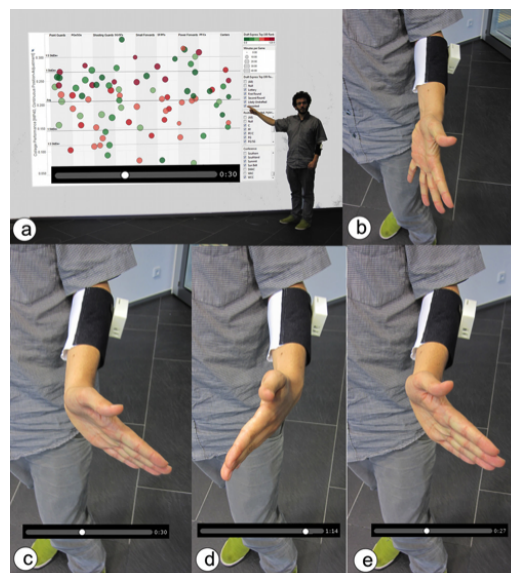


Figure 1: Using the embodied device (slider) to control the playback of a video during a presentation.

We further, by proposing a framework several embodied devices: joystick, slider (shown in example above), combo-box, dial, and toggle. All together they provide truly ubiquitous hands-free I/O interactions, which can take place in the most extreme mobile conditions, such as walking or running.

3 The challenge of closing the I/O loop: Integrating actuation and sensing

For sensing we use an medical-grade Electromyography (EMG) circuit. For actuating we use medical-grade EMS circuitry used in our previous publication. Given the difference in magnitude between the amplitudes of typical EMS-based actuation (between 10 V to 30 V) and sensing (around 1 mV to 5 mV), concurrent stimulation and sensing read through the same electrode pair is not trivial. The consequence of a simultaneous read-write is that the potential required to actuate a muscle through the skin damages the precision differential amplifiers found in EMG sensors for reading.

Therefore to achieve concurrent muscle input and output, the proposed approach is to switch back and forth between stimulation and sensing states. This is accomplished using a custom built time-division multiplexing (TDM) circuit with sample holds per channel, based on as many channels as needed of optically isolated transistors (commonly optocouplers) which are interleaved at 40 Hz by an arduino nano microcontroller, minimal C code for pin switching code allows for time-wise stability. The circuit was simulated using CircuitLab and is shown in Figure 2, with only one reading and sensing channel for simplicity.

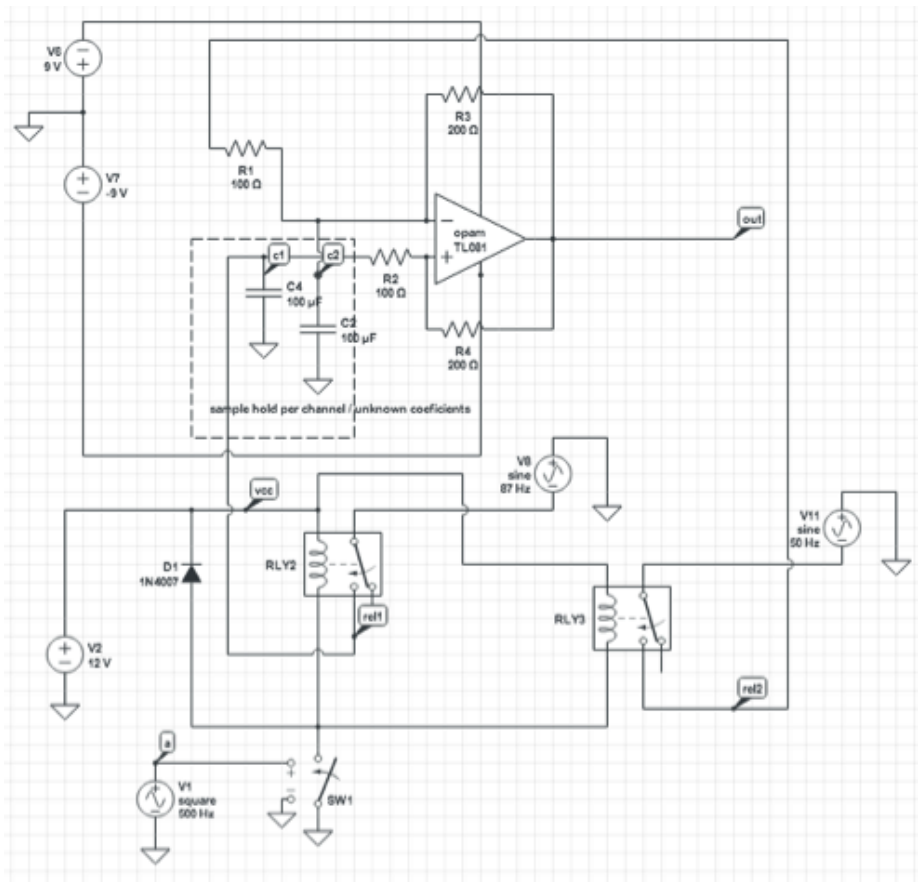


Figure 2: Simulated Time-domain Multiplexing circuit using CircuitLab, notice the highlighted capacitors for sample-hold feature (dashed rectangle).

Given that sensing requires a smaller time-window than stimulation, we opted for an unbalanced distribution of the time slots: stimulation is on for 80% of the period (i.e., ns) and sensing for only 20%, as depicted on Figure 3. Furthermore, given that voltages current dissipates over skin and muscles, we add a safety 100 ns gap between the uptime of each channel.

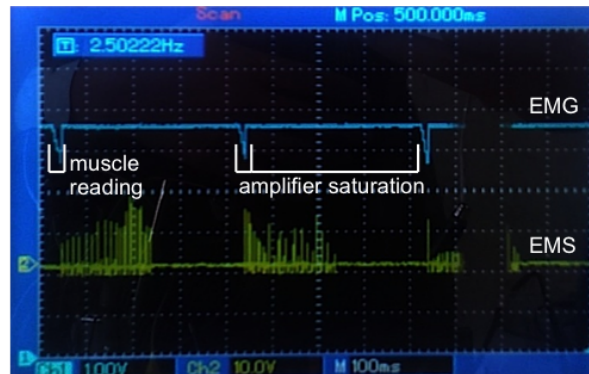


Figure 3: Realtime scope-view of multiplexing windows, for sensing and actuation.

3.1 Input using Electromyography

In order to achieve control over the user’s hand pose, the muscle configuration of the hand pose must be known. We achieve this by sensing the muscle contractions with Electromyography (EMG), and inferring the pose from a trained data model.

In order to achieve control over the user’s hand pose, we use a custom op-amp based Electromyography sensor. To read the potential of a muscle contraction, three electrodes are used, two at the extremities of the muscle, and the third close to a neutral area, preferably bone. The circuit will pick up the difference in voltage from the two electrodes attached to the muscle region, and use the electrode neat the bone as a reference signal (the zero). The hardware design is comprised of four stages: three operational amplifiers with differential design; and a rectifier. The gain on the rectifier allows for a fine calibration of the sensor’s smooth coefficient at the rectification stage. The op-amps are powered using two sources +9 V and -9 V (inverted). Sensor output is sampled using a arduino nano microcontroller.

3.2 Discussion

Naturally the concept of embodied devices needs not to be restricted to the hands. The whole body presents interesting muscle groups for rendering embodied devices. For instance, Wagner et al. identified 18 places of interest for on-body interaction [8]. In Figure 4 we show two designs of envisioned embodied devices for the biceps and the feet. These additional possibilities open up interaction spaces for scenarios in which the hands are occupied with another task.

Pursuing a whole-body exploration of embodied devices, which includes heavy limbs, such as the whole arm or legs, requires new developments on the actuating

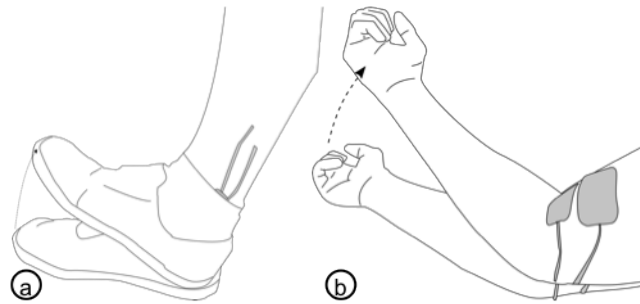


Figure 4: Additional form factors: (a) pivoting the feet up and down as an on/off switch, and (b) a lever formed by raising the forearm through biceps flexion.

technique, since EMS-based actuation is currently limited to strong muscles with lighter loads (fingers, hands, biceps), i.e., it is hard to control the lift of a heavy limb such as the upper and lower leg.

4 On-going work and projects under submission

- Embodied devices: using the body for input/output (*working title*), Pedro Lopes, and Patrick Baudisch
- HapticTurk: Mobile Force Feedback Based on People (*working title*), Lung-Pan, Pedro Lopes, and Patrick Baudisch

We proposed the concept of Embodied Devices–I/O devices that are shaped out of the user's hand muscles, using electrical muscle stimulation. With this approach, extreme mobility is achieved because the same piece of hardware is able to render multiple devices on the fly. Through a user study we will assess if the unification of I/O channels reduces the translation step between input and output vocabularies.

We demonstrated how to effectively close the loop on body actuation by simultaneously sense and actuate the user's muscles, using time-multiplexed electromyography and electrical stimulation with sample hold circuitry.

As future work we are plan to explore how other parts of the body might suit for interaction using embodied devices.

5 Acknowledgments

We would like to acknowledge our colleague Uwe Henschel for his input on analogue circuitry.

References

- [1] Andreas Gaggioli and Ralf Breining. Perception and cognition in immersive virtual reality. In *Emerging Communication: Studies on New Technologies and Practices*, pages 71–86.
- [2] Sean Gustafson, Christian Holz, and Patrick Baudisch. Imaginary phone: learning imaginary interfaces by transferring spatial memory from a familiar device. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 283–292, New York, NY, USA, 2011. ACM.
- [3] Ali Israr and Ivan Poupyrev. Tactile brush: drawing on skin with a tactile grid display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2019–2028, New York, NY, USA, 2011. ACM.
- [4] Anne Roudaut, Andreas Raus, Christoph Sterz, Max Plauth, Pedro Lopes, and Patrick Baudisch. Gesture output: Eyes-free output: Using a force feedback touch surface. In *Proceedings of the 2013 annual conference on Human factors in computing systems*, CHI '11, pages 543–552, New York, NY, USA, 2013. ACM.
- [5] T. Scott Saponas, Desney S. Tan, Dan Morris, Ravin Balakrishnan, Jim Turner, and James A. Landay. Enabling always-available input with muscle-computer interfaces. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, pages 167–176, New York, NY, USA, 2009. ACM.
- [6] Emi Tamaki, Takashi Miyaki, and Jun Rekimoto. Possessedhand: techniques for controlling human hands using electrical muscles stimuli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 543–552, New York, NY, USA, 2011. ACM.
- [7] Dzmitry Tsetserukou, Katsunari Sato, and Susumu Tachi. Exointerfaces: novel exoskeleton haptic interfaces for virtual reality, augmented sport and rehabilitation. In *Proceedings of the 1st Augmented Human International Conference*, AH '10, pages 1:1–1:6, New York, NY, USA, 2010. ACM.
- [8] Julie Wagner, Mathieu Nancel, Sean G. Gustafson, Stephane Huot, and Wendy E. Mackay. Body-centric design space for multi-surface interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1299–1308, New York, NY, USA, 2013. ACM.

Adaptive Optimizations for Data Structures in Virtual Runtime Environments

Tobias Pape

Software Architecture Group
Hasso-Plattner-Institut
Tobias.Pape@hpi.uni-potsdam.de

Implementing data access semantics of programming languages that focus on simplicity and purity often requires non-trivial compromises to achieve good performance. While approaches exist to alleviate the penalties of certain data access semantics, these either impact the simplicity and purity of the language design or require static adjustments to language implementations or programs written in them. We present an approach to improve the performances of data access dynamically at runtime that is based on automatic, transparent data structure inlining. Initial benchmarks suggest that our approach is a viable solution.

1 Data access: design *versus* performance

One of the key goals of programming language implementations is to be fast. Among others, data access is an important part of language design and implementation that needs attention to ensure good performance. While the design goals of languages may include simple, consistent, and pure semantics for data access, it might not be possible to implement them in a straight-forward way so that data access is actually fast. Techniques such as just-in-time (JIT) compilation, method inlining, or meta-tracing are used to speed up language implementations, however, performance gains with respect to data access are often only achieved when either sacrificing simplicity in the language design or correctness in the language implementation.

This is more severe the more machine model and language semantics differ. Functional languages such as ML [9] or Lisp often focus on simple, pure data semantic: Lisp's fundamental data structure is the linked list built from a simple data structure with just a data field and a next pointer. Long lists simply consist of many of those interlinked cells. More complex data structures have the same basic foundation. This differs from the common, prevalent machine model that favors contiguous memory, with rare exception like the Lisp machines [7] that were built to follow the language and not vice versa. This mismatch is accounted for by either changing the language, changing programs, or implementing the language with non-trivial compromises.

We argue that it is possible and viable to retain simple and pure data access semantics in a programming language and at the same time optimize data access dynamically at runtime with less compromises necessary.

2 Data access in language implementations

Our approach is based on the notions of data access semantics and data access implementation in the context of programming language implementations. This section introduces these concepts, traditional optimization techniques and establishes our working example.

2.1 Data access design and semantics

Language designers and implementors have to make compromises on different levels to ensure consistency in language semantics and appropriate performance when implementing those semantics.

As an example, the string data structure in Smalltalk is conceptually, merely a collection of characters. Yet, in all relevant implementations, a string is not represented in memory as a collection of character objects, but a separate structure. For an implementation the memory requirement for the intuitive collection-of-characters variant are, for each string, an object to represent the collection and one for each individual character. Compared to an opaque C string that merely provides collection-like semantics, but is otherwise handled by specialized routines, the naïve variant traditionally is both slower and more space-intensive. Hence, Smalltalk as a language [6] prefers the more efficient variant to the more “pure” one, although purity has always been one of the key goals of Smalltalk.

This is an instance of the “worse-is-better” approach [5], i.e., preferring implementation simplicity and performance to absolute correctness and completeness. Similar compromises can be found in other languages that otherwise aim to be “pure” in the above sense, e.g., in Scheme [1] and other Lisp dialects. In fact, only few languages, if at all, support the idea of strings being actual collections of first class characters. Generalizing this observation, language designers can

- choose pure, consistent data access semantics and accept potential performance penalties in language implementations,
- provide specialized data access semantics for every possible case (e.g., special semantics for strings, collections, fixed collections like arrays, objects, numbers and so forth) thus enabling implementors to always choose the fastest implementations, or
- compromise between both and only break data access semantic purity in special cases, e.g., for strings.

2.2 Current optimization strategies

Given data access semantics, language implementor can optimize data access in several ways, with the most predominant ways following.

Should the semantics permit it, language implementors can provide different specialized data access mechanisms for different data types. The best known example from

this category might be fixed-sized arrays known from most imperative languages, e.g., C as a language even requires arrays fixed in size and specialized in their containing data type. For virtual machine (VM) based languages, Java exhibits a similar behavior.

If language semantics require data access that would otherwise be slow when implemented naively, language implementors can choose to present the said semantics to the language level while pursuing a more efficient implementation *hidden* from the language level. As long as no direct access to data is made, the hidden, high-performance form can be retained and upon direct access, the semantically correct form is *reified* from the hidden form and presented to the language level. An example for this behavior can be found in the Cog Smalltalk VM [10], where the access to stack objects is done this way. Reified data structures may introduce caching-like problems: when do the hidden data change, when to update the language level data and so forth.

Language implementors can chose to not adhere to specified data access semantics at all and just implement the fastest possibles way of data access. Obviously, this can introduce incompatibilities between different implementations of the same language.

Either way, given sufficiently strict language semantics for data access, language implementors have to sacrifice either performance or correctness.

2.3 Working example characteristics

For the rest of this paper, we use a working example to clarify our solution. As a foundation, we assume an ML-like [9] language, i.e., functional with (by default) *immutable* data. As with most functional languages, we assume that lists use the head–tail representation, i.e., data access on lists is defined as:

- the first element of a list can be accessed directly (head or *car*¹)
- the rest of the list (all but the first element) can be accessed as a list (tail or *cdr*)
- the rest of t the last element of a list is the empty list or *nil*.

This is equivalent to saying that lists are simply linked list, with the payload in the first and the next-pointer in the second field of a two-field cell (commonly called *cons* cell). Hence, a simple n-element list in our example takes the form as can be found in Figure 1. Note that this is indeed the representation that most closely conveys the languages semantics, but also is the most inefficient one, especially with respect to random access.

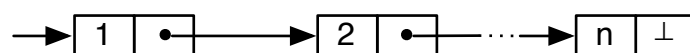


Figure 1: Languages like Lisp or ML represent collections as linked *cons*-cells

¹cell, *cons*, *car*, and *cdr* are the Lisp names of those concepts

3 Stable adaptive data structure inlining

Our approach to increase the data access performance for the scenario outlined previously is twofold. First, the combination of inlined data structures and a meta-tracing JIT compiler eliminates operations necessary to access nested data structures; the inlining rules are adaptive. Second, the warm-up time typically present in adaptive optimization is mitigated by re-using profiling information of previous executions.

3.1 Data structure inlining

We combined the recognition of patterns in data structures [4] with meta-tracing just-in-time (JIT) compilers [2] to provide a semantically correct but also fast means of data access that works for highly nested data structures. The *structure* of a data structure instance or *cell* (cf. section 2.3) is stored internally as its *shape* ($\langle \square \rangle$) and shared among equally structured cells, as depicted in Figure 2. Often occurring patterns of nested data structures are recognized and subsequently replaced with internal data structures that use longer, array-based storage. Individual cells are *reified* upon access. The overhead imposed by this recurring allocation is mitigated by employing a meta-tracing JIT compiler.

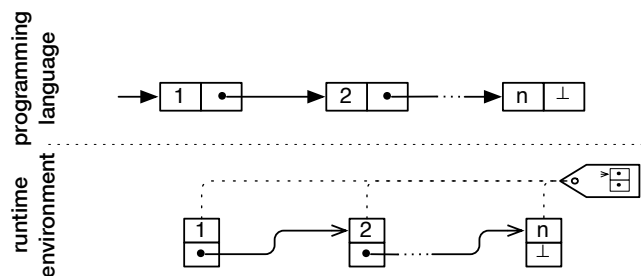


Figure 2: Data Structures and shapes. Top: the language view; bottom: the default implementation view with shapes

Without loss of generality, we consider cells to comprise *storage*, i.e., an indexable piece of memory, and a *shape*. A *shape* essentially describes a (possibly nested) cell; it comprises the abstract, structural representation of the cell a shape describes (the cell's **structure**), a mapping $Index \times Shape \rightarrow Shape$ describing what new shape this shape can be replaced with when a certain shape is found at a certain index in the cells storage (the **transformation rules**), and profiling data built up during cell creation to aid the creation of new transformation rules (the **history**).

A second type of shapes denotes that for a cell such shape describes no modifications have been made. Hence, these for these cells the language level access semantics apply directly; we denote this by a \bullet for every cell field in the cells structure as in Figure 2. These direct access shapes are commonly the leaves of a shapes structure tree. Moreover, every *type* of cell has a default shape, a direct access shape. In Figure 2, the default shape for a two-element cell is shown, denoted by the two \bullet .

3.1.1 Data structure merging

Under the assumption of immutable data (cf. section 2.3) data structures can only be altered by creating new data structures. With this premise, our optimization technique works by inlining cells upon their creation.

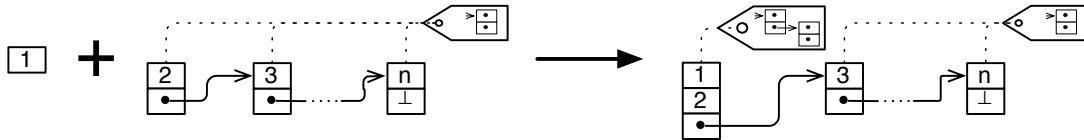


Figure 3: When creating a new cell that should contain “1” and the list as shown, a new cell that merges the “1” with the “2” cell and a different shape is created instead.

When a new cell is to be created, we handle the default shape s for the cell type and the cell contents c as specified in algorithm 1. E.g., for the first cell from Figure 2, the contents would be “1” and the rest list, and the default cell would resemble “s1” from Figure 4. The simplified process is shown in Figure 3. We iterate over the cell contents, and for each new child, we look up a possible new shape in the shape’s transformation table. First, we encounter “1” at position 0. For this example, we say that the transformation table does not contain a mapping for “1” at position 0, thus s' will be s and we continue with the next index.

At position 1, we find the rest list, whose shape is “s1”. In the transformation table, the entry for “1, s1” holds a replacement shape, “s2”. Thus, we *splice* the immediate cell storage of the rest list into the current one as c' , which now has three elements. Note that not the shape of the rest list $child_{\{shape\}}$ is changed but rather the shape of the cell to-be-created s . To allow for further transformations, we rewind the running index and start over, and repeat this until no new transformations can be found. The transformed shape and the altered storage are then used to create a new cell. If every cell would be inlined up to the maximum cell size $m+1$, the list would look like Figure 5.

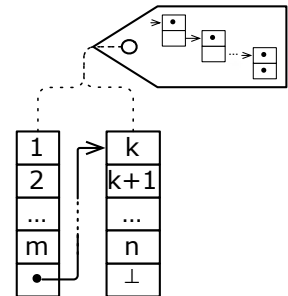


Figure 5: The list, m times inlined

Transparent child access Access to a cell’s children is reified, i.e., new objects are created when children of cells are accessed. This technique bears similarities to dynamic deoptimization [8], which is used in JIT compiled code to re-created original code during debugging. In the case of the example, accessing the tail of the first element of the list, would create a new cell with contents similar to Figure 5, however, starting at “2” rather than at “1”.

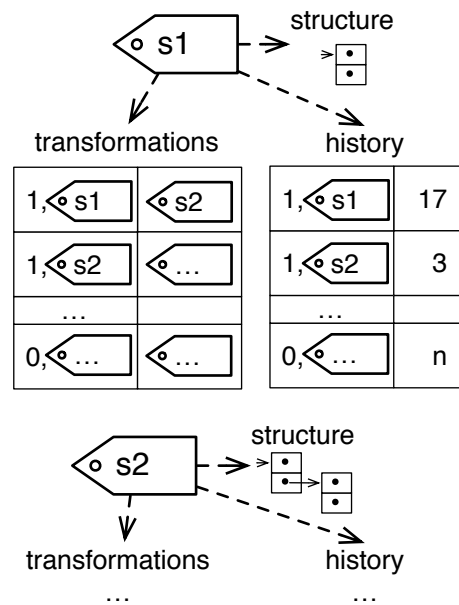
3.1.2 Meta-tracing

While this approach on its own is suboptimal due to the numerous allocation during the merging process and the reification step, when combined with a meta-tracing JIT compiler the performance is substantially better. The JIT compiler can recognize accesses

```

Input:  $s : Shape, c : [Cell]$ 
 $i \leftarrow 0$ 
while  $i < |c|$  do
   $child \leftarrow c_i$ 
   $s' \leftarrow S\{transformations\}_i, child\{shape\}$  OR  $s$ 
  if  $s' \neq s$  then
     $c' \leftarrow [c_0, \dots, c_{i-1}, child\{storage\}, c_{i+1}, \dots, c]$ 
     $s \leftarrow s'$ 
    // rewind over new storage:
     $i \leftarrow 0, c \leftarrow c'$ 
  else
     $i \leftarrow i + 1$ 
  end
end
return  $s, c$ 

```



Algorithm 1: Merging nested cells based on shape. Figure 4: Shape components: *structure*, *transformations*, and *history*.
 $merge : Shape \times [Cell] \rightarrow Shape \times [Cell]$.

to reified objects and shapes and eliminates them, as the typical reason to access a field of a cell is to make it part of a new one, hence, the reified objects themselves are only short-lived.

For the example list, a recursive implementation of list reversal (with accumulator) traditionally needs n allocations, one for each cell of the reversed list. This number initially is higher with inlining, $2n$, due to reification. With the tracing JIT compiler, fewer than n allocations are necessary, as the short-lived allocations are eliminated and the new list can be directly created in its inlined form (cf. Figure 6). Therefore, with an inlining depth of m , cells of size m can be handled in one step with only one allocation, and the number of allocations would be m/n .

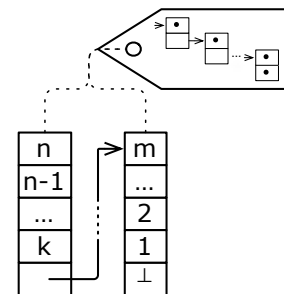


Figure 6: Reversed list

3.1.3 Shape recognition

While it is possible to provide the transformation rules for the merging process ahead of time, such rules would be static and possibly not fit for all workloads or applications. To tackle this problem, we record profiling data of how often what shapes are encountered during cell creation. E.g., in the shape “s1” in Figure 4 has this profiling information as its “history” property. In this case, we 17 times encountered a cell with shape “s1” at position 1 and three times a cell with shape “s2” and so forth. When the number of encounters reaches a threshold, we derive a new shape that reflects this often-encountered structure. E.g., for the 17 times a cell with “s1” was found at position 1, a new shape is created with a structure similar to that of “s2” in Figure 4; it is nested in position 1. For “s1”, a new transformation rule is added, to reflect the transition to the new shape, (1, “s1”→“s2”) in this case.

The effect of this recognition step is, the more often a certain structure is used, the deeper the merging can be and, hence, the higher the possible speed-up can be. To not create unbeknown large cells, we bound the merging depth with a threshold.

3.2 Stability for sustainable performance

The shape recognition approach shares a characteristic with typical optimizing JIT compilers. They collect profiling data during un-optimized execution that serve as reasoning base for creating optimized code. This is often called the *learning phase*.

3.2.1 Warm-up times

Collecting the profiling data takes time, especially because profiling happens during un-optimized execution. This *warm-up* time is more severe in our shape recognition approach, as new shapes and transformations between shapes can only happen incrementally. It is not possible to go from an un-inlined cell to a three-times inlined cell in one step in the first run; there are several transformations from an un-inlined to a one-time inlined cell necessary to actually create a two-times inlined cell in the first place. This is, however, intentional: to not clutter the transformation table of a shape with too many transformations, these transformations are only created after a certain number of observing a shape, which is recorded in the shapes history (cf. Figure 4). This threshold has also to be reached for every subsequent shape observation, to create a new one describing a more inlined cell.

The meta-tracing JIT compiler does not help either, in this case, as it is explicitly disabled during the learning phase; the JIT compiler would trace operations that are not meant to be present once a stable base of transformation rules exists.

3.2.2 Cross-run profiling data

To minimize warm-up times, we introduced a persistent cache for our profiling data and, moreover, the transition rules derived from them. These data are work-load-specific and, hence, are typically only used once. The structure of our profiling data and transition rules cope for that. The worst that could happen if cached data from a previous execution is used for a non-fitting work-load is that no optimizations are applicable and the runtime has to re-do the learning phase and warm-up its profiling data. I. e., the execution for the non-fitting work-load would act as if no persistent profiling data was present whatsoever.

If, however, the work-load fits that of a previous run close enough, the profiling data and, more important, the transformation rules can be used right away. This effect can be seen in 7(b), where all warm-up that is present in 7(a) (thin lines, low number of observations) is gone. There are fewer intermediate shapes and fewer branches, indicating a higher performance.

Re-using profiling and transformation data leads to increasingly stable performance for similar work-loads, since more optimized data structures can be used directly after start-up. Note that the recognition mechanism is not, in fact, disabled when profiling and

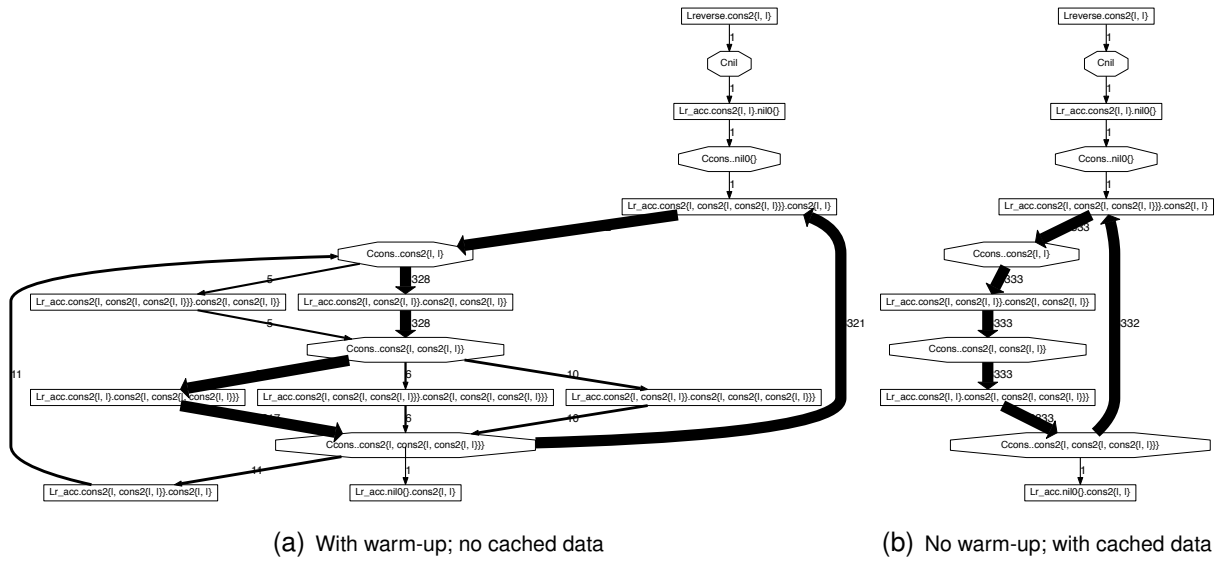


Figure 7: Transitions between observed shapes during the execution of reversing a 10 000 element list. The stronger the arrow, the more observations. The shapes are denoted as “nameN{sub-shape. . .}” with N being the number of fields in a described cell; direct access shapes are denoted as “|”.

transformation data are re-used but rather is invoked less often. When a no-fitting workload is executed, the recognition mechanism updates the profiling and transformation data accordingly. These new learned data are also persisted at the end of execution.

4 Preliminary results

To evaluate our approach, we implemented a prototypical runtime that uses the data structure inlining and profile data caching approaches. This section provides the prototype’s characteristics and the results of a micro-benchmark.

4.1 Prototype characteristics

We implemented an execution environment based on a strict λ -calculus that has ML-like semantics. Data is represented as *constructors*, i.e., generalized *cons*-cells with an arbitrary number of fields and a *tag* for type identification. Behavior is specified using λ functions with simple pattern matching based on constructor structure. All data is immutable. However, there is no language to speak of, yet. The prototype is built using the RPython toolchain of the Pypy project [2] with its meta-tracing JIT compiler.

4.2 Setup and Results

As a micro-benchmark, we chose a simple list reversal, matching the description in section 3.1.2. We profiled the timings of reversing lists of different sizes under no

optimization, with pre-defined transformation rules (cf. section 3.1.1), and with automatic recognition of transformation rules (cf. section 3.1.3). Second, this experiment was repeated once without and once with cross-run profiling data (cf. section 3.2.2). We had our prototype reverses lists of up to 100 million elements. The runtime of the first experiment part is given in Figure 8.

All benchmarks were run on a 64 bit Linux version 3.2 on a multi-core, 2.0 GHz Intel Xeon E5-2650 processor with 16 GB of RAM available. Since all runs were done un-parallelized, the number of cores (four) was irrelevant to the experiment. The machine was dedicated to the benchmarks.

As thresholds for the recognition algorithm indicated in section 3.1.3, we chose a maximum cell size of seven fields and require at least 23 occurrences of a shape before new shapes are created.

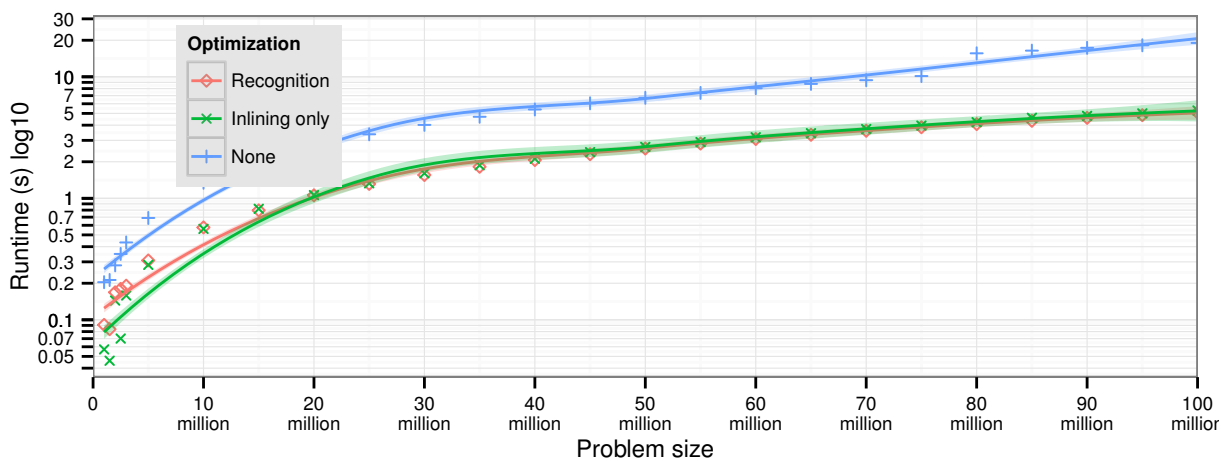


Figure 8: Runtime results for reversing list of different lengths. Configurations as indicated. The data points were smoothed using local regression [3]; the semi-transparent areas are based on standard deviation of each data point. Note the logarithmic scale.

The results for the cross-run optimizations, however, only show minor improvements.

4.3 Discussion

The benchmark results indicate a systematic speed-up of our inlining approach compared with the un-optimized case. Improvements range from 0.6 times to an order of magnitude faster. The speed-up even increases with increasing problem size. Both inlining variants, without and with automatic shape recognition, experience these speed-up rates.

Comparing the two inlining variants, the differences in runtime are minor, with the automatic shape recognition variant experiencing a higher speed-up rate with increasing problem size. This indicates that our automatically recognized transition rules match those that were hand-crafted and created specifically for this benchmark. Therefore, our recognition approach seems to be a viable optimization basis.

During the experiment, we also noticed that the memory demand of the non-optimized variant was significantly greater than for the optimized variant, with a peak of 9500 MB for the 100 million element test run. Future experiments will include more thorough memory measurements.

5 Related work

Our inlining approach is related to earlier work, mainly concerned with memory saving for statically typed functional languages. During the 1990's, several approaches for inlining lists in ML were proposed, with Shao, Reppy, and Appel [11] being pioneers in so far as they provided a formally verifiable approach. For the special case of object inlining, Wimmer [12] presented recent work, that is targeted at the Java language

These approaches work at the language level and not, like our approach, on the language implementation level. This typically requires programs to be prepared, e.g., by re-compiling or rewriting. Our approach does not require any language-level preparation. In fact, while the traditional approaches rely on ahead-of-time compilation, we optimize the programs at runtime only with JIT compilation. This allows for more workload-specific optimizations.

The shape-recording approach is related to the *hash-consing* approach, a popular optimization technique in functional languages, with recent results by Filliâtre and Conchon [4] for ML-like languages. The idea of hash-consing is to cache similarly structured data structures after they are no longer needed to avoid the allocation of new structures. This is a different intent, but the basic idea to optimize on the data's structure is similar. Note that hash-consing, too, is a language-level optimization.

6 Conclusion and outlook

Our approach to data optimization can improve the performance of data access while retaining simple, consistent language semantics. We observed speed-ups up to an order of magnitude at best. The difference between the approach with pre-defined transition rules and automatically derived transition rules is minor which suggests that the automatic recognition is a viable way to generate workload-specific optimizations.

The improvements of the cross-run optimizations are still minor.

Evaluation up until now only includes micro-benchmarks for selected data structures. We plan to conduct micro-benchmarks for different, more heterogenous data structures as well as equip an ML implementation with our optimization to test it under more realistic conditions.

References

- [1] Norman I. Adams, IV, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel Paul Friedman, Robert Halstead, Chris Hanson, Christopher Thomas Haynes, Eugene Kohlbecker, Don Oxley, Kent M. Pitman, Guillermo Juan Rozas, Guy Lewis Steele,

- Jr., Gerald Jay Sussman, Mitchell Wand, and Harold Abelson. Revised⁵ report on the algorithmic language scheme. *SIGPLAN Not.*, 33:26–76, September 1998.
- [2] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin 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*, ICPOOLPS ’09, pages 18–25, New York, NY, USA, 2009. ACM.
- [3] William S Cleveland, Eric Grosse, and William M Shyu. Local regression models. *Statistical models in S*, pages 309–376, 1992.
- [4] Jean-Christophe Filliâtre and Sylvain Conchon. Type-safe modular hash-consing. In *Proceedings of the 2006 Workshop on ML*, ML ’06, pages 12–19, New York, NY, USA, 2006. ACM.
- [5] Richard P Gabriel. Lisp: Good news, bad news, how to win big. *AI Expert*, 6(6):30–39, 1991.
- [6] Adele Goldberg and David Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman, Boston, MA, USA, 1983.
- [7] Richard D. Greenblatt, Thomas F. Knight, John T. Holloway, and David A. Moon. A lisp machine. *SIGIR Forum*, 15(2):137–138, March 1980.
- [8] Urs Hölzle, Craig Chambers, and David Ungar. Debugging optimized code with dynamic deoptimization. *SIGPLAN Not.*, 27(7):32–43, July 1992.
- [9] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML, revised edition*. MIT Press, 1997.
- [10] Eliot Miranda. Under cover contexts and the big frame-up. Online, September 14 2009. <http://www.mirandabanda.org/cogblog/2009/01/14/under-cover-contexts-and-the-big-frame-up/> (accessed 2012-02-29).
- [11] Zhong Shao, John H. Reppy, and Andrew W. Appel. Unrolling lists. *SIGPLAN Lisp Pointers*, VII(3):185–195, July 1994.
- [12] Christian Wimmer. *Automatic object inlining in a Java virtual machine*. PhD thesis, University of California, 2008.

Challenges and Approaches of Interaction Techniques for Multi-Perspective Views

Sebastian Pasewaldt

Computer Graphics Systems Group

Hasso-Plattner-Institut

sebastian.pasewaldt@hpi.uni-potsdam.de

This paper discusses challenges and approaches of interaction and navigation techniques for multi-perspective views (MPVs) in the context of 3D geovirtual environments. In contrast to single-perspective views, MPVs seamlessly combine multiple views from different perspectives in one image to reduce occlusion and visual clutter while providing additional context information. Current 3D geovirtual environments often provide a single-perspective view and corresponding interaction techniques, which are based on ray-casting. Applying ray-casting interaction techniques to MPVs introduces a number of technical challenges. We exemplarily show how existing interaction techniques can be adapted to be applicable to multi-perspective city panoramas. Further, we introduce interaction techniques that enable a fast and directed exploration of virtual 3D building models based on a multi-perspective detail+overview prototype.

1 Introduction

Current interactive 3D geovirtual environments (3D GeoVEs, e.g., Google Earth) mostly rely on single-perspective views because they emulate the human's optical system and perception. These single-perspective views have a number of drawbacks, such as occlusion, perspective distortion, and visual clutter, which complicate the communication of geoinformation. To reduce occlusion and increase screen-space utilization, landscape artists seamlessly combine multiple perspective views in one image. Recent advances in computer graphics hardware, in particular the programmable graphics pipeline, facilitate the implementation of interactive MPVs (Fig. 1 e.g., [9] and [10]).

MPVs can be generated (1) by deforming the geometric representation (the 3D scene) [10], (2) by modifying the projection of the virtual camera [6], or (3) by combining multiple images using a multiple center-of-projection camera model [12]. To achieve interactive frame rates, these modifications are applied during image synthesis using programmable shaders of computer graphics hardware, i.e., the MPVs are generated on a per-frame basis, while the underlying geometric representation remains unchanged.

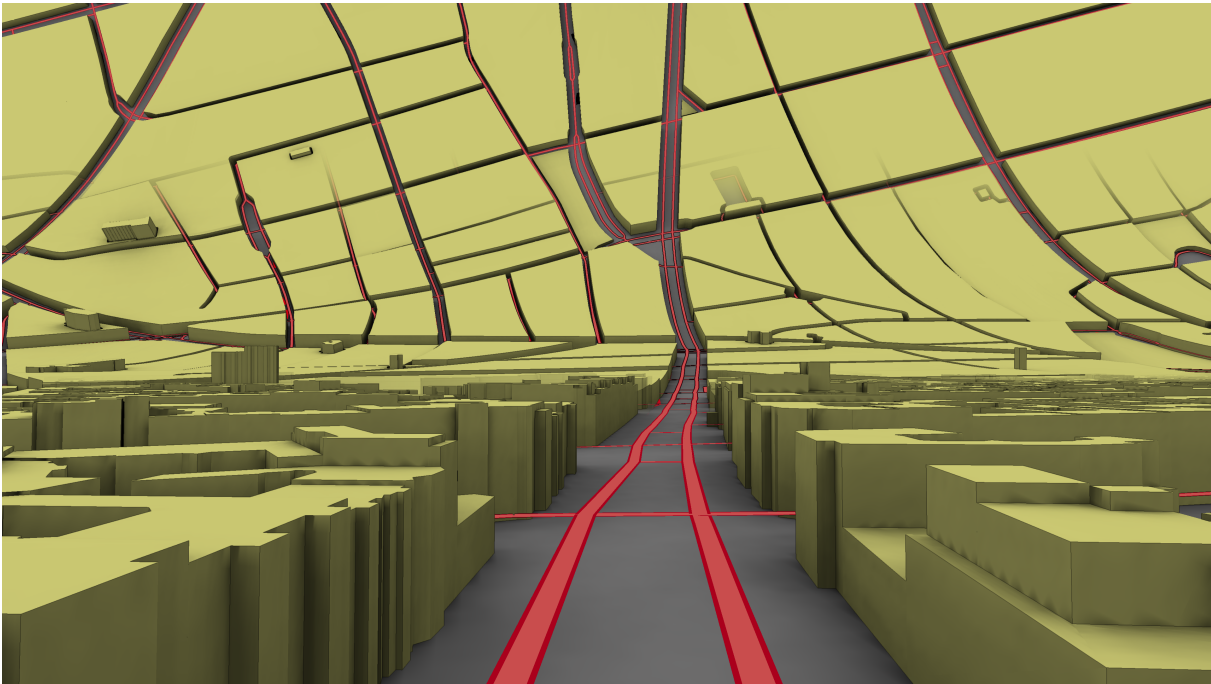


Figure 1: Example of a city panorama [10]. The virtual 3D city model is bended towards the user to increase screen-space utilization and reduce occlusion. To reduce the visual complexity, different level-of-abstraction of the virtual city model are seamlessly blended.

Current interaction techniques often rely on ray-casting [13] (Fig. 2): A ray $R = (O, D)$ that originates from the mouse-position and is perpendicular to the viewplane is cast into the 3D scene. The ray is projected into the coordinate system of the 3D scene, yielding its origin O and direction D . A list of intersections points $\mathbb{I} = \{I_0, \dots, I_n\}$ of R with the 3D scene's objects is computed, by traversing the complete 3D scene and calculating the individual intersections. Generally, the first intersection I_0 of R with the 3D scene determines the object, which should be interacted with, or the position, where the camera should be placed at. Since the current image does not depict a view of the original 3D scene, but a modified multi-perspective image, the ray-casting must not be executed on the original scene in order to yield correct intersections. For example, a selection of an object in the upper parts of the MPV results in a ray that runs above the scene and thus yields no intersections.

In this paper we propose an image-based approach for interacting with city panoramas using G-Buffers [14]. Beside the color image, additional images that encode vertex, normal and object identifier information, are generated during image synthesis. Instead of ray-casts, the G-Buffer is sampled at the mouse position to determine the object and its position under the mouse cursor. We present how existing MPV, such as city panoramas [10], can be adapted to be used with image-based interaction and navigation techniques. Further, we demonstrate how G-Buffer information can be used to implement fast and directed navigation techniques for multi-perspective detail+overview visualization of virtual 3D building models.

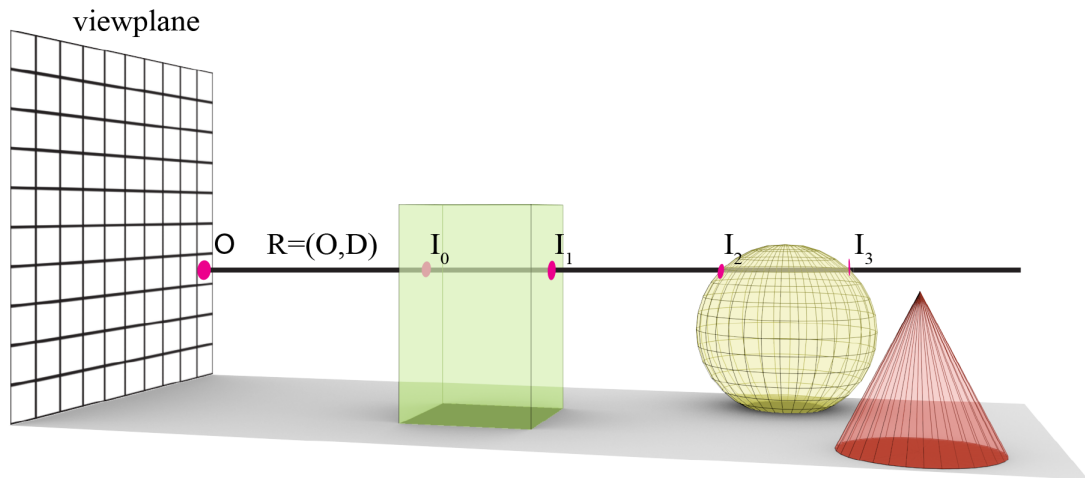


Figure 2: A ray $R = (O, D)$ is cast perpendicular to the viewplane into the 3D scene. The intersection $\mathbb{I} = \{I_0, \dots, I_n\}$ of the ray with the scene determines the objects that should be interacted with.

2 The Interaction Process

Interaction is an important aspect of information systems, such as 3D GeoVEs, because it enables the analysis, exploration and management of data, which would otherwise be hidden in the data set. The interaction process can be subdivided into three system components (Fig. 3): (A) the user, (B) the information system and (C) the user-interface. In the following, the interaction process is illustrated based on an uni-modal process model, i.e., only one output and input channel exists for the communication of the user with the system. The user interface of the 3D GeoVE presents geoinformation via an output channel (1.) (e.g., a display). The output information (2.) include a visual representation of geoinformation (the 3D geovisualization) and user-interface elements (e.g., buttons). To support a user (3.) to process the output information, cognitive capabilities, such as pre-attentive perception and pattern recognition, can be taken into consideration for the design of 3D geovisualization techniques. Based on the extracted information and the current task, a user may want to modify the visualization. Utilizing input devices (e.g., mouse or keyboard), the user emits interaction inputs (4.) via the input channel (5.). The information system (6.) processes the interaction input and updates its internal state (e.g., modifies the position of the virtual camera).

Interaction techniques are algorithms that process the interaction input (4.) and update the information system (6.). An update of the information system includes actions directly influencing the visualization process, such as the filtering of data, the modification of data mapping and the configuration of the rendering process. For example, a user can drill-down the information by filtering the raw data or by adjusting the mapping of data to visual variables. Further, a user can modify the current view, for example by changing the virtual camera. Thus, interaction techniques enable a user to move and navigate through the 3D GeoVE.

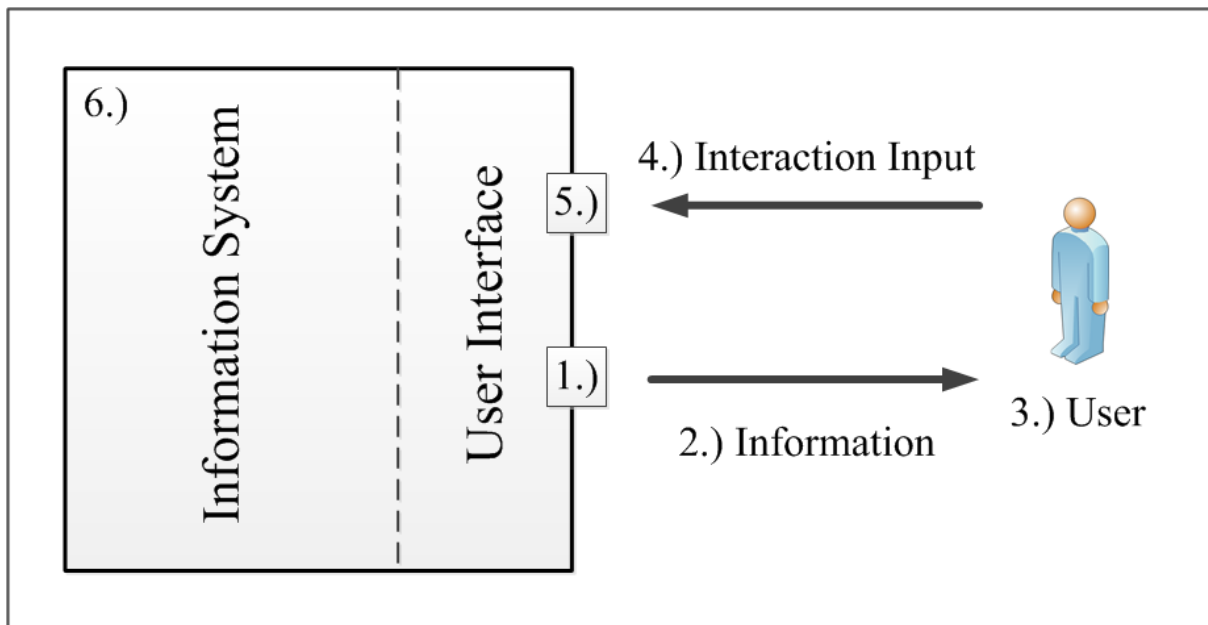


Figure 3: The interaction process is a feedback loop, where a user interacts with an information system utilizing a user interface. To change the presented information (2), which are provided by an output channel (1), a user provides interaction input (4) using an input channel (5). The input is processed by the information system and the output channel is updated.

Navigation is the "aggregate task of wayfinding and motion" [4]. Wayfinding is the cognitive element of navigation, i.e., it does not involve movement of any kind but only tactical and strategic parts that guides the movement. In the following, interaction techniques that enable a movement through the 3D GeoVE (i.e., an update of the virtual camera), are called navigation techniques.

Navigation techniques can be categorized into basic and assisting techniques. Basic techniques allow a user to directly manipulate the six degrees-of-freedom of the virtual camera, and thus an unconstrained movement. For example, orbit-navigation techniques map interaction inputs of the mouse to a camera rotation around a pivot point. This unconstrained movement can frequently lead to confusing and disorienting situations, since staying oriented in the virtual environment, i.e., relating elements of the 3D GeoVE to a known spatial context, is a non-trivial task [2]. Assisting camera-interaction techniques try to cope with this challenge by reducing the virtual camera's degrees-of-freedom. Further, they can prevent collisions of a user with parts of the virtual environment and can also adapt the movement speed of the virtual camera [8].

In addition to optimizing navigation techniques to assist a user during a task, the reduction of necessary interactions can improve a user's performance. This can be achieved by a task-specific design of the output information (2). For example, in the context of navigation systems for 3D virtual city models, city panoramas are a promising approach [10]. By displaying additional context information and reducing occlusion, more task-relevant geoinformation are presented, which otherwise would only be visible after multiple user interactions. To ease the exploration of thematic information

mapped to 3D virtual building models, multi-perspective building panoramas can be used [11] in addition to a perspective 3D view. The building panorama provides an overview and, thus, eases the processing of output information (3.) (e.g., by assisting the user to identify patterns in thematic data). Further, the building panorama can be used to implement direct navigation techniques such as the point-and-click navigation technique. A single click on a point-of-interest in the panorama places the virtual camera directly over the point-of-interest. This reduces the number of necessary interactions significantly, because panning and rotating are replaced by a single click.

The following sections do not focus on the design of MPVs for 3D GeoVEs, but on how existing ray-cast based interaction and navigation techniques can be adjusted in order to be applicable for MPVs (Section 3). For detailed information on the design process and implementation of MPVs, we refer to [10, 11]. Further, this paper demonstrates the potential of building panoramas to implement directed navigation techniques (Section 4).

3 City Panoramas

The presented city panoramas are based on the multi-scale, multi-perspective visualization technique of Pasewaldt et al. [10]. The virtual 3D city model is bended towards the user to reduce occlusion, perspective distortion, and visual clutter while increase screen-space usage (Fig. 1). The panorama is generated by deforming (i.e., translating and rotating) every vertex of the scene's geometry based on a parametric curve. Due to the geometric complexity of the scene (e.g., millions of vertices), a CPU-based implementation of this compute-intensive process could result in non-interactive frame rates. Instead, the scene is deformed on a per-frame basis during image synthesis, using programmable computation units (e.g., vertex shaders) of the graphics card, while the original scene remains unchanged.

Due to the differences between the original and the per-frame deformation of the 3D scene geometry, ray-casting interaction techniques are not applicable. A ray that is cast into the deformed parts of the multi-perspective image (Fig. 4A) would yield no intersections, since the intersection computation is executed on the CPU using the original (undeformed) 3D scene (Fig. 4B). A possible approach to cope with this challenge is (1) to deform the 3D scene for every ray-cast or (2) to deform the ray as well. The first approach often results in latencies during interaction for complex scenes, since the compute-intensive deformation process must be executed in order to implement the ray-casting. The deformation of the ray is also not applicable because of the mathematics properties of the projection/deformation (e.g., no inverse function is defined). Further, the combination of multiple level-of-abstractions [5] during runtime introduces another challenge: It is unclear which level-of-abstraction is currently displayed under the mouse cursor.

To cope with this challenges, image-based navigation techniques [8] can be used, which uses G-Buffers to determine the object or pivot point, which should be interacted with. Instead of casting a ray into a 3D scene, the ray is cast into a 2-dimensional image-based representation of the 3D scene to compute intersections. The G-Buffer is generated during image synthesis, by rendering the deformed 3D scene into vertex,

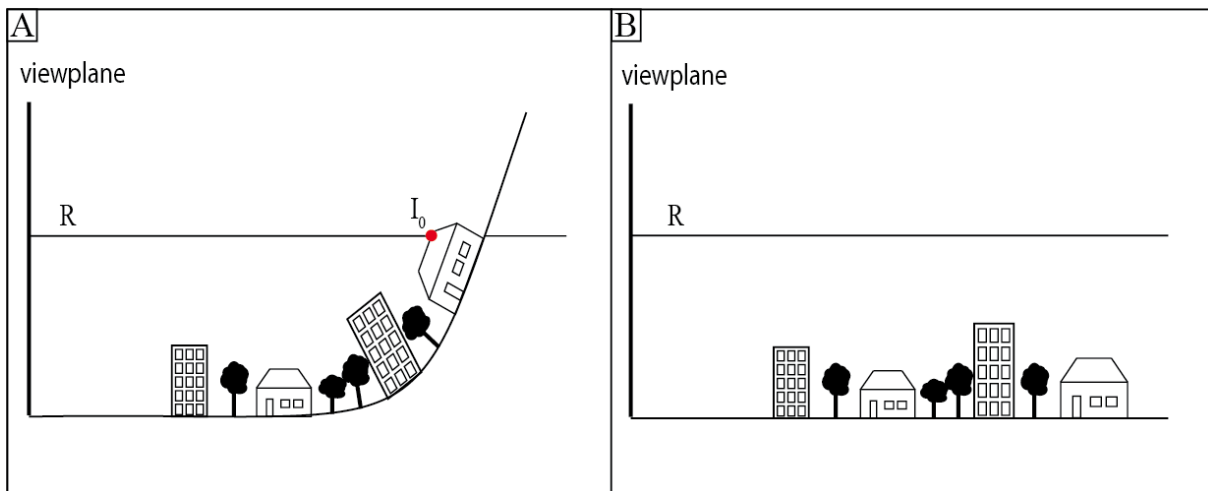


Figure 4: Since ray-casting is performed on the undeformed scene (B), a ray cast of the user that should yield an intersection I_1 with a building in the deformed scene (A), does not result in any intersection.

normal and object-ID images. As a result, the G-Buffer contains the deformed multi-perspective image and ray-casting yields a correct result.

For city panoramas, the image-based navigation technique of McCrae et al. [8] has been adapted. It enables navigation with 3D scenes of any size, i.e., from indoor scenes to virtual globes, by modulating the movement speed based on distance information and avoiding collisions. The distance information is sampled from an omnidirectional cube map with depth information. In order to combine McCrae's navigation technique with the city panoramas, the following modifications have to be implemented: (1) the vertex shader that is responsible for the deformation of the MPVs is also applied for the cube-map rendering; (2) the G-Buffer information of the MPV has to be made available for the navigation technique.

4 Building Panoramas

Building panoramas are MPVs that arrange building façades side-by-side in one image [11]. They offer an overview of the complete virtual 3D building model and they further reduce perspective distortion by depicting each façade with an orthographic projection. The building panorama (top) complements a 3D perspective view (bottom) in a detail+overview prototype for the interactive exploration of thematic information (e.g., heat transfer or solar insolation) of virtual 3D building models (Fig. 6). The multi-perspective overview facilitates the identification and comparison of patterns and the 3D view enables a detailed exploration. Both viewports are linked bi-directional using G-Buffer information, i.e., interactions in the overview are propagated to the detail view and vice versa. Thus, the multi-perspective building panorama does not only offer an overview, but also enables the implementation of fast and directed interaction techniques.

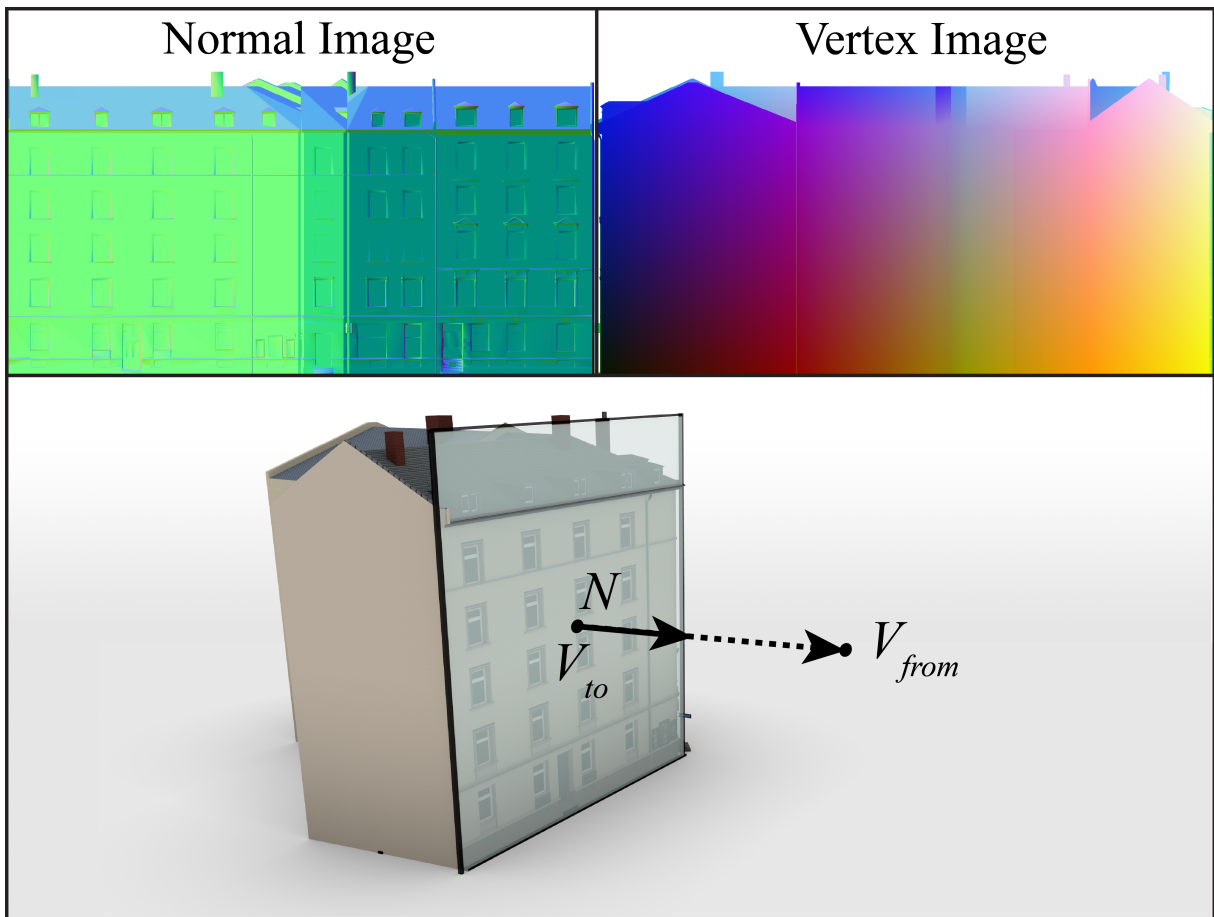


Figure 5: The vertex image of the G-Buffer is used to determine the look-to vector V_{to} of the virtual camera. The look-from vector V_{from} is computed by translating V_{to} along the normal vector N , which is extracted from the normal image.

We suggest the following navigation techniques:

(1) *Point-and-Click interaction*: Clicking on a point-of-interest in the overview directly places the 3D view on this point of interest. Compared to a combination of multiple zooming, panning and rotation, the point-and-click interaction reduces the number of interactions to a single click. Further, it is more precise, since users have to control only two degrees of freedom in the 2D image instead of six in the 3D scene.

(2) *Hover-interaction*: Instead of clicking in the overview, a hovering of the mouse continuously updates the virtual camera of the 3D view. As with the point-and-click interaction the camera's viewing direction is perpendicular to the building façade, to reduce perspective distortion. The hover-interaction technique enables an exploration of the building model similar to the Hovercam of Khan et al. [7].

The placement of the virtual camera, i.e., the look-from vector (V_{from}) and the look-to vector (V_{to}), is calculated based on G-Buffer information (Fig. 5). V_{to} is a position on the façade and can be extracted by sampling the vertex image. To place the camera perpendicular to the virtual 3D building model, the normal N is sampled from the vertex image. The final look-from vector is computed by translating V_{to} along N .

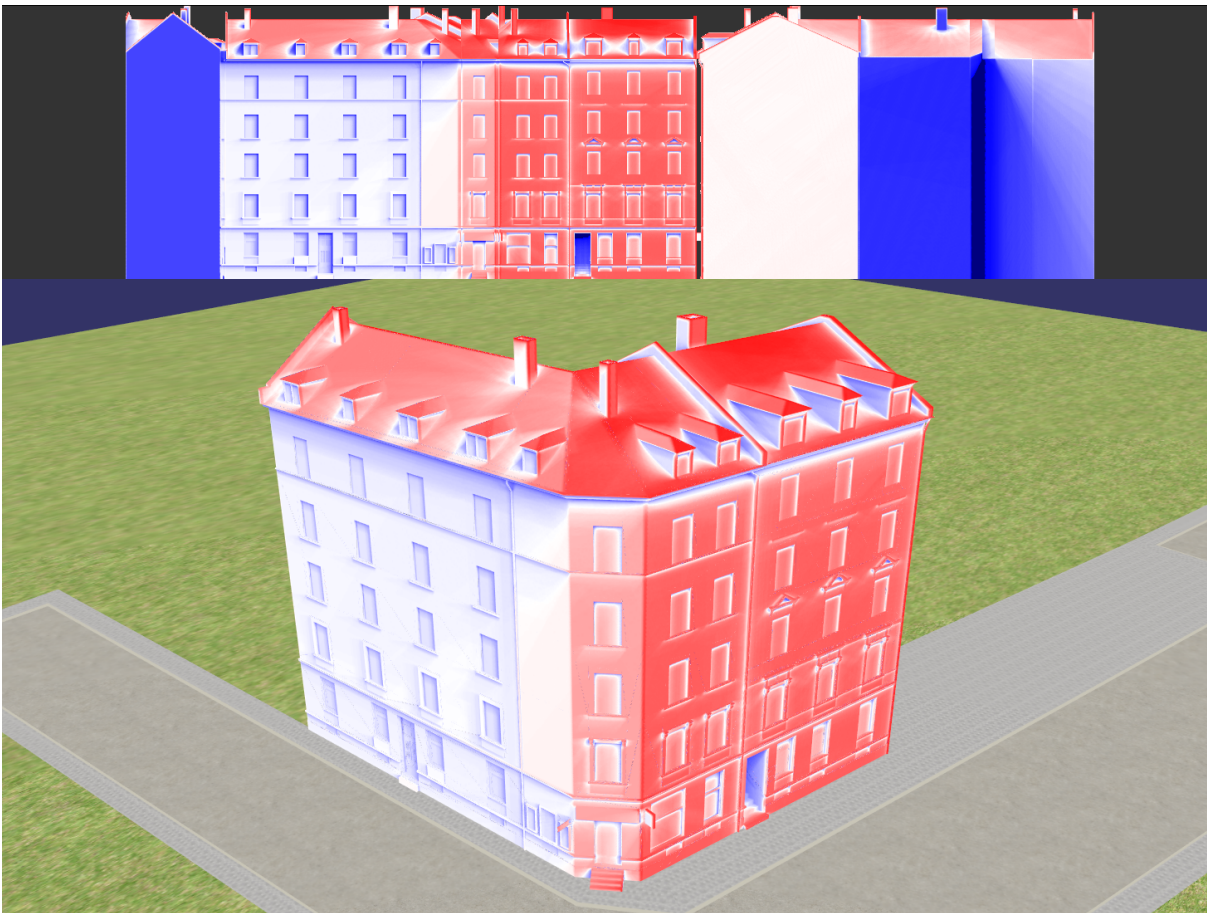


Figure 6: Detail+overview prototype for the exploration of virtual 3D building models [11]. The 2D multi-perspective view (top) offers overview and is used for direct and fast interaction with the 3D detail view (bottom).

We experienced that the user can get lost in the 3D scene at a high zoom level of the 3D view, since the mental mapping between the 3D view and the 2D overview can hardly be established due to missing context information. To cope with this challenge, a G-Buffer based highlighting of the 3D view frustum in the building panorama has been implemented.

5 Discussions

The representation of the 3D scene geometry by G-Buffers enables the implementation of image-based interaction techniques for MPVs. One drawback of the G-Buffers is that they are only a 2-dimensional representation of the 3D scene. Due to occlusion, they only contain parts of the scene and as a result, a ray cast into the image always yield at most one intersection. Interaction techniques, such as a selection of all scene objects along a ray cannot be implemented. One approach to cope with this challenge, is to use A-Buffers [3]. These buffers do not only include the "visible" geometric information of the frontmost object, but also all occluded objects.

A second drawback are the additional rendering overhead and memory requirements to generate the G-Buffers. Using multiple render targets (MRT [15]) the rendering overhead can be significantly reduced, since no additional rendering passes must be executed. Further, G-Buffer are often a byproduct of today's image-based post processing techniques, such as screen-based ambient occlusion [1], which are used to increase depth perception.

The proposed navigation techniques for the building panorama are a promising approach for the interactive exploration of virtual 3D building models, since they reduce the number of necessary interactions to move to a point-of-interest. Nevertheless, user tests will have to proof, whether this reduces the task completion time.

6 Conclusions & Future Work

Multi-perspective views are novel and promising visualization techniques to communicate 3D geoinformation. Due to the design and implementation of MPVs, ray-cast based interaction techniques are often not applicable. We presented how G-Buffer information of today's 3D geovisualization techniques can be used for the implementation of image-based navigation and interaction techniques. Further, we outlined the potentials of building panoramas to facilitate the implementation of navigation techniques for a detail+overview visualization system.

Effective interactions and navigation techniques are one approach to optimize the interaction process and thus decrease a user's task completion time. An orthogonal approach is to reduce the number of necessary interactions. This can be achieved by a user- and task-specific design of the information presentation, i.e., visualization technique. For example, the exploratory visualization of virtual 3D building models requires a user, to generate an overview of the data set before a detailed analysis can be performed. Since a single-perspective view frequently is used, large parts of a virtual 3D building model are occluded. As a result, a user has to needs a series of pan, rotation and zoom operations to adjust the virtual camera to get an overview. This is a time-consuming and tedious task. Using the proposed multi-perspective building panorama to provide an overview of all building façades in one image could supersede these interactions.

Up to this point, these are theoretically considerations. Currently, we are preparing a series of user studies to prove that our concepts are suitable for building and city panoramas. Beside the task-completion time, a crucial part of these studies are eye-tracking tests, since they can be used to gain insights into the perception of a user. Based on the insights, the design of MPVs will be adjusted.

References

- [1] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH'08 talks*, SIGGRAPH '08, 2008. ACM.
- [2] Henrik Buchholz, Johannes Bohnet, and Jürgen Döllner. Smart and Physically-Based Navigation in 3D Geovirtual Environments. In *Proc. IEEE IV*, pages 629–635. IEEE Computer Society Press, 2005.
- [3] Loren Carpenter. The A-buffer, an antialiased hidden surface method. *ACM SIGGRAPH'84*, 18(3):103–108, ACM, 1984.
- [4] Rudolph P. Darken and Barry Peterson. Spatial Orientation, Wayfinding, and Representation. In *Handbook of Virtual Environments: Design, Implementation, and Applications*, pages 493–518, 2001.
- [5] Tassilo Glander, Matthias Trapp, and Jürgen Döllner. Abstract representations for interactive visualization of virtual 3D city models. *Computers, Environment and Urban Systems*, 33(5):375–387, 2009.
- [6] Peter M. Hall, John Collomosse, Yi-Zhe Song, Peiyi Shen, and Chuan Li. RT-cams: A New Perspective on Nonphotorealistic Rendering from Photographs. *IEEE Trans. Vis. Graph.*, 13(5):966–979, IEEE, 2007.
- [7] Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice, and Gordon Kurtenbach. HoverCam: interactive 3D navigation for proximal object inspection. In *Proc. I3D*, pages 73–80. ACM, 2005.
- [8] James McCrae, Igor Mordatch, Michael Glueck, and Azam Khan. Multiscale 3D navigation. In *Symposium on Interactive 3D Graphics*, I3D '09, pages 7–14. ACM, 2009.
- [9] Sebastian Möser, Patrick Degener, Roland Wahl, and Reinhard Klein. Context Aware Terrain Visualization for Wayfinding and Navigation. *Computer Graphics Forum*, 27(7):1853–1860, 2008.
- [10] Sebastian Pasewaldt, Matthias Trapp, and Jürgen Döllner. Multiscale Visualization of 3D Geovirtual Environments Using View-Dependent Multi-Perspective Views. *Journal of WSCG*, 19(3):111–118, 2011.
- [11] Sebastian Pasewaldt, Matthias Trapp, and Jürgen Döllner. Multi-Perspective Detail+Overview Visualization for 3D Building Exploration. In Wen Tang Silvester Czanner, editor, *Proceedings of 11th Theory and Practice of Computer Graphics 2013 Conference (TP.CG.2013)*, pages 57–64. The Eurographics Association, 2013.
- [12] Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani. The graph camera. *ACM Trans. Graph.*, 28(5):1, ACM, 2009.

- [13] Scott Roth. Ray Casting for Modeling Solids. *Computer Graphics and Image Processing*, 18(2):109–144, 1982.
- [14] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. *ACM SIGGRAPH'90*, 24(4):197–206, ACM, 1990.
- [15] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 2.0)*. The Khronos Group Inc., 2004.

Architectures for Highly-Available Applications with Non-HA Infrastructure

Daniel Richter

Operating Systems and Middleware Group
Hasso-Plattner-Institut
daniel.richter@hpi.uni-potsdam.de

Next-generation data centers rely less on resilient infrastructure such as power supply, cooling, storage, or processors than previous generations. The availability of applications running with such infrastructure should be ensured by an appropriate architecture of these applications. DB Systel as a project partner wants to develop applications for next-generations data centers and needs recommendations for those architectures.

1 Introduction

One of the main concerns of IT operations is business continuity—companies rely on the availability of their information systems to run their operations. Today's data centers achieve this availability with the help of reliable infrastructure or platforms. Reliable infrastructure means for example the use of redundant hardware like multiple power supplies, RAID storage systems, or specialized hardware architectures like VAX. Because of great expenses for those hardware, there is the intention to make future software-intensive systems high-available with the help of software and middleware rather than with the help of hardware and platform. [7]

One mean to make applications more reliable and available is to make it fault tolerant. The phases of fault tolerance [8] are—after a latent fault was activated and became an error—error detection and error processing. Error processing consists of error recovery or error mitigation and can be solved using the following patterns: [9]

- Reconfiguration: Change the functional structure (add, remove or replace components), or change the internal processing configuration (so the external functionality remains the same) [1] [6] [10]
- Retry: Use redundancy in time (execute the same action again later), space (use different instances of services) or information (use an other data presentation)
- Repair: Remove the fault that causes the error (what in most cases requires manual action)

Those error detection and processing methods can be applied on different levels of abstraction. The following enumerations shows some examples:

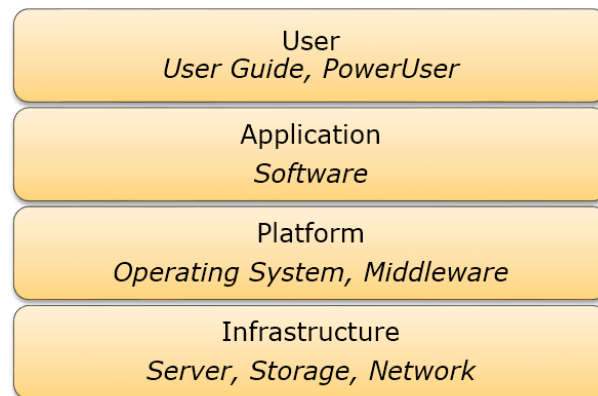


Figure 1: Levels of Error Processing

- **Infrastructure:** Usually, high-availability at infrastructure level is achieved by using specialized hardware and redundancy (in space). Examples are arrays of independent disks, network cards, power supply, etc. Those solutions generally cause high expenses.
- **Platform:** A popular way to make a platform high-available is to use replication. So the a hypervisor is able to replicate or migrate virtual machines. It is possible to run two virtual machines in fail-over configurations such as active-active or active-passive. In case of an error (at infrastructure level) the whole platform can be restored on or migrated to other infrastructure. Challenges at this level are to detect possible (future) error states [12] and restore/migrate a virtual machine without long downtime. [14] [5] [11]
- **Application:** Timely, spatial, and informational redundancy is also used on this level and it depends on the specific application, what methods are used. For example clustered databases are made high-available by using multiple hosts—usually the challenge is to keep the different nodes in sync so all data remains consistent.
- **User:** Sometimes errors only can be handled manually by human intervention. In this case you either need power users or experts that know how to get a broken system back to work or manuals are given.

Objective of our current work is to investigate if it is possible to make applications high-available at application and middleware level without having reliable infrastructure or platforms.

2 Challenges

The most effective way to make applications fault tolerant and high-available without having reliable infrastructure is to many redundant application instances with different hardware instances. So defective hardware will only harm a little part of the whole

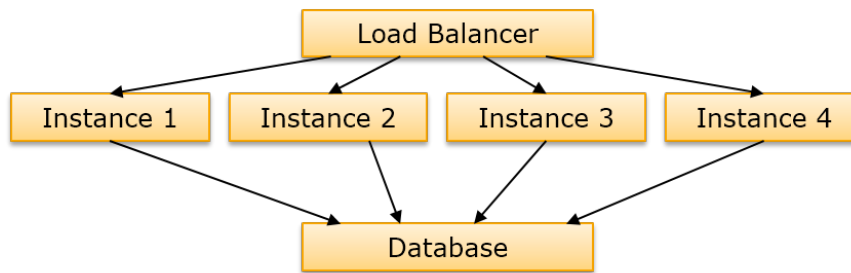


Figure 2: Bottlenecks in HA architectures

software system that can be easily replaced by other ones. One key point is to eliminate as much components that form any kind of bottlenecks. Another challenge is self-awareness of application instances, so they can act self-adaptive. [13] [4]

2.1 Shared Data

Although cluster databases already exist, some of them have disadvantages. To achieve high-availability, usually spatial redundancy is used. Data is replicated over multiple hosts. But according to the CAP theorem [3] [2] any shared data within distributed systems only can have two out of the following three attributes: Consistency, availability and tolerance to network partitions.

- Consistency and availability—tends to network partition intolerance
- Availability and network partition tolerance—tends to inconsistency
- Consistency and network partition tolerance—tends to unavailability

Today's database management systems focus on strong consistency. So either they lose availability (to gain network partition tolerance) or they lose network partition tolerance (to gain availability). Because network partition tolerance is important to make applications really scalable and parallelizable—so multiple instances are independent and a database can not become the bottleneck—the question is whether there is an actual need for (immediate) consistency.

Because immediate consistency often is not needed, one approach is to investigate the possibility to use NoSQL databases [15] where data is basically available and eventual consistency.

2.2 Load Balancing and Session Handling

Another shared component is the load balancer. It needs to be analyzed whether the load balancer can become a bottleneck—particularly in context of intercontinental distributed applications—and how possible unavailability can be solved.

3 Conclusion and Future Work

To achieve high-availability without reliable infrastructure only on application and middleware layer is difficult intention. For the next steps, we plan to setup a test environment with different kinds of applications provided by our project partner Deutsche Bahn Systel GmbH. After that, we first want to evaluate whether there are any architectural patterns that can applied to existing or new applications so you can save expensive infrastructure or make applications more available with little effort.

References

- [1] Shane Brennan, Serena Fritsch, Yu Liu, Ashley Sterritt, Jorge Fox, Éamonn Linehan, Cormac Driver, René Meier, Vinny Cahill, William Harrison, and Siobhán Clarke. A framework for flexible and dependable service-oriented embedded systems. In Antonio Casimiro, Rogério de Lemos, and Cristina Gacek, editors, *Architecting Dependable Systems VII*, number 6420 in Lecture Notes in Computer Science, pages 123–145. Springer Berlin Heidelberg, January 2010.
- [2] Eric Brewer. CAP twelve years later: How the "Rules" have changed. 45(2):23–29, 2012.
- [3] Eric A. Brewer. Towards robust distributed systems. In *PODC*, page 7, 2000.
- [4] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaella Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, number 5525 in Lecture Notes in Computer Science, pages 1–26. Springer Berlin Heidelberg, January 2009.
- [5] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, page 273–286, Berkeley, CA, USA, 2005. USENIX Association.
- [6] Marco Eugênio Madeira Di Beneditto and Cláudia Maria Lima Werner. A declarative approach for software compositional reconfiguration. In *Proceedings of the 11th International Workshop on Adaptive and Reflective Middleware*, ARM '12, page 7:1–7:6, New York, NY, USA, 2012. ACM.
- [7] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '08, page 57–62, New York, NY, USA, 2008. ACM.
- [8] Robert Hanmer. *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.

- [9] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. 37(7):56–64, 2004.
- [10] Arun Mukhija and Martin Glinz. Runtime adaptation of applications through dynamic re-composition of components. In Michael Beigl and Paul Lukowicz, editors, *Systems Aspects in Organic and Pervasive Computing - ARCS 2005*, number 3432 in Lecture Notes in Computer Science, pages 124–138. Springer Berlin Heidelberg, January 2005.
- [11] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. 2005.
- [12] Andreas Polze, Peter Tröger, and Felix Salfner. Timely virtual machine migration for pro-active fault tolerance. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, volume 0, pages 234–243, Los Alamitos, CA, USA, 2011. IEEE Computer Society.
- [13] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. 4(2):14:1–14:42, May 2009.
- [14] Felix Salfner, Peter Tröger, and Matthias Richly. Dependable estimation of downtime for virtual machine live migration. 5(1 and 2):70–88, June 2012.
- [15] Michael Stonebraker. SQL databases v. NoSQL databases. 53(4):10–11, April 2010.

Towards a Secure Multi-tenant SaaS Environment

Eyad Saleh

Internet Technologies and Systems
Hasso-Plattner-Institut
eyad.saleh@hpi.uni-potsdam.de

In this report, I summarize my research activities during the past two years in the area of Software-as-a-Service (SaaS) and multi-tenancy. Then, I discuss the research direction that I will follow in the next year and highlight the research questions that will be addressed in my thesis. Finally, I will close with the current industry cooperation and my next steps.

1 Introduction

In my first technical report [1], I selected multi-tenancy and Software-as-a-Service (SaaS) as my research area and introduced several challenges in SaaS and multi-tenancy, such as software migration to the cloud. In the next reports [2] [3], I discussed some of the challenges mentioned in the first report in more details, in particular, software customization and data privacy and security, I discussed the related work and existing approaches, then I introduced our approaches that resulted in a number of publications [4] [5] [6].

In this report, I explain the research direction that I will follow in the upcoming year and highlight the research questions that will be addressed in my thesis.

2 Problem Statement

Software-as-a-Service (SaaS) has been growing rapidly over the last years and seems to be a promising software delivery model [7]. SaaS offers several advantages to both service providers and users, such as the reduction of *Total Cost of Ownership* (TCO), better scalability, and better resource utilization. On the other hand, users can use the service anywhere and anytime, and benefit from cost reduction by following the pay-as-you-go model [8].

However, users are still concerned about the security and privacy of their data. Improper disclosure of information causes privacy issues. Due to the nature of SaaS and the cloud in general, where the data and the computation are beyond the control of the user, data privacy and security become a vital factor in this new paradigm. Several research studies [9–11] reported that security and privacy are cited as the biggest concerns in adopting cloud computing. Additionally, in multi-tenant SaaS applications,

the tenants become more concerned about the confidentiality of their data since several tenants are co-located onto a shared infrastructure.

In response to those concerns, my thesis investigate several approaches to secure multi-tenant SaaS environments. We already developed *SignedQuery*, which is a technique that prevent one tenant from accessing others' data. An extnsion to *Signed-Query* as well as another approach that targets HANA and column-store databases is currently being investigated with SAP. We also plan to study different encryption techniques to trade-off between security and performance; and to come-up with a practical solution that can be used by the SaaS vendors. Finally, we will also consider other approaches than encryption such as secure tenant-placement (STP), where a secure-aware tenant-placement algorithm will be developed.

3 Research Questions

I aim in my thesis to answer one main question: **How To Secure Multi-tenant SaaS Environments?** to answer this question, I argue that we need to investigate the following research problems.

Data Confidentiality: Data confidentiality is considered one of main concerns for organizations in utilizing cloud computing. By adopting the cloud computing model, the data of the company will be beyond their control, and hence, they need to ensure that their data is safe, and no one can tamper with or get access to.

Security Isolation: In multi-tenant SaaS environments, several tenants are consolidated into one database instance. A compromise to a server or a bug in the software may lead to data leakage or data being accessed by other parties. Therefore, strong security isolation between tenants is necessary.

Tenant Placement: Researchers study the tenant placement problem from a workload perspective to optimize the performance and decrease the cost. We would like to consider a new parameter to the formula, that is security. We will investigate the possibility of developing a security-aware placement algorithm that minimizes the threats of placing several tenants on the same physical host. We believe that this problem is integrated with security-isolation mentioned above.

In particular, we need to address the following research questions:

- A)** How to protect the tenant's assets in terms of data confidentiality?
- B)** How to ensure that tenants cannot access other tenants' data?
- C)** How to securely place the tenant in the infrastructure?

4 Threat Model

The main objective of my thesis is securing multi-tenant applications built for the SaaS industry. We argue that we can achieve this by addressing the questions in the previous section.

In our work, we assume that the software provider (SaaS vendor) is trustworthy, while the infrastructure provider (IaaS provider) is not necessarily. We consider hardware compromise either from inside or outside the boundaries of the provider. However, we assume that the IaaS provider is protecting his assets using all possible physical security measures ranging from hardware to software techniques, such as cameras, firewalls, intrusion-detection, and intrusion-prevention systems (IDS/IPS).

The threat model assumes that an attacker/adversary with a root access to the DBMS server is able to get access to or temper with others' data. Thus, protecting the data on the DBMS server even if an attacker has a root access to it is one of the goals of my thesis. We also assume that several tenants are sharing a single database instance, and hence security-isolation between tenants becomes a key issue in our work.

The contribution of the thesis is proposed to protect the confidentiality of the users' data and ensure that no one can access it except the legitimate tenant.

5 State of the Art

Protecting users' data is an essential task in current systems. Researchers investigate new ways and develop solutions to maximize the confidentiality of users' data. We discuss below the efforts and approaches that researchers have developed to protect the data.

5.1 Encrypted Databases

Data encryption is a common approach to protect the confidentiality of user's data during transmission and storage [31] [32]. The challenge of processing encrypted data is the main obstacle in adopting this model. However, it has been shown that performing certain tasks on encrypted data such as search is applicable [12–17] [22] [24] [25]. On the other hand and in reference to arbitrary computation, a fully homomorphic encryption scheme has been introduced by [18] as a result of joint efforts between Stanford and IBM. This scheme supports computation over encrypted data. However, we believe that such a technique is still in the early stages of development, would require huge extra cost, and support only limited functionality.

Another promising approach is *Silverline* [27] that supports keeping the data at the server-side confidential by encryption in away that is transparent to the application and being able to have some functionality on it as well. *Silverline* proposed to dynamically analyse the application to determine which parts of the data can be functionally encryptable; it assumes that any data that is never interpreted or manipulated by the

application is encryptable. For instance, a SELECT query in a Human-Resource Management Application (HRM) that looks for all records matching the employeeID 'Jan' is not required to interpret the actual string 'Jan' and can execute the query if it would be encrypted. As for key-management, it divides the users into groups, and assigns a single encryption key to this group, facilitates encryption and information sharing at the same time. While *Silverline* seems to be valid and practical, the main drawback is that it requires analysis of the application and the data to determine which parts can be encrypted, also a repetition of this process will be required whenever a change to the application or upgrade is taking place. Also, major part of the data will still be stored in plain-text, thus privacy and data compromise issues still open.

A recent approach *CryptDB* [19] has been introduced by colleagues in MIT. The idea is to execute SQL over encrypted data using a collection of SQL-aware encryption schemes. I discussed this approach with SAP engineers during my stay in Rot, and we believe that this solution might be applicable for small websites such as message forums or conference review applications (These were actually the use cases of *CryptDB*), but will not work for enterprise applications or real multi-tenant SaaS applications. Moreover, *CryptDB* doesn't support *Stored Procedures* as they mentioned in the paper (where the SQL code is integrated into the DBMS itself) and this is the current standard in developing enterprise applications.

5.2 Privacy-Preserving Techniques

Social networks is considered an interesting area to study the impact of security and privacy issues on. *FlyByNight* [20] and *Persona* [21] are mainly designed to work with social networks, such as Facebook. They propose to store an encrypted version of the messages on Facebook's servers in away that is transparent to Facebook functionality. Thus, users will continue to use Facebook as usual while maximizing the level of their privacy. The main issue with such approaches is that they require the application to be rewritten to support encryption/decryption techniques. Also they are designed specifically for social networks and cannot be applied to other domains.

5.3 Secure Execution Environment

Trusted cloud computing platform (TCCP) has been proposed by [23]. The idea is to provide a closed-box execution environment for the consumers, guaranteeing that the cloud provider cannot tamper with the users data. Moreover, it allows the consumers to remotely check whether the server is running a TCCP implementation or not. The main limitation of this approach is the major change that the infrastructure provider has to accommodate since they need to adopt TCCP first.

5.4 Tenant-Oriented Security Requirement

A tenant-oriented security-management architecture called TOSSMA has been proposed by Almorsy et al. [28]. TOSSMA enables SaaS providers to use it on the platform-level to serve several applications. Also, it allows the tenants to define and

customize the security requirement they need without (re)engineering the existing applications or write security integration code. Further, it allows the tenants to define the security requirement on different levels, such as component, class, or method. The main difference to our work is that TOSSMA is mainly targeted to securing the resources at the SaaS-side in general, and focus on the application code more than the data itself.

5.5 Information Disassociation

Another approach can be referred to as *information disassociation* [26], where the information is separated into parts, these parts are stored in geographically-distributed locations. Therefore, any party involved cannot benefit from the data it hosts. Cloud-RAID [29] [30] is the work of our colleague Maxim Schnjakin in our chair. His main idea is to split the data into chunks and distribute it on several providers in away that enhance the security of the data while minimizing the cost.

Based on the above discussion, we believe that data confidentiality and security-isolation in multi-tenant SaaS environments are still open issues that need to be addressed, and that would be our mission in the next years.

6 Contributions

We describe in this section the contributions we have so far in reference to the research questions mentioned in Section 3. First, We introduce *SignedQuery* [4], a mechanism designed to enhance the security isolation between tenants by preventing any tenant from accidentally or maliciously accessing other tenants' data. Second, We present *HPISecure* [5], a software prototype designed to secure the data stored on the cloud that allows the user to store an encrypted version of his data on the cloud without breaking the functionality of the application.

6.1 SignedQuery

We propose the usage of a signature to sign the tenant's request, so the server can recognize the requesting tenant and ensure that the data to be accessed is belonging to this tenant. *SignedRequest* uses a custom HTTP scheme based on a keyed *Hash Message Authentication Code* (keyed-HMAC) for authentication. To create the signature, we concatenate selected elements of the HTTP request to form a string. Then, we use the tenant's key (provided by the SaaS provider) to create the HMAC of that string. This HMAC is the *signature* used to authenticate the request. Finally, we add the signature to the request as a parameter of the headers section.

When the system (server-side) receives a request, it checks whether the request contains a signature or not. If the signature is not present in the request, we assume that it comes from illegal source and drop the request followed by sending an error message to the requester explaining that the signature is not present in the request.

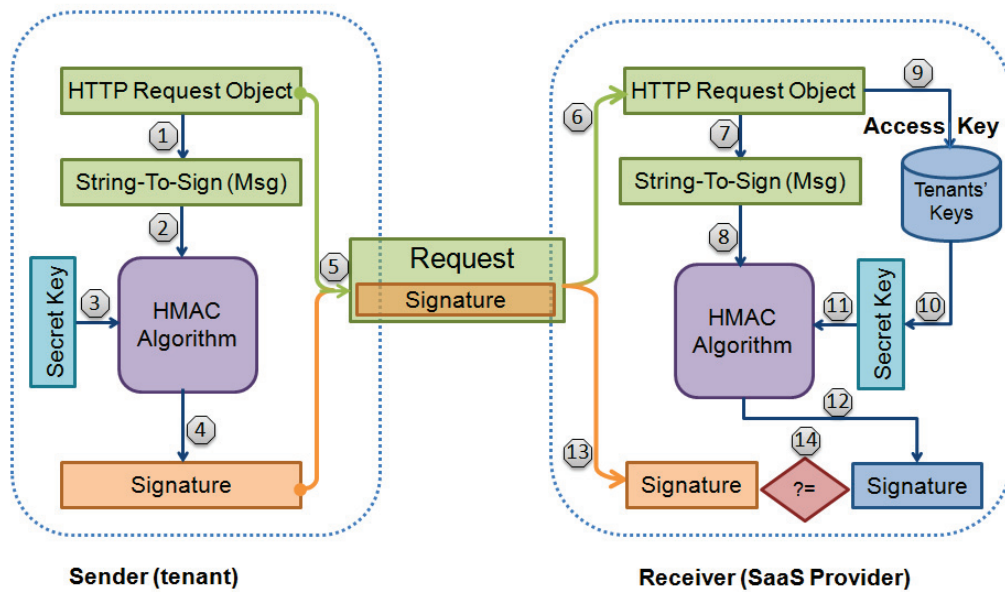


Figure 1: Signature Creation and Validation Process

If the signature is present, the process of validating the request starts, by fetching the tenant's secret key, and uses it to compute the signature of the received request the same way the tenant did. Then, we compare the signature we just calculated with the one present in the request, if they match, we conclude that the signature comes from the tenant who have access to the secret key, and therefore consider it as the authorized tenant to whom the key has been issued. If the two signatures do not match, the request is dropped and an invalid-signature error message is sent to the requester (tenant).

We implemented *SignedQuery* on top of Fiddler [33] as an HTTP proxy installed on the client and server machines, where the Request/Response objects of the HTTP protocol are intercepted, signature is authenticated, and proceed further according to the validity of the signature.

6.2 HPISecure

We propose an end-to-end encryption approach using public-key cryptography to secure the data. The focus of our approach is the documents-based applications, such as Google Docs.

Currently, there are several applications that offer securing the content on the cloud by utilizing cryptographic techniques [34] [35]. However, such applications limit the user to use his personal computer whenever he needs to access his data, because the application, the encryption/decryption keys, as well as other information is stored on his machine along with the application. And hence, he can not use other computers to access his data, which violates the basic concept behind the cloud (access your date anywhere and anytime). Therefore, our approach overcomes this limitation by

introducing the concept of the *Facilitator* as shown in Figure 2.

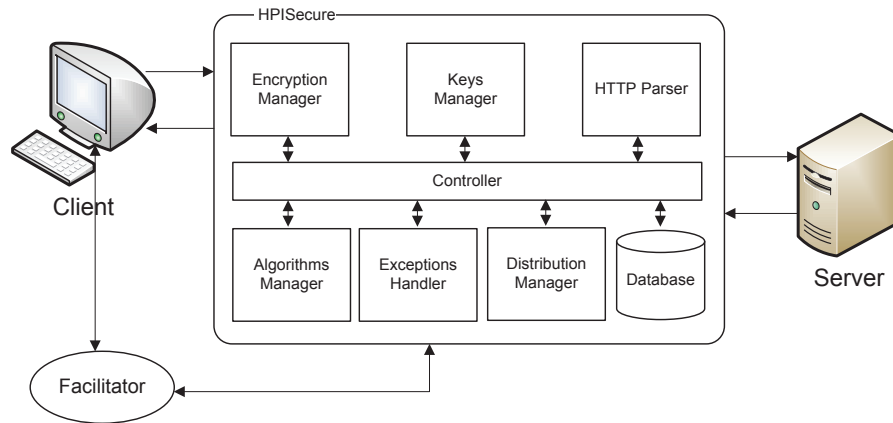


Figure 2: HPISecure Architecture

The *Facilitator* here could be the company that the user works for, it could be another cloud provider, or it could be a USB device. The idea of the *Facilitator* is simple, we need to alleviate the client machine from storing the cryptographic keys and other related data, and move this stuff to the facilitator. In this case, the client machine will be transparent to our approach, i.e., the user can use any computer to securely access his data, not only the machine that he used for the first time.

If the user works for a company, the *Facilitator* would be the internal web server of the company, where *HPISecure* will be installed. When the client try to create/save his data on the cloud for the first time, the request will be routed through the company's internal server, where the request will be intercepted, the data will be encrypt according to the preferences configured in *HPISecure* for this user, then proceed with the request to store the data securely in the cloud. For the upcoming requests when the user browse his data, the request for fetching the data will be also routed through the company's internal server, when the encrypted data retrieved, it will be intercepted, decrypted according to the user preferences by *HPISecure*, then sent to the user machine.

Individuals can use the same concept, but the *Facilitator* in this case refers to a third-party cloud provider, where the keys and related information will be stored. The same process mentioned earlier will be applied here. Finally, if the user doesn't belong to an organization (i.e., individual) and does not prefer to use third-party providers, the USB device is the choice. Using the USB device allows the user to store the cryptographic keys, configuration files, and any related information on a USB device, whenever he needs to use a new computer, all he requires is installing *HPISecure*, plug the USB, retrieve the keys and configuration files, then securely send/receive his data.

7 Teaching and Industry Cooperation

I participated in teaching during the last semester by conducting a master seminar in the topic of SaaS and multi-tenancy. Further, I had a visit to SAP Labs in Shanghai where I met with BusinessOne engineers and discuss the architecture of the new version of the application (OnDemand). BusinessOne OnDemand is currently being re-designed to be completely web-based and will work with row-store databases such as SQL Server as well as HANA.

I am finalizing this report while I am setting in SAP, Rot office, since I have a visit in the period of Oct 7–14th where I meet with Business ByDesign architects and engineers to discuss their security aspects and practices of multi-tenancy. In addition, I am working with our colleague Jan Schaffner from SAP on a project regarding the security of columns-store databases.

Furthermore, I will have a chain of visits to *SAP Certified Cloud Providers* in Europe, including Versino in Czech Republic, ZUTOM in Slovakia, and T-Systems in Hungary. These visits will be mainly to have an on-site view of the current deployment and usage of SAP BusinessOne, to discuss application and data security mechanisms in place, and to investigate open challenges and possible research cooperation.

8 Summary and Future Work

In this report, I summarized my work in the area of SaaS and multi-tenancy in the past two years. Then, I discussed the state of the art along with the problem statement and the research questions that will be addressed in my thesis. Further, I highlighted our contributions that have been published so far. In my next steps, I will focus on data-confidentiality and the secure-tenant placement challenges. Worth to mention that the next phase of our work will be in cooperation with SAP.

References

- [1] Eyad Saleh. *Multi-tenans SaaS: Challenges and Approaches*. Technical report, Hasso-Plattner-Institut, Potsdam, Germany, April 2012.
- [2] Eyad Saleh. *Migrating Traditional Web Applications into Multi-Tenant SaaS*. Technical report, Hasso-Plattner-Institut, Potsdam, Germany, Oct 2012.
- [3] Eyad Saleh. *Protecting Users Data in Multi-tenant SaaS Environments*. Technical report, Hasso-Plattner-Institut, Potsdam, Germany, April 2013.
- [4] Eyad Saleh, Ibrahim Takouna, and Christoph Meinel: *SignedQuery: Protecting Users Data in Multi-tenant SaaS Environments*. IEEE ICACCI, Mysore, India, 2013.
- [5] Eyad Saleh and Christoph Meinel: *HPISecure: Towards Data Confidentiality in Cloud Applications*. C4BIE workshop, IEEE CCGrid, Delft, Netherlands, 2013.

-
- [6] Eyad Saleh, Nuhad Shaabani, and Christoph Meinel: *A Framework for Migrating Traditional Web Applications Into Multi-Tenant SaaS*. INFOCOMP, Venice, Italy, 2012.
- [7] A. Konary, S. Graham, and L. Seymour: *The future of software licensing: Software licensing under siege*. International Data Corporation, White Paper, 2004.
- [8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and M. Zaharia: *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report, University of California, Berkeley, USA, 2009.
- [9] Future of Cloud Computing Survey. "<http://finance.yahoo.com/news/2013-future-cloud-computing-survey-110000960.html>". North Bridge. 2013 [retrieved: Oct, 2013]
- [10] Survery: Cloud Computing "No Hype", But Fear of Security and Cloud Slowing Adoption. "http://www.circleid.com/posts/20090226_cloud_computing_hype_security". Kelton Research. 2009 [retrieved: Oct, 2013]
- [11] Security is Chief Obstacle to Cloud Computing Adoption. "<http://www.darkreading.com/perimeter/security-is-chief-obstacle-to-cloud-comp/221901195>". Launchpad Europe. 2009 [retrieved: Oct, 2013]
- [12] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi: *Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions*. Journal of Cryptology, Vol. 21, pp. 350–391. 2008.
- [13] D. Boneh and B. Waters: *Conjunctive, Subset, and Range Queries on Encrypted Data*. Theory of Cryptography, Lecture Notes in Computer Science, Springer, pp. 535–554. 2007.
- [14] Y. Chang and M. Mitzenmacher: *Privacy Preserving Keyword Searches on Remote Encrypted Data*. Applied Cryptography and Network Security, Lecture Notes in Computer Science, Springer, pp. 442–455. 2005.
- [15] P. Golle, J. Staddon, and B. Waters: *Secure Conjunction Keyword Searches Over Encrypted Data*. Applied Cryptography and Network Security, Lecture Notes in Computer Science, Springer, pp. 31–45. 2004.
- [16] E. Shi: *Evaluating Predicates Over Encrypted Data*. PhD Thesis, Carnegie Mellon University, 2008.
- [17] D. Song, D. Wagner, and A. Perrig: *Practical Techniques for Searches on Encrypted Data*. IEEE Symposium on Security and Privacy, Berkeley, USA, 2000.
- [18] C. Gentry: *A fully homomorphic encryption scheme*. PhD thesis, Stanford, 2009.
- [19] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan: *CryptDB: Protecting Confidentiality with Encrypted Query Processing*. In 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, 2011.

- [20] M. M. Lucas and N. Borisov: *Flybynight: mitigating the privacy risks of social networking*. In Proc. of ACM WPES, Virginia, USA, 2008.
- [21] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Strain: *Persona: An online social network with user-defined privacy*. In Proc. of ACM SIGCOMM, Barcelona, Spain, 2009.
- [22] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano: *Public key encryption with keyword search*. In proc. of Eurocrypt, Interlaken, Switzerland, 2004.
- [23] N. Santos, K. P. Gummadi, and R. Rodrigues: *Towards trusted cloud computing*. In Proc. of HotCloud, San Diego, USA, 2009.
- [24] H. Haclgümüs, B. Lyer, C. Li, and S. Mehrotra: *Executing SQL over encrypted data in the database-service-provider model*. In Proc. of ACM SIGMOD, Wisconsin, USA, 2002.
- [25] H. Wang and L. V.S. Lakshmanan: *Efficient secure query evaluation over encrypted XML databases*. In Proc. of VLDB, Seoul, Korea, 2006.
- [26] K. Zhang, Y. Shi, Q. Li, and J. Bian: *Data Privacy Preserving Mechanism based on Tenant Customization for SaaS*. In Proc. IEEE MINES, Wuhan, China, 2009.
- [27] K. P. N. Puttaswamy, C. Kruegel, and B. Y. Zhao: *Silverline: toward data confidentiality in storage-intensive cloud applications*. In Proc. of ACM SOCC, Cascais, Portugal, 2011.
- [28] M. Almorsy, J. Grundy, and A. S. Ibrahim: *TOSSMA: A Tenant-Oriented SaaS Security Management Architecture*. In Proc. of IEEE Cloud, Hawaii, USA, 2012.
- [29] M. Schnjakin, R. Alnemr, and C. Meinel: *A Security and High-Availability Layer for Cloud Storage*. CISE 2010, Hong Kong, China, 2010.
- [30] M. Schnjakin and C. Meinel: *Implementation of Cloud-RAID: A Secure and Reliable Storage above the Clouds*. GPC 2010, Seoul, Korea, 2013.
- [31] C. Wang, Q. Wang, K. Ren, and W. Lou: *Ensuring data storage security in cloud computing*. In Proc. IWQoS 09, Charleston, South Carolina, USA, 2009.
- [32] A. Saha: *Computing on encrypted data*. In Proc. ICISS 2008, Hyderabad, India, 2008.
- [33] Fiddler website. [Online]. Available: <http://www.fiddler2.com> [retrieved: Oct, 2013]
- [34] BoxCryptor website. [Online]. Available: <http://www.boxcryptor.com> [retrieved: April, 2013]
- [35] TrueCrypt website. [Online]. Available: <http://www.truecrypt.org> [retrieved: April, 2013]

Visualization of Varying Hierarchical Data with Treemaps

Sebastian Schmechel

Computer Graphics Systems Group

Hasso-Plattner-Institut

sebastian.schmechel@hpi.uni-potsdam.de

Space-restricted techniques for visualizing hierarchies generally achieve high scalability and readability (e.g., tree maps, bundle views, sunburst). However, the visualization layout directly depends on the hierarchy, that is, small changes to the hierarchy can cause wide-ranging changes to the layout. For this reason, it is difficult to use these techniques to compare similar variants of a hierarchy because users are confronted with layouts that do not expose the expected similarity. Voronoi treemaps appear to be promising candidates to overcome this limitation. However, existing Voronoi treemap algorithms do not provide deterministic layouts or assume a fixed hierarchy. In this paper we present an extended layout algorithm for Voronoi treemaps that provides a high degree of layout similarity for varying hierarchies, such as software-system hierarchies. The implementation uses a deterministic initial-distribution approach that reduces the variation in node positioning even if changes in the underlying hierarchy data occur. Compared to existing layout algorithms, our algorithm achieves lower error rates with respect to node areas in the case of weighted Voronoi diagrams, which we show in a comparative study.

1 Introduction

Space-restricted visualization of hierarchical data has been a well investigated research field for the last decades. Although most of the existing techniques perform well with respect to scalability, readability, and the aspect ratio of the items (i.e., the graphical representation of hierarchy nodes), the stability of the layout represents a major challenge. If the depiction of similar hierarchies (e.g., varying hierarchies) does not expose similar layouts, the usability is substantially restricted. Users need to analyze and correlate the visualization results to compare the hierarchy visualization and to detect changes (e.g., new, deleted, or moved hierarchy components).

We use the term *layout stability* to describe a layout algorithm's 'tolerance' against changes in varying input hierarchy-data with respect to the arrangement and layout of resulting visual representations. In our case, we consider a layout algorithm to be stable, if small changes to the input hierarchy may cause only local changes to the layout of the resulting visualization. Layout stability is considered essential for effectively and efficiently performing visual analysis tasks such as comparing hierarchies and attributes of such hierarchies' nodes, and tracking changes to hierarchies over time [2, 7].

A key reason for this fact is that users explore visual representations and create their own mental maps [5]. Such maps significantly aid their orientation in the visualized information space by providing a reference system similar to road maps in the physical world. As with navigation in the physical world, such map is useless if changes to the input data result in items not being placed as expected. Thus, users can only use their mental map if the context – the spatial position of hierarchy nodes – is more or less stable, i.e., within some small frame of placement error. Otherwise, a new mental map has to be constructed. For example, in software visualization, large sequences of versioned module hierarchies (e.g., source-code trees) demand for a degree of coherence in corresponding visual representations to facilitate comprehension of the underlying software evolution.

2 Varying Hierarchical Datasets

There exists a wide range of varying hierarchical datasets. To illustrate this diversity, we pick two examples, illustrating a set of change operations that induce the variation of the datasets, before we define a formal model of the datasets that we consider in this paper.

File-Systems on Hard disks Files on a hard disk are structured strongly hierarchical (excluding links) using folders. In addition to this parent-child-relationship meta information on files, e.g., the attributes size and modification date, exist. A folder's size is then the aggregation of the sizes of all files contained in such folder. Furthermore, a file system can be modified by editing files, and through it modifying their file size (operation *changeAttribute*), *deleting* files as well as folders, and *adding* new files or folders.

Tree of Life The tree of life shows a hierarchical structure of different organisms (the tree's leaves) grouped by species. As in the file-system example, attributes exist for the leaves. A typical attribute here is an organism's population. Analogous to the file system, these attributes change over time, too (*changeAttribute*). Organism or even species become extinct (*delete*) while others evolve (*add*).

A Formal Model We define a formal model for varying hierarchical dataset $H = (N, \mathbb{E})$, which contains a set of nodes $N = \{n_0, \dots, n_i\}$ and a set of versioned edges $\mathbb{E} = \{E_0, \dots, E_j\}$ as follows:

$$E = \{k_0, \dots, k_j\} \quad k = (n_p, n_c) \quad (1)$$

k in E are considered as edges between nodes described by a tuple of nodes (a parent n_p and a child node n_c) with the following properties: (a) Only one tuple exists in E where $n_p = null$ and through this the child of that n_p is considered as root node. (b) Every child node has only one unique parent node. (c) There are no (transitive) cycles in E .

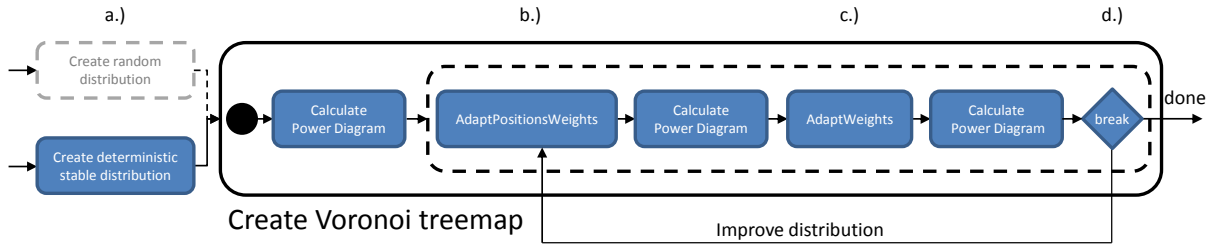


Figure 1: Process of computing a Voronoi treemap with weighted sites.

Hence, a tuple $T = (N, E_i \in \mathbb{E})$ defines one variant of our hierarchical dataset. We say that subsequent variants of such tuples are similar to each other, i.e., $\forall_{i \in [0, j-1]} : T_j \sim T_{j+1}$. We further define a function $\text{uid} : N \times E \rightarrow \mathbb{N}$ that yields a unique identifier per path $p : N \times E \rightarrow N \times \dots \times N$ such that:

$$\forall_{n_i, n_j \in N, E \in \mathbb{E}} : p(n_i, E) \neq p(n_j, E) \Leftrightarrow \text{uid}(n_i, E) \neq \text{uid}(n_j, E) \quad (2)$$

Here, $p(n, E)$ returns the path from node n to the root node n_r defined by E . Note that each variant $E \in \mathbb{E}$ can define a different path p for a node $n \in N$. In other words, our data model considers nodes that are *moved* in H as *different* nodes and they will have more than one unique identifier uid . Last, there exists a set of functions $\text{attr}_i : N \times E \rightarrow \mathbb{R}$ returning numerical attributes per node $n \in N$ and versioned edges E .

3 Visualization of Varying Hierarchical Datasets with Voronoi Treemaps

Voronoi treemaps appear to be promising candidates to increase layout stability for visualizing varying hierarchies [1]. Due to the way Voronoi diagrams are constructed, changes in the input data (a point distribution) typically only induce locally constrained changes to the output. One of the main disadvantages of existing layout algorithms for Voronoi treemaps [1, 6] consists in using a random initial distribution, which can result in essentially different depictions of similar hierarchies. We present an approach for deterministically computing an initial point distribution as input for the Voronoi Treemap computation stage instead of the random initial distribution used by Nocaj and Brandes (see Fig. 1a). As the distribution algorithm is tolerant against changes in the input hierarchy, we ensure that hierarchy nodes are position-stable in the resulting layout, and achieve a high degree of layout stability for varying hierarchies. Further, our approach reduces the error of achieved target areas for weighted Voronoi treemaps. For it, we show three optimizations to the latest algorithm: (see Fig. 1b) a less restrictive but holding criteria for the prevention of empty cells, (see Fig. 1c) a different calculation for increasing and decreasing the cell sizes through the iterative process, and (see Fig. 1d) a more precise break condition.

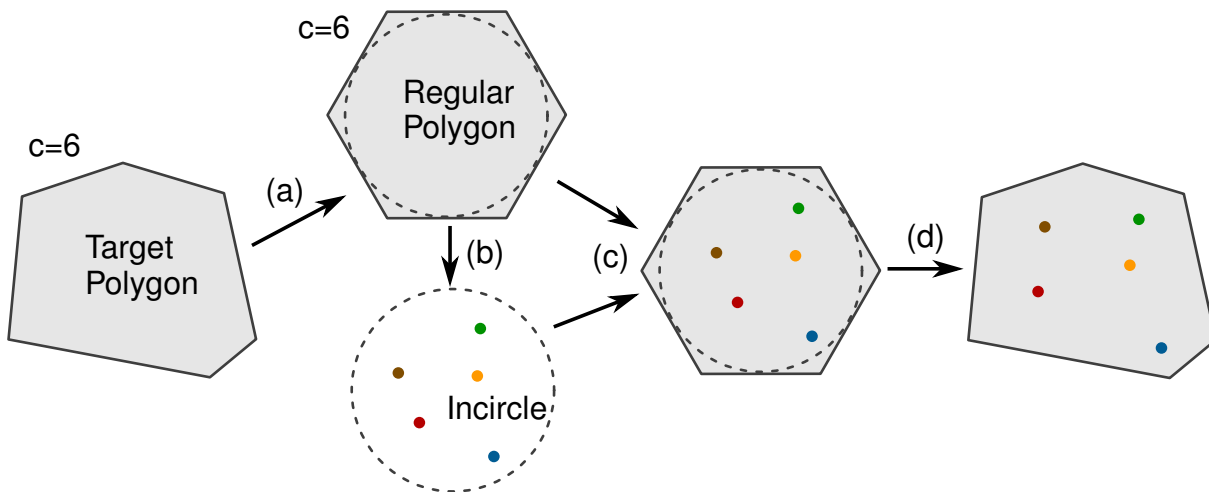


Figure 2: Workflow for computing a deterministic initial site distribution within a target polygon (left). (a) A regular polygon with the same number of vertices as the target polygon is created. (b) Sites are distributed by our deterministic approach into a unit circle and (c) transformed into the incircle of a regular polygon. (d) The regular polygon and the sites distributed within are transformed into the target polygon by using Wachspress coordinates.

3.1 A Stable Initial Distribution

Nocaj and Brandes construction of Voronoi cells is deterministic – except that they start with nondeterministic initial positions for the Voronoi sites. To achieve stability, we thus need to find a method that is able to create deterministic initial positions. That is, our approach has to fulfill the following requirements:

- The initial-distribution algorithm for the Voronoi sites has to be deterministic.
- If change operations such as *add* or *delete* occur to a parent node, they should not affect positions of its children already positioned in an earlier evolution step.
- The placement of sites relative to each other has to be stable, even if polygon's shape of their parent has changed.
- *changeAttribute* operations on nodes should only cause small changes in the resulting layout.

Voronoi treemaps allow for using arbitrary shapes as root item, within which any subsequent direct and indirect child items are contained. For each created shape that represents a child node, the algorithm can further be applied recursively to this node's children.

We start with a target polygon having a given number of vertices (*corners* c) that represent the root item in which a set of Voronoi sites (S) should be distributed. Next, we calculate a regular polygon with the same number of corners (Fig. 2a). Given a set of nodes (S), which are assumed to have a uniquely identifier, we are able to define consistent Cartesian coordinates in a unit square for each node. In our case,

we compute such unique identifier (uid) as a hash $\langle\langle x, y \rangle\rangle$ (an integer) from a node's path p . We then encode the first and last half of this hash with $x \in [0..1]$ and $y \in [0..1]$. Next, these two coordinates are transformed into the incircle of the regular polygon (Fig. 2b and 2c) by calculating polar coordinates (see Equations in (3)).

$$\begin{aligned} apothem &= \cos\left(\frac{\pi}{k}\right) \\ \langle\langle x, y \rangle\rangle &= hash \\ \phi &= 360 \cdot x \\ r &= apothem \cdot \sqrt{y} \end{aligned} \tag{3}$$

As a last step, we compute Wachspress coordinates for the regular polygon and the sites distributed within. They then describe the sites position inside a convex polygon as weighted terms of the polygon's vertices. By using the vertices' weights, each point of the distributed sites is transformed into the target polygon (Fig. 2d) [4]. Since it is guaranteed both that the target polygons in Voronoi treemaps are convex at any time, and that the incircle and Wachspress coordinates are well defined for convex polygons, the whole workflow effectively ensures that the distributed points are always placed inside the respective target polygon.

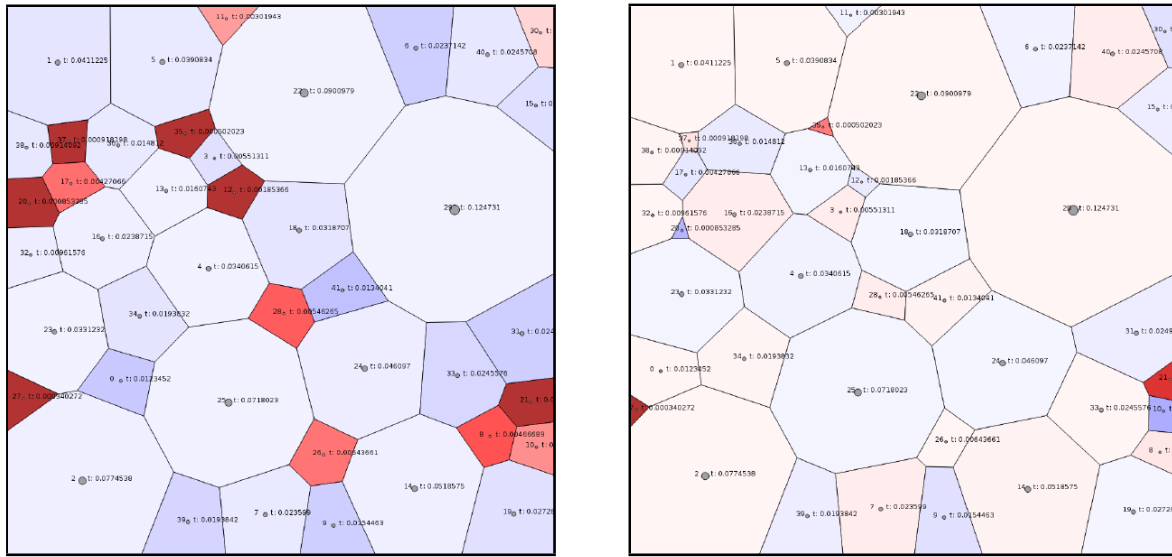
As an additional benefit, we do not need to recompute the Wachspress coordinates' weights of a polygon if the target polygon's vertices change their position slightly while the polygon itself keeps its dimension. This can happen, e.g., when changes occur in the item's occur. It thereby allows for a fast recalculation of the target distribution. The Wachspress coordinates only need to be recomputed if the number of vertices of the target polygon increase or decrease.

3.2 Optimized Layout Computation

After the initial distribution, the actual Voronoi treemap is computed (Fig. 1 b, c, d). Although the original algorithm [6] describes a rather fast way for computing Voronoi treemaps, we identified several optimizations for achieving more precise results with respect to the size of target area size of created Voronoi cells. These optimizations concern the functions used in the iterative positioning process (Fig. 1 b, c) as well as the break condition (Fig. 1 d) that decides whether a distribution is sufficiently precise.

3.2.1 Precision of Target-Area Size

As the initial positioning of the Voronoi sites does not necessarily reflect the targeted weights, Nocaj and Brandes propose an iterative optimization based on Lloyd's algorithm. Lloyd's algorithm, also known as Voronoi relaxation, in its original form is used to calculate Voronoi diagrams where the sites' location coincide with the centroids of its Voronoi cell [3]. Nocaj and Brandes use power diagrams to iteratively adapt the sites' weights and positions during each iteration. For it, the *current areas* (current size of a cell) of the cells are iteratively adapted towards the *target areas* (cell size that should



(a) Approach of Nocaj and Brandes

(b) Our Approach

Figure 3: Errors in target-area size shown by color (red = too big, white = correct, blue = too small), i.e., color encodes how much the actual size of a target area deviates from its expected size (given by the respective attribute value mapped to area size). Comparison between the results from the algorithm of Nocaj and Brandes (left) and our approach (right). In comparison to Nocaj and Brandes, our optimizations yield higher precision with respect to the error in target-area size.

be achieved). That is, they loop through the functions `AdaptPositionsWeights` and `AdaptPositions` until a break condition is satisfied. As pointed out by Nocaj and Brandes, empty cells have to be prevented during the iterative optimization of weights and positions: A centroid is required for optimizing the a site’s position, but cannot be computed for empty cells. Such a site’s empty cell can emerge if the site is encircled by a circle defined by the weight of another site. Consequently, empty cells can be avoided by limiting the site’s weights such that the constraint in Equation 4 is satisfied.

$$\forall s, t \in S, s \neq t : ||s - t|| > \max(\sqrt{ws}, \sqrt{wt}) \tag{4}$$

However, Nocaj and Brandes propose a criterion that is too strong in several cases. They limit a site’s weight to the distance of the cell that it belongs to. This often results in many cells being too small and thus – counterintuitively – especially cells that should be very small are far too large. We propose a weaker limit for a site’s weight as follows: A site’s weight in `AdaptPositionsWeights` is limited to the minimum distance to any other site, just like Nocaj and Brandes did in `AdaptWeights`. In most cases, this criterion is weaker, but it always satisfies the constraint in Equation 4. Our Algorithm 2 uses the same method to determine the distance to the nearest neighbor as proposed by Nocaj and Brandes in Algorithm 1. The distances can be calculated by means of a Voronoi diagram in $\mathcal{O}(n \log n)$. To overcome the problem of *oscillating* site locations, f_{adapt} is limited to $1 \pm \rho$ if its first derivative (increasing/decreasing the weight) starts oscillating as described in Algorithm 3. Note that even though the distance is described

in Algorithm 2 and 3 as being the distance between two sites, it can also be computed as twice the distance to the cell from a site. It is not necessary to know which site is the nearest neighbor. Also note that most distances can be calculated as the squared distances which does not require calculating any square roots. The effects of our optimizations are shown in Fig. 3 and are evaluated and discussed in detail in Section 3.3.

Algorithm 1: Two methods used to adapt the positions and weights in an iterative optimization algorithm proposed by Nocaj and Brandes.

```

1 AdaptPositionsWeights( $p, \mathcal{V}_s, S, W$ )
2   foreach site  $s \in S$  do
3      $a \leftarrow \text{centroid}(\mathcal{V}_s)$ 
4      $\text{distanceBorder} \leftarrow \min_{x \in \overline{\mathcal{V}_s}} \|x - s\|$ 
5      $w_s \leftarrow (\min(\sqrt{w_s}, \text{distanceBorder}))^2$ 
6 AdaptWeights( $p, \mathcal{V}_s, S, W$ )
7    $NN \leftarrow \text{Nearestneighbor}(S)$ 
8   foreach site  $s \in S$  do
9      $A_{\text{current}} \leftarrow A(\mathcal{V}_s)$  /* current area */
10     $A_{\text{target}} \leftarrow A(\Omega) \cdot \frac{v(s)}{v(p)}$  /* target area */
11     $f_{\text{adapt}} \leftarrow \frac{A_{\text{target}}}{A_{\text{current}}}$ 
12     $w_{\text{new}} \leftarrow \sqrt{w_s} \cdot f_{\text{adapt}}$ 
13     $w_{\text{max}} \leftarrow \|s - NN_S\|$ 
14     $w_s \leftarrow (\min(w_{\text{new}}, w_{\text{max}}))^2$ 
15     $w_s \leftarrow \max(w_s, \epsilon)$ 

```

Algorithm 2: Optimized version of *AdaptPositionsWeights* with less-restrictive empty cell prevention.

```

1 AdaptPositionsWeights( $p, \mathcal{V}(S), S, W$ )
2   foreach site  $s \in S$  do
3      $a \leftarrow \text{centroid}(\mathcal{V}_s)$ 
4    $NN \leftarrow \text{Nearestneighbor}(S)$ 
5   foreach site  $s \in S$  do
6      $w_{\text{max}} \leftarrow \|s - NN_S\|$ 
7      $w_s \leftarrow (\min(w_s, w_{\text{max}}))^2$ 

```

3.2.2 Break Condition

The iterative optimization to calculate Voronoi diagrams where the cells have a target area requires a break condition (see Fig. 1d) that is defines when the iterative optimization has finished. Nocaj and Brandes propose to cancel the optimization process when

Algorithm 3: Optimized version of `AdaptWeights` that prevents “oscillation”.

```

1  $f_s \leftarrow$  initialize with zeros
2  $AdaptWeights(p, \mathcal{V}_s, S, W)$ 
3    $NN \leftarrow Nearestneighbor(S)$ 
4   foreach site  $s \in S$  do
5      $A_{current} \leftarrow A(\mathcal{V}_s)$ 
6      $A_{target} \leftarrow A(\Omega) \cdot \frac{v(s)}{v(p)}$ 
7      $f_{adapt} \leftarrow \frac{A_{target}}{A_{current}}$ 
8     if  $f_s \neq 0$  and  $\text{sgn}(f_{adapt} - 1) \neq \text{sgn}(f_s - 1)$  then
9        $f_{adapt} \leftarrow \min(1 + \rho, \max(f_{adapt}, 1 - \rho))$ 
10     $w_{new} \leftarrow \sqrt{w_s} \cdot f_{adapt}$ 
11     $w_{max} \leftarrow ||s - NN_S||$ 
12     $w_s \leftarrow (\min(w_{new}, w_{max}))^2$ 
13     $w_s \leftarrow \max(w_s, \epsilon)$ 
14     $f_s \leftarrow f_{adapt}$ 
    
```

the sum of *area errors* is below a certain threshold. They define the area error as the difference between the current area and target area of a Voronoi cell. Furthermore the maximum number of iterations is limited.

Unfortunately the area error often does not converge to zero and the minimum error that is reached in a reasonable number of iterations highly depends on the number of sites and the target areas of the Voronoi cells. Consequently, the threshold is often reached after only a few iterations or not reached at all and the maximum number of iterations is reached. In the first case more iterations would reduce the sum of area errors. In the second case fewer iterations would probably not imply a much higher sum of area errors.

We propose a threshold for the difference between the maximum area error of different iterations which could be seen as the slope of the maximum area error. More formally, the optimization process is canceled when $e_{diff} < threshold$. e_{diff} is defined in Equation 5 where e_{-l} is the maximum error l iterations ago and e_0 is the maximum error in the current iteration.

$$e_{diff} = \max(e_{-l}, e_{\frac{l}{2}}) - e_0 \quad (5)$$

A disadvantage of this method is that the area error of the resulting Voronoi diagram does not have an upper bound. However, we know that more iterations would probably not reduce the area error.

3.3 Comparative Evaluation

To evaluate the optimizations of our approach (Alg_{OA}) (presented in Section 3.2) in comparison to the algorithms of Nocaj and Brandes (Alg_{NB}) [6], we tested both algorithms distributing different numbers of sites within the same parent polygon. For it,

3 Visualization of Varying Hierarchical Datasets with Voronoi Treemaps

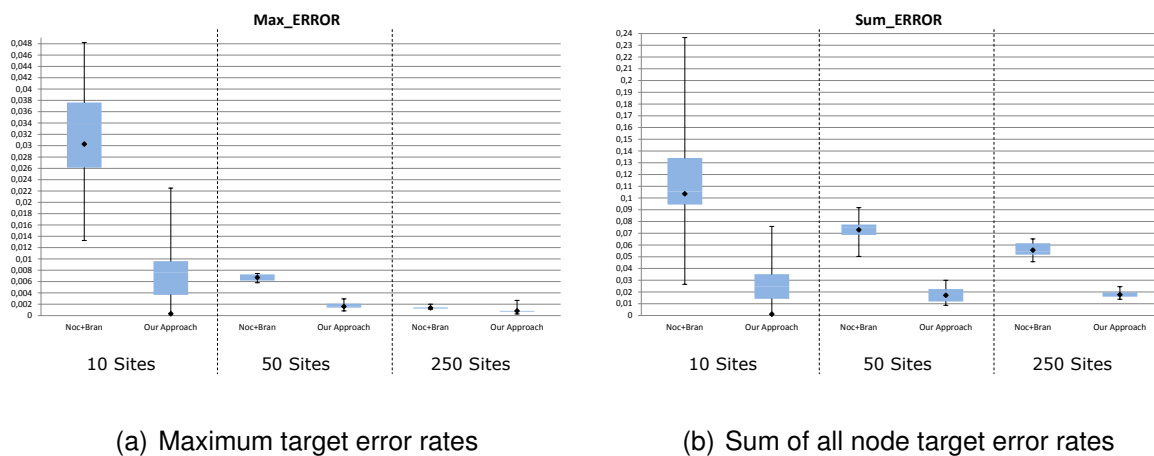


Figure 4: Results of the maximum target error rate (a) and the sum of all target error rates (b) evaluated by a comparative study between [6] and our optimization approach to improve the precision of the target areas for different numbers of sites (10, 50, 250) with randomized weights distributed in their parent node.

we computed distributions of 10, 50 and 250 sites with random polygon weights with 1000 iterations per distribution. To achieve comparable results we use the same seed to compute the weights for each run. The dependent variables of our study are: Time needed to compute the distribution within 1000 iterations, maximum error of the target sizes and the sum of target-size errors. Table 1 shows the evaluation setup with (in)dependent variables in detail.

Seed	Alg	Time	Max Err	Sum Err
s_1	Alg_{OA}	t_1	err_{max_1}	err_{sum_1}
s_1	Alg_{NB}	t_2	err_{max_2}	err_{sum_2}
s_2	Alg_{OA}	t_3	err_{max_3}	err_{sum_3}
s_2	Alg_{NB}	t_4	err_{max_4}	err_{sum_4}
...
s_n	Alg_{OA}	t_{2n-1}	$err_{max_{2n-1}}$	$err_{sum_{2n-1}}$
s_n	Alg_{NB}	t_{2n}	$err_{max_{2n}}$	$err_{sum_{2n}}$

Table 1: Dependent and independent variables for the comparative study (example for 10 sites).

The mean computation time of Alg_{OA} compared to Alg_{NB} shows equal results with each number of sites (Table 2 shows the mean computation times in detail), our algorithms shows better error rates in both, the maximum area error and the sum of area errors for every number of distributed sites (shown in Fig. 4 a,b).

We have presented an extension to an existing layout algorithm for Voronoi treemaps by using a deterministic initial distribution for the Voronoi sites. Since the resulting layouts are stable with respect to varying input hierarchies and varying size of items, this enables the use of Voronoi treemaps for *comparing* hierarchy variants. Such datasets

Number Of Sites	Alg	Mean Time
10	<i>Alg_{OA}</i>	355 ms
10	<i>Alg_{NB}</i>	353 ms
50	<i>Alg_{OA}</i>	1686 ms
50	<i>Alg_{NB}</i>	1701 ms
250	<i>Alg_{OA}</i>	8937 ms
250	<i>Alg_{NB}</i>	8914 ms

Table 2: Mean computation times for the comparative evaluation. The computation times of our algorithm are equal compared to the ones of Nocaj and Brandes.

emerge, e.g., from versioned source-code trees of software systems. Fig. 5 shows an example of such a varying hierarchy, depicting the *cc* project from the *Chromium* Git repository (branch: master) over several revisions (annotated with the commit-hashes). Our comparison of target-error rates further show that we achieve a lower target-error rate than existing layout algorithms. We conclude that the resulting layout represents attributes of the input data more accurately than previous techniques.

References

- [1] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 165–172. ACM, 2005.
- [2] Stuart K Card, Bongwon Sun, Bryan A Pendleton, Jeffrey Heer, and John W Bodnar. Time tree: Exploring time changing hierarchies. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 3–10. IEEE, 2006.
- [3] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM review*, 41(4):637–676, 1999.
- [4] Michael S Floater, Kai Hormann, and Géza Kós. A general construction of barycentric coordinates over convex polygons. *advances in computational mathematics*, 24(1-4):311–331, 2006.
- [5] Robert M. Kitchin. Cognitive maps: What are they and why study them? *Journal of Environmental Psychology*, 14(1):1 – 19, 1994.
- [6] Arlind Nocaj and Ulrik Brandes. Computing voronoi treemaps: Faster, simpler, and resolution-independent. In *Computer Graphics Forum*, volume 31, pages 855–864. Wiley Online Library, 2012.
- [7] Arlind Nocaj and Ulrik Brandes. Organizing search results with a reference map. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2546–2555, 2012.

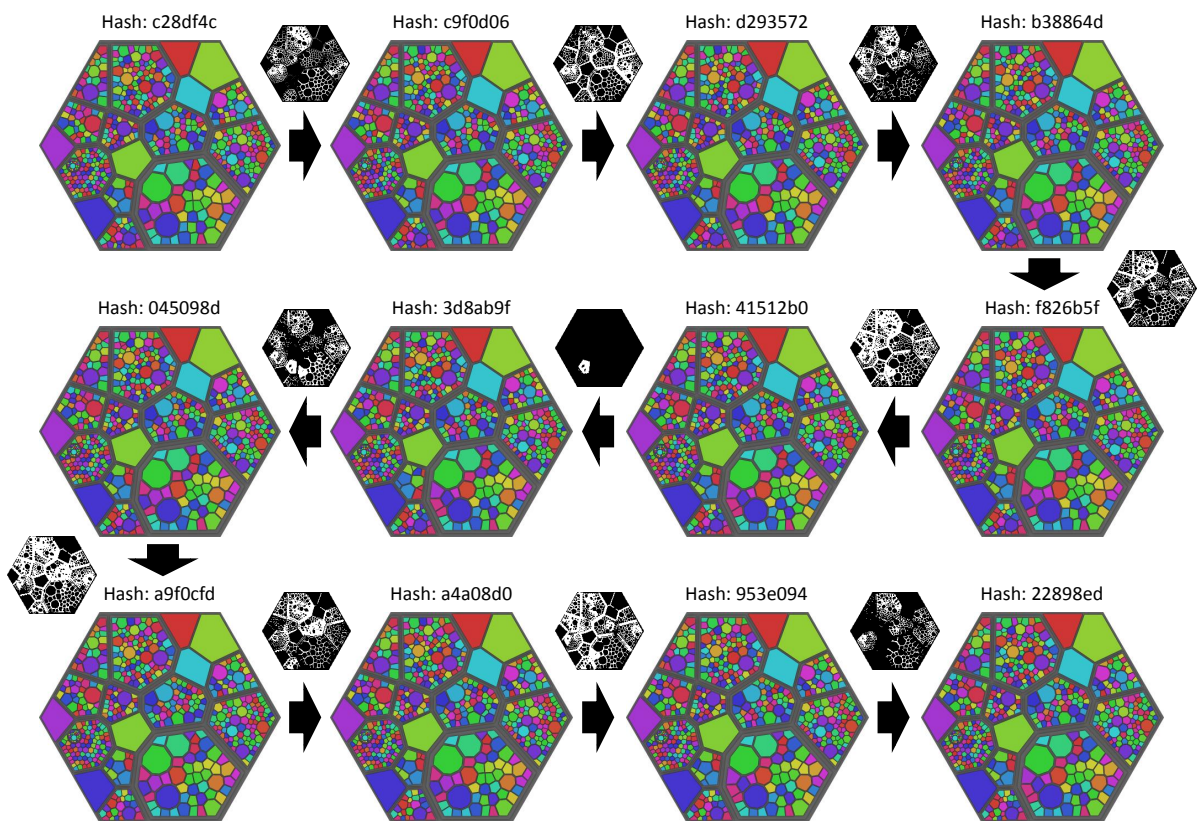


Figure 5: Visualization of the hierarchical folder structure (with about 500 nodes) of 12 revisions from the *cc* project of the *Chromium* Git repository (master). The area of the cells is mapped to the corresponding file size of the represented node. The nodes' unique identifier – created from the paths to the root node – is shown as color. Although several changes (operations: *changeAttribute* in 169 files, 5 files *added*, 2 files *deleted*) in the data are made, the layout appears as stable and through it memorable over all revisions. Between two depictions of a revision a difference mask is shown. Stable areas are represented by black pixels, while white pixels indicate layout differences.

No Tools But Objects: Towards Direct Manipulation Programming Environments

Marcel Taeumel

Software Architecture Group
Hasso-Plattner-Institut
marcel.taeumel@hpi.uni-potsdam.de

Direct manipulation user interfaces provide promising virtues that should reduce users' cognitive efforts. For programming tools, however, the three main principles about representations, actions, and feedback are difficult to implement upfront because the interfaces' suitability varies among task domains. Even in a single domain, the repertoire of tasks grows with the software system and so does the *interpretational gap* towards the tool interface, which often only supports plain programming language concepts. Unfortunately, tools are rarely adapted to close this gap because the cost-benefit ratio is unattractive for unplanned situations.

Our approach is to make programming tool behavior better accessible and adaptable. For data browsers, we propose a *query mechanism* that encourages programmers to apply modifications and accommodate domain-specific needs in situ.

1 Introduction

Programmers usually build complex software systems with the goal to fulfill customers' requirements and fix inadvertent bugs. However, programmers rarely modify the one important software system they use day-by-day: their own programming environment. They rather keep on struggling with common problems such as information overload or difficulties in information access than accommodating domain-specific scenarios in situ [9] [6]. We argue that the cost-benefit ratio for adapting programming tools is often unattractive. For example, try answering the following questions for your environment: Which lines of code are responsible for creating the first column in your class browser? How do you restore the current situation after adaption? Are the sources of your package browser available at all? Transparency is a real issue here; this makes tool adaption often not feasible. But why do programming tools need to be adapted for domain-specific problems anyway? Is there no perfect programming environment?

Programmers benefit from integrated programming tools, which reduce cognitive effort and thus reduce the time to complete tasks [7]. When considering cognition, we know from the field of usability engineering that *direct manipulation* [4, 15] serves as a valuable conceptual framework for designing efficient graphical user interfaces. The basic assumption is that the commitment of fewer cognitive resources results in

a feeling of directness. In principle, tools having direct manipulation interfaces should offer (1) a continuous representation of task-related objects and actions, (2) a simple mechanism to invoke those actions, and (3) immediate feedback plus a way to reverse inadvertent results. Which objects and which actions are relevant for programmers?

Basically, all programming activities involve exploring and modifying an extensive information space where numerous objects are connected via multiple relationships. For this, programming tools provide views on subsets of this space to support focusing on information that is relevant for a given task. Having this, for example, class browsers communicate with code editors or debuggers visualize call stacks while directly talking to object explorers nearby. Considering the programming language concepts in use, is it possible to build a programming environment that minimizes cognitive effort when creating software for all possible domains?

The sensation of directness is always relative [4]. There are many factors such as people, task, and domain, which influence the value of a given tool at any given point in time. Thus, there cannot be one programming environment that fits all. One prominent example is the recurrent problem of *crosscutting concerns* [18]: Several corresponding software artifacts are spread across the dominant system decomposition (e.g. methods in the class hierarchy) but tools only support views on the dominant language representation (e.g. class outlines and file listings). Hence, there have to be either ways to express a spread concern more concisely [5] or—as in our case—mechanisms that support tool adaption in situ to create appropriate views that efficiently exploit screen real estate.

We argue that it is possible to leverage current software browsing tools by making adaption-while-using an essential part of their conceptual model¹. We want to better implement direct manipulation principles (i.e. representations, actions, feedback) to reduce cognitive effort when adapting the tools in use and thus when building complex software systems for customers. We believe that if the cost-benefit ratio for adaption is more attractive, programmers will modify tools more often and thus save time in their daily programming activities.

In section 2, we explain to which degree traditional programming environments succeed to implement direct manipulation principles and extract unsolved issues. Then, section 3 gives a brief overview of related research projects that already tried to alleviate these issues. As the main part of this report, section 4 describes our approaches in more detail. For validation purposes, section 5 then explains our experience with our research programming environment VIVIDE in use and sketches open hypotheses. Finally, we conclude this report in section 6 and give a brief outlook on next steps.

2 Background

Direct manipulation principles can be found in many graphical user interfaces these days including traditional programming environments such as Eclipse² and Squeak³.

¹We refer to the definition of Norman [10, p. 189]

²<http://www.eclipse.org>, multi-language programming environment

³<http://www.squeak.org>, portable open source Smalltalk implementation

The virtues of direct manipulation [4] are encouraging: Novices can learn functionality quickly, experts can even create new operations, immediate feedback plus reversibility allows for simply changing the direction of activity if needed while starting to predict the interface's responses. As a result, the cognitive effort is reduced because programmers can directly map their intentions to tool interactions thus quickly making progress in their current activities.

This section explains current implementations of direct manipulation principles found in traditional programming environments. Additionally, we sketch our solution ideas for open problems, which will be described in detail in section 4.

2.1 Representation

“Continuous representation of the objects and actions of interest with meaningful visual metaphors.” —Shneiderman [15]

In the field of object-oriented programming, meaningful metaphors are needed for abstract domain concepts, which may have no representation in the real world. For this, programming tools already provide meaningful representations from an information visualization perspective [14]: (1) neat lists of objects to grasp structure and extent of system parts and (2) powerful text input fields (e.g. code editors, REPLs, search fields supporting regular expressions) to express thoughts in the programming language of your choice. These views can be arranged in overlapping windows (Squeak) or side-by-side in a tiled layout (Eclipse).

However, the predefined set of views has to be reused for any kind of domain-specific task regardless of whether the amount of visible information is appropriate or not. The whole workflow is tool-centric and not object-centric as it should be. This discussion leads to verb-noun vs. noun-verb interaction styles [13, pp. 59]. We argue that programmers naturally think about nouns (artifacts) in the first place and then about what to do (tools) with them.

Our idea is to support programmers to work with software artifacts in a more direct fashion. Once they collected a set of relevant artifacts, it should be possible to describe the information need in place instead of having to switch to another tool and retrieve the artifacts there again.

2.2 Actions

“Physical actions or presses of labeled buttons, instead of complex syntax.”
—Shneiderman [15]

Programming environments invoke actions, like other GUI applications, via buttons, tool bars, and pop-up menus, which are embedded in tool views. From a programmers' perspective, this simplifies communication and navigation between various tools such as code editor, declaration browser, or debugger.

However, the tools fail to support creating new (browsing) actions without losing the current task focus. For example, modifying columns in browsers is rather difficult

and thus unattractive in domain-specific, unplanned situations. This affects especially the problem of *crosscutting concerns* [18] where new domain- and task-specific views would be necessary for efficient navigation and understanding of the problem at hand. Regular search mechanisms—even with reusable results—are often not appropriate.

Our idea is to reveal the recipes that are responsible for selecting and rendering the objects on screen. We want to support programmers to map any scriptable object transformation to new browsing views, which can be reused later on. They should be able to simply locate and adapt the corresponding source code without having to suspend the current activity for too long.

2.3 Feedback

“Rapid, incremental, reversible actions whose effects on the objects of interest are visible immediately.” —Shneiderman [15]

Programming environments update tool views on data model changes automatically. Changes in object editors, e.g. source code editors, can often be easily undone. Thus, the effect of changes is made visible and reversible. For longer lasting operations such as searching or compiling, environments provide progress indications and show time-to-complete estimations.

However, tools fail to provide generic mechanisms for model change notification and progress indication if new domain-specific views are introduced to the environment.

Our idea is to reveal the reasons why browser windows update at all and to allow for modifying those reasons. Additionally, a generic progress indication mechanism should free programmers from blowing up their object transformation scripts unnecessarily.

3 Related Work

Representing source code artifacts efficiently on screen has been researched for several years. A recent example by Bragdon et al. [1] is called Code Bubbles—a graphical interface for Eclipse that arranges compact *bubbles* representing source code artifacts on a scrollable, two-dimensional canvas. There is a similar implementation for Visual Studio called Debugger Canvas [3], which is optimized for debugging tasks. Considering Smalltalk being strongly related to our approach, Olivero et al. [11] created Gaucho—a graphical interface for the Pharo⁴ environment that arranges source code artifacts in nestable containers called *pampas*⁵. Similar to our approach, these prototypes provide flexible containers and layouts to focus on task-related artifacts.

We argue that the flexibility of browsing actions in programming tools depends on the expressiveness of queries. For example, SOUL [2] is a logic program query language that allows programmers to verify assumptions on programs. Implementations can display the results as intentional views [8] thus being able to show, for example, all artifacts of one crosscutting concern on the screen. These queries can be reused similar to *saved searches* in common email clients or database-oriented applications. In

⁴<http://pharo-project.org>, portable open source Smalltalk implementation

contrast, our approach does not introduce an additional scripting language but makes use of the dynamic programming language Smalltalk, which programmers use anyway to write their code in this environment.

Immediate feedback and reversibility of actions is a resource-intensive problem for programming environments. There are approaches that update important caches automatically on code changes such as Steinert et al. [17] executes tests in background. Reversibility, however, is more of a conceptual problem: Which actions should be tracked, grouped, and reverted? Steinert et al. [16] develop CoExist—a tool that continuously keeps track of any tiny source code modification to free programmers of thinking about possible risks upfront while allowing for going back at any time. Our approach introduces a new way to provide unanticipated progress feedback for long running operations that block the main event loop without having programmers to introduce hooks into their queries.

4 Our Research Programming Environment

In this section, we explain the status quo of our programming environment VIVIDE, which implements several ideas of direct manipulation interfaces applied to the domain of programming. Due to space constraints, we sketch basic ideas only.

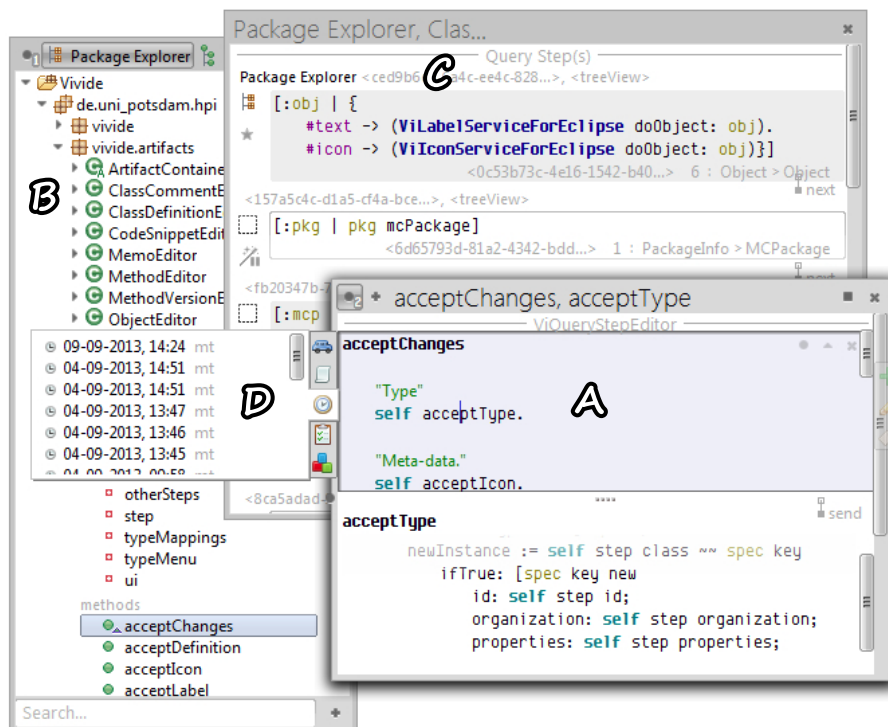


Figure 1: Our environment VIVIDE allows programmers to work with all kinds of software artifacts side-by-side. Object editors (A) represent detailed, editable views for single objects such as methods. Scriptable views (B) support exploring object relationships as needed. Programmers can modify underlying queries (C) to adapt views directly. In-place queries (D) provide immediate access to related objects.

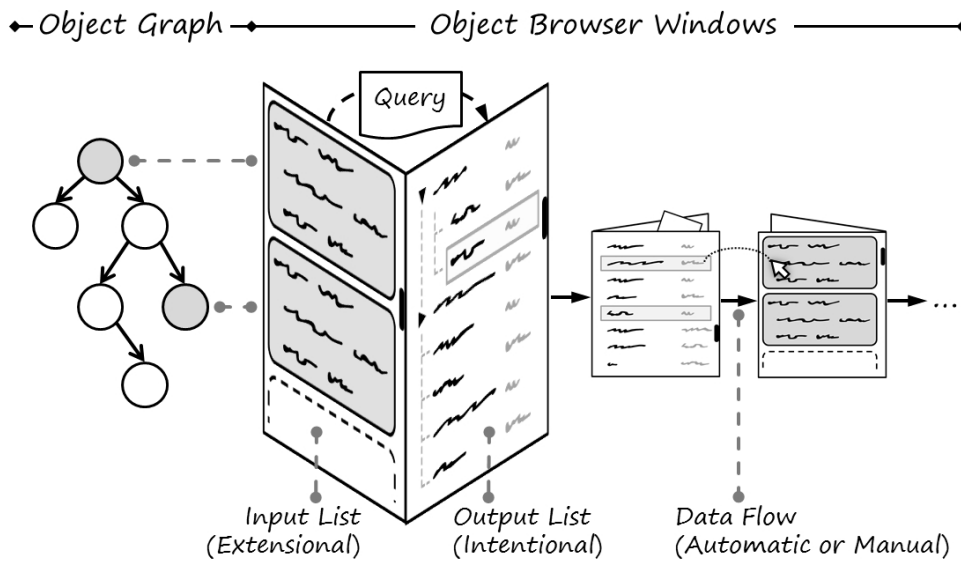


Figure 2: Object views consist of multiple independent windows each having two sides: input and output. The input side stores a user-selected list of editable single-object views. The output side shows a multi-object list view that is generated by a query evaluated on the input. Data flows between windows via propagating selections or drag-and-drop.

4.1 Representing Nouns Before Verbs

We argue that programmers naturally reason about relevant software artifacts in the first place and then about what to do with them. Thus, our prototype (Figure 1) favors a noun-verb interaction style [13, pp. 59] and unifies the concept of object views meaning that there actually is no tool in the traditional sense⁵. This supports programmers to follow up on any clue when reasoning about software artifacts—not the tools are focused but objects and relationships. Given several task-related objects, they can choose from a set of *queries* to reveal interesting (static or dynamic) relationships such as the methods of a class or possible arguments for a method. Having this different point of view, programmers do not have to look for appropriate tools that hopefully provide the desired information but rather to think directly about the information itself. We believe that this removes one level of indirection, which will reduce the cognitive effort.

The windows in VIVIDE (Figure 2) have two sides: input and output. The input side is an extensional list, which is filled manually by the programmer who collects objects of interest while browsing the system. The output side is an intentional list, which is filled when evaluating a user-defined query. Having this, all kinds of common browsers can be created—even debuggers where the input is a list of stack frames and the output is a view on the objects of the particular system state.

⁵The definition of the term “tool” is diverse. We mean traditional tools such as class browser, file browser, or SVN browser.

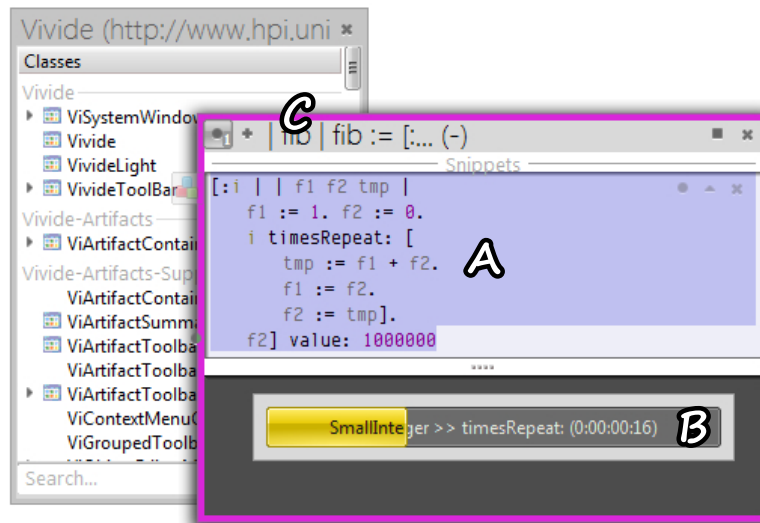


Figure 3: Feedback for long running operations (A): A watchdog supervises the main event loop of the programming environment, analyses its call stack frequently, and extracts progress information (B) from recurring stack frames. Responsible windows will be highlighted (C) to pinpoint work of hidden ones.

4.2 Object Queries as Adaptable Browsing Actions

We argue that programmers who are focused on the task at hand can easily pinpoint problems in their tool views meaning information overload or need. With our prototype, programmers can simply reveal the recipe that is used to produce any particular view (Figure 1 C). Having this, they can adapt those recipes directly and see the result immediately without losing sight of the objects of interest. There is no need to dive into a tools sources to alter its behavior; programmers can stay focused on their current activity while adapting the views. We call these recipes *object queries*.

An object query consists of a linked list of *steps* where each step has a *definition*, which is a piece of source code. Thus, steps and definitions are unit of adaption and reuse. Steps can either *transform* objects (including select, sort, and group) or extract several *properties* to be displayed in object views. Both kinds of steps can interleave thus giving programmers the chance to describe more advanced views such as heterogeneous trees. When a query is being evaluated, objects are “flowing” through the steps while *data model nodes* are being generated at each consecutive list of property steps. This means that there has to be at least one (default) property step for a query. In contrast to traditional linked lists, one cycle is possible to allow for recursively describing data models of dynamic size. Having many lists of those query steps, any step in a list can be used as entry point and thus serve as a query of its own.

4.3 Unanticipated Progress Feedback

We want to support programmers to focus on the current programming activity. Thus, we provided a simple way to adapt object browsers if necessary by means of query steps and definitions as described above. However, there are two important questions

that need to be answered to fulfill the third direct manipulation principle about immediate feedback: (1) When to reevaluate queries to propagate object changes? (2) How to handle long running queries? The first question is answered with additional source code for each query definition that is intended to subscribe to an observer mechanism. We argue that this place is good to decide when the outcome of an object transformation might change. False-positives would decrease tool performance, false-negatives could be mitigated with manual query reevaluation.

In the scripting world⁶, programmers write pieces of source code that are “good enough” for a particular purpose. Reusing those scripts under different conditions can result in unexpected long response times and—as in our case where reusing queries all the time—thus leading to an unresponsive user interface. We argue that programmers take more time and produce less readable scripts when having to consider progress indication. Our approach (Figure 3) is to let a *feedback thread* observe the main application thread by taking frequent⁷ call stack samples. Within these samples, recurring stack frames are mapped to tasks with current progress according to previously defined *strategies*. Having this, the operation description is independent of the progress description and the performance overhead is independent from the operation duration. It scales up: The overhead is inversely proportional to the CPU speed because the call stack analysis needs always the same amount of CPU cycles for a given setup of strategies.

5 Evaluation

As for now, the most valuable feedback about applicability has been collected by using VIVIDE to modify itself and implement new features as needed since January 2013. Additionally, we conducted two pilot case studies: Several computer science students had the possibility to accomplish software projects within VIVIDE in an undergraduate lecture and a graduate seminar. Our goal was to collect insights from programmers who are not directly involved in this research project.

In this section, we explain the experimental conditions and results, which cannot be generalized but serve as starting point for further studies.

5.1 Method

At first, participants got a brief tutorial to learn about the basic concepts and functions of the tool. We allowed questions and gave support throughout the whole period, which was from the beginning of April to the end of July 2013. VIVIDE provides an integrated feedback mechanism, which gives users a way to supply questions, ideas, problems, or praise with a single button click. Thus, we were able to provide direct support. Nevertheless, participants could decide to stop using VIVIDE at anytime if

⁶We assume that code is executed in the main event loop and that concurrency is not an option because the result is important for the user to continue work.

⁷We choose a frequency of 4 Hz as a trade-off between analysis overhead and perceived responsiveness.

they wanted to. After completing their tasks, we provided a simple questionnaire for debriefing purposes:

1. Did you use VIVIDE for programming? (yes/no)
2. Why did you do so? (free text)
3. Did you create custom queries? (yes/no)
4. If you did, which ones? (free text)

5.2 Participants and Task

There were 68 undergraduate students in the lecture and 18 graduate students in the seminar. All of them were familiar with the Squeak/Smalltalk environment and thus knew about how to use the standard toolset. None of them did use VIVIDE before. In both cases, students had to work on new software projects in teams of three to four.

5.3 Results

Only 50 out of 86 responded to our questionnaire. From this, we know that at least 19 students tried to use VIVIDE for their projects. However, 12 of them stopped using it after a while mainly because the standard toolset seemed to be better suited for their projects and there was no interest in learning a new tool.

Students fully ignored it because: “normales Squeak hat ausgereicht / keine lust wieder neues UI einzulernen” (Plain Squeak was good enough. Didn’t feel like learning new UI), “Angst vor neuem :P” (Afraid of new things), “bei gemstone, seaside, gemtools, pharo und so weiter, brauchte ich nicht auch noch eine neue IDE” (So much new libraries to learn. Did not want to learn a new IDE as well). It seems that some students did not want to pose additional risks to project outcome by trying out a new research prototype that may impede progress.

Students stopped using it because: “Die Performance war unter Ubuntu [...] miserabel [...]” (It was unusable slow with Linux), “Der Bug [...] ließ anfänglich kein angenehmes Arbeiten dazu. Danach habe ich aufgehört vivide zu nutzen.” (There was a bug that impeded our workflow. Then we stopped using VIVIDE). Thus, we got valuable information about serious bugs and showstoppers.

Students kept on using it because: “Vor allem wegen der kleinen, flexiblen Fenster. Die Andockfunktion ist effizient nutzbar.” (Mainly because of the small, flexible windows. The docking function is efficient). There was one team of undergraduates that used VIVIDE throughout their project because its topic naturally fitted: “Im Rahmen des Projekts Expert Knowledge Mining. Queries sind eine gute Sache, um erste Codesnippets auszuprobieren, die Listen liefern sollen.” (For our project Expert Knowledge Mining. Queries are well suited to try out first code snippets that should produce lists of objects). This supports our main assumption that programmers can better focus on their current activities while browsing domain-specific object relationships.

The feedback mechanism was used 20 times by 9 different students. There were students that kept on using VIVIDE until the project end but never submitted any feedback report. There were 4 questions, 3 ideas, 12 problems, and no praise. Each report

had the current screenshot attached, which allowed us to understand the concern more quickly. In the end, we could solve all problems, answer all questions and got valuable information about where to improve our programming environment.

5.4 Hypotheses

Our experience during the last twelve months revealed several hypotheses, which we are going to prove either by experimentation or literature studies:

Correctness All domain-specific browsing tasks can be accomplished with queries.

Practicability If programmers can easily see *which rules* produce a particular view, they will adapt those rules to accommodate domain-specific scenarios.

Impact If programmers can work with task- and domain-specific views, they will need less time to accomplish programming tasks because of the reduced cognitive effort.

6 Conclusion and Next Steps

Our VIVIDE programming environment got into a useful and usable state, where we are able to fully replace all standard tools that are not for refactoring or version control. Programmers can perform important programming activities such as code creation, modification, navigation, and debugging with the help of over 80 queries, which are distributed among 30 different object types. In the regular setup, we cover three different domains: the Smalltalk standard library, test-driven run-time analysis [12], and the query system itself.

VIVIDE shows that, using our approach, tools can take advantage of additional information to be derived from dynamic programming systems such as Squeak/Smalltalk. As for browsers, there can be a user interface that allows programmers to modify its views in situ. There is no need to switch to a different code editor for this purpose since code can be entered and evaluated everywhere.

For the next steps, we are going to validate the hypotheses as described above, think about how queries can support program modification in addition to browsing, and explore the applicability of VIVIDE to further domains outside Squeak.

References

- [1] Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeptura, and Joseph J. LaViola Jr. Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, pages 2503–2512. ACM, 2010.
- [2] Coen De Roover, Carlos Noguera, Andy Kellens, and Vivane Jonckers. The SOUL Tool Suite for Querying Programs in Symbiosis with Eclipse. In *Proceedings of the*

-
- 9th International Conference on Principles and Practice of Programming in Java*, pages 71–80. ACM, 2011.
- [3] Robert DeLine, Andrew Bragdon, Kael Rowan, Jens Jacobsen, and Steven P. Reiss. Debugger Canvas: Industrial Experience with the Code Bubbles Paradigm. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 1064–1073. IEEE Press, 2012.
- [4] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct Manipulation Interfaces. *Human-Computer Interaction*, 1(4):311–338, 1985.
- [5] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-oriented programming*. Springer, 1997.
- [6] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information Needs in Collocated Software Development Teams. In *Proceedings of the 29th International Conference on Software Engineering*, pages 344–353. ACM/IEEE, 2007.
- [7] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software Maintenance Tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, December 2006.
- [8] Kim Mens, Bernard Poll, and Sebastián González. Using Intentional Source-Code Views to Aid Software Maintenance. In *Proceedings of the 19th International Conference on Software Maintenance*, pages 169–178. IEEE, 2003.
- [9] Yoshiro Miyata and Donald A. Norman. Psychological Issues in Support of Multiple Activities. *User Centered System Design*, pages 265–284, 1986.
- [10] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 1988.
- [11] Fernando Olivero, Michele Lanza, Marco D’Ambros, and Romain Robbes. Enabling Program Comprehension through a Visual Object-focused Development Environment. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 127–134. IEEE, 2011.
- [12] Michael Perscheid, Bastian Steinert, Robert Hirschfeld, Felix Geller, and Michael Haupt. Immediacy through Interactivity: Online Analysis of Run-time Behavior. In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 77–86. IEEE, 2010.
- [13] Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional, 2000.
- [14] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the Symposium on Visual Languages*, pages 336–343. IEEE, 1996.

- [15] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 5th edition, 2009.
- [16] Bastian Steinert, Damien Cassou, and Robert Hirschfeld. CoExist: Overcoming Aversion to Change. In *Proceedings of the 8th symposium on Dynamic languages*, pages 107–118. ACM, 2012.
- [17] Bastian Steinert, Michael Perscheid, Martin Beck, Jens Lincke, and Robert Hirschfeld. Debugging into Examples: Leveraging Tests for Program Comprehension. *Testing of Software and Communication Systems*, pages 235–240, 2009.
- [18] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton Jr. N Degrees of Separation: Multi-dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119. ACM, 1999.

Communication-Aware and Memory-Aware VMs Consolidation

Ibrahim Takouna

Internet-technologies and Systems
Hasso-Plattner-Institute
ibrahim.takouna@hpi.uni-potsdam.de

This report presents two of my recent research contributions. First, it presents communication-aware VMs consolidation for communicative VMs that host parallel or dependent jobs. Second, it presents memory-aware VMs consolidation for non-communicative VMs that host independent jobs. Finally, it depicts my thesis contributions and next steps.

1 Communication-Aware

Consolidating parallel applications based on server resources (e.g., CPU or memory) might can be inefficient, causing a very significant degradation in the performance of the parallel applications. Walker compared the performance of NPB MPI using Amazon EC2 and the National Center for Supercomputing Applications (NCSA) [1]. The result showed that the programs CG, FT, IS, IU, and MG had greater than 200% performance degradation. We conjecture this to the random placement of these applications. For instance, Figure 1 shows the mapping between jobs and servers for 10 parallel applications, where each has four jobs. In this example, the VMs are scheduled based on CPU without considering the correlations among VMs. It is clear that not all VMs that belong to a single application are scheduled on the same server. For instance, each VM of App2 are placed into different server where VMs 1 and 3 are placed on Server 1, and VMs 4 and 5 are placed on Server 5. This placement causes high performance degradation for parallel applications and also inefficient network utilization. There is little research on communication-aware load balancing schemes for parallel applications in virtualized data centers. For instance, Qin et al proposed a communication-aware technique for enhancing the performance of communication-intensive applications through an efficient utilization for the network in non-virtualized cluster environments [2].

1.1 Methodology

Our goal is to achieve an efficient scheduling for communicative VMs and minimize energy consumption by servers and network components. Usually, the approaches that are not communication-aware can cause a random placement of the VMs. This might improve the resource utilization at the server level but worsen the network bandwidth

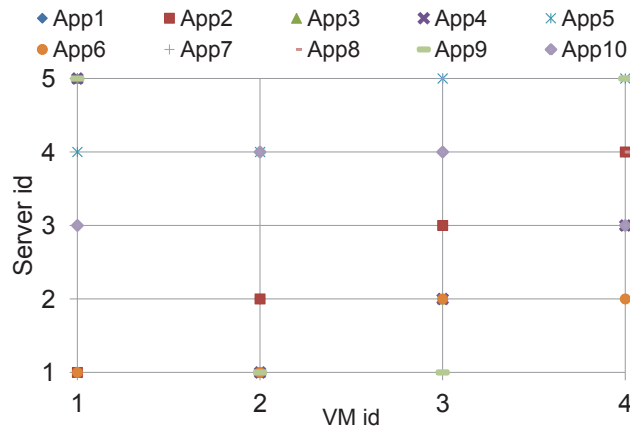


Figure 1: CPU based placement approach

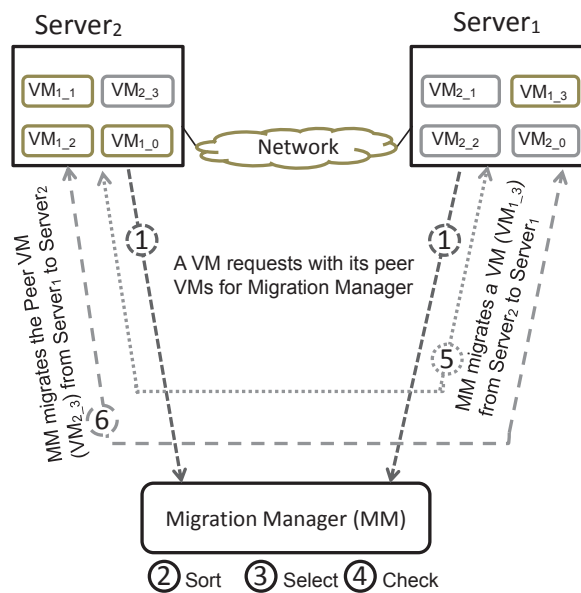


Figure 2: The proposed approach: Peer VMs Aggregation

utilization at the switch level and the performance of these VMs. Our approach is to aggregate the communicative VMs exchanging traffic between each others to be placed in the same server. This assists in localizing the traffic and minimizing the communication delay among communicative VMs. To this end, we developed the Peer VMs Aggregation (PVA) approach, which is presented in algorithm 1. Our approach requires VMs to be involved with the Migration Manager (MM) to determine the communication patterns among VMs.

Our approach is built based on the following issues. The jobs in virtualized environments are executed on VMs. A VM can easily determine the VMs that communicate with it, but the VM is not capable of determining the communication pattern of the whole application. Determining the communication pattern is the role of the Migration Manager. Furthermore, we consider that the VMs communicate with one another through a shared network. However, when VMs are scheduled on the same server, they can

communicate through a shared memory. Importantly, the servers always have enough free resource capacity for using migration (e.g., 10% to 20% of CPU).

Algorithm 1: Peer VMs Aggregation (PVA)

1 Initialization

Each VM_i has communication with other peer VMs sending a request with its peer VMs for Migration Manager (MM), $VmsRequests.add(VM_i, peer_VMs)$

While ($VmsRequests$ is not empty) **do**

2 MM Sorts VMs in descending order based on the the number of in/out traffic flows from the VM's server to its peer VMs' servers

3 MM Selects the first ranked VM to be migrated to the destination server ($Server_2$) of the peer VMs

4 MM Checks the suitability of the destination server
If is suitable

MM Migrates the VM from the source ($Server_1$) into destination ($Server_2$)

Else

5 MM Migrates a VM from the source ($Server_1$) to the destination ($Server_2$) whereas this VM is not one of the VM's peers and is suitable to be hosted by the source ($Server_1$)

6 MM Migrates the VM from the source ($Server_1$) to destination ($Server_2$)

Now, we discuss our approach. The communicative VMs initially send requests to the MM. The request for migration contains a list of peer VMs of the requesting VM. For example, these VMs, VM_{1_0} , VM_{1_1} , VM_{1_2} , and VM_{1_3} , belong to an application where VM_{i_j} indicates VM j of application i . Thus, the request of VM_{1_3} contains VM_{1_0} , VM_{1_1} , and VM_{1_2} . Similarly, the request of VM_{2_3} , which belongs to another application, contains VM_{2_0} , VM_{2_1} , and VM_{2_2} . The MM is responsible for four procedures: Sort, Select, Check, and Migrate. Importantly, MM performs these procedures iteratively on servers until it the algorithm converges. The convergence of the algorithm means that all VMs belong to the same application are scheduled on the same server. We next illustrate each procedure.

- **Sort:** MM sorts the VMs, in descending order based on a number of the in/out traffic flows, after compiling these requests to discover the communication patterns and determining the current placement of these VMs on the servers. As VMs have no knowledge of whether they are hosted on the same server, MM determines that and ignores the requests of VMs scheduled on the same server.
- **Select:** MM selects the top ranked VM to be migrated to the destination server that contains its peer VM. For instance, the selected VM (i.e., VM_{1_3} , which is hosted in the server $Server_1$) has very mutual communication with other VMs

(i.e., VM_{1_0} , VM_{1_1} , and VM_{1_2} , which are hosted $Server_2$). Thus, the selected destination server is $Server_2$.

- **Check:** MM checks the suitability of the destination server in terms of CPU, memory, and network, as the resource demand of a VM is become known at the run time.
- **Migrate:** MM directly migrates the selected VM If the destination server is suitable. Otherwise, MM tries to migrate a VM (i.e., VM_{2_3}) from the destination server ($Server_2$) to make room for the selected VM (VM_{1_3}) to be placed in the same server ($Server_2$) of its peer VMs. However, the selected VM (i.e., VM_{2_3}) should be also suitable to be migrated to $Server_2$. In this example, VM_{1_3} is migrated from ($Server_1$) to ($Server_2$) and VM_{2_3} is migrated from $Server_1$ to $Server_2$, simultaneously. However, if there is no VM to be moved from the destination server, MM can place the selected VM on a server shared the same edge switch with the server of its peer VMs.

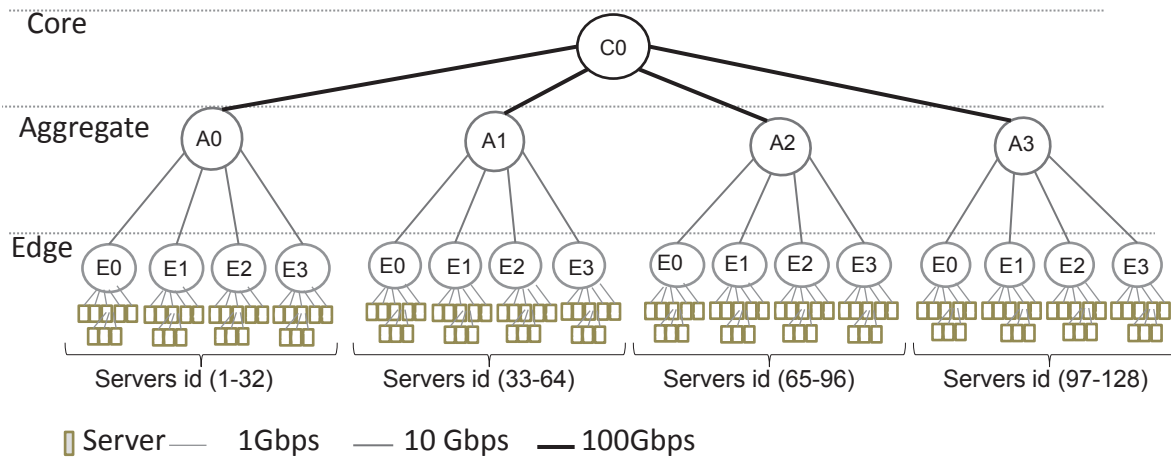


Figure 3: The simulated data center

1.2 Evaluation

To evaluate our approach, we generated 200 different parallel applications of NPB benchmark, inducing BT, SP, IS, CG, LU, and MG. The parallel application generated four jobs whereas each was executed in a VM. Each job required 1000 MIPS. The ideal execution time of the jobs is 1000 seconds. We also simulated the communication pattern of these applications. The total number of servers in our simulation was 128 servers. The simulated server model is HP ProLiant DL380 G7 (3.07 GHz, Intel X5675 processor with 6 Cores) and its memory-bus bandwidth is 40 GB/s. We considered that the total capacity of the server was 9000 MIPS and each server could host at maximum 8 VMs to keep CPU resource for migration overhead. Seeking simplicity, we assumed that all jobs were similar in their start time and length. We simulated the memory demand of these jobs. We simulated the placement of the 800 VMs (i.e., jobs)

based on CPU utilization random fashion. This is considered as the initial placement of the VMs where the CPU based approach is unaware of communication pattern among VMs. We then simulated our proposed approach, which can discover the patterns and online rearrange the places of the VMs using migration.

During the simulation, we measured the following metrics: performance degradation, the VMs placement, memory-bus utilization for each server, network utilization for each link, number of migrations, energy consumption of servers and switches. Using these metrics, we can evaluate the efficiency and the effectiveness of our approach in comparison to the CPU based placement.

1.2.1 VMs placement

Figure 4 shows the distribution of VMs on the servers comparing our approach and CPU based placement approach. In Figure 4, the x-axis represents the VM index of a specific application, and the y-axis represents the server index, which hosts the VM. Figure 4 clearly shows a uniform shape. This means that our approach aggregates all the VMs belongs to an application into the same server. Contrarily, CPU based placement shows a random placement of VMs due to the lack of awareness about the communication among VMs. Thus, our approach, PVA, is capable of achieving a perfect placement after determining the communication pattern of communicative VMs.

1.2.2 Network utilization

Figure 5 shows the average network utilization of the main links, which connect the switches together. For instance, the link C0-A0 represents the link between the core switch with $id = 0$ and the aggregate switch with $id = 0$ as shown in Figure 3, which depicts the data center network architecture. Importantly, the average utilization of the link (C0-A0) significantly decreases using our approach compared to the CPU based placement approach. This is because the link (C0-A0) aggregates the traffic of the servers (1 to 32) as shown in Figure 3 and our algorithm quickly rearranges the VMs in these servers after some time of the simulation. However, the average utilization of the link (C0-A2) is almost the same using both approaches because our algorithm converges when the VMs placed into servers (65 to 96) almost finish executing their jobs. Our approach reduces the average utilization of the network by 25% where the average utilization for all links of the network is 27% by using our approach while it is 36% by using CPU based placement. Our approach outperforms the CPU based placement in terms of reducing the network utilization. As our approach moves the communication between VMs from shared network to shared memory by aggregating the communicative VMs into the same server.

1.2.3 Number of migrations

As we exploited migration, we calculated the number of migrations for each VM. Figure 6 depicts the inverse CDF of the number of migrations for each VM. It shows that 30% of VMs were migrated at least once, and only 5% of VMs were migrated more than 4

times. Algorithm convergence depends on the randomization of VMs, the number of VMs, how many parallel migration can be performed.

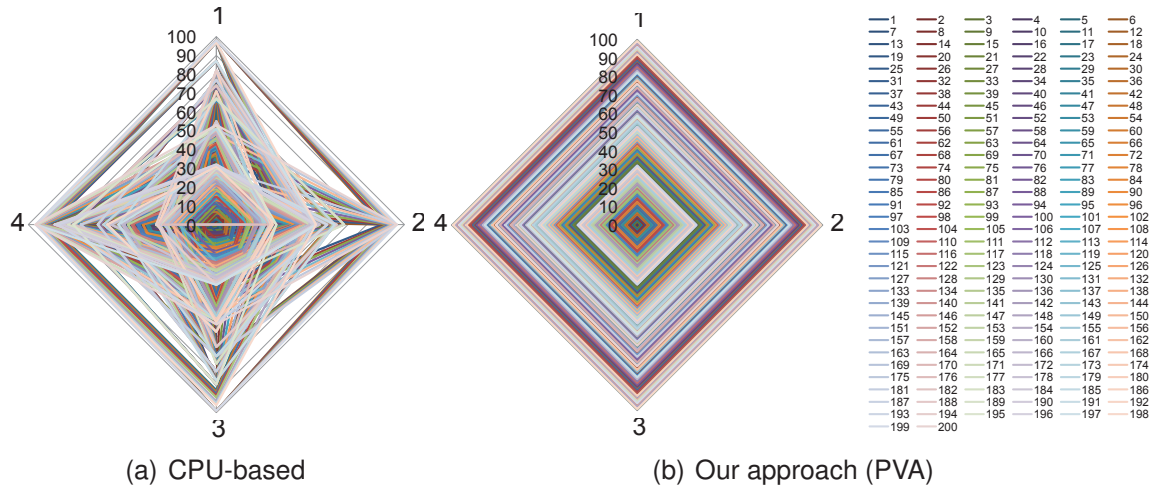


Figure 4: The place of VMs into servers: CPU based vs. our approach placement

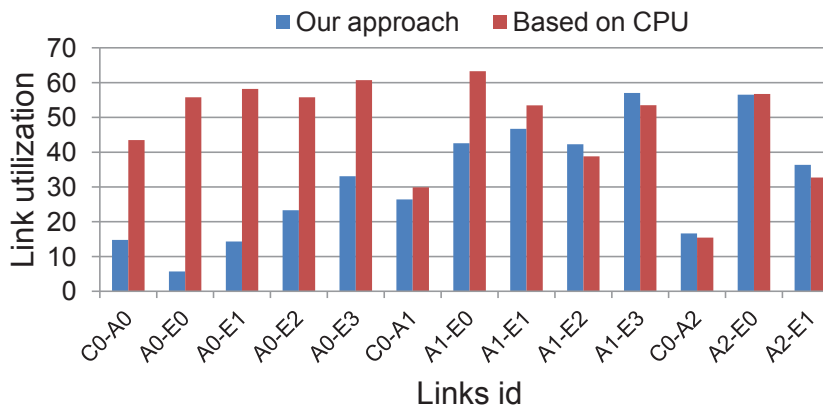


Figure 5: The average utilization of the links between switches

2 Memory-Aware

Our goal is to improve performance of the overall system by consolidating VMs efficiently based on memory-access demand. To this end, we propose a memory-access-pattern-aware consolidation approach exploiting the heterogeneity of the applications memory access and demand. We implemented our approach into CloudSim simulator, which is well known for simulating VM consolidation and energy consumption. To evaluate our approach, we simulated the access and demand of 8 programs of NPB-OMP benchmark suite based on real measurement from [3]. We also emulated the shared memory-bus of a server as RR-FCFS scheduling algorithm. To compare the

results, we measured the performance of the application when it is run alone, CPU based consolidation, and our approach. The results show that we can achieve balance in memory-bus utilization and improve of the system compared to CPU-based consolidation.

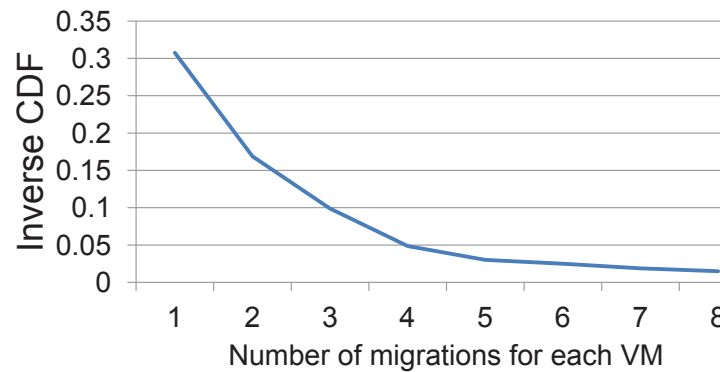


Figure 6: The CDF of the number of migrations for each VM

2.1 Methodology

Memory-bus access unaware VM consolidation can cause high system degradation. Unfortunately, the demand of memory-bus access of an application is determined on-line. We exploit live migration to balance memory-bus utilization among the running servers. Thus, we present Memory-bus Load Balancing Algorithm 2. MLB algorithm takes hosts of cluster and a numerical value $Alpha$, which specifies the intensity of balancing as input and returns the migration map, which contains VMs to be migrated and the destination host. MLB algorithm selects a host that its memory-bus is highly utilized (i.e., maxHost) and a host that has the lowest memory-bus' utilization (i.e., minHost). Then, it compares the variation of the VMs' access demand of the hosts using Wilcoxon unpaired test. Based on the $P.value$ of Wilcoxon test and $Alpha$, MLB algorithm decides whether migrate VMs or not. If $P.value$ less than $Alpha$, MLB algorithm selects a VM with highest memory access from the host with high memory-bus utilization and a VM with lowest memory access from the host with low memory-bus utilization. Then, it swaps their placement and adds them to migration map to be migrated. The maxHost changes each iteration of i to get the next maxHost.

2.2 Evaluation

To evaluate our proposed approach, we conducted several simulations. We study the performance of our approach against the default placement of VMs based on CPU utilization. The evaluation includes the memory-bus utilization, the performance degradation, the average latency in each server, energy consumption, and the number of migrations. We simulate 100 VMs where each VM runs one of NPB benchmarks. These VMs are hosted in 13 hosts. Furthermore, we use different values of $Alpha$ where a low value (i.e., $Alpha = 0.1$) represents low balanced and a high value (i.e., $Alpha = 0.8$) represents high balanced.

Algorithm 2: Memory-bus Load Balancing (MLB)

```

Input: clusterHostList, Alpha
Output: migrationMap
1  $l \leftarrow \text{clusterHostList.size()} / 2$  , migrationMap  $\leftarrow$  null
2 for  $i \leftarrow 0$  to  $l$  do
3   maxHost  $\leftarrow$  descendingOrderMemBwHost().get(i)
4   minHost  $\leftarrow$  ascendingOrderMemBwHost().get(0)
5   P.value  $\leftarrow$  Wilcoxon(maxHost.vmList(), minHost.vmList())
6   if P.Value  $<$  Alpha then
7     /* maxVM and minVM must not be in migration */
8     maxVm  $\leftarrow$  maxHost.findMaxMemBwUtilizedVm()
9     minVm  $\leftarrow$  minHost.findMinMemBwUtilizedVm() /* The swap function
10    sets the new mapping of the VMs */
11    migrationMap.add(swap(maxVm , minVm))
12    if migrationMap  $\neq$  null then
13      return migrationMap
  /* In case the for loop ends without finding suitable VMs
  to be swapped, the algorithm returns null */
12 return migrationMap

```

2.2.1 Memory-bus utilization

Figure 4 shows the VMs mapping into servers and shows stacked memory-bus demand for servers. It includes CPU-based approach and our approach with different values of $Alpha$. Figure 4-(a) shows stacked memory-bus demand of CPU-based approach. The memory-bus of servers 5 and 6 are highly utilized because these servers hosts memory-intensive benchmark (i.e., IS). The stacked memory-bus demand of servers 5 and 6 is more than 14000MB. This placement causes high degradation of performance in these servers. On the other hand, Figure 4-(a) shows a very low memory-bus utilization of servers 1 and 2. It is clear that not considering memory-bus demand causes unbalance of server's memory-bus and degradation in performance of the system.

Figure 4-(b) shows the result of our algorithm with $Alpha = 0.1$. It clearly shows the difference between our algorithm and CPU-based. In this case, the stacked memory-bus demand of all servers does not exceed 8500 MB. Figures 4-(c), (d), and (e) depicts the results of our algorithm with higher value of $Alpha$, which means higher balancing of memory-bus. Our algorithm with $Alpha = 0.4, 0.6, \text{ and } 0.8$ allows the stacked memory-bus demand to not exceed 8000MB. However, Our algorithm with $Alpha = 0.6$ can be considered as the best. Importantly, reducing the memory-bus frequency (i.e., DVFS of memory) can be applied after our algorithm to reduce energy consumption by memory sub-system.

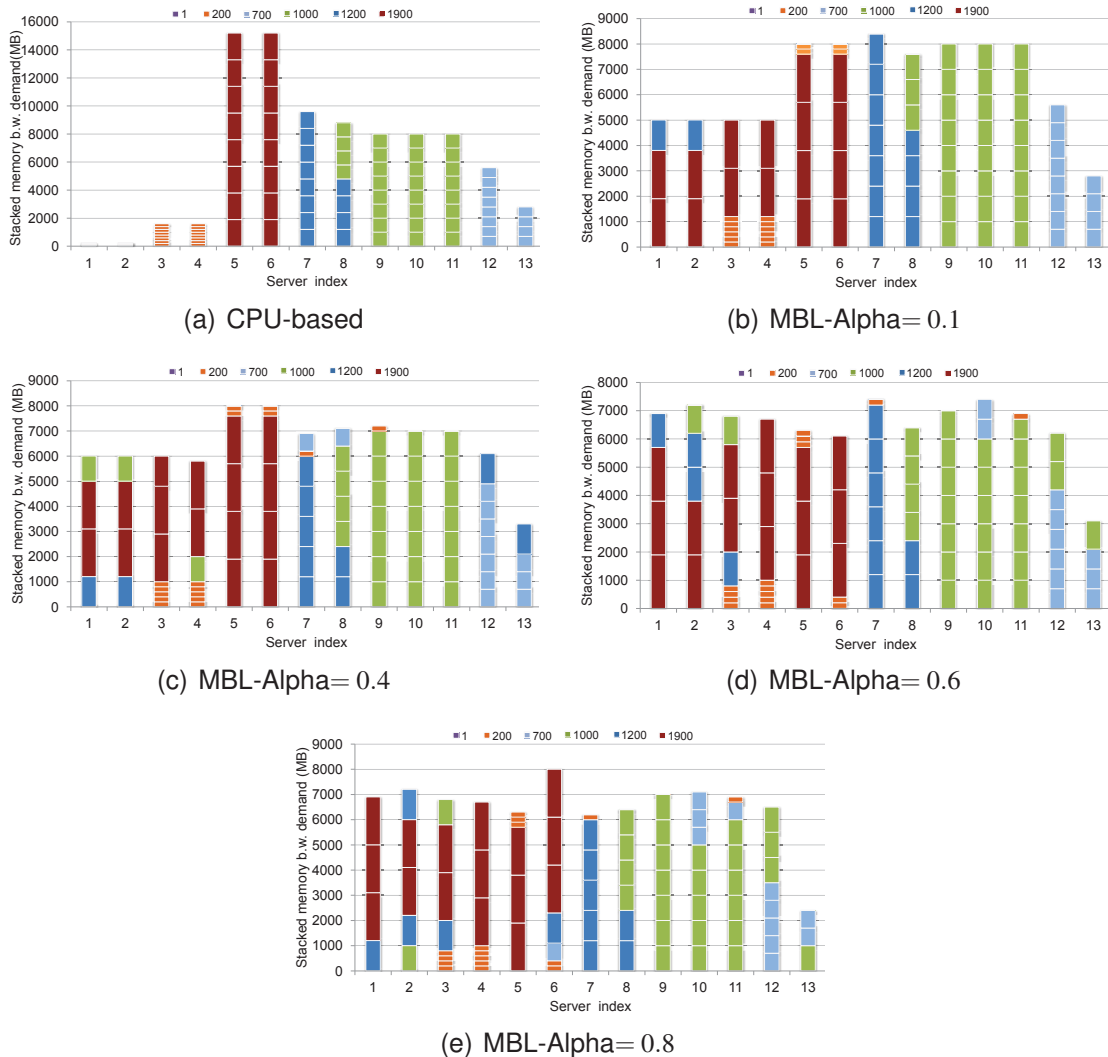


Figure 7: VMs placement: CPU based vs. our approach

2.2.2 Performance degradation

Figure 8 depicts CDF of performance degradation. For instance, we can find that 60% of VMs experienced performance degradation from 15% and over by using CPU-based approach. On the other hand, by using our algorithm, only 40% of VMs experienced performance degradation from 15% and over. Thus, our algorithm outperforms the CPU-based algorithm.

3 Thesis contributions

My thesis targets two categories of VMs: independent/non-communicative VMs and dependent/communicative VMs as shown in Figure 9. Here is a short summary for each contribution.

1. A proactive robust optimization for energy management to mitigate undesirable

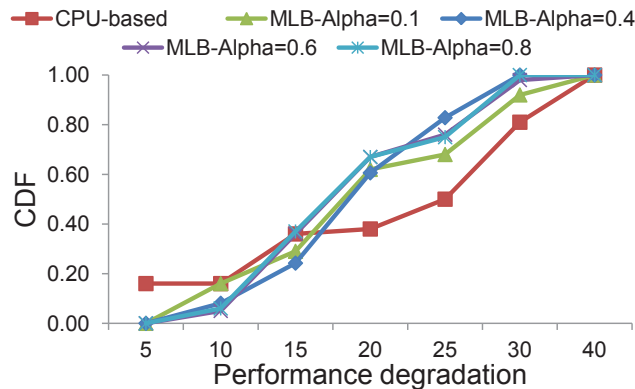


Figure 8: Cumulative distribution function (CDF) of performance degradation for CPU-based approach and our approach (MLB)

changes in the power-state of the hosts by which increases in the hosts' availability for hosting new VMs and reliability against a system failure during power-state change [4].

2. A robust dynamic VM consolidation for efficient energy and performance management to achieve equilibrium between energy and performance trade-off. Our approach reduces the number of VMs migration, which increases the energy consumption by network infrastructure and SLA's violations, and the number of power-state change by 74.8% and 38%, respectively [5].
3. Energy-efficient scheduling of HPC VMs using Host and VM Dynamic configuration to assuage the trade-offs between energy and acceptance ratio of jobs where our approach reduces energy by 20% for executing jobs and increases the system utilization by 45% compared to pure DVFS approach [6] [7].
4. Communication-aware and energy-efficient scheduling for parallel applications to enable dynamic discovery of communication patterns and reschedule VMs based on the determined communication patterns using VM migration. The result shows that our proposed approach reduces the average of the network's utilization by 25% and achieves energy savings of about 60% by reducing the number of active switches and higher VM performance compared to CPU-based placement [8] [9].
5. Energy and performance efficient scheduling of independent VMs by exploiting memory demand heterogeneity. Our approach reactively redistributes the jobs according their utilization of memory-bus using VM migration to improve the performance of the overall system [10].
6. Cooperative approach for thermal-aware and energy-efficient scheduling of HPC VMs involving host power-state change, thermal model of data centers, and scheduling policies.

4 Next Steps

- **Thermal and energy cooperative management:** During my visit to Chinese Academy of Science data center, I obtained real traces of thermal and energy. By using these traces, we can build accurate models of thermal and energy of the data center. I had preliminary results during my visit there for building 3D thermal model for data center to achieve thermal-aware VMs placement. Furthermore, we can cooperate thermal awareness and energy management with resource management scheduling policy (e.g., IBM Platform LSF) to decide which hosts should be turned off.
- **Thesis writing:** I am planning to start writing my thesis in December 2013 after conducting more experiments and achieving results for thermal and energy cooperative management.

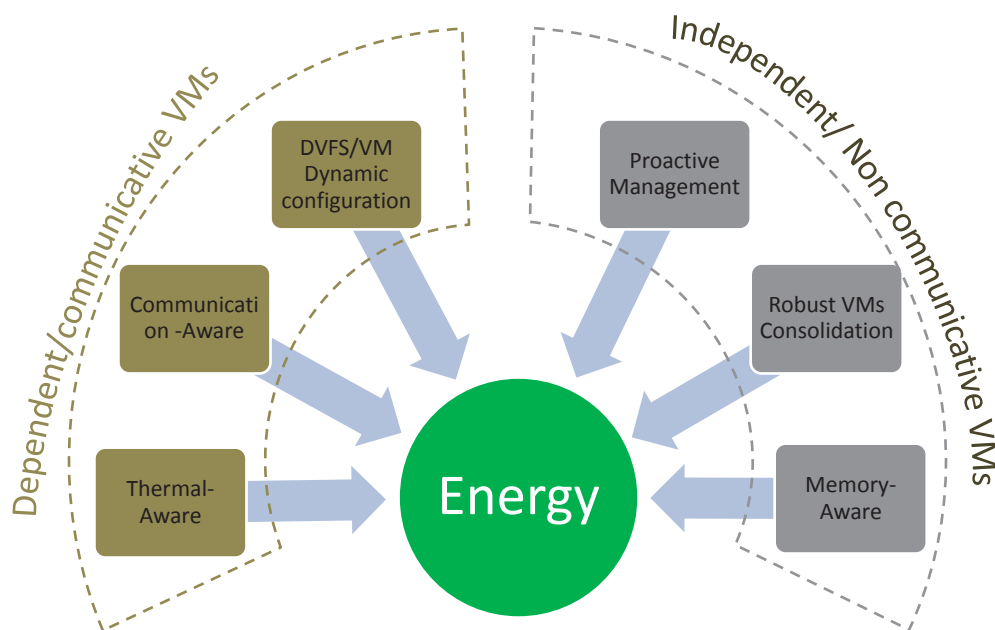


Figure 9: Thesis contributions

References

- [1] Edward Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *Usenix Login*. 33(5):18-23, 2008.
- [2] Xiao Qin, Hong Jiang, Manzanares, Adam Manzanares, Xiaojun Ruan, and Shu Yin. Communication-Aware Load Balancing for Parallel Applications on Clusters. *IEEE Transactions on Computers*. 59(1):42–52, 2010.
- [3] Uday Kiran Medisetty, Vicenc Beltran, David Carrera, Marc Gonzalez, Jordi Torres, and Eduard Ayguad. Efficient HPC application placement in

- Virtualized Clusters using low level hardware monitoring. [Online]. Available:<http://www.udaykiranm.com/hpcvirt.pdf>. [retrieved: Apr, 2013].
- [4] Ibrahim Takouna, Kai Sachs, and Christoph Meinel. Multiperiod Robust Proactive Energy Management in Virtualized Data Centers. *Computing Journal Springer*, 2013. (Under review)
 - [5] Ibrahim Takouna, Kai Sachs, and Christoph Meinel. Robust Dynamic Virtual Machine Consolidation for Efficient Energy and Performance Management in Virtualized Data Centers. In Proceedings of *Fifth ACM/SPEC International Conference on Performance Engineering, (ICPE)*. 2014. (Under review)
 - [6] Ibrahim Takouna, Wessam Dawoud, and Christoph Meinel. Energy Efficient Scheduling of HPC Jobs on Virtualized Clusters using Host and VM Dynamic Configuration. *ACM SIGOPS Operating Systems Review*. 46(2):19–27, 2012.
 - [7] Ibrahim Takouna, Wessam Dawoud, and Christoph Meinel. Accurate Multicore Processor Power Models for Power-Aware Resource Management. In Proceedings of *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. Sydney, Australia, pp. 419–426, 2011.
 - [8] Ibrahim Takouna, Roberto Rojas-Cessa, Kai Sachs, and Christoph Meinel. Communication-Aware and Energy-Efficient Scheduling for Parallel Applications in Virtualized Data Centers. In Proceedings of *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. Dresden, Germany, 2013.
 - [9] Ibrahim Takouna, Wessam Dawoud, and Christoph Meinel. Analysis and Simulation of HPC Applications in Virtualized Data Centers. In Proceedings of *IEEE International Conference on Green Computing and Communications (GreenCom)*. pp. 498–507, 2012.
 - [10] Ibrahim Takouna, Wessam Dawoud, and Christoph Meinel. Energy and Performance Efficient Scheduling of Independent VMs in Virtualized Data Center by Exploiting VMs' Memory Demand Heterogeneity. In Proceedings of *GCM'2013 in conjunction with 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. Dresden, Germany, 2013.

Using Omniscient Debuggers

Arian Treffer

Enterprise Platform and Integration Concepts
Hasso-Plattner-Institut
arian.treffer@hpi.uni-potsdam.de

This paper presents how Omniscient Debuggers allow the advanced navigation of a program execution, which greatly improves the time required to find the source of a failure.

1 Introduction

A large part of developing software consists of finding and removing bugs in existing code. For this, the debugger is considered an important tool. By improving the usability of debuggers, we hope to reduce the time developers need to find bugs, which leaves more time for fixing them.

This leads to the following research questions:

- Which additional operations should a debugger provide to allow the most efficient navigation of a program execution?
- How can the program state be visualized to help the developer to identify important values?
- How can a repository of valid execution traces be used to find errors?

2 Foundation

This section introduces basic terminology and describes a typical debug session, which will be used as a running example throughout the paper.

2.1 Stages of a bug

In a running program, a bug goes through three stages.

A *fault* is a defect in the source code. A fault does not necessarily correspond to a single location, but can be spread over multiple lines or even files. For instance, it could be debated whether the fault is that a method is called with a null argument, or that the method is not able to handle it.

When faulty code is executed, an *error* can occur when the fault causes a part of the program's state to become invalid. Depending on the fault, no error might occur in some situations, which are often also the most common ones. Furthermore, the

error can also disappear when the deviation from the valid state did not impact further execution and the invalid value is overwritten or cleaned up at some point. On the other hand, an error may also grow when it affects other, possibly fault-free, parts of the system, according to the garbage-in-garbage-out principle.

Finally, the error can cause a *failure* if it affects the observable behavior of the system. The most obvious failures are crashes or error messages, but the display of wrong values or incorrect communication with other systems are failures, too. When a failure is not detected and handled, real-world damage can follow.

2.2 Finding faults with a debugger

Very often, it is only through a failure that the existence of a fault is brought to attention. Thus, the developer tasked with finding the fault will first try to reproduce the failure by finding the circumstances under which it occurs.

Then, assuming she is not able to immediately guess the fault, she tries to find the error by examining the program state as the failure happens, using a debugger. Once the error is identified, its source has to be found, which may lie in the same method (e.g., for a variable), but might as well be in an entirely different part of the system (e.g., for a field that is changed in many places). There are two ways to find the source. For both, the debug session has to be restarted.

Firstly, the developer can set a breakpoint at a candidate location. This allows to examine the program state right as the error occurs. However, this approach is not feasible if there are many candidate locations.

Secondly, a breakpoint can be set before the error occurs. Then, the program is stepped through at a high level, skipping methods that are unlikely to produce the error. This approach works better when there are many candidate locations, but less so when they are spread over a longer stretch of time.

Using either approach, the source of the error is eventually found. If it is another error in the program's state, the process has to be repeated until the actual fault is found.

This process can be greatly speeded up with educated guesses based on a good understanding of the code. However, it usually takes several iterations of debugging to get to the fault.

Frequent restarts of the debug session can consume a large amount of time. Breakpoints can be hit multiple times before the point of interest is reached, which requires high concentration from the developer, as she carefully steps through the program. Each restart of the debugger breaks the flow of the actual activity, hunting errors, and consumes valuable short-term memory of the developer.

Furthermore, the process assumes that the developer is able to spot erroneous values. However, the error may not always be obvious, as it might depend on other values whether a given value is valid or not. The developer might not even be aware of these dependencies.

3 Related Work

Backwards and omniscient debuggers have been implemented by Lieberman and Henry [4], Lewis [3], Hofer et al. [2], Pothier et al. [10], and Lienhard et al. [5]. However, these works focused more on the efficiency of the implementation and less on the usability of the debugger.

Daikon is a widely-used application for invariant detection [1] and serves as a foundation for ongoing research [7]. While it is not interactive, like a debugger, the same algorithms can be applied in our context. The work of Perscheid et al. [8] focuses more directly on finding faults by tracing automated tests.

4 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 [2–5]. Many of them internally work like omniscient debuggers, but do not reveal this to the user.

4.1 Modeling the execution trace

Debuggers are a special kind of runtime analysis tools. Basically, there are two ways to implement a runtime analysis.

A live analysis evaluates the program as it is executed; as soon as, or even before, the program terminated, the result of the analysis is available. Common debuggers typically fall into this category.

A post-mortem analysis first records aspects of the programs execution and then analyses the recorded data. Sometimes, this approach has the advantage that multiple analyses may be run iteratively, without having to re-execute the program. This disadvantage of this approach is that, depending on the granularity of the recorded data, it requires much more memory.

Backwards debuggers can be implement with both the live and the post-mortem approach. In the scenario described above, where the debug session begins at the occurrence of a failure, it does not make much of a difference. In other use cases, however, the look-ahead that is possible with the post-mortem approach can make the difference between a backwards and an omniscient debugger.

Many strategies have been proposed to reduce the amount of data that has to be captured to allow a replay of the execution. However, since we aim for an omniscient approach, we will record almost everything.

Figure 1 shows how the execution trace is modeled. The trace model consists of two parts.

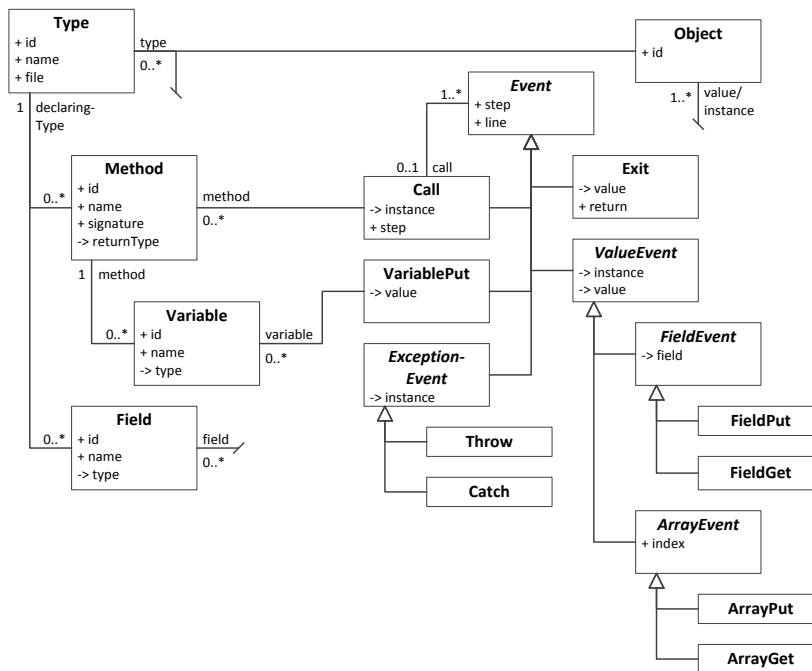


Figure 1: Trace Model

The static part describes the application’s code on a high level. Classes, fields, methods, and variables are represented with enough properties to find them in the code base, so that they can be referenced from the actual trace data.

The dynamic part identifies all objects that occurred during the execution and contains events that happened with these objects. Every event is identified and ordered by a step number and contained in a parent method call (except for the root call).

The program flow is described with call events, which reference the method and the instance on which it was called, and their respective exit events which provide the result value and indicate whether the method terminated via return or exception. Additional program flow is provided by exception throw and catch events.

Value events indicate changes and side-effects of the program state. Represented are field and array reads and writes, as well as variable changes. This way, the state of an object or the variable assignments at a certain point in time can be easily derived from the latest respective set events.

Based on this model, it is possible to develop a debugger that can revert and replay the execution of a program and provide snapshots of the state at any point in time.

4.2 Advanced navigation

As described above, a recurring task is to find the source of a value. It seems obvious how a backwards debugger can improve the time required to find the source of an error.

Restarting the debug session becomes virtually unnecessary. Once an error is identified, the developer can step backwards to its source. If a method call is stepped over by accident (in either direction), the operation can be easily reverted.

Nevertheless, this may still require stepping (backwards) through large parts of the application. An omniscient debugger, on the other hand, immediately knows where the value was set.

Name	Value
this	TimeIntervalParser (id=48)
paramText	"10h49m789, 12h" (id=34)
params	String[2] (id=50)
[0]	"10h49m789" (id=55)
[1]	"12h" (id=56)
result	Long[2] (id=52)
index	0
param	"10h49m789" (id=55)

Figure 2: Variables View in Eclipse

Figure 2 shows the variable view of Eclipse's debugging perspective, which is typically used to spot erroneous values. With the omniscient debugger extension, the developer can directly jump back in time to the assignment of a value simply by double clicking it. This changes the debugging process as follows:

The developer finds the value and double-clicks to jump to its source. She finds that the value is built from three other values, using a formula that seems to be correct. However, she is not sure which of the input values is erroneous. Thus, she bookmarks the current point-in-time and begins to investigate the first value, again by double-clicking it.

Once she stepped around through the value's creation, she is certain that this value is valid and uses the bookmark to return *back to the future*. Then she begins to investigate the second value. When she realizes that it is invalid, this process is repeated until the fault is reached.

As the example shows, another important task is to determine whether a value is valid by examining how it is produced. Here, the omniscient debugger can assist in multiple ways.

Firstly, instead of showing just the current stack trace, the omniscient debugger can provide a tree of previous and subsequent invocations (cf. figure 3). Especially after jumping backwards, the developer may have to regain orientation, where this additional context can be helpful.

Secondly, the debugger can show the history of a variable, or even an entire object (cf. figure 4). Mostly, this is helpful when a value is created in a loop or if an object is changed in multiple, different parts of the application over a longer stretch of time.

Finally, the debugger knows whether a given value is used again or at all. By greying out variables and fields that are not accessed again (at least not before their values are changed), the program state that has to be examined by the developer is effectively reduced.

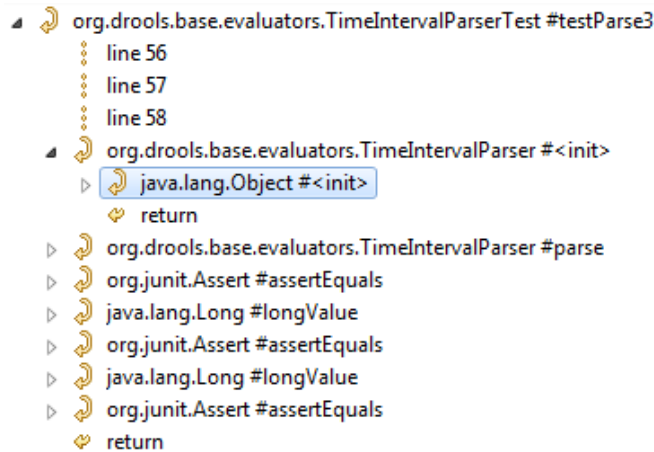


Figure 3: Call Tree

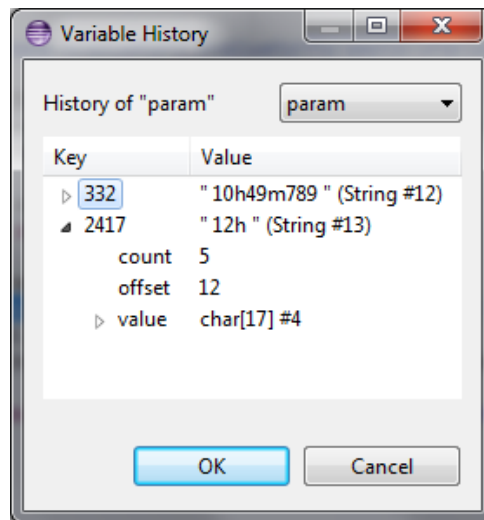


Figure 4: Variable History

5 Hyper Debugging

Texts, by their nature, are one-dimensional strings of characters. A hypertext is a text that contains references to other locations in both other texts and itself. Thus, combined hypertexts form a multi-dimensional structure where each individual text gains additional value from the texts it references. This makes it both easier and more interesting to investigate a complex body of information [6].

A program trace is a one-dimensional sequence of events (or multiple interleaved sequences in multi-threaded applications, but still basically linear). Likewise, a hyper debugger is a debugger that is able to link events in the current execution to events of other execution traces.

5.1 Trace repositories

Figure 1 has shown how a single execution trace is modeled. To store multiple traces of an application, each event has to be associated with a trace id. The static part of the model remains unchanged.

As a source for valid execution traces, automated tests come to mind. An extensive test suite is a representative show case for the expected behavior of the application and can be used for many helpful analyses [8]. Another advantage is that the repository can be easily rebuild when parts of the code base changed. However, tests for exceptional cases should be removed from the repository.

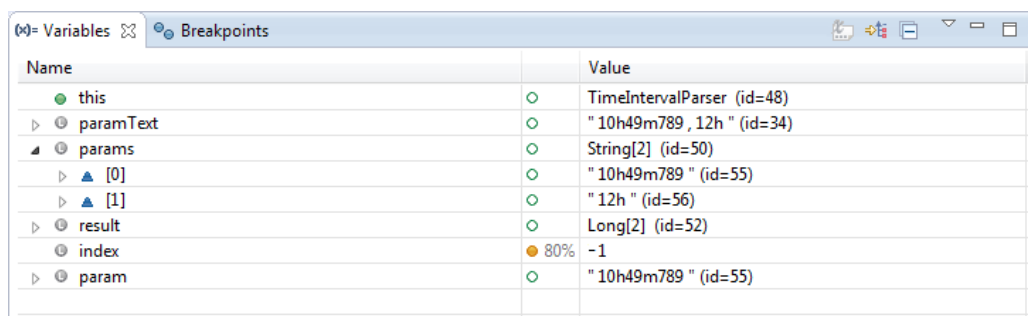
It is often discussed in literature that program traces are too large to be handled efficiently [5, 10]. This applies even more to repositories of many traces. However, we expect that this problem will be solved by the ongoing decrease of memory cost and modern main-memory data structures [9].

5.2 Detecting Invariants

Execution traces can be used to detect invariants or assumptions in the code. The quality of the detected invariants for a given piece of code rises with the number and variety of its invocations [1].

The main problem of regular invariant detection is that the number of potential invariants grows with the size of the code base. Fortunately, in our scenario we are only interested in invariants for the current context, i.e., the variables of a single method or the fields for a single object.

By detecting invariants for a small context in real-time, it is possible to detect potentially invalid values in the current program state, helping the developer to estimate the extent of the current error. Figure 5 shows how the variable view was extended to show the violation of invariants. Additionally, the confidence (derived from the number of exemplary invocations in the repository) is shown. A mouse-over shows the invariants that were violated.



Name	Value
this	TimeIntervalParser (id=48)
paramText	" 10h49m789 , 12h " (id=34)
params	String[2] (id=50)
[0]	" 10h49m789 " (id=55)
[1]	" 12h " (id=56)
result	Long[2] (id=52)
index	80% -1
param	" 10h49m789 " (id=55)

Figure 5: A potentially invalid value

In the best case, this feature helps the developer to quickly identify incorrect values which makes it easier to find the underlying fault. In the worst case, this analysis produces too many false positives, which hopefully encourages the developer to write more and better automated tests.

6 Future Work

The improved navigation features of the omniscient debugger have yet to be validated with a user study. Further research might show better ways to visualize values in the program and their dependencies and changes over time.

The repository of test traces can be used even outside a debug session to help the developer with relevant information while she is writing code. Future work can investigate how relevant information can be found and shown to the developer.

References

- [1] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007.
- [2] Christoph Hofer, Marcus Denker, and Stéphane Ducasse. Design and implementation of a backward-in-time debugger. *NODE 2006*, pages 17–32, 2006.
- [3] Bil Lewis. Debugging backwards in time. *Computing Research Repository*, cs.SE/0310016, 2003.
- [4] Henry Lieberman. Reversible object-oriented interpreters. In *ECOOP' 87 European Conference on Object-Oriented Programming*, volume 276 of *Lecture Notes in Computer Science*, pages 11–19. Springer Berlin/Heidelberg, 1987.
- [5] Adrian Lienhard, Tudor Gîrba, and Oscar Nierstrasz. Practical object-oriented back-in-time debugging. In Jan Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, number 5142 in *Lecture Notes in Computer Science*, pages 592–615. Springer Berlin Heidelberg, January 2008.
- [6] Randall Munroe. Tab explosion. <http://xkcd.com/609/>, 2009. Accessed on October 10th, 2013.
- [7] Saeed Parsa, Behrouz Minaei, Mojtaba Daryabari, and Hamid Parvin. New efficient techniques for dynamic detection of likely invariants. In *Adaptive and Natural Computing Algorithms*, number 6593 in *Lecture Notes in Computer Science*, pages 381–390. Springer Berlin Heidelberg, January 2011.
- [8] Michael Perscheid, Michael Haupt, Robert Hirschfeld, and Hidehiko Masuhara. Test-driven fault navigation for debugging reproducible failures. *Computer Software*, 29(3):188–211, 2012.
- [9] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.

- [10] Guillaume Pothier, Éric Tanter, and José Piquer. Scalable omniscient debugging. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, OOPSLA '07, page 535–552, New York, NY, USA, 2007. ACM.

Enabling Adaptation in Cyber-Physical Systems

Sebastian Wätzoldt

Systems Analysis and Modeling Group
Hasso Plattner Institute
sebastian.waetzoldt@hpi.uni-potsdam.de

This report contains preliminary research results concerning an adaptable software layer for cyber-physical systems (CPS). It introduces a component model and an approach to enable reflection capabilities of the software system using special monitor and effector ports. Furthermore, first ideas of realizing the new port concepts are shown with the help of a complex example from our CPS laboratory.

1 Introduction

Many of today's systems must be highly adaptable to react on changing requirements and environmental conditions. Often, these kind of systems cannot be stopped and updated for economical or safety reasons. Especially, safety-critical systems as embedded and (networked) cyber-physical systems (CPS) [1, 3, 5, 9], ranging from small medical devices to complex cars, planes and trains up to intelligent, distributed traffic management systems or smart factories, must fulfill high expectations on reliability and predictability to operate in an potentially open environment and interact with humans. Usually, feedback loops are used to monitor the physical parts of the system and react according to the current situation.

Following the external adaptation approach from Salehie et. al. [8] and therefore a clear decoupling of adaptation and business logic has several benefits. On the one hand, the adaptation engine can be independently developed and maintained. This decrease complexity and enables an easy distribution on different computational nodes of these two parts. On the other hand, the adaptation engine has to use well defined interaction points/interfaces of the business logic that usually leads to a better overall system design and a reduced error rate. Furthermore, it enables the protection of intellectual property (IP) treating the specific business logic as black boxes.

The basic interaction of adaptation engine and adaptable software system is depicted in Figure 1. The adaptation engine on top contains different activities according to the approach from Kephart and Chess [4] from the autonomic computing domain, namely, **Monitor**, **Analyze**, **Plan** and **Execute** around a common **Knowledge base (MAPE-K)**. The feedback loops interact with the system only via sensing in the monitor activity and effecting during the execute activity using well-defined interfaces. As knowledge representation, we use different kinds of runtime models as for example requirement, context or system models. The MAPE activities are specified in the EU-REMA modeling language from Vogel [10].

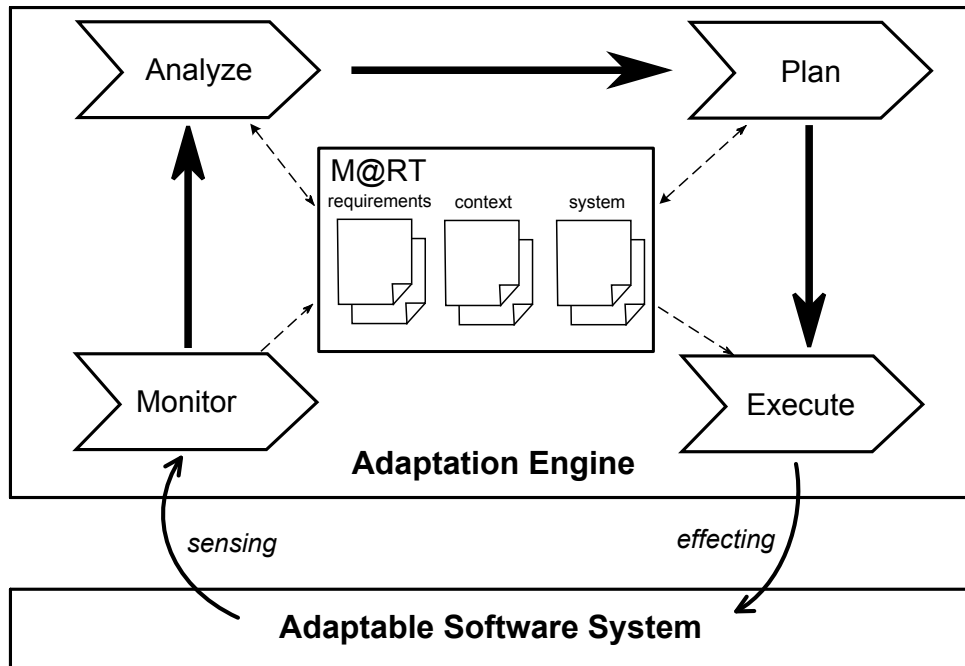


Figure 1: MAPE-K adaptation loop in the adaptation engine and interaction with the adaptable software system.

A problem for embedded and cyber-physical systems using the external adaptation approach can be the interaction between adaptation engine and system logic (see lower part in Figure 1). Key challenges are a missing reflective, dynamic framework as OSGI or the Java reflection API capabilities for dynamic object manipulation (create, delete, link) at runtime. Furthermore, we have to consider resource limitations (memory, energy or CPU power) and timing constraints. The adaptation engine cannot simply interact with the running system without potential violations of these constraints. For CPS the questions are: How can the software provide reflective characteristics without introducing much overhead and still protecting implementation details (IP)? How can the interaction between adaptation engine and software system still guarantee no constraints violations?

In this paper, we try to answer these questions and introduce a component model that enables the interaction between adaptation engine and embedded real-time system. This model can be handled by the adaptation engine as first class entity at runtime. Furthermore, we show a concept of enabling reflective behavior in the CPS architecture, which is part of our component model, too. As a result, we enable static and dynamic reconfiguration on different layers in the CPS. We evaluate our approach by introducing an example from our CPS laboratory¹, where three autonomous robots run in a distributed factory simulation.

The rest of the paper is structured as follows: We discuss related work in Section 2. Section 3 introduces a running example, our component model and the basic concepts of the reflective ports as well as the realization of these concepts. Finally, we discuss future work and conclude in Section 4.

¹www.cpslab.de

2 Related Work

In this paper, we enable adaptation for CPS introducing different communication pattern. Therefore, we touch a number of existing work concerning modeling adaptation of systems as well as using and handling runtime models. In the following paragraphs, we describe some of these ideas and how they influence our work.

Oreizy et al. [7] describes a set of architectural runtime changes such as component addition, removal and replacement and a very basic component model. Furthermore, an approach to describe, use and manipulate the static architecture during runtime using an ADL is introduced. However, the approach does not describe the consequences of changing the system architecture taking timing constraints and adaptation on different abstraction level into account. We adopt the described architectural changes and show how they applied in our approach.

The idea of decoupling adaptation and business logic is described in [8]. Kephart and Chess [4] provide a more detailed description of the adaptation engine from the autonomic computing domain in form of an adaptation loop with the four activities **M**onitor, **A**nalyze, **P**lan and **E**xecute around a common **K**nowledge base (MAPE-K loop). Moreover, Vogel et. al [11] describe the role of the knowledge base in more detail and provide a categorization of runtime models. Furthermore, Vogel introduces the EU-REMA modeling language in [10] for specifying MAPE-K loops. We follow the idea of decoupling the adaptation and business logic and use the modeling language of Vogel to describe the adaptation logic of our running example. However, all mentioned approaches do not describe the interaction of adaptation engine and adaptable software system in more detail. We fill this gap by providing specific interaction pattern and a mapping of them to an implementation skeleton.

Becker et al. [2] present an approach for static reconfiguration in embedded systems using the AUTOSAR² standard from the automotive domain that switches between different system configurations at runtime if necessary. This approach is appropriate for hard real-time systems because all configurations are deployed in the system during development time (a reconfiguration is more or less a simple switch between components). They do not consider an interaction with an adaptation engine. We extend this approach considering less timing predictable (soft real-time) parts of the system and an interaction with an existing adaptation engine.

3 Reflective CPS Architecture

Enabling reflective capabilities in our CPS architecture, we will focus on the bottom adaptable software system layer in Figure 1. In [10], this layer is treated as black box *Software Module* at the lowest layer by modeling adaptation activities.

We claim that the adaptation of a CPS is different to non safety-critical systems, which leads to the following additional considerations. At first, sensing and effecting the system by the MAPE-K loop should not violate timing constraints. Therefore, a black box view without any information about the system can not work. We structure

²<http://www.autosar.org/>

the software layer into two logical parts and we enable the modeling of additional manipulation possibilities. Typically we can identify special software entities as for example sensors and actuators in a CPS with different manipulation characteristics. If we model the interaction of these entities taking timing constraints and manipulation possibilities into account and handle the information as first class entities at runtime, we enable further investigations at the adaptable software as well as the adaptation engine level. For example, we can use model checking techniques to proof whether all constraints still hold after a planning step of a new system configuration. Furthermore, we can identify interaction pattern between different functional parts of our software architecture.

Providing some architectural information about the system and modeling it in form of interconnected components is state of the art in robotic, automotive, CPS and other embedded, safety-critical systems. Therefore, we develop a component model that can be used to provide some more information about adaptation capabilities of the underlying software. Benefits for breaking the black box view of the adaptable software are the identification of communication pattern between components and a clear realization of reflective behavior on different abstraction levels without timing constraint violation.

In the following subsections, we introduce a running example for this paper that is also used in the implementation section later on. Afterwards, we explain our component metamodel. Finally, we describe the realization of reflective properties.

3.1 Example

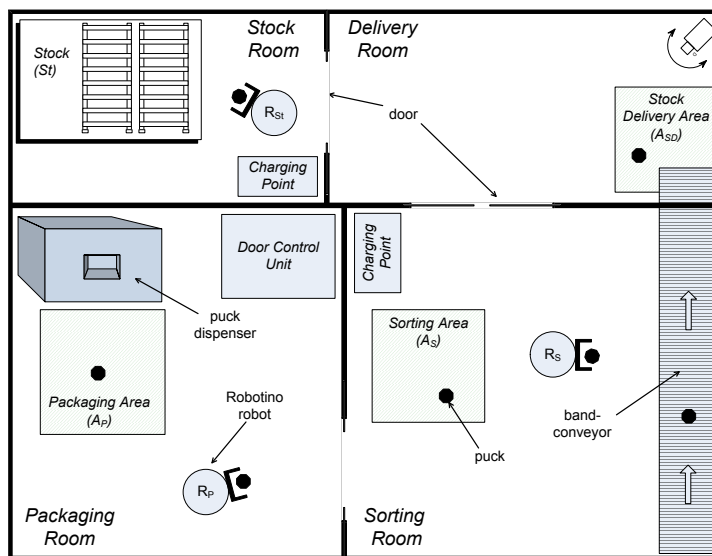


Figure 2: Factory automation robot scenario.

Figure 2 shows a structural sketch of our CPS laboratory simulation environment. Three autonomous robots have to transport pucks between different rooms to fulfill an overall task. Additional constraints (e.g., saving energy, maximize throughput), a changing environment (e.g., doors can be closed or opened, robots may fail, obstacles) and a different set of sensors and actuators on each robot lead to a complex scenario that must be highly dynamic at runtime.

For the sake of simplicity, we describe only a part of one robot's software architecture. Furthermore, we omit collaboration details between the robots in this paper. So in our example, the robot must move around (for puck transportation), load its battery before it becomes exhausted and calculate a path through obstacles and different rooms. Additionally, a single adaptation loop optimizes the throughput of puck transportation as well as the power consumption of the battery.

In the context of CPS, we have to deal with timing constraints in the functional part of the system. Therefore, we logically separate the adaptable software layer into two parts. At first, the core software system, as it is depicted at the bottom in Figure 3, includes all functional parts that fulfill hard real-time constraints. Second, the extension part contains the non and soft real-time behavior of the software system. This separation is a pure logical grouping of the system that is not visible in the metamodel in the next Section 3.2, but potentially has total different characteristics in terms of dynamic and static adaptation capabilities.

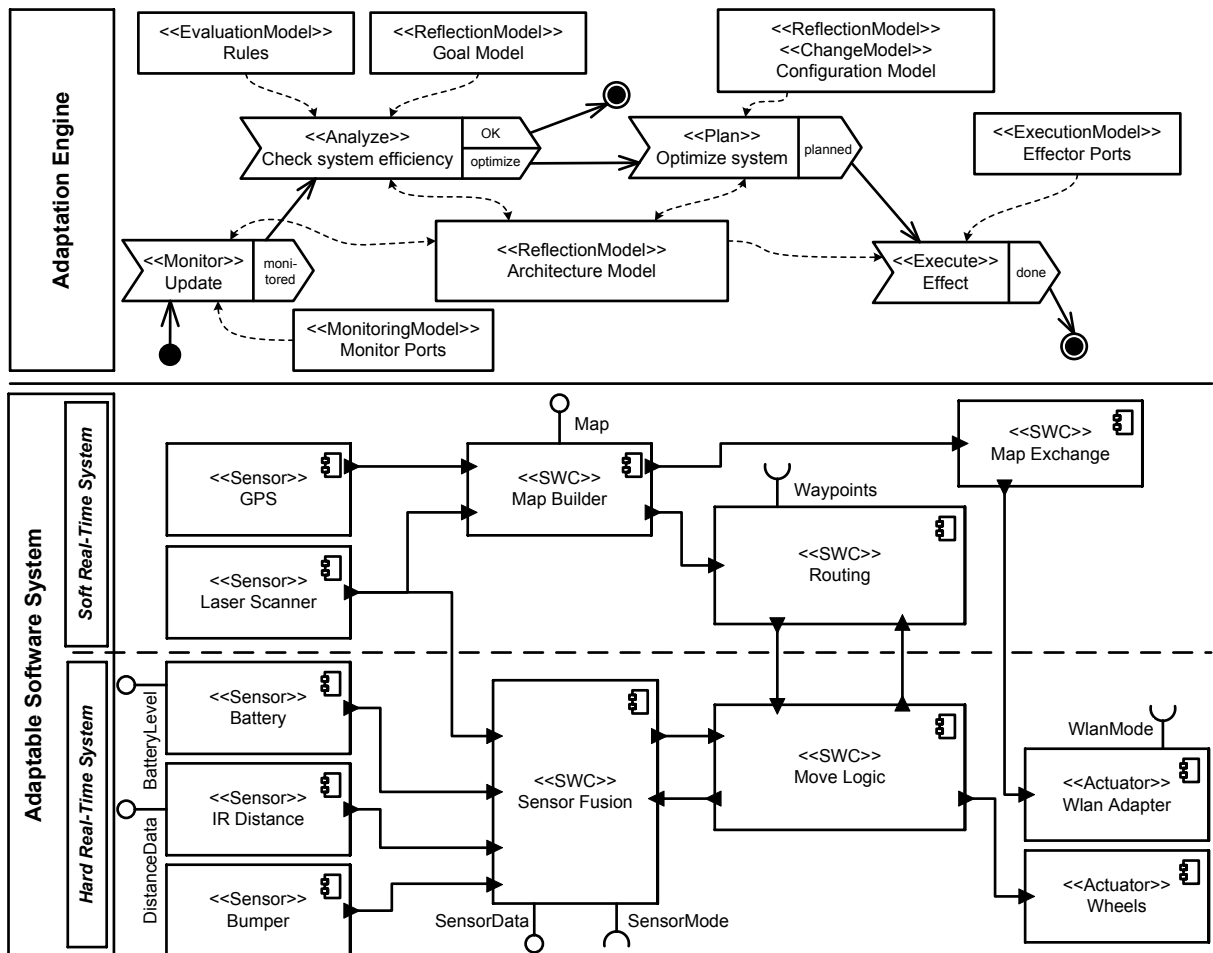


Figure 3: Example: Adaptable software system of one robot with adaptation engine.

At the bottom of the adaptable software layer (see Figure 3), we have three sensor components to detect obstacles at a short distance (one infrared sensor array and a bumper) and to get the current energy level of the battery. The sensor data is analyzed

by a sensor fusion component and used afterwards to calculate moving commands, which are sent to the wheel actuator of the robot. Additionally, the robot has a wireless communication actuator sending messages to other robots or a main station. At the extensional layer, the robot can use advanced sensors for localization or scanning the environment. Additional components can create a map representation of the environment, can calculate optimized routes through the scenario or send the internal map to other laboratory participants.

Each component has an implementation that is split up in so-called functional units. These units are mapped to tasks for execution, which can be handled by a scheduler of an arbitrary real-time operating system. We do not look inside the specific implementation of these units for protecting IP.

On top of the adaptable software, the adaptation engine is realized by one MAPE-K loop that optimizes the robot behavior in terms on throughput (how much puck transportation tasks are processed in a certain time slot) and battery lifetime. At first, the monitor activity of the loop senses the current architectural situation of the robot and updates it in a corresponding runtime model. The activity uses special reflective interaction points of the robot system to retrieve these information. According to the current situation, an analyze activity has an overview about constraints and current goals and checks whether they are fulfilled or not. Additionally, it determines with the help of a rule set if the efficiency of the system can be optimized. If all constraints are fulfilled and no optimization is necessary, the adaptation loop will stop. Otherwise, a planning activity tries with the help of a configuration runtime model to improve the robot behavior. At the end, an execution activity reads the planned changes from the runtime model and forces them to the running software system. For more information concerning syntax and execution semantic of the MAPE activities as well as runtime models see [10].

In summary, we have a self-, context- and requirement-aware adaptive system.

3.2 Model for CCPS

In the following, we describe our metamodel of an adaptive component-based cyber-physical system (CCPS) with real-time constraints. We define only those properties, we will need for component interaction at the adaptable software level and for specifying the monitoring as well as effecting activity by the adaptation engine.

We cover two major parts of a CPS architecture (cf. Figure 4). First, we describe the static structure in form of components and its interconnections. Each component consists of different functional units (FU) that implement the behavior of the system, internal data (e.g., shared variables) and ports to communicate with other components. A component can exchange data via ports if an outgoing port is connected to an *In*-port with the same interface.

With respect to the execution of the functional units within the component structure, we map these units to a set of tasks in the second part of our architecture modeling. The task set is handled by the scheduler of the real-time operating system. We do not require a special scheduling strategy or operating system, because this is very specific for different problem domains. However, the scheduler should support priorities for tasks as well as a periodic and/or triggered execution start mechanism. Each functional

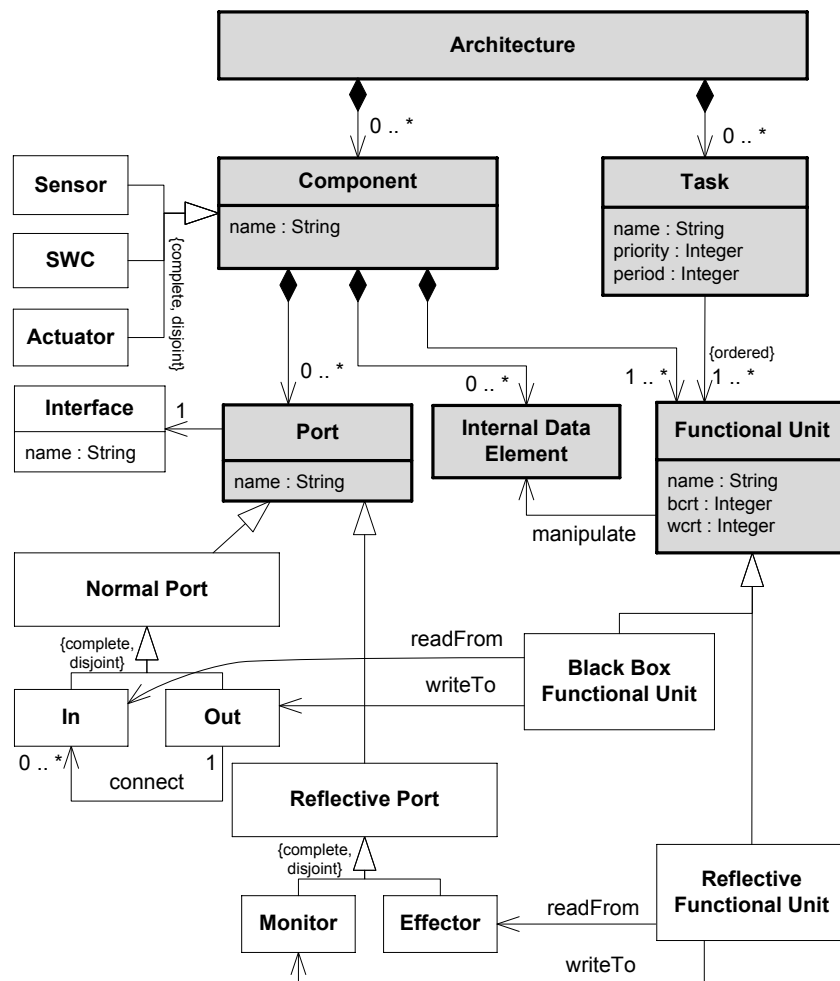


Figure 4: Metamodel for the adaptive component-based cyber-physical system.

unit of one task is executed after the other according to the ordered position in the mapping.

Because FUs encapsulate and hide implementation details for protecting IP, we must know the response time properties for each FU given by the *best-case response time* (*bcr*t) and *worst-case response time* (*wcr*t) attributes for analyzing component interaction inside the adaptable software layer and for the realization of data exchange with the adaptation engine later on. The response time of a FU is the time from enabling it in the task until it finishes its execution. Because we abstract from a specific scheduling algorithm, the response time can be greater than the execution time due to task preemption. In that time span, a FU can read and write data to assigned ports at any point in time. The *bcr*t is always smaller or equal to the *wcr*t. On the one hand, specifying these two timing properties is very minimalistic compared to a full white box view of the system enabling adaptation and data exchange. Furthermore, we can map these information to formal models to simulating or model checking the system as we do it in [6]. On the other hand, it must be done for most embedded, safety-critical system anyway guaranteeing hard and soft timing deadlines. If for one FU the $wcr\ t = \infty$ (unknown), the corresponding *wcr*t of the task is ∞ as well and can only realize soft real-time constraints. If a component contains only FUs with $wcr\ t = \infty$, it can be placed at the logical soft real-time layer in the adaptable software system (cf. Figure 3).

We distinguish three special kinds of components, namely sensor, software component and actuator.

- A sensor is a component that has only outgoing ports. It is the software representation of a hardware sensor in the CPS. It can retrieve, convert, filter and aggregate real (analog) data and provide it to other software components.
- A software component contains the functional behavior of the CPS. It can receive data from sensors. The data is processed internally and can cause control data that is sent to actuator components.
- An actuator is a component that has only incoming ports. It is the software representation of a corresponding hardware actuator part. It derives as well as converts input data into hardware specific control values.

In our component model, we abstract from the real hardware using sensor and actuator components. Furthermore, we make no assumption about the deployment of tasks to more than one execution nodes. If we have multiple or distributed processor nodes, the operating system or the scheduler must fulfill our requirements concerning task execution with different priorities and triggers.

3.3 Interaction Pattern

We distinguish two cases for component interaction. First, components communicate and exchange data via ports inside the adaptable software layer. Second, the underlying adaptable system provides reflective interaction points for monitoring and effecting by the adaptation engine. In this report, we discuss the last case in detail in the following.

The adaptation engine must have monitoring and affecting interaction points with the running software system. For this reason, we introduce reflective ports provided by a component (cf. Figure 4). We distinguish *Monitor* and *Effector* ports. The former provides information for the adaptation engine, the latter receives control values (data) that have a direct impact on the internal behavior of the component. Figure 5 shows an example for each component type. Sensor components normally encapsulate a HW sensor part and therefore, they can have only monitoring ports. In this example, the infrared sensor provides its distance values. The same line of argument holds for the actuator components that manages a corresponding HW part and can only have effector ports. The Wlan actuator can be switched on or off from outside (e.g., for saving energy). A software component may have both or an arbitrary subset of both reflective port types. In our example, the sensor fusion component provides more complex (aggregated) information about obstacles and the current mode merging the sensor data. Additionally, the internal mode can be effected from outside that changes the sensor evaluation algorithm (e.g., robust and safe versus a fast incomplete evaluation of sensor data).

It must be noted that the reflective ports can not only be used by the adaptation engine. Other components may use these ports, too. However, components inside the reflective software layer (cf. Figure 3) should use normal ports for communication and data exchange due minimizing the overhead of reflective port realization.

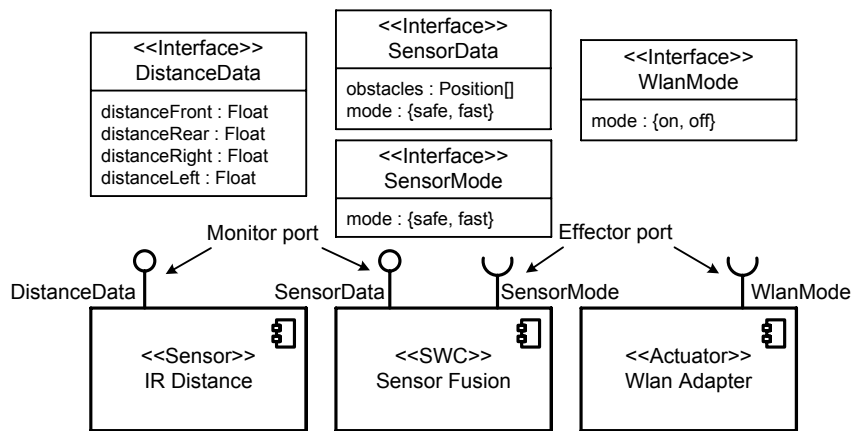


Figure 5: Reflective port example with interface specification for a sensor, software and actuator component.

According to our metamodel, each component must have at least one functional unit that represents its (black box) implementation. If we want to add reflective capabilities for a component, we have to decouple this functionality from the normal execution logic ensuring that we do not violate existing timing constraints of the system. In addition, we execute the reflective part with a lower priority only if we have enough execution time left.

Figure 6 illustrates the basic idea adding a monitor (I) or effecting (II) port to a component. In both cases, we have to add a (reflective) internal data element (*R_Data*), which contains the accessible runtime information and an extra reflective functional unit (*R_FU*), which reads/writes the data from/to the reflective port. In the first case (moni-

tor port), at least one functional unit updates the internal data element. The additional *R_FU* checks during execution whether the *R_Data* element is completely updated. It provides the data to the reflective port, where it can be read from outside or triggers the listener directly. Reading and writing the internal *R_Data* can be seen as atomic operation. Our framework ensures that multiple functional units can update the data as well as the reflective functional unit can check if it is complete and a new value.

In the second case (effector port), the reflective part updates an extra internal data element with incoming control values. The normal functional unit independently processes the data (if available) in its next execution cycle.

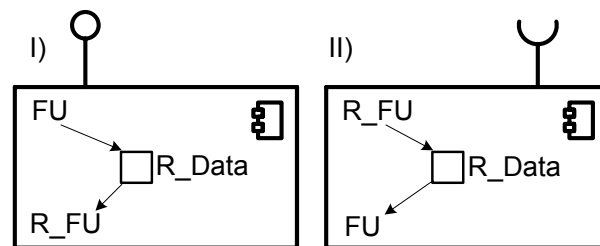


Figure 6: Implementation of the reflective port concept. I) Monitor port II) Effector port.

In our design of adding reflective capabilities, we strictly separate the normal behavior from the reflective part for the following reasons. First, we minimal disturb the normal execution branches of the component. We reduce the overhead of adding reflective properties to an additional write on an internal data element (a simple variable in most cases) by the functional units in the I) monitor case, which can be add to the end of the execution of these units or an additional read (e.g., at the beginning of normal execution) in the II) effector case. Second, we can map the normal and reflective functionality to different tasks for system execution. If the reflective task runs with a lower priority, it can be simply interrupted by the important (real-time) behavior of the system. As a result, it can take several cycles until the values are propagated or effect the system. Our assumption is that there are enough unused time slots executing the lower prioritized reflective task. Of course, you can map the normal and reflective functional units on the same task or choose the same priority for different tasks, but this may lead to unexpected (timing) behavior of the system execution. Third, we are able to simply add or remove the reflective part at runtime during system execution in the most cases and therefore, we can react to new situations and changing system requirements. Fourth, we can easily combine the monitor and effector port within on software component.

4 Conclusion and Future Work

In this report, we have shown a component model for enabling reflective properties for the adaptable software layer by explicitly modeling monitor and effector ports. Moreover, we discuss a concrete realization of this port concept. As future work, we will generate from the component model different code fragments that realize all necessary interactions. This generated code must be integrated in our existing toolchain

(cf. [12]). Furthermore, we want to investigate communication dependencies between components that have different timing constraints.

References

- [1] Acatech. Cyber-Physical Systems: Driving force for innovation in mobility, health, energy and production. Technical Report POSITION PAPER, December 2011.
- [2] Basil Becker, Stefan Neumann, Martin Schenk, Arian Treffer, and Holger Giese. Model-Based Extension of AUTOSAR for Architectural Online Reconfiguration. In Sudipto Ghosh, editor, *Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*, volume 6002 of *Lecture Notes in Computer Science (LNCS)*, pages 83–97. Springer-Verlag, 2010.
- [3] Holger Giese, Bernhard Rumpe, Bernhard Schätz, and Janos Sztipanovits. Science and Engineering of Cyber-Physical Systems (Dagstuhl Seminar 11441). *Dagstuhl Reports*, 1(11):1–22, 2012.
- [4] Jeffrey O. Kephart and David Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.
- [5] Edward A. Lee. Cyber Physical Systems: Design Challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, January 2008.
- [6] Stefan Neumann, Norman Kluge, and Sebastian Wätzoldt. Automatic transformation of abstract autosar architectures to timed automata. In *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems, ACES-MB '12*, pages 55–60, New York, NY, USA, 2012. ACM.
- [7] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based runtime software evolution. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society.
- [8] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.
- [9] Mark-Oliver Stehr, Carolyn Talcott, John Rushby, Patrick Lincoln, Minyoung Kim, Steven Cheung, and Andy Poggio. Fractionated Software for Networked Cyber-Physical Systems: Research Directions and Long-Term Vision. In Gul Agha, Olivier Danvy, and José Meseguer, editors, *Formal Modeling: Actors, Open Systems, Biological Systems*, volume 7000 of *Lecture Notes in Computer Science*, pages 110–143. Springer Berlin / Heidelberg, 2011.

- [10] Thomas Vogel and Holger Giese. Model-Driven Engineering of Adaptation Engines for Self-Adaptive Software: Executable Runtime Megamodels. Technical Report 66, Hasso Plattner Institute at the University of Potsdam, Germany, April 2013.
- [11] Thomas Vogel, Andreas Seibel, and Holger Giese. The Role of Models and Megamodels at Runtime. In Juergen Dingel and Arnor Solberg, editors, *Models in Software Engineering, Workshops and Symposia at MODELS 2010, Oslo, Norway, October 3-8, 2010, Reports and Revised Selected Papers*, volume 6627 of *Lecture Notes in Computer Science (LNCS)*, pages 224–238. Springer-Verlag, May 2011.
- [12] Sebastian Wätzoldt, Stefan Neumann, Falk Benke, and Holger Giese. Integrated software development for embedded robotic systems. In Itsuki Noda, Noriaki Ando, Davide Brugali, and James Kuffner, editors, *Proceedings of the 3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN)*, volume 7628 of *Lecture Notes in Computer Science*, pages 335–348. Springer Berlin Heidelberg, 10 2012.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
82	978-3-86956-266-7	Extending a Java Virtual Machine to Dynamic Object-oriented Languages	Tobias Pape, Arian Treffer, Robert Hirschfeld
81	978-3-86956-265-0	Babelsberg: Specifying and Solving Constraints on Object Behavior	Tim Felgentreff, Alan Borning, Robert Hirschfeld
80	978-3-86956-264-3	openHPI: The MOOC Offer at Hasso Plattner Institute	Christoph Meinel, Christian Willems
79	978-3-86956-259-9	openHPI: Das MOOC-Angebot des Hasso-Plattner-Instituts	Christoph Meinel, Christian Willems
78	978-3-86956-258-2	Repairing Event Logs Using Stochastic Process Models	Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, Mathias Weske
77	978-3-86956-257-5	Business Process Architectures with Multiplicities: Transformation and Correctness	Rami-Habib Eid-Sabbagh, Marcin Hewelt, Mathias Weske
76	978-3-86956-256-8	Proceedings of the 6th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering	Hrsg. von den Professoren des HPI
75	978-3-86956-246-9	Modeling and Verifying Dynamic Evolving Service-Oriented Architectures	Holger Giese, Basil Becker
74	978-3-86956-245-2	Modeling and Enacting Complex Data Dependencies in Business Processes	Andreas Meyer, Luise Pufahl, Dirk Fahland, Mathias Weske
73	978-3-86956-241-4	Enriching Raw Events to Enable Process Intelligence	Nico Herzberg, Mathias Weske
72	978-3-86956-232-2	Explorative Authoring of ActiveWeb Content in a Mobile Environment	Conrad Calmez, Hubert Hesse, Benjamin Siegmund, Sebastian Stamm, Astrid Thomschke, Robert Hirschfeld, Dan Ingalls, Jens Lincke
71	978-3-86956-231-5	Vereinfachung der Entwicklung von Geschäftsanwendungen durch Konsolidierung von Programmierkonzepten und -technologien	Lenoi Berov, Johannes Henning, Toni Mattis, Patrick Rein, Robin Schreiber, Eric Seckler, Bastian Steinert, Robert Hirschfeld
70	978-3-86956-230-8	HPI Future SOC Lab - Proceedings 2011	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Doc D'Errico
69	978-3-86956-229-2	Akzeptanz und Nutzerfreundlichkeit der AusweisApp: Eine qualitative Untersuchung	Susanne Asheuer, Joy Belgassem, Wiete Eichorn, Rio Leipold, Lucas Licht, Christoph Meinel, Anne Schanz, Maxim Schnjakin
68	978-3-86956-225-4	Fünfter Deutscher IPv6 Gipfel 2012	Christoph Meinel, Harald Sack (Hrsg.)
67	978-3-86956-228-5	Cache Conscious Column Organization in In-Memory Column Stores	David Schalb, Jens Krüger, Hasso Plattner

ISBN 978-3-86956-273-5
ISSN 1613-5652