Proceedings of the 5th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering

hrsg. von
Christoph Meinel, Hasso Plattner, Jürgen Döllner,
Mathias Weske, Andreas Polze, Robert Hirschfeld,
Felix Naumann, Holger Giese

Technische Berichte Nr. 46

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

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

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 46

Proceedings of the 5th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering

herausgegeben von

Christoph Meinel
Hasso Plattner
Jürgen Döllner
Mathias Weske
Andreas Polze
Robert Hirschfeld
Felix Naumann
Holger Giese

Universitätsverlag Potsdam

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

Universitätsverlag Potsdam 2011
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 3474
E-Mail: verlag@uni-potsdam.de

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

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

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2011/5147/
URN urn:nbn:de:kobv:517-opus-51472
http://nbn-resolving.org/urn:nbn:de:kobv:517-opus-51472

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-129-5

Contents

Towards a Truly Retargetable Decompiler **1**
Jan-Arne Sobania

Unifying the definition of megamodels:
Toward describing service-oriented system's development **15**
Regina Hebig

A Granular Approach for Information Lifecycle Management in the Cloud **25**
Johannes Lorey

Data in Business Process Modeling **35**
Andreas Meyer

Semantics Detection for Data Quality Web Services **47**
Tobias Vogel

A Shared Platform for the Analysis of Virtual Team Collaboration **59**
Thomas Kowark

Programming Models for Parallel Heterogeneous Computing **67**
Frank Feinbube

A Study on Mobile Real-Time Middleware **77**
Uwe Hentschel

Understanding Service Implementations Through Behavioral Examples **89**
Michael Perscheid

Modeling Browser-based Mashups by Means of Meaningful Choreographies **101**
Emilian Pascalau

Multiple Runtime Models and their Relations for Self-Management **111**
Thomas Vogel

Recent Developments in JCop –
Context-oriented Concurrency Control and Compiler Optimization **123**
Malte Appeltauer

Towards Service-Oriented,
Standards- and Image-Based Styling of 3D Geovirtual Environments **133**
Dieter Hildebrandt

Modeling and Verification of Self-Adaptive Service-Oriented Systems **149**
Basil Becker

Parsing Behavior: The Hierarchical Nature of Concurrent Systems **159**
Artem Polyvyanyy

Categorization and Use of Identity Trust **169**
Ivonne Thomas

Enabling Reputation Interoperability through Semantic Technologies **179**
Rehab Alnemr

A Proactive Service Registry With Enriched Service Descriptions **191**
Mohammed AbuJarour

Towards Automated Analysis and Visualization of Distributed and
Service-based Software Systems **201**
Martin Beck

Towards Efficient Camera Interaction in Service-based 3D Geovirtual
Environments **211**
Jan Klimke

Towards Synchronization of Partitioned Applications **221**
Felix Geller

Towards a Truly Retargetable Decompiler

Jan-Arne Sobania
jan-arne.sobania@hpi.uni-potsdam.de

Moore's Law has been the source of growth of computing performance for decades, as the exact same binary programs would get speedups simply by moving them to newer hardware. However, this has changed in the last years, as further miniaturization is no longer utilized to accelerate single-threaded code, but to provide parallelism. To fully leverage this potential, applications now need to be written with parallelization in mind. This presents a challenge for specialized custom applications. These can generally not be replaced by commercial-of-the-shelf software, but due to cost and time constraints, traditional porting or re-development for new architectures is also not feasible. In addition, in-house applications may use old or obsolete development environments which may either no longer be available, not supported on new hardware, or even the source code to the original application might been lost.

This report proposes decompilation as a solution. Decompilers have traditionally been written to support only one particular processor type, although remarkable parts of their core analysis algorithm are independent of the processor type or even the processor family. Few researchers have worked on retargetable decompilers in the past; i.e., decompilers supporting processors from different families with non-identical feature sets. However, these approaches generally failed to deliver a single core analysis that worked as well as a processor-specific one while still supporting changing the target architecture.

We discuss the current status of the research and propose an architecture for a general, retargetable decompiler for current processor families. An example of an experimental decompiler is shown that translates Microsoft .NET CLI code into OpenCL for execution on appropriate accelerators. Finally, we discuss how the presented techniques are applicable to more general processor architectures like the Intel Single-Chip Cloud Computer (SCC).

1 Introduction

The first decompilers were developed during the 1960s. In contrast to today's machines, most computers were not backwards compatible back then, so decompilers were seen as a convenient means to help in porting legacy code. At later times, decompilers have been used for a variety of other purposes, including documenting, debugging and alterating programs [5], or malware/security research, besides the obvious use case of reconstruction of programs whose original source code has been lost [15]. Furthermore, analysis techniques typically used by decompilers are also present in modern runtime environments like the Java Virtual Machine [13] or the .NET CLI [9] for verifying code prior to execution.

Despite the broad field of use, all these approaches have something in common: they are limited to a particular, definite task for a single target processor. However, their core analysis algorithm is largely independent of the processor architecture, as it depends only on the set of features offered by the processor, not the respective representation of those features in the instruction set.

Independent of the specific processor, machine code generated from a high-level language typically preserves aspects of the structure of the original code. For example, instructions are typically attached to a single function, and control flow statements are represented by branch instructions. Therefore, in order to reconstruct the control flow graph (and then the control flow statements), it is sufficient to only consider branch instructions. Furthermore, the specific encoding of the instruction on the machine is irrelevant; what matters is what parameters it uses and what the target operation is.

This observation of the core analysis mainly being independent of the target processor has led to research into *retargetable decompilers*, examples of which include DCC [5,6] and Boomerang [1]. However, these approaches are limited in several ways, as discussed below.

The main contribution of this report is a refined architecture of a truly retargetable decompiler. We discuss challenges that modern processors pose for decompilation which are not sufficiently dealt with in other decompilation approaches, and we highlight on how corresponding analysis passes can be integrated into our architecture. Based on these preliminaries, we present an experimental decompiler for translating .NET CLI code to OpenCL, which can be used to speed up computations if a suitable accelerator is installed in the system. Finally, we will discuss how these principles could be applied in a more general case, using the example of the Intel Single-Chip Cloud Computer (SCC).

This report is organized as follows: section2 summarizes previous efforts to create retargetable decompilers. Section 3 reviews common processor features available to assembler programs and relates them to the operation of a decompiler. Section 4 then outlines the decompilation process and discusses the operations performed by a compiler that need to be reversed in order to reconstruct source code. Section 5 shows an example of how these principles can be applied to convert code generated by a .NET compiler for execution on OpenCL-compatible devices. A discussion of a broader use case follows in section 6, which outlines on how decompilers can help in porting code to modern many-core architectures like the Intel SCC. Finally, section 7 concludes this report.

2 Related Work

According to Cristina Cifuentes [5], the first decompilers date back to the 1960s and where used to aid in porting programs from older machines to new architectures; as an example, the decompiler D-Neliac is mentioned which decompiles Univac M-460 machine code to Neliac, a variant of Algol 58 (see [11]).

Many decompilers have been developed since, dealing with various problems in both machine languages and high-level languages. In our discussion below, we con-

centrate on efforts towards the creation of retargetable decompilers:

- *DCC* [4], although meant as a general decompiler, does only support 16-bit x86 code as input, and its intermediate representation is crafted to support commonly-used x86 features. However, it lacks support for non-x86 processor features like branch delay slots, register renaming, predicated instructions and so on.

  Furthermore, the handling of pointers in the program is severely bugged, which basically allows it to be used only for programs consisting of only a few kilobytes of binary code.

- *Boomerang* [1] was started as an open-source project to create a general retargetable decompiler, but development has stopped for some years now. It supports x86, SPARC and PowerPC code as input, but has several shortcomings listed on the project page [2]. For example, several analysis only work for certain targets and are not handled in a general way, like partial register accesses (e.g., on x86, `AH` is an alias for bits 8 to 15 of `AX`). Currently, these are only for the x86 family, with the underlying algorithm being unsuitable for other architectures like SPARC.

- *UQBT* [7], the University of Queensland Binary Translator, as its name implies, is a binary translator which also contains features of a decompiler. Retargeting is implemented via different front ends for different processor architectures, which also perform processor- and operating-system-specific pre-processing.

  UQBT relies on emulating the source processor's instruction stream if decompilation of a method is deemed impossible (e.g., if instructions are encountered for which no appropriate representation in a high-level language exists). Indirect jumps or calls via function pointers rely on tables that map the code address from the source architecture to its counterpart on the target. A program's data section is copied verbatim, so there is no need to reconstruct any data types beyond the basic arithmetic ones. In addition, being a binary translator, no importance is attached to generating human-readable code.

3 Processor Features

This section reviews features found in current processor architectures and how they relate to decompilation.

3.1 Stack vs. Register Machines

Probably the most basic distinction of modern processors is whether they represent *stack* or *register* machines. Virtual environments like Java [13] or .NET [9] typically use a stack only, whereas all "'normal"' processors (x86, IA-64, SPARC, ARM etc.) rely on a combination of registers and a stack, which might be just a software convention; e.g., a memory region and general-purpose register reserved for the stack pointer.

Stack machine code can be ported in a straight-forward way to a register machine, provided that certain constraints are met. Especially, this task is trivial if the size of the allocated stack frame is known at each program point – as is always the case with valid Java and .NET programs, because their respective byte code verifier would prevent other binaries from running. Therefore, for decompilation, we can assume the input program to be written for a register machine; the number of registers may not be constant for each program (e.g., some Java programs might use more stack locations simultaneously than others), but their count is still finite and statically known.

3.2 Basic Instructions

Standard operations on register machines include the following:

- Arithmetic instructions, which are usually separated into integer and floating-point instructions (on machines having native support for the latter).
- Memory accesses. RISC machines typically have separate instructions to transfer data between registers and memory, whereas CISC machines might also combine memory accesses and other operations (like arithmetic ones) in a single instruction.
- Control flow instructions, like branches or calls to subroutines.
- Processor management instructions. Typically, these are used by operating systems to manage the execution environment for single processes (like instructions to configure the protection level or page tables).

Processor management instructions present a challenge for a decompiler, as their effects can usually not be represented in a machine-independent manner, like the effects of instructions from the other categories. However, this seems not to present a problem, as compilers usually do not emit them in the first place.

3.3 Register Operands

Another challenge is the addressing of operands of an instruction. We can always assume these to be registers (on CISC machines, memory operands can always be loaded into a temporary pseudo register), but we cannot assume a single named register to contain only one program-level variable at each point in time. For example, on x86, the register operands *AL* and *AH* specify the lower and upper byte, respectively, of the register AX; in 32-bit mode, AX itself specifies the lower-order 16 bits of the 32-bit register EAX.

Other processors similarly store multiple user-visible values in a single register; for example, the SSE instructions of x86, as well as the parallel- arithmetic instructions of IA-64, might separate single 128-bit registers into 4 32-bit values, interpret each of them as a single-precision floating point value, then perform 4 additions in parallel. On vector processors, instructions typically contain bit masks denoting which parts

of these sub- divided registers are being used in an operation; unused contents are ignored and keep their contents, even if the register is a destination operand.

Therefore, a general decompiler should also support accesses to parts of a register *natively*; i.e., it should be capable of representing such parts without resorting to higher-level constructs, like the bit operations a programmer would use in the C language to represent such accesses on long integer variables.

3.4 Control Flow Instructions Revisited

The above section on basic instruction types already mentioned control flow instructions like branches or calls, stating that their effects can usually be represented in a machine-independent manner. There are, however, certain special cases of these instructions that are in fact machine-*dependent*, as we will discuss now.

Architectures like SPARC or MIPS use so-called *delay slots* after branches, calls and other similar instructions that change the address of the next instruction explicitly. The delay slots immediately follow their corresponding branch instructions in memory, and contain regular instructions themselves. During execution, the new program counter is loaded when the branch is being executed, but the branch takes effect only *after* the instruction in the delay slot has been executed.

Special care must be taken when reconstruction a control flow graph of a program on such an architecture, as delay slot instructions can occur in two places with completely different semantics:

- In the delay slot of a branch. In this case, the program behaves as described above.
- As the *target* of a branch. In this case, the instruction executes as usual, and serves as the first instruction of a new basic block.

3.5 Register renaming

Certain architectures like ARM, SPARC or IA-64 support implicit register renaming; i.e., a single register operand like *r21* might refer to different physical registers, depending on which state the processor is in. Regular analysis algorithms in decompilers like the ones introduced in [5] expect register names to be unique, so they cannot be used as-is on such architectures without disambiguating registers first.

On ARM, the situation is simple, as registers are really just *banked* instead of *renamed*. That is, register mapping depends only on the processor mode (like user, supervisor or interrupt), not on other implicit status. Compilers are not expected to change the processor mode in order to get access to the banked registers; therefore, a decompiler can safely assume that register names are unique, as long as no mode switching occurs. SPARC contains limited support for register renaming on subroutine calls: of its 32 general-purpose registers, only 8 are global (i.e., never renamed), whereas the other ones are categorized into three groups consisting of 8 registers each:

- Local registers are allocated per function and cannot be read or written to by subroutines.
- Output registers behave as local registers, but supply arguments to subroutines, as well as results to the caller.
- Input registers contain parameters passed to the current routine by its caller, and can be used for return values as well.

On a subroutine call, the output registers of the current routine are renamed such that they become input registers of the subroutine; in addition, the subroutine gets new local and output registers its caller does not have access to. On return, the local and output registers are deallocated, and the input registers can now be accessed using their old name again, so the caller can read the routine's return value. The IA-64 architecture uses an extended register renaming scheme. It specifies 128 general-purpose registers, of which 96 are renamed; in comparison to SPARC, it uses the same notion of local, input and output registers, but the size of the corresponding registers windows is not fixed. Instead, a function uses the *ALLOC* instruction to specify their number. The register windows are not limited in size, the only constraint is that at each time, the total number of allocated registers must not be greater than 128. In addition to these variable register windows, IA-64 also supports another renaming scheme to implement *modulo-schedules loops*. When executing the *ALLOC* instruction, in addition to the size of the local and output register window – input registers are treated like local ones from the point of view of the function – there is also a window of *rotating registers*. As the name suggests, these registers are renamed in a rotating fashion, with the next rotation step being initiated by special branch instructions. This allows constructing software-pipelined loops in which each iteration processes values in its own local register (prepared by the previous iteration), thus reducing the number of data dependencies and increasing parallelism. Control flow differs quite a bit for regular and modulo-scheduled loops, and reconstruction requires static knowledge of the set of rotating registers [16]. However, if this information is available, reconstruction is straight-forward.

3.6 Speculative Execution

IA-64 provides special instructions for performing speculative memory accesses. For example, a program can request to load a value into a register a long time before it is required; if execution finally reaches the instruction in need of the value, it is already present in the register, so there is no additional overhead.

There are two types of speculation available on IA-64: control and data. A control speculation works just like an ordinary memory load operation, except that it places a special value called *NaT* (*not-a-thing*) in an invisible part of the target register if the operation fails (e.g., because the address is invalid). On the other hand, a data speculation records the address being accessed in the so-called *advanced load address table* (ALAT); if the address is later being written to, the processor removes the corresponding ALAT entry. Therefore, the program can check whether the target value

might have been changed before using the register in other operations and retry the operation if necessary. Control and data speculation can also be used at the same time. For a decompiler, speculative execution poses quite a challenge, as accurately reconstructing success and failure of speculative accesses requires analysing all other parallel activity on the system in question. However, we assume this not to be needed for most programs: as speculation might fail because of intervening accesses by other programs (e.g., an ALAT overflow), we think that speculative accesses can be represented just like normal memory accesses, and all speculations in the program can be assumed to have succeeded.

3.7 Invoking Code on Co-Processors

Some processors support direct invokation of code written for other processors or processing modes. For example, IA-64 provides a special jump instruction to execute x86 code, and ARM provides a similar instruction to switch between routines running in normal and thumb mode (which has a different instruction set). Some custom processors might also contain coprocessors capable of executing own programs (within a completely different memory space and register set), and appropriate call instructions might resemble those for normal subroutines.

There are two basic ideas how decompiling such cases can be handled:

- The coprocessor call is handled like a library function for which no other information is available. If parameters are being passed – for example, by directly reading from or writing to the coprocessor's registers – these accesses must then be represented by library calls as well.
- On the other hand, if the routine running on the coprocessor is known and the decompiler supports the other instruction set, the call can be handled just like a normal subroutine call. In this case, accessing the coprocessor's register does not differ from accessing the normal processor's registers.

4 Decompiler Architecture

4.1 The Compiler Pipeline

As a general decompiler is expected to reverse the process of compilation for a wide range of target architectures, we have to provide a general model for this process first. This general model, which we call *the compiler pipeline*, is displayed in figure 1; it is similar to the model of a "'language processing system"' shown in the "'dragon book"' [3, p. 4]. In comparison, the linker and loader have been separated into own stages, in order to reflect that these processes are quite distinct.

In this model, the program is transformed step by step by various subprocesses. First of all, a compiler reads the input code and emits assembler instructions suitable for the target processor. This assembler code is then converted into object files, which are combined with additional (static) libraries and *linked* to form the program's binary

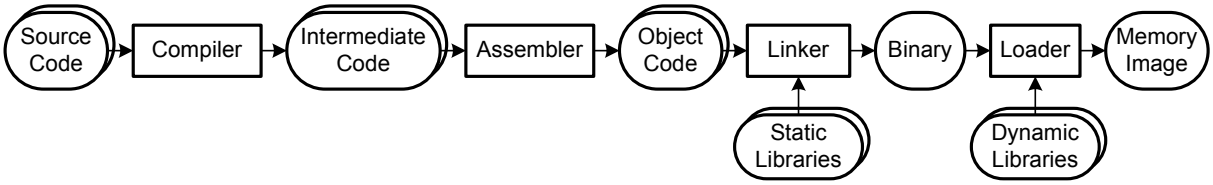


Figure 1: The Compiler Pipeline

file (e.g., an *elf* file for UNIX or an *exe* file for Windows). If this binary file and required dynamic libraries are available, the operating system's loader can then create a runnable image of the program in memory.

The model applies to machine code programs (e.g., C programs compiled by gcc) as well as high-level programs written in languages like Java or .NET, although the latter use slightly different terminology. For example, the Java Compiler already contains the corresponding assembler, and its result is called a class file instead of an object file. The linker step does not exist at all, although packaging of class files into a JAR archive could be seen as a form of linking.

As can be seen, the final executing program does not solely contain code written by the user, but also consists of static and dynamic libraries, as well as possible other code generated during any of the phases in the model. Therefore, it is not sufficient for a general decompiler to reverse the process of a compiler *alone*; instead, to reconstruct code that is as close as possible to the original source, it should reverse the entire compiler pipeline, making use of any additional information gathered during the phases, if available.

4.2 Decompilation Overview

Similar to the compiler pipeline shown in figure 1, we propose the model shown in figure 2 for a general decompiler. This is an extension of the model proposed for *DCC* in [5].

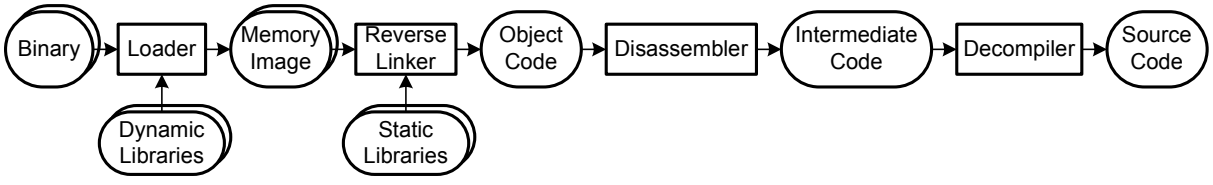


Figure 2: Proposed Decompiler Pipeline

Decompilation starts with binary files of the program, which are used to build *virtual memory images* of the target program via a special loader. If no binary file is available, the decompiler might also start directly from a memory image of the real system under investigation.

The reason we specify multiple images is that programs may load different modules at the same address ranges over time, whereas the decompiler might need to analyze the whole program at once. For example, overlays can be represented by multiple

memory images, as can self-unpacking code or code running on co-processors as discussed in section 3.7.

Once memory images are known, code from each image is checked for static libraries by means of a *reverse*-linker; this step is similar to the recognition of library signatures as described in [5]. As an extension, we propose to also process symbol information from corresponding static libraries at this point to reconstruct parts of the program's symbol table. That is, if a section of a known object or library file is found in the memory image, our reverse-linker can resolve relocation records from the object file to get symbol names and addresses in the memory image. Depending on which information is contained in the object files, this might also return function signatures or type information for later parts of decompilation.

After reverse-linking, it is known which parts of the binary represent libraries and which represent user code; therefore, the decompiler can focus on the latter ones, as they are what compiling the original source code has produced. For decompilation, instructions are now disassembled and converted into an intermediate code, which will be refined during the subsequent process until it resembles the original source code as close as possible.

4.3 The Decompiler Core

Beginning at the memory images with static libraries identified, decompilation can proceed as follows:

The disassembler produces decoded instructions, which are then converted into C-like intermediate code. For example, if an instruction like `add eax, edx` is encountered, it can be converted into a C-type statement like `eax = eax + edx`. Special instructions like those subject to register renaming are marked and resolved later, as the corresponding C expression does also depend on other state not known yet.

Based on these instructions, boundaries of subroutines are reconstructed. As a simple heuristic, the target of a call instruction is assumed to be the beginning of a new function; the list of subroutines may be extended during later phases. Processor- or operating-system-specific heuristics can also be used to get a list of entry points. For example, when decompiling a DLL under Windows, its export table can be used to get base addresses of publicly-callable functions.

The instruction lists are used to reconstruct a control flow graph for each function. This phase might also include processor-specific activity; for example, on SPARC on IA-64, register renaming can be detected and converted into appropriate C constructs, as can modulo-scheduled loops [16] or predicated instructions. Once this phase is complete, all processor-specific features have been resolved into expressions in the intermediate language.

Global data-flow analysis [5] is used to track passing of parameters and return values in registers, and to detect operations on common runtime structures like the stack. This allows to give an initial estimate of procedure signatures.

Up to this point, almost no information on the data types used by the program are known (besides width of registers and memory accesses, if available on the target processor). There has been work on how to reconstruct types in the past [8, 14], but

some basic problems of this step from a decompiler's point of view have still remain. For example: how to accurately represent values and types split over multiple memory locations (as is the case for 64-bit arithmetic on 32-bit x86), and how to find a representation of the decompiled code (that adheres to a reconstructed type system) while at the same time resembling code that a human programmer might have written.

With type information, expressions can be re-written to resemble C-style accesses more closely. For example, a memory access via a pointer and scaled operand can be converted into an array access. Once expressions have been rewritten in this regard, the decompiler can invoke its final stage to produce readable source code. As discussed in [6], another post-processing step may be invoked to convert this raw C into higher language constructs; for example, to detect virtual method calls on objects and represent them accordingly.

5 Decompilation Example: .NET CIL to OpenCL

Based on the principles discussed above, we have built an experimental decompiler for translating .NET CIL code into OpenCL. It reads the native CIL stream generated by a source language compiler and outputs OpenCL-compatible C code; this code is then sent to the vendor-provided OpenCL compiler to generate the binary for execution on the OpenCL compute device.

Choosing CIL as input (instead of, for example, source code in a high-level language targeting the .NET runtime) allows invoking the decompiler from within the same .NET assembly that already contains the user code. This way, we could selectively offload certain compute-intensive tasks to accelerators if available, but still rely on the CPU for other systems without appropriate hardware. For the prototype, it is specified explicitly which code shall be executed on the GPU, by using a syntax similar to the Task Parallel Library [12].

CIL uses the abstraction of a stack machine, but individual instructions still closely resemble semantics of operations in a high-level language. Therefore, in comparison to machine code for "'normal"' processors, CIL is particularly easy to decompile.

As OpenCL-compatible devices provide a rather constrained environment for code execution in comparison to CPUs, we have not implemented every aspect of CIL in our decompiler and C code generator. Currently, we support the following:

- Control flow statements (like *if*, *switch*, *for*)
- Intrinsic data types (8 to 64-bit integers as well as single and double-precision floating-point values) and associated basic arithmetic operations
- Accesses to one- and multi-dimensional arrays, as long as the target array is statically known.

Specifically, memory allocations, exceptions and pointers to arbitrary memory locations are not supported, and code containing these features is rejected by the instruction decoder. Exceptions are a special case, though, as they may be thrown as a

side-effect of an instruction, rather than being used explicitly. In the latter case, the CIL code is just rejected and never compiled to OpenCL. On the other hand, instruction side effects that could result in an exception being thrown (e.g., if an array index is out of range) are silently discarded.

6 Intel SCC: Many-Core and Beyond

The Intel SCC is a research microprocessor developed at Intel Labs Braunschweig, Germany, and other locations around the globe. It combines 48 P54C cores (software-compatible to a Pentium 90) on a single die – 6 times as many cores as on Intel's current high-end CPU Nehalem EX – together with 4 memory controllers, an on-chip network, power management and hardware support for message passing [10].

Cores are organized in 24 so-called *tiles*, which are arranged in a 6x4 mesh. Power management is software-controlled, and both the operating frequency and voltage can be adjusted freely to increase energy efficiency. Frequency is set per tile, whereas voltage is controlled on a 2-by-2 tile basis (*voltage island*). The SCC consumes 50W to 125W, depending on the voltage required for a specific frequency; further savings are possible by disabling cores or even by powering down entire voltage islands.

All communication between cores, memory controllers and the I/O subsystem is accomplished via high-speed message passing – basically, the cores send messages directly into each other's level-1 cache. In comparison to other modern CPUs, there is absolutely no hardware cache coherency support; if coherency is required, it needs to be implemented in software.

The lack of hardware cache coherency eliminates snoop traffic – which can already limit scaling in current multi-core systems – and is believed to be a key in allowing to scale processors to 100 cores and beyond. However, it provides quite a challenge in actually *programming* the machine, as most mainstream programming models rely on multiple threads having the same (cache-coherent) view of the main memory.

For example, when using traditional multi-threading under Windows or Unix, all threads use the same memory space; i.e., a write performed by one thread can be seen immediately by all others, and pointers to memory buffers can be passed unchanged to any other thread. On some system, actual bandwidth and latency of memory accesses may depend on a particular physical core or socket (*ccNUMA*), but memory is still shared from the viewpoint of the application.

The traditional means for applications to use more than one address space is to spawn multiple processes. However, when these processes are running on the same node, operating systems still allow using shared memory in this case (e.g. via *MapViewOfFile* under Windows or *mmap* under Unix), which again is assumed to be cache-coherent. If an application shall be able to execute on different nodes, it needs to use communication explicitly; for example, via sockets or MPI on cluster systems.

When trying to port legacy applications into such an environment, for example with the aid of a decompiler, there are at least three types of challenges:

- *Identifying* potential parallelism in what might be a sequential algorithm

- *Mapping* parallel constructs to a non-cache-coherent architecture
- *Assessing* whether parallel execution is beneficial for execution

A decompiler can produce abstract syntax trees for source machine code, but obviously that is not sufficient for identifying parallelizable constructs. However, it could be extended by integrating techniques originally developed for optimizing compilers for cache-coherent multi-core systems; for example, methods to detect memory dependencies in sequential code. A similar analysis could be employed by a decompiler to reconstruct fields of data structures being accessed, so part of the information required for identifying parallelism would already be available.

The same applies to mapping abstract syntax to a non-cache-coherent architecture. If it can statically be deduced which parts of buffers are accessed, data structures can also be rearranged for efficient execution in distinct address spaces. However, this could result in quite some overhead if done at runtime (or even depending on which processing cores are active, their respective memory latency, and so on); therefore, it needs to be assessed upfront whether parallelization should be performed at all, or whether it is more beneficial to the optimization goal (which may not be just performance, but also power consumption, system efficiency etc.) if the computation is just executed sequentially.

7 Summary and Conclusions

We have presented a number of challenges when developing a retargetable decompiler, and strategies on how to deal with them. In general, when designing a single representation for multiple input languages, a choice must be made as to which features to integrate natively, which to re-build via more basic constructs, and which not to include at all. For our decompiler, we propose a tradeoff that is *pragmatic*: if a compiler is able to generate certain code for a certain target, the decompiler should support it. However, we feel no need to support every aspect of a particular instruction set that compilers do not support as well.

We then presented a generalized architecture for a decompiler and related it to its well-understood counterpart for compilers. Based on this, we introduced an experimental decompiler for Microsoft .NET CIL code with an attached OpenCL code generator. The next steps include extending the basic decompiler framework to support other source architectures and produce easily-readable high-level code, for example to aid in porting existing binary programs to other processor architectures like the Intel Single-Chip Cloud Computer.

References

[1] Boomerang – a general, open source, retargetable decompiler of machine code programs. http://boomerang.sourceforge.net/.

[2] Boomerang – what needs to be done. http://boomerang.sourceforge.net/tobedone.php.

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.

[4] Cristina Cifuentes. http://www.itee.uq.edu.au/~cristina/dcc.html.

[5] Cristina Cifuentes. *Reverse Compilation Techniques*. PhD thesis, School of Computing Science, Queensland University of Technology, 1994.

[6] Cristina Cifuentes. An environment for the reverse engineering of executable programs. *Asia-Pacific Software Engineering Conference*, 0:410, 1995.

[7] Cristina Cifuentes, Mike Van Emmerik, and Norman Ramsey. Uqbt - a resourceable and retargetable binary translator. http://www.itee.uq.edu.au/~cristina/uqbt.html.

[8] E. N. Dolgova and A. V. Chernov. Automatic reconstruction of data types in the decompilation problem. *Program. Comput. Softw.*, 35(2):105–119, 2009.

[9] ECMA. Common language infrastructure (cli). Standard ECMA-335, ECMA International, June 2005. www.ecma-international.org/publications/standards/Ecma-335.htm.

[10] J. Howard et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *ISSCC*, 2010.

[11] Maurice H Halstead. *Machine-independent computer programming*. Spartan Books, 1962.

[12] Daan Leijen and Judd Hall. Optimize managed code for multi-core machines. *MSDN Magazine*, October 2007. http://msdn.microsoft.com/en-us/magazine/cc163340.aspx.

[13] Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[14] Alan Mycroft. Type-based decompilation, 1998.

[15] Mike Van Emmerik and Trent Waddington. Using a decompiler for real-world source recovery. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 27–36, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Miao Wang, Rongcai Zhao, Jianmin Pang, and Guoming Cai. Reconstructing control flow in modulo scheduled loops. In *ICIS '08: Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pages 539–544, Washington, DC, USA, 2008. IEEE Computer Society.

Unifying the definition of megamodels: Toward describing service-oriented system's development

Regina Hebig

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

Service-oriented systems as well as model driven engineering are applied to affect the performance of software development and the degree of reuse of software. Thus, developing SOAs with MDE can lead to synergistic effects, but also to the compensation of SOA's effect of these two factors. Therefore, capturing the development with MDE is crucial for optimizing the way of building SOAs. Megamodels, that are used to capture models and their interrelations, are a natural candidate for that task. However, there is no unique definition of the term megamodel. In this report we survey different approaches and come up with a unified definition of the term megamodel.

1 Introduction

Today, model-driven engineering (MDE) is used in an increasing number of software development projects. Thereby, an increasing amount of models are applied to develop and realize the desired system. These models may show overlapping aspects of the system under development at different levels of detail, which is suggested in the unified process [8] by developing a system in different phases (e.g., requirements, analysis, design, implementation, etc.). Additionally, a key feature of MDE is automation. Thus, models are automatically created from other models by means of model transformations.

Service-oriented systems are applied, in order to increase the performance of software development and to support reuse of software. However, both factors performance and reuse are also influenced by the usage of MDE. Consequently, we assume that an unsuitable form of MDE might counteract the SOA goals, while a suitable MDE can strongly complement with SOA in exhausting the development performance and reuse of software. Therefore, the ability to capture the used models, model types, and model operations - the form of the MDE - is crucial, for optimizing the way of building SOAs.

In recent years scientific approaches arose, which deal with the question of how to capture the dependencies between models. The most prominent examples are

megamodels proposed by J. Bézivin [2] and J.M. Favre [4] or R. Salay's macromodels [10][1]. The dependencies of models that already exist are not the only thing in focus of research. Approaches that deal with transformation chains [3] aim to predefine and automatically perform compositions of model transformations that are defined between models.

There is already a considerable amount of literature about megamodels and related approaches [1, 2, 4, 5, 10, 11, 13].[2] However, the understanding and desired usage of megamodels differs from author to author. Therefore, we collect characteristics that can reasonably be applied to megamodels for classifying existing approaches and for providing an overall understanding of megamodels. In order to provide a common understanding of the concept megamodel, we unify the different definitions and provide a corresponding metamodel that reflects the definition. To the best of our knowledge there is currently no approach that gives an overview about megamodel approaches or provides a unified definition.

In Section 2 we survey different megamodel-related approaches. In Section 3 we present a unified definition for the term megamodel. A full version of the presented findings, is published in [7][3].

2 State of the Art

In this section, we discuss approaches that define the term 'megamodel' or related concepts like the 'macromodel'. We focus on how terms are defined and on the intention for the application of these types of models.

2.1 Definition of Megamodels and Related Terminologies

Most approaches that deal with megamodels come up with their own definition of the term. In the following we introduce the different definitions.

BÉZIVIN is one of the first founders of the term megamodel. His view of the term megamodel has grown during the recent years. The first definition was given in the year 2003: *"A megamodel is a model with other models as elements"* [2]. In 2007 this definition is further complemented: *"A megamodel contains relationships between models"* [1].

FAVRE can be seen also as one of the founders of the term megamodel. Nevertheless, in comparison to BÉZIVIN he has a strong theoretical focus on megamodels. In 2004 he gave the following definition: *"... the idea behind a megamodel is to define the set of entities and relations that are necessary to model some aspect*

[1] We will use the term megamodel for all approaches throughout this paper, since the main part of the surveyed papers use this term.

[2] Due to space limitations we only focus on a few approaches and omitted approaches that strongly overlap with them.

[3] This is a joint work with Andreas Seibel and Holger Giese

about MDE" [4]. In a follow-up paper, FAVRE provided an extension of the definition: *"A megamodel is a model that represents this kind of complex arrangements without entering into the details of each artifact"* [5].

SALAY provides an approach that uses the term macromodel, which is related to the term megamodel but has a different intention. A first definition is provided in 2007: *"A macromodel is a graphical model whose elements denote models and whose edges denote model relations"* [10]. Later, he defines the term macromodel as: *"A macromodel consists of elements denoting models and links denoting intended relationships between these models with their internal details abstracted away"* [11].

SEIBEL is also influenced by the terminology of BÉZIVIN and provides the following definition: *"A megamodel is a combination of high-level traceability models and low-level traceability models"* [13]. A high-level traceability model shows thereby relations between models whereas a low-level traceability model shows relations between element of different models.

2.2 Comparing the definitions

All approaches have in common that they use megamodels or macromodels in order to illustrate the dependencies between models. We further want to compare the definitions used by the different approaches. Thereby, some definitions come along with a metamodel for megamodels. These are the definitions of SEIBEL, FAVRE, BÉZIVIN, and SALAY. BÉZIVIN, FAVRE, and SALAY further allow extensions of their metamodels, such that the possible relations can be adapted to the current needs ('extensibility').

Besides the opportunity to extend the metamodel, some approaches have constraints on possible megamodels. For example, the metamodel that is introduced by BÉZIVIN excludes that a megamodel can be used as reference model. There, a megamodel is modeled as terminal model. Other constraints can arise if the types of relations that are allowed to be part of the megamodel are predefined. Most definitions do not restrict the possible relations, although they predefine relation types. BÉZIVIN only predefine the 'conformsTo' relation for associating a model to its reference model. However, he does not restrict the possible types of relations between the models. SALAY does not restrict the types of relations within the macromodel. In SEIBEL's approach relations are named 'traceability links', which can be fact links or obligation links. While fact links only state that there is a relationship, obligation links also define how a change can be propagated to restore the consistency of a relationship. Furthermore, SEIBEL allows the definition of the type of the traceability links via the 'isOfType'-relation. Thus, SEIBEL does not restrict the possible relation types, since the type and semantic of a required traceability link can be defined by the user of the approach. FAVRE introduces his metamodel in order to identify a set of relations that are sufficient to cover the needs of MDE. Naturally, the possible relation types are restricted to the relations already identified by FAVRE, such as 'RepresentationOf' , 'DecomposedIn', 'ConformsTo', or 'IsTransformedIn'. New relation types would require an extension of this metamodel. Therefore, we say that FAVRE restricts the possible types of relations.

	SALAY	FAVRE	BÉZIVIN	SEIBEL
Provide Metamodel	✓	✓	✓	✓
Extensibility	✓	✓	✓	-
Constraints (Megamodel is no Reference Model)	-	-	✓	-
Constraints on Relations	-	✓	-	-
Hierarchy of Megamodel	✓	✓	✓	✓
Hierarchy of relations	✓	-	-	✓
Inter-Level-Relations	✓	-	-	✓

Table 1: Properties of different Megamodel-definitions

As a further characteristic of a megamodel definition we want to highlight the question whether it supports *hierarchy*. That includes on the one hand the question, whether a megamodel can contain further megamodels. On the other hand there is the question whether relations can contain other relations. The most approaches allow megamodels explicitly to contain further megamodels. FAVRE allows systems to contain further systems via the 'DecomposedIn' relation. Similarly, SEIBEL allows a 'subElement' relation between models. Since a model that contains models conforms to the definition of a megamodel, one might say that SEIBEL allows a megamodel to contain further megamodels. BÉZIVIN allows a megamodel to contain models, while he sees a megamodel as a model. Thus, BÉZIVIN allows a megamodel to contain further megamodels. In a similar manner SALAY allows the same.

SALAY and SEIBEL allow relations to contain further relations. BÉZIVIN does not directly allow a relation to contain a further relation, but provides weaving models that can be associated to relations and contain relations between the elements of models that are associated to that relation. There is a further characteristic that is fulfilled by SEIBEL. He allow '*inter-level-relations*', this means relations that connect modeling artifacts on different hierarchical levels. For example, such a relation can connect a model element with a model. Following the formal definition of macromodels in [10] SALAY also allows '*inter-level-relations*'.[4]

2.3 Application of Megamodels

After looking at the definition of the term megamodel we want to summarize the intended applications of the different approaches. Thereby, we add the approach of DIDONET as example for transformation chain approaches. However, since these approaches deal with a kind of model operation, one can consider to use megamodels as definition language for transformation chains.

BÉZIVIN proposes different applications of megamodels. In [2] megamodels are applied to support model-driven software development by using it for model management. Thus, megamodels provide a global view on models. Whereas, in [1]

[4]Confusingly, SALAY's metamodel in [11] seems to allow no relations between macromodels and models.

megamodels are applied to facilitate traceability between models and their elements.

FAVRE applies megamodels to model MDE [4,5]. He does not focus on the applicability of megamodels but on reasoning about relations that can exist in the context of MDE. He defines exemplary patterns in MDE by using the megamodels.

SALAY shows in [10,11] how to apply macromodels to capture the modelers intention how different models, showing different views of the system, are related to each other. Therefore, a macromodel can be a type that represents a pattern that defines a certain intention of a modeler and it can be an instance that represents the current models. He provides an automated analysis to examine whether the intention of the modeler is satisfied.

SEIBEL shows in [13], similar to [1], that megamodels are predestined for the application of traceability in MDE. Additionally, relations can be associated with behavior, such as model transformations. This 'behavior of relations' is used to resolve inconsistencies that may occur when models change.

M. DIDONET introduces in [3] an approach for supporting the combination of rules and automated execution of multiple transformations. Thereby, he already proposes the integration of their language into a megamodeling platform, such that the interrelation of multiple transformation, models and metamodels can be better handled.

2.4 Characteristics of Megamodel Approaches

Besides the intention of a megamodel, it is interesting to look at the operations that are performed on a megamodel. SEIBEL and BÉZIVIN automatically *adapt* their megamodels to the current situation. In contrast, SALAY *compares* his macromodels with the current situation and verifies, whether the constraints defined in the macromodel are satisfied. In addition to the automated derivation of the megamodel, SEIBEL automatically *interprets* his megamodels in order to restore consistency between models. Also FAVRE interprets his megamodels in order to identify mega-pattern automatically. Since DIDONET does not use megamodels, he cannot perform operations on it. However, the specification he uses instead is interpreted within the tool.

Models in classical software engineering are used to illustrate the structure, the behavior, and the function of a system (e.g., feature models [12]). Transferred to megamodels '*behavior*' means that the approach treats the megamodel as a behavioral model and '*structure*' means that the approach does only consider the megamodel as a representation of the models including their relationships. Most approaches use megamodel for the illustration of structure, which might, for example, be compared to a current situation. Only SEIBEL also defines behavior, since he defines the trigger for actions to reestablish consistency as well as the corresponding actions within the megamodel. None of the approaches uses megamodels to illustrate function.

We differentiate between four types of relations. Thereby, the first type is the relation that illustrates a relation that exists in the represented system. Typically this type of relation occurs in 'normal' models and is in most cases not expected to occur in megamodels. The other three types of relations should not be illustrated with non-megamodel models. First, there is the *'overlap'-relation* that indicates that models have overlapping aspects. This relation is, for example, used by SALAY to indicate that two models show different views of the same system. SEIBEL uses this relation as basis for identifying consistency and inconsistency between two models. In [1] BÉZIVIN uses megamodels for traceability issues and not as repository as in [2]. Therefore, he uses 'overlap'-relations in [1]. The *'model operation'-relation* is a representative of a model operation that was/can be/will be performed with models as input and models as output. For example, FAVRE introduces the 'isTransformedIn' relation, which indicates that one model was transformed to another model. Similarly, BÉZIVIN support this relation type. SEIBEL uses the 'model operation'-relation for automatically restoring the consistency between two models. The last type of relation is called *'static'-relation*. We introduce this type for the reason of capturing a huge amount of relations, such as 'conformsTo', which represents the relation between a model and a reference model, or the 'contains' relations, which indicates that a model is contained by another model. This might also include the 'represents' relation between a models an the represented system, which is used by FAVRE. We do not provide a full list of relations that can be handled as 'static'-relations. Like FAVRE denotes, there is still research on the question of which relations between models are part of MDE. Most approaches, like BÉZIVIN, work with the 'conformsTo' relation. SEIBEL uses the 'isOfType' relation for links. Also SALAY uses to show the 'instanceOf' relation in his example models. A model might be used to describe a current situation (*descriptive*) or to define a desired situation or behavior (*prescriptive*). The approaches of BÉZIVIN consider the megamodel to be created automatically and, thus, to describe the actual state of the system. Also SEIBEL works with an automatically derived megamodel and thus descriptive megamodels. However, SEIBEL defines operations for reestablishing consistency between models and, thus, prescribes also behavior. FAVRE captures a current picture, where he also models mega-patterns of relations, which he uses to describes possible types of activities in MDE. SALAY uses his megamodel to describe a current situation as well as to prescribe desired situations with constraints. As the goal of DIDONET is the definition of a chain of transformations, he specifies behavior, which is the application of transformations in the specified order. Thus, DIDONET's approach is prescriptive. Since, DIDONET uses no megamodel he defines no relations. However, he specifies transformations and, thus, deals with model operations.

3 Unified Definition of Megamodel

As shown above, megamodel-related approaches have different motivations, and come along with different meanings of the term megamodel. We first provide a 'unified' definition of the term megamodel, which is a synthesis of the definitions of the investigated approaches from Section 2.1. Based on this unified definition we provide a generic

metamodel of a megamodel. Finally, we define what we expect from a megamodel.

The basic structure of a model can be considered as *a graph that contains nodes and edges between them*. A classical software model is typically a graph [9] and thus is *a model that contains model elements and relations between them* with model elements are nodes and relations are edges. A megamodel could be considered as a classical software model. Thus, we define a megamodel as: *a model that contains models and relations between them*. However, there is a major difference to classical software models that is a megamodel explicitly considers models instead of model elements. This difference is visually shown in Figure 1.

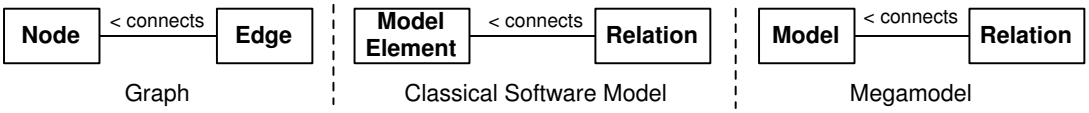


Figure 1: Core of structural models

Based on this unified definition of a megamodel we define a metamodel, which is free of implementation aspects and can be extended to the needs of the different approaches. The metamodel is shown in Figure 2. This metamodel contains additional concepts like models can contain models, models can contain relations and relations can depend on other relations. These additional concepts are motivated in the following.

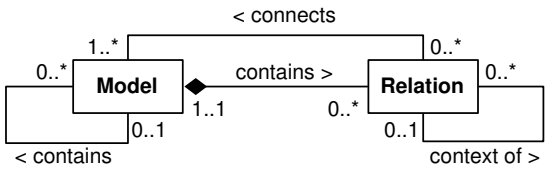


Figure 2: Core metamodel for megamodels

But first, why is a relation not considered to be a model? Basically, because an edge is not considered as a graph. In the megamodel case, the relation itself only describes that models are related in some specific way (dependent on the type of a relation). For example, FAVRE introduces a relation of the type 'isTransformedIn', which reflects that one model is the result of a transformation that was applied to another model. If an approach like SEIBEL's requires for example a transformation specification to be associated to a relation, the core metamodel can be extended, such that a relation can have a corresponding association to a model.

A model can contain models for at least two reasons. First, a megamodel is a model that contains models and thus a model should be capable of containing models (hierarchical megamodel). In this case, the model is a megamodel that can contain models as well as megamodels. Second, model elements in classical software models can be interpreted as models and thus models can contain models. In this case, the model is a model element, which does not contain further models. However, this

depends on how models are internally structured and how they are interpreted. To the best of our knowledge, there is currently no consensus about how to structure models right. A prominent example is UML. Technically, a UML model is a single monolithic model. However, conceptually a UML model may contain different models, e.g, class diagrams, sequence diagrams, etc. Nevertheless, these are only model elements in a UML model. This interpretation of model elements depends on the level of detail that should be considered in a megamodel.

Additionally, a model can contain relations. A megamodel is a model that contains relations between models. Thus, the model should reflect the capability of containing relations, too. In the core metamodel relations are not directed. A direction can be introduced by extending the core metamodel. Finally, a relation between models does not necessarily exist in isolation. Certain relations may require another relation. In [6] a transformation explicitly requires another transformation as its own context. In [13] it is shown that relations at different levels of detail can have a dependency between each other. We further expect that models in a megamodel, which are not megamodels itself, do only contain relations that are necessary to describe their original. All other relations should not be part these models but should be explicitly contained in the megamodel that contains these models. For example, relations like 'conformsTo', 'contains' (if they are used for structuring models only), 'isTransformedIn', etc. should be explicitly reflected in a megamodel but not in the models itself.

All in all, our proposed core metamodel of megamodels is extensible, allows hierarchies of models (and thus of megamodels) and relations, and supports 'inter-level-relations'. In comparison to BÉZIVIN, we do not provide specific types of models nor relations. Furthermore, we do not declare a megamodel to be a terminal model, which is a specific kind of model. FAVRE's metamodel of megamodels does not distinguish between the concept of models and relations. He also predefines specific relations (e.g., 'conformsTo') in the metamodel, which is however not necessary in all application domains of megamodels. SALAY's metamodel of macromodels does explicitly distinguish between macromodel and model and further between macrorelations and relations. We do not make this explicit distinction because we define a megamodel to be a model. SEIBEL's metamodel of megamodels is somehow similar to this metamodel but it is more detailed. It explicitly distinguished between models and model elements. Furthermore, it is distinguished between two kinds of relations. Subsuming, our proposed core metamodel provides a consistent view on megamodels, which covers all surveyed approaches.

To give an idea of how to extend the core metamodel of the megamodel, we show an exemplary extension in Figure 3. (a) shows a metamodel extension, which additionally contains three kinds of relations like 'Transformation', 'conformsTo' and 'isTransformedIn' and four kinds of models like 'M2Model', 'M1Model', 'M1ModelElement' and 'Tool'. 'M2Model' represents metamodels whereas 'M1Model' represents models, which conform to 'M2Model' reflected by the 'conformsTo' relation. The 'Transformation' relation is specified by a 'M1Model', which contains an executable specification of a model transformation. This specification can be executed by a tool 'Tool'. It takes a transformation with a specification connected and creates an 'isTransformedIn' relation between M1Models it has transformed.

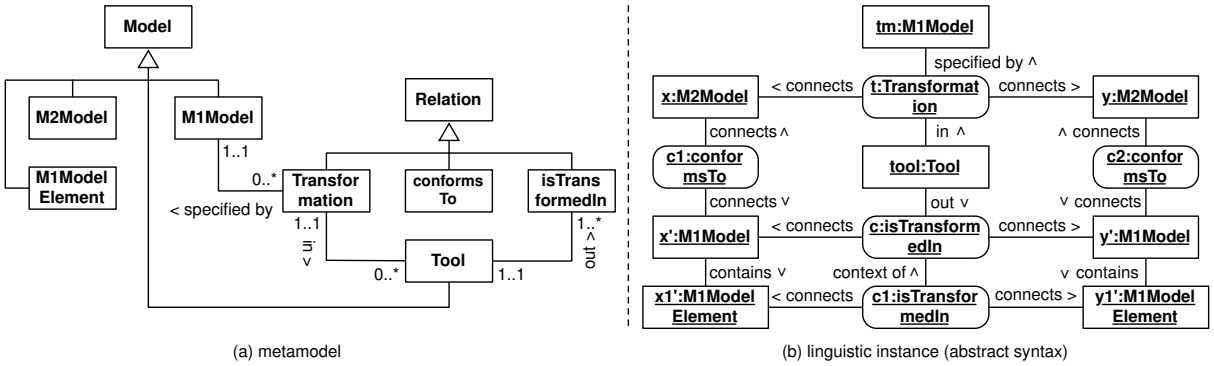


Figure 3: Exemplary extension of the core megamodel metamodel

(b) shows a linguistic instance of the metamodel extension. ‘tm’ is the specification of a model transformation ‘t’ between the two ‘M2Model’-metamodels ‘x’ and ‘y’. ‘tool’ applied ‘t’ on ‘x” (which conforms to ‘x’) and ‘y” (which conforms to ‘y’). Thereby, ‘tool’ produced the ‘isTransformedTo’ relation ‘c’. A strength of the hierarchy can be seen in the example, too. It is possible to illustrate not only that the model ‘x” was transformed to the model ‘y”, but also that in this context, the model element ‘x1” was transformed to the model element ‘y1”.

Note, this is only one possible extension of the megamodel-metamodel. For example, the semantic of the specified ‘isTransformedIn’ relation might not be appropriate for each megamodel approach. In (b) there is an ‘isTransformedIn’ relation between model ‘x” and model ‘y”. What happens if model ‘x” is affected by changes? In the semantic of this metamodel extension, there is no reason that the ‘isTransformedIn’ relation will be removed. This is appropriate if the changes on ‘x” do not influence ‘y” or if the megamodel approach is only interested in the fact, that the transformation happened. However, other megamodel approaches might require the transformation to be redone. It might even be desired to introduce the notion of validity to the transformation ‘isTransformedIn’, such that the relation would become invalid if ‘x” changes. This example shows that even terms that are strongly associated to megamodels, such as transformations, can have quite different semantics in different megamodel approaches.

4 Conclusion & Future Work

Different approaches provided different understandings of the term megamodel. We introduced a unified definition of the term megamodel including a core metamodel that will help to improve discussions about megamodels. We were able to show that megamodels are used for very different purposes, but also for the illustration of behavior. Further, we introduced a unified core definition for the term megamodel. Based on this core definition, we can, in future work, specialize megamodels for capturing the MDE development of SOAs. Further, we will develop metrics for comparing different forms of MDE development, in order to find the most suitable form.

References

[1] Mikael Barbero, Marcos Didonet Del Fabro, and Jean Bézivin. Traceability and Provenance Issues in Global Model Management. In *ECMDA-TW'07: Proc. of 3rd Workshop on Traceability*, pages 47–55. SINTEF, June 2007.

[2] Jean Bézivin, Sébastien Gérard, Pierre-Alain Muller, and Laurent Rioux. MDA components: Challenges and Opportunities. In *Proc. of First International Workshop on Metamodelling for MDA*, pages 23 – 41, York, UK, November 2003.

[3] Marcos Didonet Del Fabro, Patrick Albert, Jean Bézivin, and Frédéric Jouault. Industrial-strength Rule Interoperability using Model Driven Engineering. Research Report RR-6747, INRIA, 2008.

[4] Jean-Marie Favre. Foundations of Model (Driven) (Reverse) Engineering – Episode I: Story of The Fidus Papyrus and the Solarus. In *Proc. of Dagstuhl Seminar on Model Driven Reverse Engineering*, 2004.

[5] Jean-Marie Favre. Megamodelling and etymology. In *Proc. of Dagstuhl Seminar on Transformation Techniques in Software Engineering*, volume 05161, 2005.

[6] Mathias Fritzsche, Hugo Brunelière, Bert Vanhooff, Yolande Berbers, Frédéric Jouault, and Wasif Gilani. Applying Megamodelling to Model Driven Performance Engineering. In *Proc. of 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 244–253. IEEE Computer Society, 2009.

[7] Regina Hebig, Andreas Seibel, and Holger Giese. On the Unification of Megamodels. In *MPM'10 at the 13th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems*, 3 October 2010.

[8] I. Jacobson, G. Booch, and Jim Rumbaugh. *Unified Software Development Process*. Addison-Wesley, 1999.

[9] Thomas Kühne. What is a model? In *In proc. of Dagstuhl Seminar*, volume 04101, 2005.

[10] Rick Salay. Towards a Formal Framework for Multimodeling in Software Engineering. In *Proc. of the Doctoral Symposium at the ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems*, volume Vol-26, Nashville (TN), USA, October 2007.

[11] Rick Salay, John Mylopoulos, and Steve Easterbrook. Using Macromodels to Manage Collections of Related Models. In *Proc. of 21st International Conference on Advanced Information Systems Engineering (CAiSE'09)*, volume 5565/2009 of *LNCS*, pages 141–155, Amsterdam, The Netherlands, 8-12 June 2009. Springer.

[12] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *Proc. of 14th IEEE International Requirements Engineering Conference (RE'06)*, volume 0, pages 139–148, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[13] Andreas Seibel, Stefan Neumann, and Holger Giese. Dynamic Hierarchical Mega Models: Comprehensive Traceability and its Efficient Maintenance. *Software and System Modeling*, 9(4):493–528, 2009.

A Granular Approach for Information Lifecycle Management in the Cloud

Johannes Lorey
Information Systems Group
Hasso-Plattner-Institut
johannes.lorey@hpi.uni-potsdam.de

Traditional data placement strategies in the context of Information Lifecycle Management (ILM) are applicable only to on-site storage systems. In contrast to this approach, Cloud storage provides a novel possibility to reduce or entirely eliminate capital expenditures for hardware. As a unique solution to buffer short-term resource demand peaks, Cloud infrastructures can be combined with on-site systems to support efficient placement of data.
The algorithms underlying this optimization must consider not only the workload as a whole, but rather variable-sized subworkloads to determine an optimal placement. As a means to identify these subworkloads, we introduce a multi-dimensional granularization approach. Based on different granules of metadata information, we propose a flexible hybrid data placement system incorporating both on-site and Cloud resources.

1 Research Context and Related Work

The amount of digital data is increasing constantly. A recent study by hardware vendor EMC yields that in 2010 the overall storage demand of newly created data will increase to 1.2 Zettabytes, a 50% increase from 0.8 Zettabytes in 2009 [6]. The expenses for purchasing and maintaining storage infrastructure on the other hand declines – but at a much slower rate [19]. Hence, evaluating current data placement paradigms offers great opportunity for reducing storage cost. Our idea for facing this challenge comprises approaches from three different research areas.

1.1 Information Lifecycle Management

To reduce capital expenditures for hardware, being able to dynamically determine the current value of a single datum is a key factor in choosing its optimal storage location. This decision process is generally referred to as Information Lifecycle Management (ILM). ILM strategies aim at reducing the cost for storage by placing and relocating data accordingly. For placement decisions, usually only on-site hardware resources are considered, such as hard disk drives or magnetic tapes [3] [11]. These resources can be classified hierarchically into different categories based upon their technical and economical characteristics. Generally speaking, higher-level storage is more costly (in

terms of money per storage unit) and more easily accessible, while lower-level storage is cheap and mostly used as a back-up solution. This relationship is illustrated in Fig. 1.

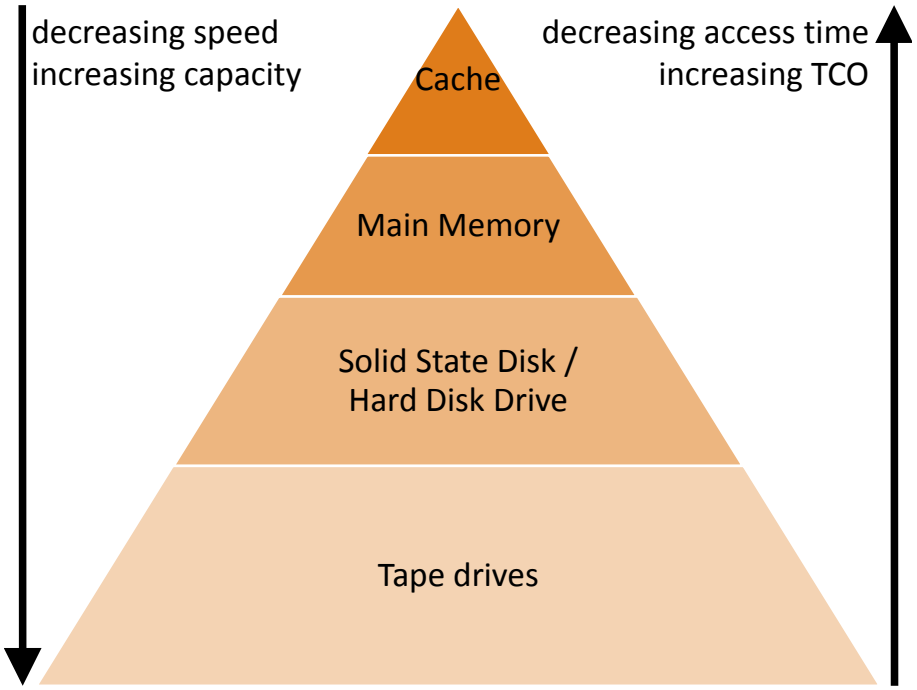


Figure 1: The Information Lifecycle Management Pyramid.

Thus, purchasing and operating a hard disk drive will result in higher cost than employing a magnetic tape with identical size. On the other hand, a record that is accessed frequently should reside on a faster hard disk drive instead of a slower tape in order to ensure time-critical operations can be executed quickly. During its lifetime, a single datum is typically moved from higher-level to lower-level storage depending on several factors, such as its access frequency or its current importance. Usually, these migration operations are based on a set of pre-defined policies [1] [18] or are sometimes even performed manually [17].

One major drawback of the traditional fixed set-up illustrated in Fig. 1 is its lack to flexibly handle varying demand in storage resources. This shortcoming is typically by-passed with higher capital expenditures, i.e., 'throwing more hardware at the problem'. Clearly, this approach cannot be regarded as a sophisticated solution. Instead, there is a demand for an alternative storage platform to flexibly buffer demand peaks without high up-front cost.

1.2 Cloud Storage

Cloud Computing and Cloud storage offer a novel approach for data handling and maintenance. Two of the main characteristics of the Cloud are its rapid elasticity and the associated metering of resource consumption. Hence, there are little to no capital

expenditures associated with using Cloud services [13]. On the other hand, long-term storage necessities may economically justify purchasing on-site hardware instead of renting resources in the Cloud. The authors of the aforementioned EMC study point out that by 2020, one third of all digital data will either live in or pass through the Cloud.

As Cloud Computing has been a major technological trend for only a short time, the decision between using on-site or Cloud resources has usually been a binary one for companies up to now. Established organizations usually have the resources required for their daily operations at hand and thus do not sense the need to outsource infrastructure. On the contrary, security and reliability concerns might even prevent them from doing so [15]. For start-up businesses on the other hand, commercial Cloud Computing offerings usually satisfy most resource demands and provide the entire hardware and software stack for them to offer services over the Internet.

However, there has been only little research on integrating both on-site and Cloud resources. Most of the ongoing projects aim at emulating certain aspects of commercial providers [10] or face the challenge of establishing a hybrid system by storing identical copies of data on-site and in the Cloud [2]. Generally, data is either regarded in its atomic form or in its entirety. Other work focuses on the technical aspects of combining local and remote storage facilities [7], but not on the actual placement decisions. Only recently have there been advances to include Cloud resources for ILM systems, but these have been rather limited to certain scenarios [9].

1.3 Distributed Databases

Wide-area networks and the accompanying dissemination of information has also resulted in great research effort in the context of distributed databases. Cloud storage services are based on the same concepts. Thus, the elementary challenges faced for highly-distributed systems are also present in a Cloud environment, such as concurrency control, reliability, or consistency [12]. It has been recognized by the research community that the old paradigms established for local databases, such as ACID, are not always suitable for or even necessary in a distributed or Cloud environment. Instead, new requirements have been proposed, such as predictability and flexibility [5]. This somewhat impedes the combination of Cloud and on-site resources.

In general, the aim of distributed databases is to provide an efficient way for querying and processing disseminated information. In this however, they are limited by the well-known CAP theorem [4]. The acronym summarizes the three concepts **C**onsistency, **A**vailability, and **P**artitioning and states that it is trivial to achieve any two of them, but impossible to satisfy all three at the same time. In the case of data distributing, partitioning is implicit whereas consistency might be sacrificed for availability or vice versa.

Besides providing greater scalability, distributed databases also offer straighforward failover and back-up mechanisms. If data is replicated across the infrastructure, losing a single system should be easy to compensate. However, whereas in single-machine environments the choice for data relocation is trivial, in a distributed set-up migrating data to satisfy queries more efficiently, i.e., by moving the data closer to the consuments, is an ongoing research challenge with many practical implications [20] [14].

2 Granular Access

The traditional approach of establishing data placement strategies in the context of ILM tends to handle all involved entities isolated from one another [3]. For example, pieces of data are only considered in their most atomic form (e.g., as an individual record or binary file). Also, the lifecycle of a datum is usually partitioned into fixed-size time frames. Moreover, data access often is monitored globally only, i.e., with no regard to individual user or user group access profiles. This stems from the fact that data storage systems were conventionally ordered hierarchically according to their access speed and respective cost per storage unit. We propose a hybrid approach to optimize placement decisions, both in terms of storage capabilities and data organization.

2.1 Fragments

As a basic abstraction model, we propose the notion of fragments. A fragment serves as a container for multiple atomic data items that are somewhat similar regarding the way they are accessed. This allows us to formulate placement decisions based on different levels of granularity by combining more (i.e., coarse-grained) or less (i.e., fine-grained) atomic data items into an individual fragment. We refer to this approach as Granular Access. The notion of a fragment is inspired by the ideas proposed in [16].

It should be pointed out that fragments are not strict partitions of the underlying dataset. Instead, multiple fragments may contain the same atomic data item. Thus, to optimize query execution by storing data in the most suitable location, there exist two alternatives: replication (i.e., having multiple copies of a data item) and relocation (i.e., moving a single data item to where it is currently best fit).

Both replication and relocation offer advantages and disadvantages. Replication leads to an increase in storage space and consistency issues, but provides an implicit back-up mechanism and avoids transfer overhead. On the other hand, relocation is very efficient in terms of space demands and consistency, however it offers no automatic failover mechanism and may result in bandwidth clogging.

2.2 Two-dimensional Granular Access

The two basic dimensions for deriving fragments based on Granular Access are data and users. Obviously, atomic data items represent the target of every access operation. Users, on the other hand, are the source for these operations. By monitoring information of user operations on data, access patterns can be established. This in turn helps discovering relationships between pieces of data or individual users. Additionally, it is usually helpful to identify the type of an operation, in particular whether the access is non-conflicting (e.g., a read) or could be potentially conflicting (e.g., a write).

Fig. 2 illustrates one possible example for such a relationship between users and data items based on different kinds of access operations. Here, $d_{1,1}$ through $d_{1,8}$ refer to atomic data items, whereas $u_{1,1}$ through $u_{1,8}$ indicate individual users. A red square represents a potentially conflicting access operation on a datum by a user, a green

square stands for a non-conflicting operation. By clustering multiple access operations, fragments can be formed.

For example, in Fig. 2 there is a strong relationship between $u_{1,1}$, $u_{1,2}$ and $d_{1,5}$, $d_{1,6}$. Hence, on a more coarse-granular level $u_{1,1}$, $u_{1,2}$ are considered as $u_{2,1}$ and $d_{1,5}$, $d_{1,6}$ as $d_{2,3}$. This in turn hints at how the fragment is composed (containing $d_{1,5}$, $d_{1,6}$) and where it should be stored ('close' to $u_{1,1}$, $u_{1,2}$). Note that the first index of $d_{i,j}$ and $u_{k,l}$ indicates how many individual data and user entities are comprised, respectively.

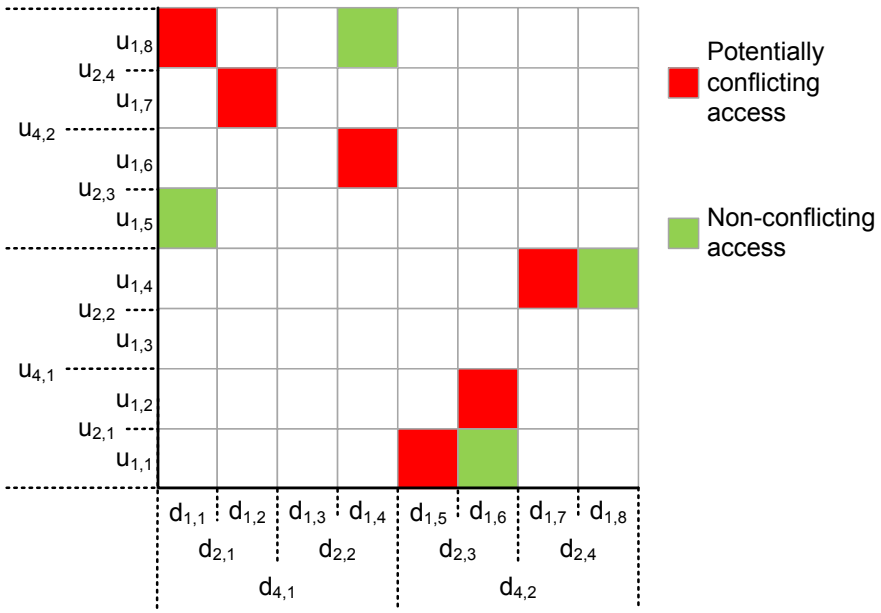


Figure 2: Granular Access based on user and data information.

2.3 Three-dimensional Granular Access

In the context of ILM, time is another essential dimension. Thus, we extend our previous model of Granular Access to include a temporal component as well. Fig. 3 visualizes the result. As with the other two dimensions, time can be considered on different levels of granularity. An atomic time interval $t_{1,i}$ might represent a minute, an hour, a day, etc. By coarsening the granularity, different access patterns might become apparent, e.g., morning/evening, weekday/weekend, or seasonal trends.

In [8], we present a formal definition for granular ILM workloads based on the data, user, and time dimension. There, we also discuss the implications of fragment composition properties on scalability, storage model, and transaction policy.

3 System Design

Fig. 4 depicts our design for an ILM system based on Granular Access. Here, the central concept are fragments, which comprise a number of atomic data items. One

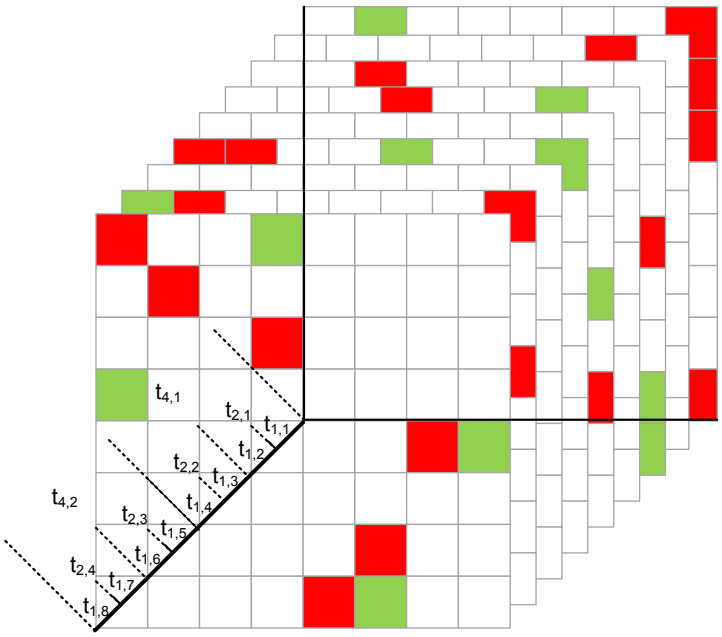


Figure 3: Granular access based on user, data, and temporal information.

individual data item might be included (i.e., replicated) in multiple fragments. The major components of the system design are briefly summarized in the following subsections.

3.1 Transaction Analyzer

The Transaction Analyzer identifies the atomic data items for a single query based on the according transaction log entry. For each datum, the corresponding metainformation (e.g., the storage location and size) is gathered. Subsequently, the lifecycle status of every datum is updated. The resulting metadatum entities are then passed on.

3.2 Fragment Cohesion Calculator

The Fragment Cohesion Calculator uses the identified metadatum items to establish possible fragment combinations. For this, the user and time information is extracted from the currently analyzed query. Possibly, more than one atomic datum is relevant to the query, hence, all possible fragment combinations are identified and a corresponding cohesion score is calculated. The cohesion score reflects how coherent the individual data items are with respect to the fragment's size. In practice, a threshold might be applied to the cohesion score in a subsequent component to avoid composing and storing fragments with only loosely connected data items.

In case a fragment is already present in the fragment metainformation database, its score is updated. Otherwise, the properties of the new fragment are stored in the database. Among these properties are the cohesion score, the identifiers of the individual data and user items, as well as the query time. This information is used to identify the proper storage location later on.

3.3 Fragment Analyzer

The Fragment Analyzer calculates the overall lifecycle information of a fragment based on the according status of its components. In addition, it determines the fragment's replication factor, i.e., how many of the fragment's data items can be found in other fragments. While the cohesion score was solely based on an individual fragment, the replication factor must be determined by comparing multiple fragments with one another.

3.4 Cost Estimator

As placement decisions are formulated based on a cost model, the Cost Estimator needs to gather information about the technical and monetary characteristics of all available datastores. By calculating the cost for storage, transfer, and processing of a fragment, the optimal storage location for its data items can then be determined. However, a cost model for leasing disk space in a Cloud can become very complex even when only regarding plain data storage features [19].

3.5 Storage Dispatcher

As pointed out in the description of the Cost Estimator and indicated in Fig. 4, datastores might present different technical features. This is observable foremost in their storage model, e.g., being relational, key/value, etc. Hence, for dynamically migrating data between datastore with different paradigms it is essential to convert data from one format to another. This is the responsibility of the Storage Dispatcher. Additionally, it executes the relocation operations by actually transferring the data items to the respective infrastructure.

4 Summary and Next Steps

In this work, we presented our granular approach for Information Lifecycle Management incorporating both on-site and Cloud resources. Additionally, we introduced the key concept of fragments and analyzed our idea of a Granular Approach on data incorporating user and temporal information. Furthermore, we gave an overview of a system design that leverages different datastores to automate and optimize placement decisions. Here, we briefly discussed the most important components.

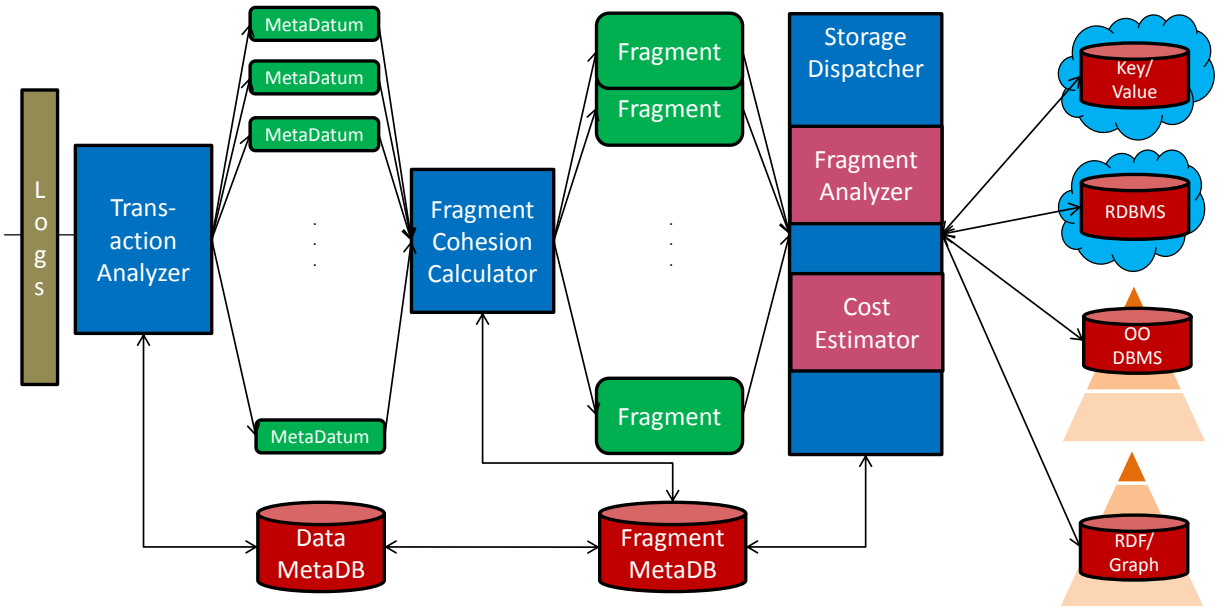


Figure 4: A design for an ILM system based on Granular Access.

Using the Granular Access approach in combination with the ideas from the research areas outlined in Sec. 1, our future work will aim at answering the following questions:

- How can the value of a datum be estimated accurately?
- How can data allocation and query processing be optimized?
- How can unnecessary relocation operations be avoided to prevent bandwidth clogging?
- How can the cost for data storage be minimized?

The answers to these questions will influence the fragments' composition, e.g., with respect to their size and replication factor. The composition approach in turn has implications on the implementation of the individual components of our system that were introduced in Sec. 3. As next steps, we will specifically focus on the Fragment Cohesion Calculator and Fragment Analyzer. We plan to evaluate our ideas on a number of use cases.

References

[1] Mandis Beigi, Murthy Devarakonda, Rohit Jain, Marc Kaplan, David Pease, Jim Rubas, Upendra Sharma, and Akshat Verma. Policy-Based Information Lifecycle Management in a Large-Scale File System. *IEEE International Workshop on Policies for Distributed Systems and Networks*, 0:139–148, 2005.

[2] Borja Sotomayor, Ruben S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13:14–22, 2009.

[3] Charlotte Brooks, Giacomo Chiapparini, Wim Feyants, Pallavi Galgali, and Vinicius Franco Jose. *ILM Library: Techniques With Tivoli Storage And IBM Totalstorage Products*. IBM Redbooks, 2006.

[4] Eric A. Brewer. Towards robust distributed systems. In *Proceedings of the 19th annual ACM symposium on Principles of distributed computing*, 2000.

[5] Daniela Florescu and Donald Kossmann. Rethinking cost and performance of database systems. *SIGMOD Rec.*, 38(1):43–48, 2009.

[6] John Gantz and David Reinsel. The Digital Universe Decade -Ű Are You Ready? `http://chucksblog.emc.com/content/DU2010text.pdf`, May 2010. IDC iVIEW.

[7] Heeseung Jo, Youngjin Kwon, Hwanju Kim, Euiseong Seo, Joonwon Lee, and Seungryoul Maeng. SSD-HDD-Hybrid Virtual Disk Consolidated Environments. In *Proceedings of the 4th workshop in Virtualization in High-Performance Cloud Computing (VHPC)*, August 2009.

[8] Johannes Lorey and Felix Naumann. Towards Granular Data Placement Strategies for Cloud Platforms. In *2010 IEEE International Conference on Granular Computing*, pages 346–351. IEEE Computer Society, 2010.

[9] M. Meisinger, C. Farcas, E. Farcas, C. Alexander, M. Arrott, J. D. L. Beaujardière, P. Hubbard, R. Mendelssohn, and R. Signell. Serving Ocean Model Data on the Cloud. In *Proceedings of OCEANS 2009 MTS/IEEE*, 2009.

[10] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[11] Oracle. Information Lifecycle Management for Business Data. `http://www.oracle.com/technology/deploy/ilm/pdf/ILM_for_Business_11g.pdf`, June 2007.

[12] M. Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[13] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing, July 2009.

[14] Mahadev Satyanarayanan, Victor Bahl, Ramon Caceres, and Nigel Davies. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 99(PrePrints), 2009.

[15] James Staten, Simon Yates, Frank E. Gillett, Walid Saleh, and Rachel A. Dines. Is Cloud Computing Ready For The Enterprise? Technical report, Forrester Research, 400 Technology Square Cambridge, MA 02139 USA, March 2008.

[16] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996.

[17] Paul P. Tallon. Understanding the Dynamics of Information Management Costs. *Communications of the ACM*, 53(5):121–125, 2010.

[18] Tetsuo Tanaka, Kazutomo Ushijima, Ryoichi Ueda, Ichiro Naitoh, Toshiko Aizono, and Norihisa Komoda. Proposal and Evaluation of Policy Description for Information Lifecycle Management. *International Conference on Computational Intelligence for Modelling, Control and Automation*, 1:261–267, 2005.

[19] Edward Walker, Walter Brisken, and Jonathan Romney. To lease or not to lease from storage clouds. *Computer*, 43:44–50, 2010.

[20] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A data placement strategy in scientific cloud workflows. *Future Gener. Comput. Syst.*, 26(8):1200–1214, 2010.

Data in Business Process Modeling

Andreas Meyer
andreas.meyer@hpi.uni-potsdam.de

The importance of data in business process modeling increased steadily during the last years motivated by the need of visualizing executable business process models as well as industry-driven with regards to process controlling and business intelligence. Several modeling notations do exist which deal with data on different levels. This report classifies these notations and discusses some of them with regards to the role of data. Additionally, chances with respect to future work in the field of data object life cycles are discussed briefly.

1 Introduction

Traditionally, business process management emerged from Workflow modeling in the mid-nineties of the last century. Workflow modeling comprised the basic representation of tasks needed to be executed to reach a specific business goal. These tasks were ordered and for each of them, conditions could be specified which guided the execution of the whole workflow. These conditions included statements about required information, material, and resources, i.e. persons and machines, to perform the actual work. [18] presents the derivation of patterns for all these perspectives from mainly workflow systems.

In the beginning, business process management focused on the design and documentation of business processes and therefore, it focused on the order of and on dependencies between tasks while data and resources played a very minor role if any; i.e. the so-called control flow dominated these business process modeling notations. The first notations utilized for modeling heavily relied on long existence approaches, e.g. workflow nets have been derived from Petri nets which have been introduced by Petri in 1962 [15]. In the beginning of this century many new approaches have been introduced, but only few are of ongoing interest in business process modeling, e.g. Business Process Model and Notation (BPMN) [13, 14]. Most of the notations introduced in the beginning of the first decade do still focus on control flow, whereas later introduced notations shifted their focus to data or at least data awareness. Two big steps in this field are the notion business artifacts from IBM [1, 6, 11] and the outcome of the Corepro project of the university of Ulm in cooperation with Daimler AG [8]. This shift of focus aligns with a widening focus in business process management. Execution of business process models, especially the ones designed using a graphical notation, became a more and more important aspect in business process management. Execution itself did already play an important role for a while and Business Process Execution Language (BPEL) [12] became the industry standard. But BPEL lacks a graphical representation (besides XML structures). Therefore, effort was put into a mapping of the

widely accepted BPMN representation to BPEL. However, the mapping is not straightforward [26] and finally, BPMN will get execution semantics in the upcoming version 2.0 [14]. And as execution got more attention also data needed to get attention as almost all process rely on documents and information; e.g. a simple order process needs to consider the order document itself, probable item lists stating availability and costs of the ordered items, the address information for delivery et cetera. All this information is visualized by the means of data objects.

Besides the need based on execution, industry asks for data support for several years now as it can be observed for instance through the keynotes as well as tutorials by industry people at the recent BPM conference, the main conference on business process management. The speakers claim for instance that control flow and data information depend on each other [17] and none of both “means” something without the existence and specification of the other (Paul Harmon).

Additionally, data flow, i.e. the usage of data objects throughout a process, shall be documented as well as used for KPI measuring and controlling and the field of business intelligence. Business intelligence requires explicit statement of data objects. Pre- and postconditions of data objects with regards to the manipulation of them during process execution are of importance as well. Usually, these data objects are provided by different, probably intersecting legacy systems installed in the organization and therefore, the origin of data or more general information must be visualized and considered in the business process models.

Tackling such talks and requirements, the aforementioned data-driven approaches emerged during the last couple of years. However, even if industry claims the importance of data, not everybody follows the approach of strict data-centric approaches as to be seen by the industry-driven development of the BPMN 2.0 standard. BPMN 2.o will get a stronger focus on data but still keeps data object as supporting modeling constructs and control flow will dominate the models. But nevertheless, they also incorporated the higher importance of data in process models.

The scope of this paper is to revise different existing business process modeling notations and discuss the role of data within these notations with regards to the requirements sketched above. After presenting these discussions, ideas for future work and improvements regarding business process modeling and data are presented. The remainder of this report is structured as follows. Section 2 generally classifies different business process modeling notations and discusses their feasibility to deal with data at an high level of abstraction. Section 3 introduces ideas on utilizing existing works on dependencies between business process models and data object life cycles. Additionally, an outlook on future research is given. Finally, this report is concluded in Section 4.

2 Data in Business Process Modeling Notations

This section presents a high-level classification schema of business process modeling notations in the upcoming subsection. Afterwards, representatives for each classification type are introduced and the role of data in these approaches briefly discussed.

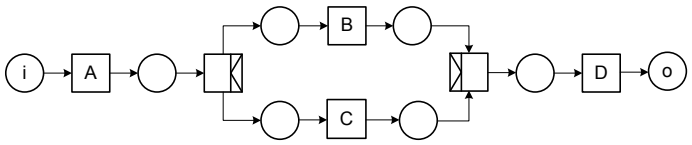


Figure 1: Workflow net

Subsection 2.5 finalizes this section by summarizing the discussion.

2.1 Classification

Generally, business process modeling notations can be classified with respect to their primary modeling constructs. Basically, three types have been identified: activity-driven, data-driven, and communication-driven approaches.

2.2 Discussion of Activity-driven Approach

The activity-driven approach mainly focuses on the representation of the control flow. In this section, workflow nets, BPMN 2.0, and BPEL are discussed as three important representatives in this area. BPEL is the industry standard in business process execution languages. BPMN gathered much attention regarding designing business process models in a graphical environment and moves onward to execution semantics specified in the 2.0 standard. Additionally, BPMN is widely accepted in industry and highly influenced by industry leaders like SAP and Oracle. Workflow nets have been the beginning in business process management and are still used as basis of other notations like BPMN to allow syntactical checks in these higher level notations.

Petri Nets and Workflow Nets. Petri nets were introduced by Petri in [15]. After several extensions covering for instance timing, hierarchies, and colors in Petri nets [4,5,22,23], they are often used to model or formalize business processes. Transitions model process activities, while places model process states and help to realize routing decisions. The edges capture the process control flow. Data or data objects are not considered at all. Hence, Petri nets focus on modeling of process activities and control flow. The idea behind workflow nets is to utilize petri nets for workflow modeling [24]. Therefore, the statements for Petri nets are valid for workflow nets as well.

In Figure 1, a workflow net containing an exclusive split after activity *A* is visualized. Data is not part of this model.

BPMN 2.0. Currently BPMN 2.0 is planned to succeed the version 1.2, see [14]. In comparison to version 1.2, BPMN 2.0 evolves with new model elements, diagram types, and model execution semantics. In essence, BPMN is an expressive graph-based modeling notation. The graph nodes correspond to process elements, e.g., events, activities, and gateways. Graph edges represent various object relations, where control flow is the most important one among the rest. Those graph nodes that are related by

the control flow relation are referenced as flow objects, while other nodes as non-flow objects. Flow objects include events, activities, and gateways, data objects are non-flow objects.

Data objects can be associated to the flow objects, where an association indicates that the flow object accesses the data object. While undirected associations capture only the fact of data access, directed associations show, whether the data object was read or written. A data object may have distinguished states, evolving through the process. Along with graphical associations, BPMN allows to specify data access through activity attributes *InputSet* and *OutputSet*. In the context of BPMN data modeling capabilities, it is relevant to mention the message exchange mechanism: different organizations communicate with each other via message exchange visualized via explicit message objects. Data objects can be associated with sequence flow or message flow edges. However, such a modeling method can be seen rather as a shortcut: the data object is the output of the edge's source node and the input of the edge's target node.

Additionally, BPMN 2.0 introduces the concept of data object collections, organizing similar data objects into groups. Further, to show that a data object is delivered by an external resource or is consumed by it, a data object is declared as a process input or output. BPMN 2.0 also introduces *data store* as an instrument for process data persistence. Generally, data store represent anything where data can be gathered or read from, e.g. paper, folder, database. In information systems, a data store is usually represented by a database.

Figure 2 presents the capabilities of BPMN 2.0 with respect to data modeling. Date objects *Data 1* and *Data 2* are read and written by different activities and passed along between participants. As the objects traverse activities, their states are updated. The message passing between two participants is modeled explicitly with the envelope associated with the message flow relation. Additionally, the model contains a *database* element persisting information relevant to the communication of *Organization 1* and *Organization 2* on the side of *Organization 2*.

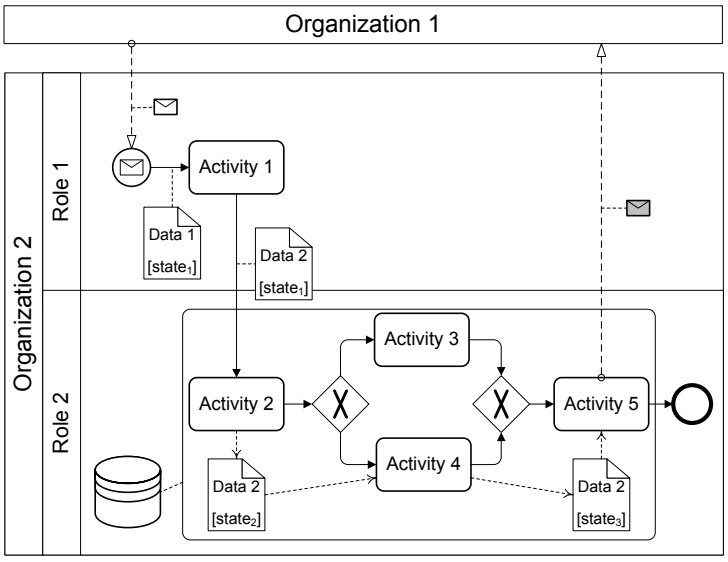


Figure 2: BPMN 2.0

Summarized, the role of data modeling increased in BPMN 2.0 and reached of data flow in BPMN 2.0. However, BPMN remains the business process modeling notation with a focus on control flow.

BPEL. WS-BPEL, Web Services Business Process Execution Language, has been introduced by IBM, BEA Systems, Microsoft, Siebel Systems, and SAP and is the current de-facto standard for web-service-based execution of business processes. The current version 2 has been introduced in [12] and focuses on activities, services linked to these activities, and their order visualized by an XML structure. Data flow is modeled implicitly only through variables contained in globally visible data containers, which are shared among the participants of the process. However, the variables represent in- and output messages of the activities. The exchange of specific data objects and the manipulation of variables may be handled through BPEL's *assign* statement as presented exemplarily in Listing 1.

```
1 <assign>
2  <copy>
3   <from container="c1" part="partX"/>
4   <to container="c2" part="partY"/>
5  </copy>
6 </assign>
```

Listing 1: Exchange of data elements in BPEL

Besides this data object passing, there are no possibilities to handle data. This includes specifications of transformations and modifications of data objects. Therefore, BPEL itself is very activity-centric and control flow focused with data flow playing a very minor role. However, Habich et al. introduced a data aware extension to BPEL in [3]. They utilize so-called Data-Grey-Box Web Services [2], which are web services enhanced with an explicit data aspect specifying how the specific data (object) is to be accessed. Additionally, they introduced a new link type to connect services from the data perspective. Altogether, the authors propose an orthogonal extension to the control flow concept: a separate data flow layer playing a supporting role in process modeling. These extension increases BPEL's data capabilities. However, the level remains on a level of BPMN 1.2 and below BPMN 2.0.

2.3 Discussion of Data-driven Approach

The data-driven approach does mainly focus on data objects and information and tries to visualize the flow of data and existing pre- and post-requirements which restrict the manipulation of single data objects with respect towards the overall business goal, the specific business process shall reach. In this section, the business artifact approach as well as the Corepro framework are discussed. In the first mentioned approach, data objects are named business artifacts around which all other business process information is centered. In Corepro, data objects are not the only driving factor, but the combination of data and control flow allows business process execution.

Besides these two modeling approaches, there exist also works dealing with data in a pre step. For instance, the product-based workflow approach [16] discovers optimal execution orders of activities primary based on data requirements and secondary based on the costs and time the execution of a specific activity consumes. Based on this information, a workflow net model is derived and presents the executable model. Following, approaches of this kind are not considered as these are no real business process modeling approaches trying to replace activity-driven approaches, but preparation steps to derive a business process model.

Business Artifacts. Business artifacts is a business process modeling approach that "shift[s] the focus ... from the actions taken to the entities that are acted upon" [1]. Initially introduced by Nigam and Caswell in [11], the approach has been discussed in a series of papers and thoroughly formalized by Bhattachary et al. in [1]. The formalization identifies business artifacts, schemata, services, and business rules as the main concepts of the approach.

Business artifacts are information entities capturing business process goals and enabling judgment of the goal accomplishment. Business artifact examples are order and invoice in the order to cash process. According to the formalization developed in [1], a business artifact is a fusion of two models: informational model and life cycle model. The informational model describes the artifact properties relevant to the process. The life cycle model defines the artifact states and allowed state transitions. The life cycle states correspond to high-level states on the path towards reaching the business process goal. While the informational model is formalized with a database schema, the life cycle model can be defined as a finite state machine or a Petri net. A collection of related business artifacts leading to the same business goal is called a schema.

Services correspond to activities in such notations as BPMN or EPCs (event-driven process chains). A service acts on business artifacts manipulating the content of artifact's informational model and changing the artifact life cycle state. Business rules determine the use cases and conditions which need to appear for allowing a service to access a business artifact.

Based on the aforementioned components, process execution does not follow a predefined order but depends on the availability of business artifacts in a specific state or business artifacts holding certain information. Hence, this approach utilizes data as the first class modeling artifact which alone drives the business process execution.

Notice that the visual notation for business artifact modeling is not settled. In this report, an example using the notation used in [1] is presented, see Figure 3. The figure presents the model of one artifact with seven attributes, constituting the informational model, and three life cycle states: *state1*, *state2*, and *state3*. The transitions between states are realized by means of activities *A1* and *A2*.

Corepro. Corepro is a framework providing an approach for enacting and changing data-driven process structures. It is presented in [9] and [10]. The core idea is to automatically create data-driven process structures which form one out of two bases to automatically create the control-flow strongly influenced by data dependencies. Alto-

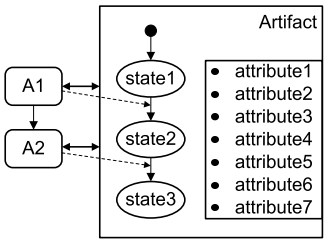


Figure 3: Business artifact example based on [1]

gether, this is a five-step-approach. First, available data objects and generally existing states of these data objects are determined on the data side. Besides, the process definition needs to be created and possible process states identified. Second, all data objects are put into an hierarchical structure before the object life cycle for each data object is created. In step four, the outcomes of the two preceding steps are utilized and the dependencies between the data states of the single data objects are determined. Lastly, this data-based structure is combined with the control-flow information leading to rules which are used for the aforementioned automatic data-driven control-flow.

Following these steps, Corepro also allows process adaptation during runtime and automatically translates the modifications made to data structures into the control-flow. These modifications include but are not limited to changes like adding or deleting data objects, changing relations of data objects, or adding external state transitions. For not yet activated elements of the process, the authors provide simple rules for the adaptation. For modifications of already started instances, specific correctness constraints are formulated and need to be fulfilled for allowing the intended adaptation.

Summarized, this approach closely relates control- and data-flow structures and the process execution is strongly influenced by data structures.

2.4 Discussion of Communication-driven Approach

The communication-driven approach works with activity- as well as data-driven approaches. It focuses on interactions between all process participants and the business goal is finally reached by a series of these interactions or communications. Therefore, communications are the primary modeling construct. Following, proclets as one representative are introduced.

Proclets. Proclets are leight-weight processes which communicate via structered messages, so-called performatives, trough channels with each other. They have been introduced in [21]. Each of these processes focus on the behavior of one specific case instead of overloading single processes with information from several cases. The then missing connections between dependent process chunks are introduced into the overall model via the aforementioned channels. Proclets itself are the specification of the general process and contain a knowledge base comprising information about previous interactions, i.e. a life cyle. Therefore, each proclet instance has a state. The working procedure of the communication channels seems to be adapted from what is known

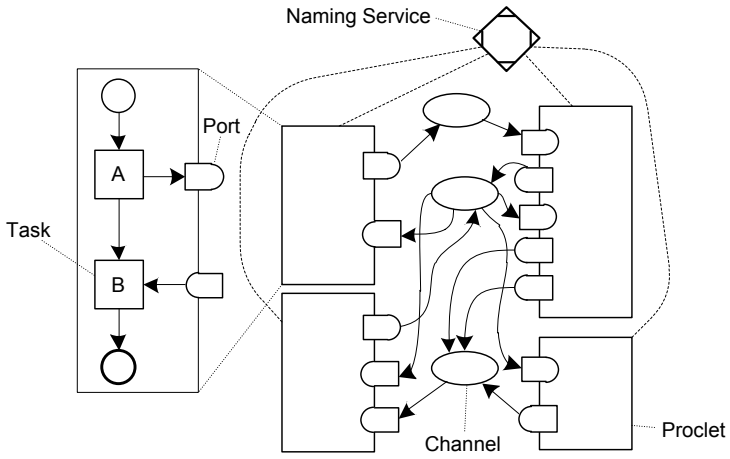


Figure 4: Proclets framework based on [21]

from the internet protocols. For instance, point-to-point and broadcast communications are allowed, different reliability channels exist comparable to the tcp/ip and udp protocols, and security and priority settings can be set to allow for encoded or real-time communication for example. Additionally, a naming service ensures that the proclets are able to find each other and to establish the communication. This naming service works on basis of identifiers.

Summarized, proclets move the main focus from pure control-flow to communication and interaction between case-based process chunks, which are supposed to be control-flow oriented. However, the processes connected by the communication channels might be of any focus. Figure 4 presents the framework of proclets, including all relevant modeling artifacts: Proclet, channel, naming service, port, and task.

2.5 Conclusion of Approaches Discussion

Independently from the type of notation and approach, data awareness is existing or currently approaching in notations which are still under ongoing development. From the presented notations, only workflow nets are not capable of dealing with data information which is based on the fact that they are an one-to-one mapping from Petri nets. Therefore, workflow nets will not change in the future. In all others, data plays a role of increasing importance. These notations offer various opportunities to model data flow and dependencies, although the activity-driven approaches do not visualize the data flow explicitly. The step of deriving it needs to be performed, which is also automatable. Besides data awareness, the different notations do have further strengths and therefore, based on the actual field of application the appropriate notation needs to be chosen. Data alone is not a sufficient indicator. For instance, the activity-driven approaches may the pick of choice regarding documentation of business processes. Data-driven approaches are more applicable in areas which require flexible process redesign, probably even during process execution, or fields of knowledge intensive processes.

3 Dependencies between Business Process Models and Data Object Life Cycles

The field of integrating data object life cycles and business processes has been initiated by Wahler [19, 20, 25]. She introduced approaches for deriving data object life cycles from business process models in activity diagram notation [19] and vice versa [7]. In the same context, she introduced the notions of *object life cycle conformance*, *object life cycle coverage*, and *life cycle compliance* which allow syntactic checks on the relation between a business process model and the appropriate data object life cycle. In fact, these notions check whether all transitions or activities respectively and all states have a counter part in the other model type. For instance, if a data object life cycle and a business process model fulfill the requirements for conformance and coverage, both models are identical from an execution point of view.

Generally, there are four main types of views on a business process model with respect to abstraction: the *management view*, the *technical view*, the *data-based view*, and the *implementation view*. The relationship between these types is visualized in Figure 5. Relationships between the very high-level management view and a rather detailed but still control-flow-oriented technical view can be highlighted by the means of behavioural profiles [27]. The mapping from the data-based view to the implementation is tackled by a current master's thesis of Ole Eckermann, the way back is currently not tackled. The linkage of the technical view with the data flow probably defined by master data management experts is currently a cumbersome and manual task and lacks a common understanding from the process side (technical view) and the data side. In this field, I like to tackle open issues based on Wahler's work and I like to determine opportunities to harmonize both worlds influenced by the achievements of the current data-driven business process modeling approaches discussed in Subsection 2.3.

These issues comprise for instance the verification of process instances based on the notions of data object life cycle conformance and coverage and providing information and correction proposals to align both representations based on environmental information. Further open issues concentrate around the fields of aggregations of data object life cycles of different data objects utilizing Wahler's synchronization points and data object aggregations and connected to these two fields also concurrent access to data objects or rather chunks of data objects.

4 Conclusion

This paper presents a discussion on the role of data in different business process modeling notations which belong to the set of activity-driven, data-driven, or communication-driven business process modeling approaches. Generally, data awareness is given in all discussed notations which were updated during this century or are currently in the update process. Especially the activity-driven approaches do not provide explicit data flow representation, but it might be derived from the models automatically. Therefore, the choice for a specific modeling notation is less dependent on data requirements, but more with respect to further requirements like the need to flexibly change process

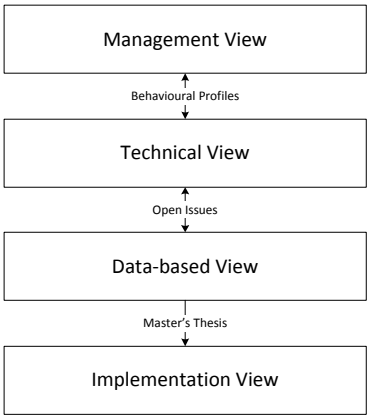


Figure 5: Views on business process models

structures at runtime or the intention to document current process structures within the organization.

Additionally, insights regarding the interplay of business processes and data object life cycles have been presented with respect to further improvements in this field of application.

References

[1] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. *Business Process Management*, pages 288–304, 2007.

[2] D. Habich, S. Richly, and M. Grasselt. Data-grey-boxweb services in data-centric environments. In *IEEE International Conference on Web Services, 2007. ICWS 2007*, pages 976–983, 2007.

[3] D. Habich, S. Richly, S. Preissler, M. Grasselt, W. Lehner, and A. Maier. BPEL-DT - Data-Aware Extension of BPEL to Support Data-Intensive Service Applications. *Emerging Web Services Technology*, 2:111–128.

[4] K. Jensen. Coloured petri nets. *Petri nets: central models and their properties*, pages 248–299, 1987.

[5] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods, and practical use*. Springer Verlag, 1996.

[6] S. Kumaran, R. Liu, and F. Wu. On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In *Advanced Information Systems Engineering*, pages 32–47. Springer, 2008.

[7] J. Küster, K. Ryndina, and H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. *Business Process Management*, pages 165–181, 2007.

[8] D. Müller. Management datengetriebener Prozessstrukturen. 2009.

[9] D. Müller, M. Reichert, and J. Herbst. Flexibility of data-driven process structures. In *Business Process Management Workshops*, pages 181–192. Springer, 2006.

[10] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *Advanced Information Systems Engineering*, pages 48–63. Springer, 2008.

[11] A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[12] OASIS. Web Services Business Process Execution Language Version 2.0, April 2007.

[13] OMG. Business Process Modeling Notation (BPMN) Version 1.2, January 2009.

[14] OMG. Business Process Model and Notation (BPMN) Version 2.0, June 2010. draft, revised submission.

[15] C.A. Petri. *Kommunikation mit Automaten*. Rhein.-Westfäl. Inst. f. Instrumentelle Mathematik an der Univ. Bonn, 1962.

[16] H.A. Reijers, S. Limam, and W.M.P. Van Der Aalst. Product-Based Workflow Design. *Journal of Management Information Systems*, 20(1):229–262, 2003.

[17] C. Richardson. Warning: Don't Assume Your Business Processes Use Master Data, 2010.

[18] N.C. Russell. Foundations of Process-Aware Information Systems. 2007.

[19] K. Ryndina, J. Küster, and H. Gall. Consistency of Business Process Models and Object Life Cycles. *Models in Software Engineering*, pages 80–90, 2007.

[20] Ks. Ryndina, J.M. Küster, and H. Gall. Consistency of Business Process Models and Object Life Cycles. In *MoDELS Workshops*, volume 4364 of *LNCS*, pages 80–90. Springer, 2006.

[21] W. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer. Workflow modeling using proclets. In *Cooperative Information Systems*, pages 198–209. Springer, 2000.

[22] W.M.P. van der Aalst. *Time coloured Petri nets and their application to logistics*. Springer-Verlag.

[23] WMP Van der Aalst. Putting high-level Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[24] W.M.P. Van der Aalst et al. The application of Petri nets to workflow management. *Journal of Circuits Systems and Computers*, 8:21–66, 1998.

[25] K. WAHLER. *A Framework for Integrated Process and Object Life Cycle Modeling*. PhD thesis, IBM Research, 2009.

[26] M. Weidlich, G. Decker, A. Großkopf, and M. Weske. BPEL to BPMN: the myth of a straight-forward mapping. *On the Move to Meaningful Internet Systems: OTM 2008*, pages 265–282, 2008.

[27] M. Weidlich, M. Weske, and J. Mendling. Change propagation in process models using behavioural profiles. In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pages 33–40. IEEE, 2009.

Semantics Detection for Data Quality Web Services

Tobias Vogel
tobias.vogel@hpi.uni-potsdam.de

Data quality is an essential key factor for economical success. It is a set of properties of data, such as completeness, accessibility, relevancy, and consistency in presentation which also includes the absence of multiple representations for same real world objects. To avoid those duplicates, there is a wide range of commercial products and customized self-coded software. These programs can be quite expensive in both, adoption and maintenance which can be a barrier for small and medium-sized companies to afford these tools. We present a novel approach, whose main characteristics are (1) minimal user interaction required and (2) self-adaption to the provided input data. Similarity measures are assigned to the provided records' attributes and a duplicate detection process is carried out. We incorporate a novel matching approach, called 1:k Mapping to classify the provided data, which is a prerequisite for properly assigning similarity measures.

1 The Need for Data Quality

Data are often captured by humans. Ignoring the correct spelling, insufficient audio quality on a phone line, stress, typos, or encoding issues are severe problems for the overall correctness of data in CRM systems filled by employees. There are validation tools that check data for soundness to some extent, e.g., syntactical correctness of telephone numbers or existence of postal addresses. However, it is unlikely – at least with a reasonable confidence – to tell correct and wrong spellings of names apart or to check a person's occupation.

As a result, the affected company will face problems of, e.g., finding the correct record when trying to solve a postal delivery issue. Furthermore, it is impossible to calculate key performance indicators correctly, such as the correct number of unique customers or the average revenue per customer. Also, the expenses for advertisement mailings are unnecessarily high due to sending different shipments to the same customer. And last but not least, the customer satisfaction suffers when clients are bothered with unnecessary mailings.

There are approaches to manage these deficiencies by being more tolerant in searching: similarity measures are introduced that relax strict comparisons by assigning a real number (typically between 0 and 1) to a pair of elements, instead. This number represents the certainty that both elements represent the same real-world entity. This makes it possible to execute fuzzy queries. The challenge is to develop good

similarity measures and to apply them to the corresponding attributes in the data to be cleaned, which is an assignment problem. Combining the different measures and selecting promising pairs of elements in the provided datasets is called *Duplicate Detection*.

Applications to perform duplicate detection are offered by database vendors, third-party software, and consultant companies. However, in general, the effort is not only of *financial* nature, i.e., in buying and maintaining the software. Often, there are many parameters to be set by domain experts to tweak the applications and to maximize the effectiveness of the batch runs, hence a *manual* effort. While large companies are able to pay for this essential process, other companies, which have the same need for high-quality, duplicate-free data, are not able to pay consultants and maintenance for a once-in-a-month-run.

Web Services[1] are an appropriate model for such on-demand invocation styles. They offer flexible and scalable processing powers and are often provided with pay-as-you-go cost models.

Providers for duplicate detection services only offer webpages, where humans can upload and download their database dumps manually. While this is cumbersome and time-consuming, the extract and load procedures are not part of the service. Manual effort has also to be put into assigning data types/semantics to the uploaded data as shown in Figure 1.

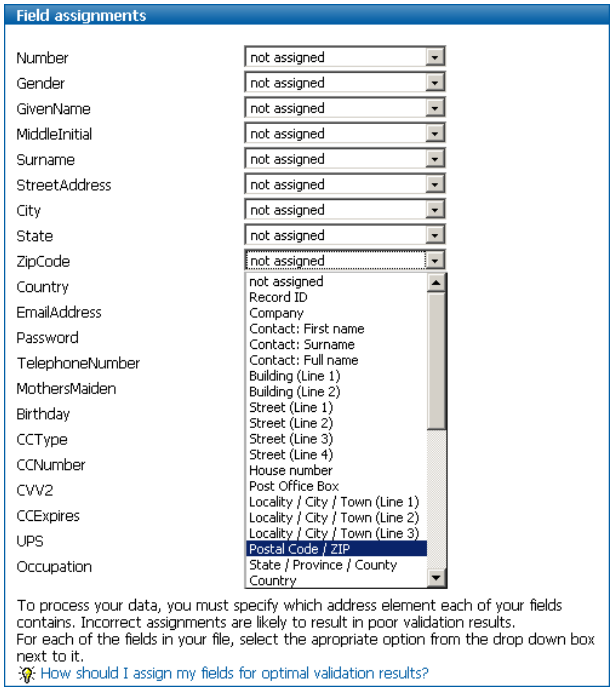


Figure 1: Screenshot of assigning semantics to attributes of uploaded input data (`addressdoctor.com`)

[1]We define a Web Service as a piece of software that serves a well-defined purpose, is typically invoked over a network, and is called by other programs rather than human users.

The conditions for different duplicate detection tasks are manifold, depending on the amount of (meta) information, this tasks bases on. For example, if the data is not in relational format, it is unclear, *which* attributes to compare, whereas if the data type or semantics are unknown, it is hard to decide, *how* to compare different attributes. We approach these challenges towards a service-based duplicate detection technique without human interaction, that is integrated into the DAQS (**Da**ta **Q**uality as a **S**ervice) project.

In this paper, we propose

- the novel 1:k mapping problem between columns and similarity measures,
- a technique to classify semantics of columns and describe an extended version of the Hungarian Algorithm to solve the 1:k mapping problem, and
- an evaluation on different datasets showing the feasibility and usefulness of our approach.

2 Workflow

The duplicate detection service works in three phases, as illustrated in Figure 2.

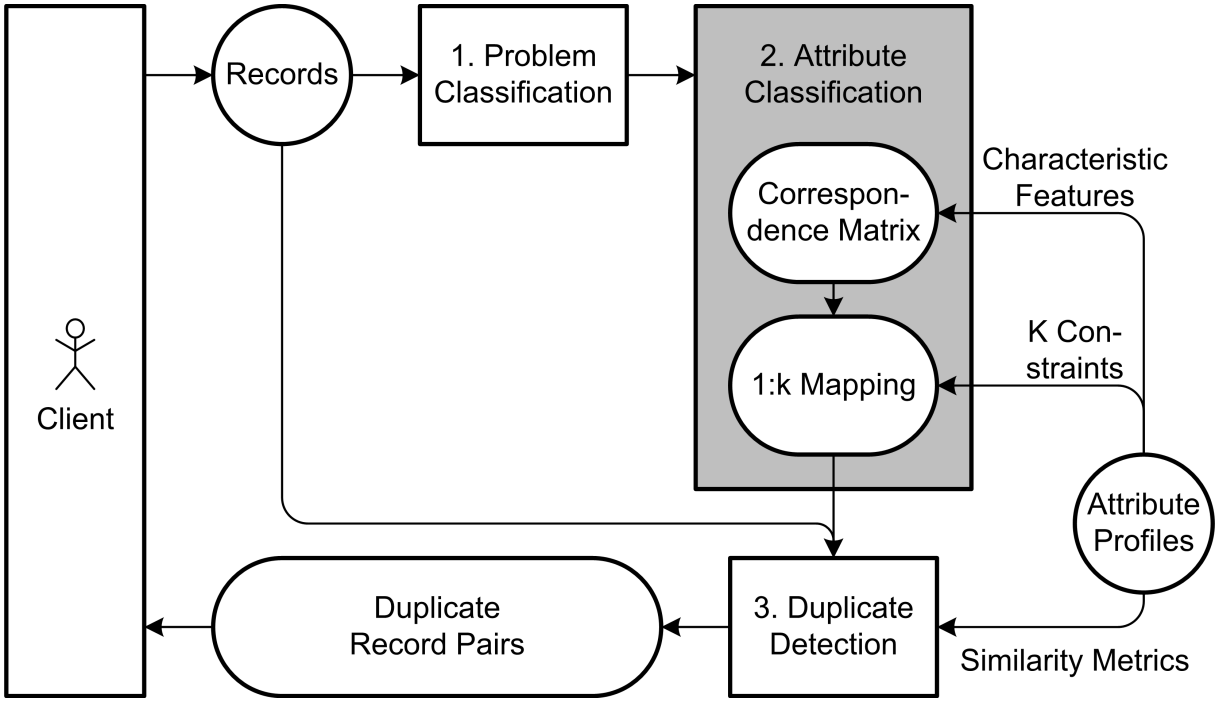


Figure 2: Workflow and architecture of a duplicate detection service (phases are in boxes)

In the *problem classification phase*, the provided data are analyzed to find out the format and how to treat them during further processing [13], e.g., whether information

retrieval techniques have to be applied, whether schema information are present, etc. This is not part of this paper. In the adjacent *attribute classification phase*, for each attribute a corresponding similarity measure is found automatically using the 1:k mapping technique. This phase is the focus of this paper and is explained in Section 3. The *duplicate detection phase* contains the actual duplicate detection logic and is not in this paper's focus.

For the remainder of the paper we will assume that we have no datatype information but attribute names and instances. We further assume that the mapping is clear, i.e., it is known which attribute from one tuple to compare with which attribute from another tuple. However, it is not clear, *how* to compare these attributes; we want to find this out. Finally, we assume that we can differentiate between several attributes and have separators (e.g., semicolons in CSV files).

3 Attribute Classification

The goal of the classification is to assign appropriate (highly specialized) similarity measures to the attributes of the input data. This is facilitated by first assigning semantics to these attributes and then derive similarity measures from those semantics. E.g., two instances of a *given name* would be compared differently than two *email addresses*, still all values being of data type `String` (or `VARCHAR`). Besides, maiden names and family names have the same characteristics, telephone and fax numbers are indistinguishable, which implies that a too detailed semantics detection is both, not feasible and not of much help, since it would result in inferring the same similarity measure. In the following, we will stick to the term *classification* to describe this assignment. Each different semantics is called *class*.

Classification is done with the help of feature vectors (c.f. next section). These vectors are created for *training data* whose classes are known and the data whose classes have to be found out, called *test data*. For classification, the test data is compared to the training data through machine learning techniques. The better the features match, the higher is the confidence that the piece of test data is of the same class as the corresponding piece of training data. Figure 3 describes the classification process in general whereas the following sections will provide more details on the classification.

3.1 Features

We define a set of boolean features, which are applied on each single attribute value, thus creating single *feature vectors*. We use the heuristic that there is a high probability that feature vectors for values from the same attribute look similar.

There are three different types of features (1. and 2. implemented predominantly with regular expressions):

1. 73 *single character features* check for existence of letters, digits or some special characters, e.g., “a”, “A”, “4”, “#”, or “@”.

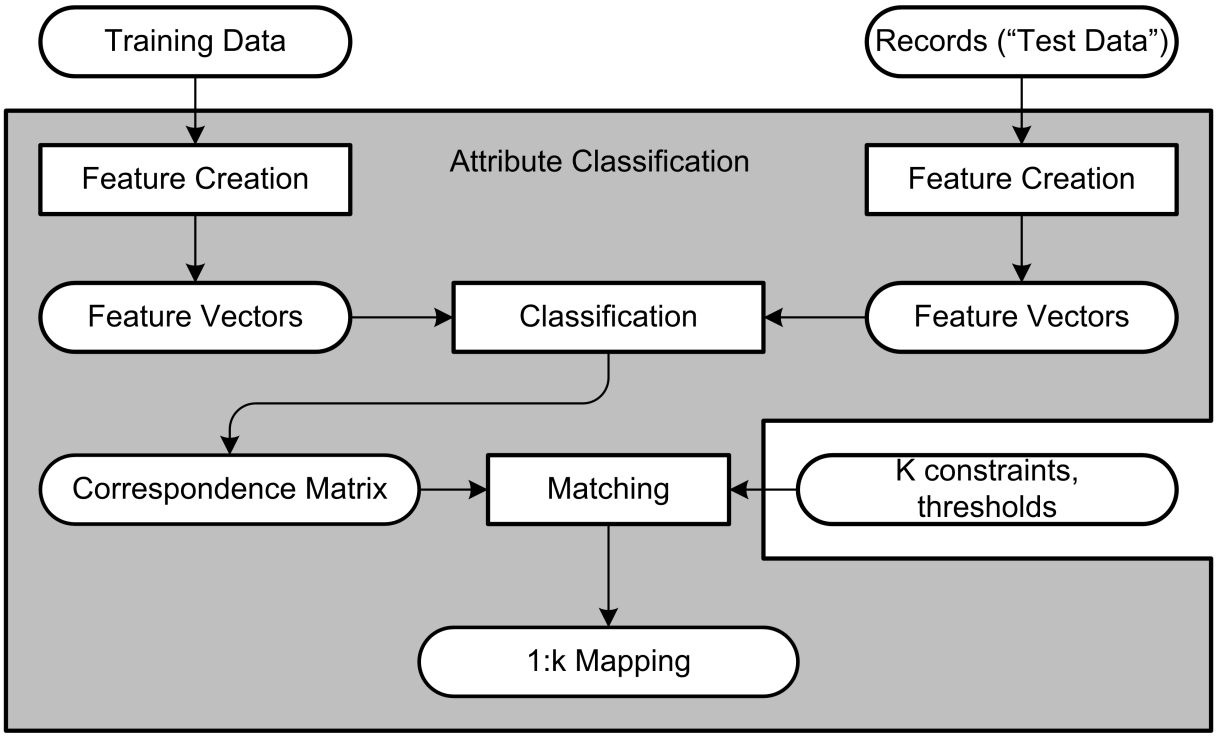


Figure 3: Classification Process Details

2. About 20 *multiple character features* check for more advanced patterns, e.g., whether a string begins with a lowercase letter (`[\\p{L}&&[^\\p{Lu}]].*`), contains a separated 4-digit number (`(^|.*\\D)\\d{4}(\\D.*|$)`), or has a length between 20 and 29.

3. 2 *lookup features* use different webpages related to names.[2]

3.2 Correspondence Matrix

Once both attribute sets are represented by feature vectors, test data (the set of *source attributes*) can be classified based on the training data (*target attributes*). With classification, we mean the assignment of the most similar target attribute, the attribute of an overall global schema, to each source attribute.

We use the Naïve Bayes classifier of Weka [4]. This classification results in a *correspondence matrix*, c.f. Figure 3. Since each source attribute is to some extent similar to each target attribute, the correspondance matrix contains many elements that are greater than zero. For an example, see Table 1.

The challenge is to select a mapping from this correspondence matrix that represents the most realistic classification. Based on this matrix, it is not trivial to derive a decision, which source attribute to match to which target attribute. The set of these matching decisions is called the mapping, hence a mapping problem has to be solved.

[2]Among others: DBpedia (`http://dbpedia.org/`), NameWiki (`http://wiki.name.com/`)

Source\Target	Firstname	Lastname	Phone	Address
Fullname	0.8	0.6	0.1	0.2
Telephone	0.0	0.0	0.9	0.2
Street	0.2	0.4	0.1	0.9
House Number	0.0	0.0	0.7	0.7
ZIP	0.0	0.0	0.5	0.3

Table 1: Correspondence Matrix for Source (First Column) and Target (First Row) Attributes with Illustrative yet Sound Values

The correspondence matrix contains n source attributes and m target attributes. A simple 1:1 mapping would assign each source attribute to one single target attribute as long as there are free target attributes left. A downside of this approach is, that there have to be enough matching partners (i.e., $n > m$). Moreover, each source attribute is forcefully matched, even if there is no correct matching partner. To solve this, an 1:1 matching with a threshold is possible. Correspondences below this threshold would not be taken in the final mapping. However, we want to allow different source attributes to map to the same target attribute, e.g., “telephone”, “mobile phone”, and “fax” to “telephone”, which is still not possible. Therefore, we could allow a 1:n mapping, having “telephone” participating several times.

While it is possible that a tuple contains several attributes similar to telephone numbers (i.e., fax numbers) it is very improbable that there are several birthdays in a tuple. The birthday class should only be able to take part once in the final mapping, while a given name might appear several times. The knowledge about target attributes comprises also information about how often such an attribute may appear in the source schema and thus, defines the number of times a source attribute can be matched (“K Constraints” in Figure 2). Therefore, we propose the *1:k Mapping* which is basically a 1:n mapping, but with changing n for each target attribute.

3.3 1:k Mapping

Assume an acyclic, directed, bipartite Graph $G = (S, T, E)$ with two sets of nodes (source and target elements) $s_i \in S, i = 1, \ldots, n$ and $t_j \in T, j = 1, \ldots, m$ respectively as well as a set of edges $e_{ij} \in E$ where $|E| = n \cdot m$. Such graph is depicted in Figure 4 where all edges resemble E. Further assume a correspondence matrix C with entries c_{ij} quantifying the similarity between source element s_i and target element t_j (c.f. Table 1). Assume also a set of k constraints $k_j \in K$, $k_\cdot \in \mathbb{N}$, $|K| = m$.

Find a mapping M with elements

$$m_{ij} = \begin{cases} 0 & \text{if } e_{ij} \text{ takes not part in the mapping} \\ 1 & \text{if } e_{ij} \text{ takes part in the mapping} \end{cases}$$

and

$$\sum_{i=1}^{n} m_{ij} < k_j \quad (\forall j = 1, \ldots, m)$$

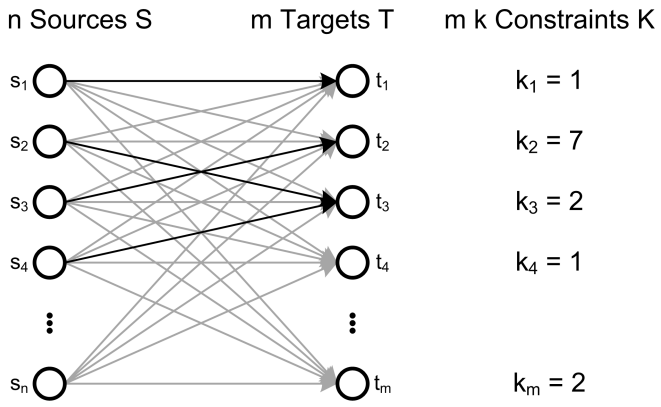


Figure 4: Sample 1:k mapping with given K (only black edges)

and maximize the overall similarity of the selected participating elements in the mapping.

$$\max\left(\sum_{\forall ij} c_{ij} \cdot m_{ij}\right)$$

The intended mapping is illustrated in Figure 4, where n source attributes are matched to m target attributes and each source attribute n_i is matched at most once. s_n, for example, is not matched. Each target attribute t_j is matched at most the number of times, the corresponding k_j allows to.

The final mapping M is calculated using a global matching algorithm gm on the correspondence matrix ($M = gm(C)$). This mapping task can be solved with an extended version of the Hungarian Algorithm [6] (ha). The original Hungarian Algorithm solves the assignment problem [10], which does not allow the multi-mappings of multiple source elements to the same target element.

The k constraints are incorporated by duplicating columns of C. The value of k_j determines the number of duplications of the j-th column. Note that this column duplication requires $\sum_{k_j \in K} k_j = a$ additional rows. The resulting matrix is called C' having n rows and $m' = m + a$ columns.

The Hungarian Algorithm requires a squared matrix and thus, $|n - m'|$ additional rows or columns representing non-existing source or target elements have to be added to the matrix for padding. Only one, rows or columns, will be added. Without loss of generality, this results into a new set of rows ($\bar{n}$) and columns ($\bar{m}$) with $\bar{n} = \bar{m} = max(n, m')$, $n \leq \bar{n}$ and $m' \leq \bar{m}$, and entries $c''_{\bar{i}\bar{j}} = 0 \quad \forall n < \bar{i} \leq \bar{n}, m' < \bar{j} \leq \bar{m}$. Consequently, the resulting matrix is called C'', with all padding entries set to zero.

See Table 2 for an example C'' with $K = \{k_1 = 3, k_3 = 2, ...\}$. It does not matter, at which position the additional columns or rows are inserted.

With this squared correspondence matrix C'', the Hungarian Algorithm can be used to create a 1:k multi-mapping: $M' = ha(C'')$. However, the result has to be modified. Since the Hungarian Algorithm involves all source attributes into the mapping, even those m_{ij} for which $n < i \leq \bar{n}$ and the final mapping M shall have the original di-

Source \ Target	Firstname	Firstname	Firstname	Lastname	Phone	Phone	Address
Fullname	0.8	0.8	0.8	0.6	0.1	0.1	0.2
Telephone	0.0	0.0	0.0	0.0	0.9	0.9	0.2
Street	0.2	0.2	0.2	0.4	0.1	0.1	0.9
House Number	0.0	0.0	0.0	0.0	0.7	0.7	0.7
ZIP	0.0	0.0	0.0	0.0	0.5	0.5	0.3
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 2: Extended Correspondence Matrix, now squared

mensions $n \times m'$, M' has to be transformed into M. This is done by taking only those matches that do not have a padding partner. Therefore $M = \{m'_{ij} : 1 < i \leq n, 1 < j \leq m'\}$.

After the mapping M is finally created, the input records can be examined for duplicates using the similarity measures derived from the semantically classified attributes.

4 Evaluation

Different decisions in the described approach can be taken independently. They are given in the follwing.

Features The feature selection (c.f. Section 3.1) relies on basic and more advanced features. Usually, the classifier automatically exploits the more distinguishing and thus relevant features by itself, however, a higher number of features raises the chance of having "good" features within the feature vector. But still, features may be designed for fitting to expected classes and their respective characteristics, e.g., the "@" character for email addresses. Furthermore, feature development is always a trade-off between calculation time and distinction quality (e.g., for lookup features that post HTTP calls, but yield a relatively confident hint for a string being a name). The calculation time may be reduced by caching and parallelization but is worth being evaluated.

Featurization The featurization style has influence on the availability of other, promising features such as aggregate functions (e.g., average length, distinctness of the attribute values, standard deviation). Feature vectors can be created attribute value by attribute value or for the whole set or multi-set of attribute values.

Classifier There are plenty of classifiers which classify a given (test) feature vector against a set of other (trained) feature vectors (c.f. Section 3.2). Naïve Bayes does this with the help of conditional probabilities, but there are also classifiers which build up decision trees or use hyperplanes to separate the vector space (Support Vector Machines). While Naïve Bayes is reasonably fast, other approaches may take longer and result in better classification quality. The trade-off

also has to incorporate the fact that a mapping process is performed afterwards which might reduce the requirement of a highly precise classification.

Mapping Having performed the classification, the correspondance matrix has to be interpreted and the mapping has to be inferred. The presented extended Hungarian Algorithm is a sophisticated approach to enable the 1:k mapping. Faster algorithms such as a greedy or majority vote algorithm are possible using the same extension mechanism described in Section 3.3.

In the following, preliminary results are shown for using all of the presented features, single-value featurization, Naïve Bayes classification and the extended Hungarian Algorithm. German-language address datasets are mostly taken from publicly available sources (`http://fakenamegenerator.com`, `deutschland-api.de`, `dbpedia.org`) and confidential corporate datasets. 500 rows are used for training as well as test datasets. The training dataset comprises 18 address-related classes from very specific ones (gender, homepage, familyname) to very general ones (number, date). Test data has always German attribute names, training data has always English attribute names.

Source	Target	correct?	confidence
plz	ZIP	yes	1.0
telefon	HOUSENUMBER	no	0.967
id	NUMBER	yes	0.822
strasse	STREET	yes	0.664
nachname	FAMILYNAME	yes	0.582
vorname	GIVENNAME	yes	0.475
ort	CITY	yes	0.457
datum	BIRTHDAY	yes	0.078

Table 3: Results for classifying a semi-synthetic corporate address dataset against the global dataset

Table 3 shows the results for matching the corporate dataset against the global dataset. 7 of 8 attributes are matched correctly. The “telefon” and the “datum” attributes are only filled sparsely, resulting in wrong or very unconfident matches. When applying a threshold on the mapping, the match between “datum” and birthday would very likely be eliminated.

Table 4 shows the results for matching a person dataset with politicians against the global dataset. 6 of 8 attributes are matched correctly. Pecularities are the presence of a URL field, which is perfectly matched assumedly due to the very feature checking for the string containing “http” and the “zusatz” field, containing very few city names. The “jobs” field is confounded with the street class. Both have similar characteristics (long strings without numbers). A wrong classification does not necessarily lead to a reduced quality in the adjacent duplicate detection process. Instead, occupation and street would end in an akin similarity measure, the street similarity measure might

Source	Target	correct?	confidence
url	HOMEPAGE	yes	1.0
geboren_am	BIRTHDAY	yes	0.982
zusatz	HOUSENUMBER	no	0.916
id	NUMBER	yes	0.804
nachname	FAMILYNAME	yes	0.583
vorname	GIVENNAME	yes	0.485
jobs	STREET	no	0.335
geboren_ort	CITY	yes	0.104

Table 4: Results for classifying a politician address dataset against the global dataset

contain additional rules to expand common abbreviations such as “str.” to “straße”. Moreover, the matching has a relatively low confidence and might be removed by a threshold, anyway.

5 Related Work

There is a large body of prior work both for duplicate detection and for the matching of different schemas.

Duplicate Detection consists of two separate fields, similarity measures that are described, e.g., by Navarro [9] and Elmagarmid et al. [2] on the one side and algorithms to select promising comparison pairs, e.g., the sorted neighborhood method by Monge and Elkan [5] on the other side. An overview on both can be found in Naumann and Weis [8].

Schema Matching is the technique of creating and selecting correspondances between two sets of elements, typically attributes of relations. Rahm and Bernstein give a survey on different methods for schema matching [11]. Another matching approach, which inspired this paper, was Naumann et al.’s classification algorithm [7]. It uses a rich feature set to create an instance-based mapping between two schemas. Instance mappings are used by iFuice [12], where knowledge about explicit connections between different schemas is exploited. However, in the use case of customer data, those hyperlink connections will not appear.

Bilke and Naumann [1] combine both fields of research and utilize known duplicates for schema matching. This paper does the opposite and uses schema matching to eventually improve duplicate detection.

Faruquie et al. [3] present a data cleansing service with an optional duplicate detection component. However, proper thresholds for the pairwise attribute comparison part of the duplicate detection have to be selected manually. Furthermore, they concentrate on argumenting for data cleansing in general and omit details about how to actually perform the duplicate detection. They present different proposals for how to transfer the data to the service provider.

There are also some existing web applications that offer data cleansing. Mostly,

this especially comprises data verification and enrichment. However, AddressDoctor[3], AdressExpert[4] and Uniserv[5] are commercial offers that comprise duplicate detection.

6 Summary and Roadmap

Duplicate detection is a crucial part of data cleansing. There are many applications for industry-scale duplicate detection, but they are not appropriate for small and medium businesses which need a continuous, once-per-month duplicate detection run. Services are a promising approach for these companies, but the existing offers are limited and cumbersome regarding to the grade of automation.

The presented approach addresses these deficiencies. Input datasets are analyzed and if there are no semantics given for the data – which is the general case – they are classified against prepared, domain-specific training data. An extended version of the hungarian algorithm solves the 1:k mapping problem for this classification and a mapping is created. Predefined similarity measures are then applied on the different attributes.

However, the classification and mapping process (c.f. phase 2 in Section 2) is not yet integrated in the overall duplicate detection workflow. Before deriving similarity measures and actual detecting duplicates, the classification and mapping process has to be improved (e.g., by better handling sparse attributes) and thorougly evaluated as described in Section 4.

Furthermore, the specific characteristics of a data quality service have to be exploited more. The test data of different customers could help enlarging the number of classes the service can tell apart. Thus, a feedback system is required. This data can only be used when being sufficiently privacy preserving. Moreover, different attributes might not be independent from each other, e.g., when a street is present, it is likely that also a house number attribute is contained in the customer's dataset, or a month implies the existence of a day and year specification. Another open point is the selection of k in the 1:k mapping. This can also be learned when analyzing several customers' datasets.

References

[1] Alexander Bilke and Felix Naumann. Schema matching using duplicates. Proceedings of the 21st International Conference on Data Engineering, 2005.

[2] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge & Data Engineering*, 19, 2007.

[3] http://www.addressdoctor.com/
[4] http://www.adressexpert.de/
[5] http://www.uniserv.com

[3] Tanveer Faruquie, Hima Prasad, Venkata Subramaniam, Mukesh Mohania, Girish Venkatachaliah, Shrinivas Kulkarni, and Pramit Basu. Data cleansing as a transient service. In *Proceedings of the ICDE*, 2010.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[5] Alvaro Monge and Charles Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD workshop on data mining and knowledge discovery*, May 1997.

[6] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[7] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura M. Haas, and Nimrod Megiddo. Attribute classification using feature analysis. In *ICDE*, page 271, 2002.

[8] Felix Naumann and Melanie Weis. *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers, 2010.

[9] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 1999.

[10] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[11] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[12] Andreas Thor. *Automatische Mapping-Verarbeitung von Web-Daten*. Dissertation, Institut für Informatik, Universität Leipzig, 2007.

[13] Tobias Vogel. Self-adaptive data quality web services. In Wolf-Tilo Balke and Christoph Lofi *, editors, *Proceedings of the 22nd Workshop "Grundlagen von Datenbanken 2010"*, volume 22, Technische UniversitÃd't Braunschweig, 38106 Braunschweig, Germany, May 2010. Institute for Information Systems.

A Shared Platform for the Analysis of Virtual Team Collaboration

Thomas Kowark
thomas.kowark@hpi.uni-potsdam.de

The traces of digital collaboration have been shown to provide indicators for beneficial or detrimental developments in project teams and, accordingly, the success of the corresponding projects. Often such observations are made in controlled settings using relatively small sample sizes. This is a drawback, as it prevents a generalization of the findings. Increasing the sample size, however, is often not feasible due to constraints in time, capacity, or funding.

In this report, we outline a Software-as-a-Service solution that tries to overcome this limitation by providing an easily accessible service for the analysis of collaboration behavior that, simultaneously, uses this data to test deduced assumptions against a continuously increasing database of team collaboration structures. In addition to presenting the high-level architecture of the proposed system, we give an overview of case studies that are conducted with the help of this system and identify implementation aspects that will be part of future research.

1 Introduction

Collaboration and communication are key enablers for the success of project teams. Especially in early phases of projects spreading knowledge amongst team members and developing efficient collaboration processes is crucial for later project success [3]. Collaboration, in that sense, comprises all activities that contribute to sharing project-relevant knowledge between project members. Ideally, these activities are predominantly verbal and non-verbal face-to-face interactions, but to a certain degree they are manifested in digital artifacts created with the help of groupware tools (e.g., email, Wikis, or version control systems).

Previous studies have shown that digital collaboration artifacts, even though they represent only a small portion of overall collaboration activity, contain indicators for possible project success [12]. A limitation of the cited studies is their relatively small sample size. The indicators found during those case studies are initially significant only within the evaluated environment. Generalizing such findings is merely possible if a much broader database comprised of projects in different, yet comparable, settings is available. This additional research work is, in most cases, not feasible to be performed by a single institution.

In this report, we present an approach to overcome this limitation by means of a Software-as-a-Service (SaaS) solution. The platform is named “analyzeD” and enables

researchers to upload, anonymize, and analyze digital collaboration artifacts of their respective case studies. By that, they benefit from the capabilities of the software for performing their own research, yet simultaneously contribute to a database of digital collaboration data. This database can, in turn, be used to verify assumptions made in only small sets of projects by trying to detect similar collaboration patterns in other projects.

The remainder of this report outlines the current state of the system and gives a short summary of previous applications. The main focus, however, is a discussion of aspects that need to be considered during the implementation. In particular, scalability, data privacy, and a formalized approach to compare different collaboration structures are topics that require close attention in the future. The report concludes with a presentation of case studies that use the platform for data evaluation and at the same time serve as test cases for further development activities.

2 Prior Work

Starting point for the development was a thorough analysis of the d.store platform [13]. It uses Semantic Web technologies to represent concepts of collaboration artifacts, such as emails or wiki pages, as ontologies. The concepts are linked through associations, e.g. a person is linked to an email by being its sender. They are also time-annotated and, accordingly, can be put in correlation with project timelines.

Ontologies define which kind of data can be uploaded to the platform. Gathered artifacts are combined into so-called team collaboration networks, which provide a unified representation of the digital collaboration happening within project teams. Data analysis is possible through visual navigation clients or a SPARQL query interface.

Initially, the platform was used to analyze the collaboration behavior of engineering design teams in a controlled classroom setting. This setting enabled researchers, for example, to deduce and test more effective project management models [11]. Building on the obvious relevance of analyzing computational collaboration artifacts in engineering design teams, a logical next step was the application in different setups, such as software engineering projects. Therefore, ontologies representing concepts of groupware tools common in software engineering (e.g., version control systems, bug tracker, or code metrics) were developed along with services for collecting and uploading the corresponding data [5].

An initial application of the platform in such a setting was performed during the course of a software engineering curriculum. 80 students of a university lecture were required to develop a single system in a joint effort. The overall development team was further split into 13 sub-teams based on responsibilities for certain aspects of the system. Various cross-cutting concerns and mutual dependencies between some teams required a high degree of collaboration.

Even though the setting did not allow for statistically significant assertions about possible detrimental or beneficial digital collaboration patterns, interesting insights could be drawn from the project. For example, the usage patterns of bug trackers and version control systems in conjunction with the project timeline provided indicators for prob-

lems with both systems. Interestingly, the corresponding usage patterns indicated the problems two weeks before they were mentioned by the students. A comprehensive discussion of the project setting and the findings of the initial case study was recently accepted for publication [6].

Besides revealing interesting collaboration patterns even during the first application of the platform in a software engineering project, more conclusions could be drawn during the lecture. On a technical level, scalability with regards to networks sizes turned out to be an issue that requires close attention. The intention to capture all available aspects of collaboration artifacts and the heavy usage of inference rules created performance bottlenecks that prevented ad-hoc analysis of the data. Furthermore, a strictly technical query interface was considered to be insufficient for intuitive data analysis.

On a meta level, the first case study elucidated that data gathered during such installments is unsuitable for deriving generalizable assertions about the significance of certain collaboration patterns for project success or failure. It became apparent that, in order to create statistical significance, means had to be created to broaden the database for such analysis without being forced to continuously perform case studies like the one previously mentioned.

As a consequence, development efforts regarding extensions of the d.store platform were moved towards creating an SaaS solution that not only allows to capture and analyze collaboration data for single projects, but enables automated comparisons with different, yet comparable, projects. In order to achieve this goal, the aforementioned problems regarding scalability and data analysis need to be solved, since without fulfilling such vital requirements, acceptance of the platform as a research tool is unlikely. Additionally, new issues pushed to the fore: Such an approach also requires simple handling of data privacy issues, for example by specification of obfuscation rules. Also a sound formalization of the comparison of different collaboration structures is inevitable.

3 Related Work

Monitoring and analyzing digital collaboration activities to deduce beneficial or detrimental developments within project teams has been subject to prior research.

An approach that aggregates data from different data sources was developed by Ohira et. al [8]. The Empirical Project Monitor relies on numerous feeder applications that parse data sources like source code management systems, bug trackers, or email archives. It provides a number of preset visualizations for this data. Additionally, an underlying communication model tries to detect flaws within the collaboration behavior based on empirical studies. The main difference to this approach is the creation process of the communication model used for deviation detection as it is not generated from an ever increasing database of collaboration data but statically implemented within the application.

Reiner [9] presented a proposal for a knowledge modeling framework that supports the collaboration of design teams. Communication information has been deduced from explicit interactions between members of a design team by a software tool that he

developed to provide a prototypical implementation for the proposed framework. This research laid the early foundation for our work but did not address generalization of the findings made during single projects by validating them in an automated manner against other gathered data.

Microsoft's Team Foundation Server [7] is a commercial collaboration tool suite that allows its users to analyze and compare the collaboration behavior of the teams using this platform. It is, however, limited to collaboration activities that are performed using one of the provided tools. If, for example, different version control, bug tracking, or email systems are used, the corresponding activities cannot be analyzed.

Beyond the analysis of digital collaboration traces through means of IT, E-Research systems are also an interesting field of related work. Approaches like the one presented by Abidi et. al [2] enable researchers to jointly work on research topics by distributing data collection and evaluation activities. Since all involved personas are using a shared system, collected data and evaluation results are available to each of them. However, such a system was, to our knowledge, not developed and applied in the area of virtual team collaboration analysis.

4 Platform Development and Application

In the following, we present the next steps of our research. They comprise activities regarding the development of the proposed SaaS solution, as well as planning and execution of case studies in the fields of software engineering and engineering design.

4.1 A Software-as-a-Service Approach to Virtual Collaboration Analysis

Software development efforts will be concentrated on implementing the analyzeD architecture (see Figure 1). The d.store will be an integral part of this software, but by adding a variety of improvements and services to the initial landscape, we will simplify setup and access especially for technically inexperienced users. In order to achieve those improvements in usability, the following extensions are necessary and and will be implemented:

- A central service for configuration and control of feeder services. This web-application serves as a service repository, i.e., it lists all available sensor clients for the supported data sources. Additionally, it provides a single point of access to enter access data, specify data obfuscation rules and upload the corresponding digital artifacts to a distinguished d.store instance.

- A visual query interface. Instead of being forced to use a rather technical query language to analyze the gathered data, this service will provide an interface that allows creating queries by selecting the attributes of interest and defining basic outline parameters for the search.

- A visualization service that is able to generate a variety of diagrams for query results in a generic manner. Insights gained during an earlier sub-project about a similar topic will be used to shorten the development time for this particular application [10].

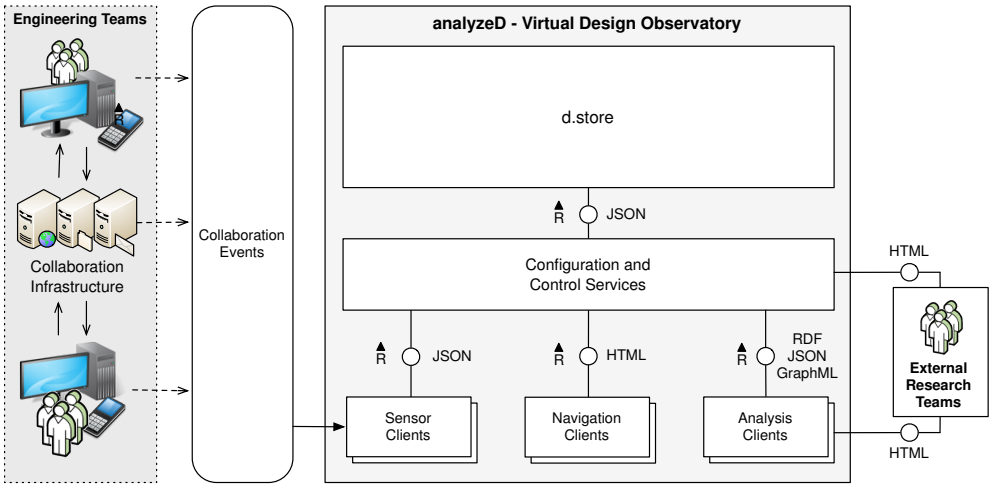


Figure 1: Architecture overview of the proposed analyzeD platform.

The central configuration service was implemented prototypically, but certain aspects like the service repository require further testing and implementation effort. Furthermore, actions towards increased scalability of the service were taken. In a first step the persistence infrastructure was moved to AllegroGraph[1], a native RDF triple store implementation. Along with a shift from extensive usage of inference rules towards explicit data representation this prevented execution times for node insertion to grow exponentially with the number of nodes present within a team collaboration network. Since related work indicates that columnar in-memory databases are also a viable option for storing RDF data [1], next steps will include a sample implementation using state-of-the-art in-memory column databases.

4.2 Case Studies

In order to test the implementation of the platform in different scenarios, two case studies will be performed during the course of the next academic year.

Software Engineering Building on experiences gained during the first case study in a software engineering lecture (see Section 2), the platform will be used again in such a setting. Contrary to the first installment, the exercise will feature two development teams that compete with each other in developing a customer relationship management system. Both teams are split into eight sub-teams and equipped with the same

[1] `http://franz.com/agraph/allegrograph/`

groupware tools. By that, comparability of the respective development teams is greatly enhanced, and the the case study provides us with the possibility to perform similarity analysis for the resulting team collaboration networks.

Additionally, first contacts have been established to other universities that are eager to implement a comparable version of the lecture in their own curriculum and use the analyzeD platform in order to analyze the resulting collaboration networks.

Engineering Design As part of the HPI-Stanford Design Thinking Research program, scientists from the Center for Design Research at Stanford university apply the platform in an engineering design context. In order to support this project, various steps will be performed to enable support for the analysis of CAD application usage. Initial investigation of the logging behavior of tools such as AutoCAD have already taken place and corresponding log file parsers are in development. The logs contain very detailed information about the working steps performed during a session. Thus, the definition of a corresponding ontology representing the identified concepts should contain the following elements:

- A class representing a CAD model with attributes such as a textual description, a set of documents belonging to the model, and an array of changes representing the evolution of the model.
- A class representing a change of the model. This class contains information about the author of the change, a detailed list of the activities performed, a (computed) score for the severity of the change, and a rating for the current iteration of the model (e.g., efficiency scores, weight-stability ratio, etc.).

Previous approaches to capture the workflow of CAD designers have struggled with a lack of semantics in the rare sequence of steps contained within the logs [4]. By adding the additional information about change severity and a rating for the respective iteration to our model, we are able to not only analyze what is being done, but also how it is being done and how successful certain styles of working are. The metrics for those measurements will be developed in close collaboration with CAD-designers of our prospective partner companies, as well as engineers of CAD software vendors. A test bed is provided by the CAD installations at Stanford University Product Realization Laboratory, part of Mechanical Engineering. Additionally, we are planning to collaborate with Autodesk and/or SolidWorks, primary supplier of CAD solutions such as AutoCAD as well as with numerous companies currently using AutoCAD.

Outlook The long-term objective of the project is to create a viable means for analyzing projects in a variety of settings based on previously gathered data. This will create a project management dashboard that indicates if certain developments within the digital collaboration structures of projects have proven to be beneficial or detrimental in the past. In order to achieve this goal in a meaningful manner, collaboration data captured in real industry projects is essential as it lacks most of the artificial bias that is intrinsic to classroom projects. Therefore, we are currently investigating possibilities

to gain access to data and participants of historical projects, as well as integrating the platform into newly started ones.

5 Summary

In this report, we have presented a vision-of and first steps-towards a platform for the analysis of digital collaboration activity that aggregates knowledge gathered during independent case studies to verify the general validity of assumptions about potentially beneficial or detrimental collaboration patterns. This is achieved by providing a flexible, easily accessible, and scalable service for the analysis of digital collaboration behavior for end users that monitors the resulting anonymized team collaboration networks for similarities. By that, lessons learned during one project can be propagated to members of completely unrelated projects just by a similarity analysis of their respective collaboration activities. We believe that such a system can greatly aid researchers in various fields, as it allows them to access a database of comparable projects, which they could not have created by themselves in reasonable amounts of time, effort, and cost.

References

[1] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18(2):385–406, 2009.

[2] Syed SR Abidi, Ashraf Abusharek, Ali Daniyal, Mei Kuan, Farrukh Mehdi, Samina Abidi, Faisal Abbas, Philip Yeo, Farhan Jamal, and Reza Fathzadeh. A service oriented e-research platform for ocean knowledge management. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 32–39, Washington, DC, USA, 2010. IEEE Computer Society.

[3] Victor R. Basili and Marvin V. Zelkowitz. Analyzing medium-scale software development. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 116–123, Piscataway, NJ, USA, 1978. IEEE Press.

[4] Andrea Casotto, A. Richard Newton, and Alberto Sangiovanni-Vincentelli. Design management based on design traces. In *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 136–141, New York, NY, USA, 1990. ACM.

[5] Thomas Kowark. Towards a service landscape for a real-time project manager dashboard. Technical report, Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering, 2010.

[6] Thomas Kowark, Jürgen Müller, Stephan Müller, and Alexander Zeier. An educational testbed for the computational analysis of collaboration in early stages of

software development processes. In *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS)*, January 2011.

[7] Microsoft. http://msdn.microsoft.com/en-us/vstudio/ff637362, Visual Studio Team Foundation Server 2010, 2011.

[8] Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, Michael Barker, and Koji Torii. Empirical project monitor: A system for managing software development projects in real time. In *International Symposium on Empirical Software Engineering*, Redondo Beach, USA, 2004.

[9] Kurt A. Reiner. *A framework for knowledge capture and a study of development metrics in collaborative engineering design*. PhD thesis, Stanford, CA, USA, 2006. Adviser-Leifer, Larry J.

[10] Victor Saar. Visualisierung temporaler Eigenschaften im Kommunikationsverhalten global verteilter Entwicklungsteams. Master's thesis, Hasso Plattner Institute, Enterprise Platform and Integration Concepts., 2009.

[11] Philipp Skogstad. *A Unified Innovation Process Model for Engineering Designers and Managers*. PhD thesis, Stanford University, Center for Design Research, 2009.

[12] Matthias Uflacker, Philipp Skogstad, Alexander Zeier, and Larry Leifer. Analysis of virtual design collaboration with team communication networks. In *Proceedings of the 17th International Conference on Engineering Design (ICED'09), Vol. 8*, pages 275–286, 2009.

[13] Matthias Uflacker and Alexander Zeier. A platform for the temporal evaluation of team communication in distributed design environments. In *CSCWD '09: Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design*, pages 338–343, Washington, DC, USA, 2009. IEEE Computer Society.

Programming Models for Parallel Heterogeneous Computing

Frank Feinbube
Frank.Feinbube@hpi.uni-potsdam.de

This paper describes my work with the Operating Systems and Middleware group for the HPI Research School on "Service-Oriented Systems Engineering".

1 Motivation

Many of the computational problems we are facing today are complex and need huge computational power to be solved. It is well-known that processors will not continue to get faster, but will get more cores instead. More cores are not only harder to utilize by an application programmer, but also challenge hardware designers. Thus various new hardware architectures are designed and evaluated in order to find the ones that will fulfill the needs of future computer systems. Prototypic configuration boards like Intels Single Chip Cloud Computer (SCC) are an attempt to deal with the ever-increasing number of cores by removing essential features of current processors like hardware cache coherency. Another approach is to accompany common general purpose CPUs with sophisticated special purpose processing units. These so called Accelerators are easier to build and very fast for specific application purposes. They are the foundation for the new trend of hybrid computer systems.

2 Domains of Hybrid Systems

2.1 High Performance Computing

The High Performance Computing (HPC) community has a steady need for an increase of floating point operations per second (FLOPS) in order to simulate natural processes with a higher accuracy. Therefore enormous computer clusters are used. These supercomputers are not only very expensive to build, but do also need a huge amount of energy, both for processing and for cooling.

A popular approach to gain more FLOPS while reducing power consumption is to use GPU computing devices like NVIDIAs Tesla modules. Some of the best-performing 500 HPC clusters already contain a great amount of these devices. They have good cost per FLOPS ratio and like Cell processors, field-programmable gate arrays (FPGA) and other special compute accelerators are very energy efficient. They are quite adequate for this application domain and it can be expected that many of the next generation supercomputers will be heterogeneous systems facilitating GPU compute devices.

2.2 Business Servers

Due to ever-increasing application of RFID-technology and clever networks of software systems, business warehouses collect more and more data every day. These data sinks are gold mines for business intelligence. In order to exploit useful knowledge about customers and processes, a big machinery for data mining and information extraction is needed. The performance needs for such analysis applications are far beyond the ones of day-to-day online analytical processing (OLAP) query processors. On the other hand these systems do not have to be as highly available as the business critical online transaction processing (OLTP) servers. Thus more cost efficient components can be used. In order to accompany their mainframes with fast and cost efficient OLAP processing, IBM presented their IBM Accelerator solution at the IBM System z University event. The idea is to build a heterogeneous computing systems consisting of a standard mainframe and an additional server of several blades for very fast, but less reliable analytic operations on cloned databases.

Another trend is to use special processors to speed up various performance critical and regularly needed algorithms like (de)compression, XML parsing, encryption and decryption, and regular expression matching. These so called accelerators are special hardware devices designed to run sophisticated calculations at high speed. GPU compute devices can be regarded as another special kind of accelerators. There are also successful applications of GPU computing in the domain of business intelligence. [4, 6, 9] Hybrid systems consisting of reliable CPUs supported by various accelerators are expected to be the future configuration of servers.

2.3 Desktop Computers

Even in the domain of Desktop computers heterogeneity has become common. The success of netbooks has confirmed that many end users are interested in cheap and mobile computers. Since these systems are so energy efficient, they also support the always-on mentality of these days. The trade-off for low power consumptions is a less powerful CPU. In contrast to this, many computers are used as entertainment devices and thus require a good performance, especially to decode and display music and video streams. To speed up these activities Atom processors are accompanied by on-die-GPUs like the Intel GMA 3150 GPU or NVIDIAs ION GPU. While these accelerators are applied for the specific application of video processing, they also can be used to speed up other classes of algorithms.

That medium-class and high-end computers can not only be used for computer games and high definition entertainment purposes, but also for realistic simulations and elaborate calculations has been proven by various BOINC projects (e.g. Folding@HOME). The parallel algorithms used in these projects have to exploit distributed parallel heterogeneous systems and thus present a great challenge to the programmer. On the other hand they are very elegant because they make use of the idle time of the participants computers that would otherwise be wasted.

2.4 Mobile and Embedded Systems

Mobile devices like cell phones have severe restrictions on power consumption because they are only powered by batteries. On the other hand as the success of iPhones demonstrates, users like fancy user interfaces, smooth visualization and entertaining applications. To support this functionality many current phones have a powerful processor and some co-processors. The upcoming smart phones conforming to the Windows 7 Phone specification will even contain a DirectX 9 compatible graphics processor. These processors may also be used for tasks that are not related to computer graphics. Some applications have shown how GPU compute devices can be used to support embedded systems liker routers [9] and home entertainment devices [10].

As for the other application domains mobile and embedded systems take advantage from hybrid approaches because they provide high special purpose performance while consuming only a small amount of power.

3 Programming Models

"This is the year when applications developed on GPU computing go into production." said NVIDIA CEO Jen-Hsun Huang. While most accelerators for hybrid systems are still at a prototypic stage, GPU computing has already achieved a variety of success stories. Especially calculation-intensive scientific applications seem to fit onto GPU computing devices very well. They were used to speed-up seismic exploration, weather modeling, computer vision, and medical imaging. The military applies GPU computing for advanced image processing and electromagnetic simulations. It is also used for business intelligence, complex event processing, speech recognition, engineering modeling, and analysis solutions.

In order to get a deep understanding of the programming model for GPU computing, we build a prototype that solved the NQueens puzzle for large board sizes on GPU compute devices. We learned that the CUDA programming model relies on SPMD kernels that work on a complex memory hierarchy in parallel. While some fundamental concepts like blocks, threads, and shared memory are made explicit, others like coalescing, occupancy, and local memory are hidden to the programmer. Understanding the explicit and implicit characteristics of a CUDA kernel execution is essential to predict the performance of a CUDA application. We presented our experiences at the ISPDC'2010 (section 5.1).

At the Summer School of the Universal Parallel Computing Research Center (UPCRC Illinois) (section 5.2) an overview about the various programming models for parallel shared memory systems was presented. The topics covered task-based approaches like Intels Threading Building Blocks (TBB), parallel for-loops like OpenMP, lambda expressions, vectorization principles, as well as GPU Computing with OpenCL.

OpenCL [1] is a standard introduced by the Khronos Group - a consortium of CPU and GPU hardware vendors. OpenCL is a programming model that allows to write programs for CPUs, GPUs and Cell Broadband Engine Architecture (CBEA) processors. It shares the concept of kernels and memory hierarchy of CUDA. In addition OpenCL

introduces the concept of streams. The OpenCL API is more low level than CUDA.

Powerful programming models and APIs like CUDA and OpenCL allow time efficient reformulation of multi-threaded code for CPUs for GPU computing. While this means little effort to execute parallel general purpose algorithms on GPUs, these will not utilize the GPU hardware well. Kirk et al. describe the situation this way: "If the application includes what we call data parallelism, it is often a simple task to achieve a 10x speedup with just a few hours of work. For anything beyond that, we invite you to keep reading!" [3]

We created a survey on best practices for optimizing GPU computing applications in order to really benefit from the acceleration GPU hardware can offer. This survey will be published in a special issue of IEEE:Software Journal in 2011 (section 5.3).

Although GPU computing is a mature accelerator category, it is still hard to write GPU computing code, even harder to utilize the GPU appropriately. Even the highly experienced developers of the National Supercomputer Center in Shenzhen were only able to reach 43 % of the theoretical peak performance for their hybrid supercomputer Nebulea. Nebulea is number two in the top 500 list of supercomputers and consists of NVIDIA C2050 GPUs. The utilization of 43% was reached for a dedicated implementation of the embarrassingly parallel LINPACK algorithm. [5]

Hybrid computing and heterogeneous computing need good programming models and tool support to overcome these difficulties and enable developers to benefit from these new system architectures. The industry is also interested in recommendations for hardware changes. Coming up with good programming models and tools for hybrid systems is hard. This is highlighted by the fact that the well-established sector of parallel and multi-core computing is still looking for appropriate programming models and empowering developer tools.

4 Research Plan

The area of my research is on parallel hybrid systems and accelerator technologies. I aim to help developers to handle the complexity of such system w.r.t. to coding experience and resulting application performance. Consequently, my focus is on parallel languages, parallel libraries, and parallel toolkits for hybrid systems.

As high-lighted again at the UPCRC (section 5.2) appropriate tools and programming models for parallel and multi-core computing are still ongoing research topics. In the field of hybrid systems, programming models and tools have not only to cope with parallelism, but also with differing execution characteristics of the processors and accelerators in a given system configuration. State-of-the-art programming is done with CUDA and OpenCL, which extend the C++ language. Especially OpenCL is a very low-level interface and thus laborious to work with. There are first approaches to reduce the burden for the programmer. Lee et al. [11] show that it is possible to use the OpenMP-API for GPU computing. Most of the other approaches are simple wrappers for higher-level languages. The problem with the current approaches is that they do not map onto the accelerators hardware very well, are only suitable for very special subset of problems, or lead to severe code bloat. This is where I want to work at.

In order to accomplish my research goal, the following tasks have to be completed:

1. Identify best practices and patterns for multi-core development and hybrid computing. (section 5.3)

2. Identify hardware capabilities and restrictions of hybrid architectures.

3. Identify common use cases and algorithms for hybrid computing. (section 5.4, section 5.5)

4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms. (first steps in section 5.6)

5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Until now I worked mainly on milestones one to three. In order to realize step four, I will start with a selection of examples (section 5.5) and evaluate how each of them can be applied to a particular architecture. The next step is to apply the optimizations described in section 5.3 manually and learn which optimizations that can be applied automatically. The findings will than be formalized and provided as a .NET library.

5 Recent Activities

5.1 Paper presentation at the 9th International Symposium on Parallel and Distributed Computing (ISPDC)

I presented my experiences with the NQueens problem and CUDA at the 9th International Symposium on Parallel and Distributed Computing in Istanbul, Turkey in July 2010. [8]

The conference serves as a forum for engineers and researches and covers topics from parallel to distributed computing. The keynotes were held by D. Keyes and Wolfgang Gentzsch. They highlighted the current and future challenges for the high performance computing (HPC) community.

The focus of the first session was GPU computing. Our presentation started with an overview of the CUDA programming model. The NQueens problem was introduced and our parallelization approach was described. The main focus was the application of various optimizations onto our solution in order to achieve a better utilization of the card. These optimization led to contrary performance implications on the two available CUDA-enabled card generations of NVIDIA.

The other sessions discussed models and algorithms, multi-cores, Web Services and Multi-Agent Systems, Interconnection Topologies, Networks and Distributed Systems, Grids and P2P Systems, and Scientific Programming.

5.2 Visiting the UPCRC Summer School

I was visiting the Summer School of the Universal Parallel Computing Research Center (UPCRC Illinois). It is a joint research endeavor of the Department of Computer Science, the Department of Electrical and Computer Engineering, and corporate partners Microsoft and Intel. It aims to pioneer and promote parallel computing research and education.

The school started with an introduction on shared memory parallelism and multicore technology by UPCRC Co-Director Marc Snir. UPCRC Principal Investigator Danny Dig presented parallelism techniques for object-oriented languages and how refactoring can be applied to transform sequential applications into concurrent ones. Clay Breshears and Paul Peterson from Intel illustrated how OpenMP and Threading Building Blocks can be used for parallelizations. In addition they introduced cutting-edge developer tools by Intel: the Intel Parallel Inspector, the Intel Parallel Amplifier and the Intel Parallel Advisor. Phil Pennington and James Rapp from Microsoft presented the C++ Concurrency Runtime and the .NET Task Parallel Library (TPL). María Garzarán gave an overview on vectorization and described various techniques to apply them. UPCRC Illinois Co-Director and PI for the world's first NVIDIA CUDA Center of Excellence Wen-mei W. Hwu introduced OpenCL. John E. Stone of the Illinois Beckman Institute illustrated CUDAs utility with a Electrostatic Potential Maps application. Marc Snir concluded the school with his Taxonomy of Parallel Programming Models. As a special final event we were visiting the Petascale Computing Facility at Illinois which will house the Blue Waters sustained-petaflop supercomputer.

Besides getting a lot of practical experiences with various wide-spread parallel programming tools and libraries, I got an overview on application classes and use case for parallel computing. In addition I learned about the challenging problems for the area of parallel computing: programming models that are easy to use and powerful in leveraging parallel platforms. I also learned about the hardware trends that will push the shift from parallel to heterogeneous computing and thus will increase the importance of good programming models and execution platforms.

5.3 Journal Paper for the IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

Modern graphic cards are able to act as additional compute device beside the processor. Parallel computing is therefore no longer a dedicated task for the CPU, the new trend is *heterogeneous computing* of main processor and graphics processing unit (GPU).

Our journal article presents a synthesis of important strategies for utilizing the additional graphic processor power. We explain the primary concepts of GPU hardware and the according programming principles. Based on this foundation, we discuss a collection of commonly agreed critical performance optimization strategies. These optimizations are the key factor for getting true scalability and performance improvements when moving from a multi-threaded to a GPU-enhanced version of your application.

It will be published in a special issue of IEEE:Software Journal in 2011.

5.4 Knowledge Sink for Use Cases, Tools and Libraries

While we were creating our survey on best practices (section 5.3), we collected a lot of literature about GPU computing and related topics. This collection is available at [7]. Besides general topics, there are special collections for use cases of GPU Computing, as well as a collection of tools and libraries.

5.5 Example Implementations of Representative Use Cases

The hands-on labs at the Summer School of the Universal Parallel Computing Research Center (section 5.2) included a variety of algorithms that can be executed on parallel platforms. These included matrix-matrix multiplication, convolution, prefix scan, quicksort, and minimum spanning tree. Starting with these, we created a collection of representative use cases for parallel computing. We used further literature to extend the collection. [3, 12] These use cases will be used to evaluate our new programming models.

5.6 Prototype to Run OpenCL-Code from .NET

The current tools and techniques to use the OpenCL API restrict programmers to write C++ like code. In order to provide a greater audience with easy access to GPU computing, Jan-Arne Sobania and I were cooperating to run OpenCL programs using the .NET Framework. We used a simple parallel loop approach that is known to .NET developers because the parallel for loop of Microsofts Task Parallel Library (TPL) has become a part of the .NET 4.0 Framework. Our prototypic implementation demonstrates that we can provide OpenCL access similar to the TPL via a .NET library. This way it is easy and intuitive for a .NET developer to benefit for GPU computing. Recently a similar approach for Java has been made available by ATI. [2]

Based on our work we want to evaluate which of the best practices for optimizations that are described in section 5.3 can be applied by our .NET library. We expect that type and meta data information available in the runtime will be useful for that approach. Ueng et al. [13] demonstrated that coalescing memory access - a very popular optimization - can be applied automatically. Some other best practices have potential for runtime support as well.

6 Conclusion

This paper introduces the research area of hybrid systems. Section 2 gives an overview of the application domains for hybrid computing systems. The High Performance Computing (HPC) always aims at more flops and smaller power consumptions. Business servers use Accelerators for for OLAP and specific application needs. Home computer need to be high-performing entertainment devices that consume very little energy. Power restrictions are more severe on mobile and embedded devices, but even in this sector applications become more and more resource intensive. Programming

models and appropriate developer tools for parallel and multi-core computer systems are active researched topics. Programming models and tools for the domain of heterogeneous and hybrid systems have not only to cope with parallelism, but also with differing execution characteristics of the processors and accelerators in a given system configuration. My research aims at help developers to handle the complexity of such system w.r.t. to coding experience and resulting application performance. I worked on surveys on best practices and patterns, hardware architectures and uses cases of hybrid computing. Currently I am working on a library for a high-level language to access OpenCL-enabled accelerators. With the help of this library I plan to apply selected best practice optimizations automatically. The usefulness will be demonstrated using the collection of use case examples I created. Section 5 provides an overview on my recent work.

References

[1] The OpenCL Specification - Version 1.1, 6 2010.

[2] Advanced Micro Devices, Inc. Aparapi. `http://developer.amd.com/zones/java/Pages/aparapi.aspx`.

[3] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 1 edition, 2 2010.

[4] P. B. Volk, D. Habich, and W. Lehner. GPU-Based Speculative Query Processing for Database Operations. In *Proceedings of First International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, ADMS, 10*, 9 2010.

[5] Jack Dongorra. Challenges for Exascale Computing. Technical report.

[6] Wenbin Fang, Bingsheng He, and Qiong Luo. Database Compression on Graphics Processors. In , 2010.

[7] Frank Feinbube. GPU Readings List. `http://www.dcl.hpi.uni-potsdam.de/research/gpureadings/`.

[8] Frank Feinbube, Bernhard Rabe, Martin von Löwis, and Andreas Polze. NQueens on CUDA: Optimization Issues. In *Proceedings of 9th International Symposium on Parallel and Distributed Computing, Istanbul, Turkey (ISPDC), 2010*, 2010. Istanbul, Turkey (to appear).

[9] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-accelerated software router. In *Proceedings of SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, New Delhi, India*, pages 195–206, New York, NY, USA, 2010. ACM. New Delhi, India.

[10] Taskin Kocak and Nicholas Hinitt. Exploiting the Power of GPUs for Multi-gigabit Wireless Baseband Processing. In *Proceedings of ISPDC '10: Proceedings of the 2010 Ninth International Symposium on Parallel and Distributed Computing*, pages 56–62, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. OpenMP to GPGPU: a compiler framework for automatic translation and optimization. In *Proceedings of PPoPP '09: Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming, Raleigh, NC, USA*, pages 101–110, New York, NY, USA, 2009. ACM. Raleigh, NC, USA.

[12] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, 7 2010.

[13] Sain-Zee Ueng, Melvin Lathara, Sara S. Baghsorkhi, and Wen-Mei W. Hwu. CUDA-Lite: Reducing GPU Programming Complexity. In *Proceedings of 21th International Workshop on Languages and Compilers for Parallel Computing, Edmonton, Canada (LCPC), 2008*, pages 1–15, Berlin, Heidelberg, 8 2008. Springer-Verlag. Edmonton, Canada.

A Study on Mobile Real-Time Middleware

Uwe Hentschel
uwe.hentschel@hpi.uni-potsdam.de

The Fontane project has shown that distributed systems connected via a mobile phone network become more and more important. This applies to devices of the everyday life as well as to industrial and medical devices. Mobile systems have characteristics like the available bandwidth which may vary strongly and make high demands on applications. In the context of the Fontane project exists the requirement to transfer a streaming electrocardiogram. This extends the demands on mobile systems to the area of real-time applications. An additional layer, the middleware, could be helpful to reduce demands on the application by concentrating often required functionality at a central point. But special characteristics of mobile systems are often not addressed by middleware typically used for distributed systems with wired connections [8, chapter 11].

Another point is if an application wants to adapt its behavior to the variable bandwidth of mobile systems the application have to know first the current value of available bandwidth. On the one side the direct measurement of the available bandwidth makes high demands on itself and influences the monitored mobile network. On the other hand on many mobile nodes like phones the network status is only monitored to be indicated to human users. This paper gives a short overview of the problem area and possible proposals to deal with this.

1 Introduction

The popularity of mobile devices, such as laptop computers or mobile phones, is unbroken within the last years. However, not only such typical mobile devices but also embedded or medical devices or hybrid forms become more and more part of the everyday life. The latter can especially be found in the areas of assisted living and vital signs monitoring of patients with special diseases like diabetes or cardiac insufficiency.

In the majority of cases these devices can be connected to wireless networks. Particularly within rural areas mobile phone networks provide the only opportunity for long distance communication. Applications using mobile connections have to deal with problems which are typical for this type of communication. For example:

- The network connectivity can temporarily and unexpectedly be lost.
- The available transmission bandwidth varies frequently and is typically lower than within stationary networks.
- The devices have often limited resources in terms of battery power, CPU operating speed or main memory.

Because the communication using mobile networks is one of the most power consuming tasks of mobile devices, they are usually involved in rather short sessions.

Another point of interest is the distributed structure of mobile systems. A patient monitoring system for example, contains typically measurement and control devices for each patient and central data storage devices, often called the telemedicine center, where medical professionals discover findings. Here mobile devices capture data and transmit this to a stationary central part. Another example are systems where autonomous units and a central unit are connected via a mobile network and the autonomous units get their commands from the the central unit. Such system structures are also called mobile nomadic systems [8, chapter 11] (Figure 1).

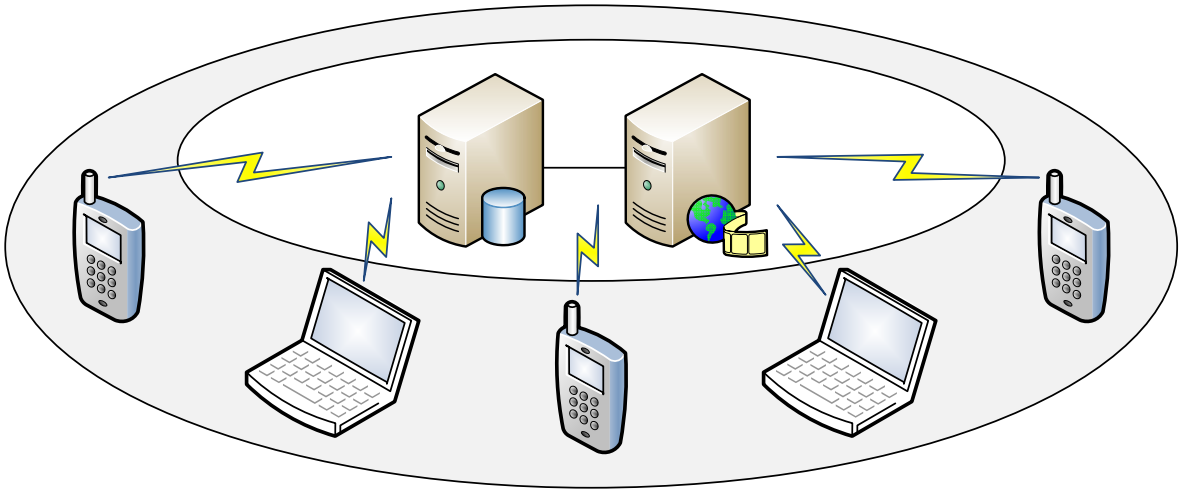


Figure 1: Mobile nomadic system

2 Middleware for mobile distributed systems

One main goal of middleware for distributed systems, such as CORBA or Java RMI, is to provide a higher level of abstraction for the application, hiding the complexity of the underlying network, protocols, and operating system. In other words the whole system shall look like a single entity and the distribution should be transparent for applications. In contrast, middleware for mobile distributed systems shall not hide the whole complexity of the underlying layers but establish awareness for mobility. This means that the application developer shall deal with some aspects of mobility, such as variable transmission bandwidth, possibly lost connectivity, or battery power.

This can be realized using two basic strategies. First, lower layers can provide parameters typical for mobile connections which allow the application to react to changes of the network. In this case the application itself is responsible for the mobile connection control. Second, an additional layer evaluates the lower layer parameters and provides special event functions what can be used by the application. The application does also not necessarily need to handle the aspects of mobility itself but the developer shall know that they exist and decide about the strategy to deal with.

Within the Fontane project, in cases of emergency a streaming electrocardiogram (ECG) shall be transmitted from the patient's mobile device to the telemedicine center. A two channel ECG with 512 samples/sec and 12 bit/sample resolution each channel requires a transmission bandwidth of about 12 kbps. Such a bandwidth cannot be guaranteed as an absolute value but in many cases as mean value within a specific time period. In general, if real-time application shall be used within a mobile distributed system, additional requirements have to be fulfilled by the network and the communication control. Typical real-time requirements are:

- The compliance with timing boundaries of given tasks, such as the transmission of a message or a part of the data.
- The predictability of specific operations.

Because of the expected unpredictable effects which influence the transmission parameters of the mobile distributed network a middleware for such networks cannot guarantee hard real-time requirements but the middleware can focus on providing soft real-time guaranties.

3 Effects which influence mobile communication

Todays mobile phone networks in Germany use the frequency ranges about 900 MHz and 1.8 GHz in the case of GSM (Global System for Mobile Communications) and about 2 GHz in the case of UMTS (Universal Mobile Telecommunications System). This section lists environmental conditions that are typically relevant for radio frequencies in the range above 100 MHz and discusses their influence on different communication layers.

3.1 Real world effects

The mobile communication between different parties can be influenced by following effects:

Geographic parameters: Geographic conditions like the distance between transmitter and receiver and their altitude are important because the waves are propagated quasi-optically.

Obstacles within the transmission path: Obstacles formed by buildings, elevations, vegetation, and animals cause shadowing effects as well as reflection, diffraction, scattering, and interferences of radio waves.

Atmospheric conditions: Precipitations, such as rain, snow, hail, and fog, attenuate the amplitude of electromagnetic waves. Electrostatic discharges in case of thunderstorm accompany with strong magnetic fields and are sources of electromagnetic disturbance. Inversions, arrangements of air in different layers with low temperature at the surface of the earth, diffract radio waves.

Radio interferences: Interferences caused by other radio services or multipath propagation make reclamation of the original signal on the receiver side difficult.

Architecture of the mobile system: The system architecture has an important influence on the quality of data transmission. Especially the design of the devices, and here especially the type and arrangement of antenna and the construction of enclosure, the positioning of radio frequency components (indoor, outdoor, ...), and the available or chosen transmission power have to be considered.

Radio cell resources: The limited resources of the radio cell itself can lead to a conflict between voice and data communication. Within a data communication the available transmission bandwidth is shared by all participants.

Radio cell change-over: In the case of a handover, such as inter-cell handover, normally the receive signal level and/or the signal quality within the new cell will be better than within the current cell [4]. But the transmission bandwidth available within the new cell can be different and perhaps lower than within the current cell.

3.2 Transmission layer

The effects that influence the transmission path outside the mobile network result in a variable receive signal level or signal quality. Indicators of signal quality are typically the carrier-to-noise ratio (CNR), signal-to-noise ratio (SNR), bit error rate (BER) and frame error rate (FER). The radio access network reacts typically on such influences by adapting its transmission parameters; the modulation and coding schema (MCS) for example, in the case of GPRS (General Packet Radio Service) or EGPRS (Enhanced GPRS) within a GSM network; or the radio access technology (RAT) in the case of an inter-RAT handover. This actions, on the other hand, influence the available bandwidth and the connection visible for the layer above.

3.3 Applications layer

The abstraction at applications level leads to the effect that the exact source which influences the receive signal cannot be identified. All above mentioned real world effects result in a variable transmission bandwidth or in the worst case in a lost connection.

Therefore it is useful not only to measure the currently available bandwidth but also to retrieve information from the transmission layer in order to get a better understanding of the current transfer situation within the mobile network (see also section 6).

4 End-to-end measurement of available bandwidth

As first approach to measure the available bandwidth within a GSM network a small client-server-program was implemented. The client is connected to the GSM network and sends a train of data packets to the server which is reachable from the Internet

(Figure 2). Every 30 minutes the client starts a measurement cycle using UDP first and afterwards TCP.

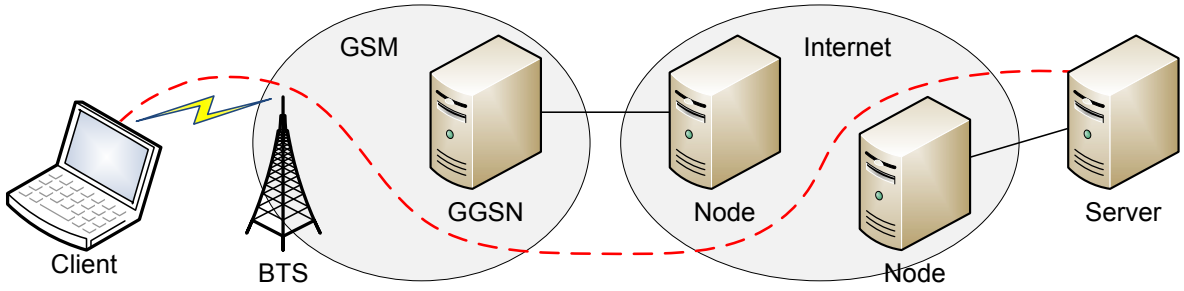
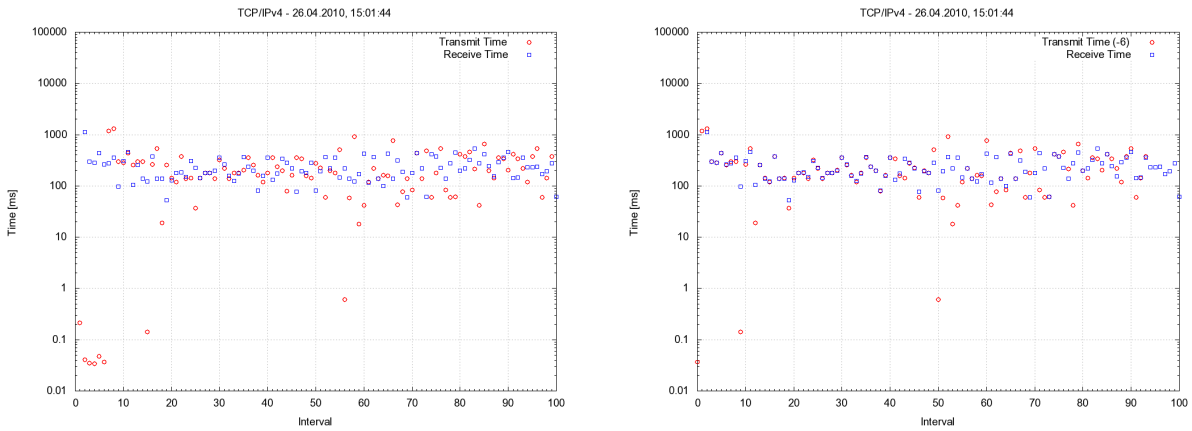


Figure 2: Network Measurement

Within this measurement client and server are not time synchronized. Both of them use their own high resolution timer. The client puts the transmit timestamps and its time resolution in the data packets. The server receives the packets and stores the client timing data and its own measurement data in a file. The assumptions here are:

- The radio link is the link with the smallest capacity and the minimum available bandwidth along the path.
- All other links along the path do not influence the distance between the packets.



(a) Measured Times (b) Shifted Transmit Times

Figure 3: Time between consecutive measurement packets

The left diagram in Figure 3 shows one of the measurements made. As one can see the captured transmit time is extremely low within the first six intervals. The first packets are filled into an empty transmit buffer and therefore the measurement shows the speed within the client and not within the network. The right diagram shows the same measurement but the transmit time values are shifted six intervals left in order to correct the influence of the transmit buffer size. Now one can see that up to the interval 50 (left half of the diagram) most of the time values from client and server are

nearly the same. In the second half the distances between the client and server values are bigger. Over the whole measurement the time values are spread over more than one decade. This indicates that some effects, which probably occur on the radio link, influence the currently transmitted data packets and the packets currently put into the transmit buffer in the same way. On the other hand, there must be some effects that have different influence on sender and receiver side.

Within the first measurement cycles many data packets were lost (Figure 4). As we have found out, the packet loss was caused by connection losses based on inter-RAT handovers from GSM to UMTS cells or vice versa. After turning-off the multi-RAT device option, the modem has used only the GSM network and no data packets were lost. This shows that even if the available bandwidth is relatively high the connection can temporarily be lost.

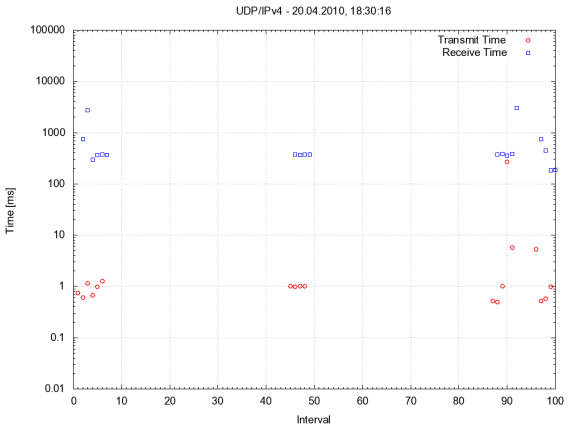


Figure 4: Effect of inter-RAT handover

The later experiment runs over a period of two days. The results are displayed in Figure 5 which shows the relative frequency of the mean values of each measurement cycle separated by the used transmission protocol.

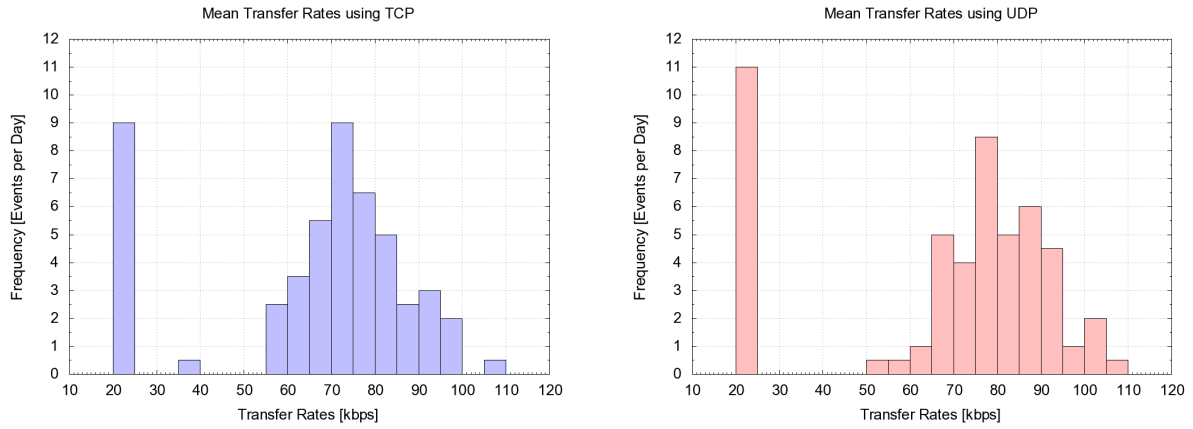


Figure 5: Transfer rates measured over a period of two days

Both diagrams show two separated groups – a small one at 20 kbps up to 25 kbps and broader one between 55 kbps and 100 kbps in the case of TCP and 50 kbps and 110 kbps in the case of UDP.

5 Proposals for solution

Depending on the tasks of the mobile devices within mobile nomadic systems, different approaches exist at applications level to deal with the specific aspects of mobility. Here two situations will be discussed. First, the mobile node can only be a data provider and second, the mobile node may act as worker node or autonomous unit.

5.1 Mobile device as passive node

Within this scenario the mobile nodes, such as sensors or patient monitor devices, capture data, preliminary analyze this data optionally and transmit it then to the stationary central part of the system. A typical way to adapt the functionality of an application on the mobile device to variable transmission bandwidths is to adapt the amount of data to be transmitted. Basically the following two methods are possible:

- Data reduction
- Data compression

Data reduction means to decrease the amount of transmitted data by dividing the captured data into more and less significant parts and disregarding the less important ones. On the contrary, data compression means to reduce the size of the data format. While the first method is lossy in every case, the second one can be lossy or lossless.

An alternative solution can be to buffer the captured data if the available bandwidth is too small and to send this data at a later time if the available bandwidth is again high enough.

5.2 Mobile device as active node

Within the second scenario the mobile nodes are instructed by the central part of the system to achieve tasks. Afterwards the mobile nodes report their results back to the instructing party. In order to be responsive to variable bandwidth and lost connections the work can be handed out to more than one mobile nodes. Generally the following models are feasible:

- Alternative worker nodes
- Redundant worker nodes

Within the first model the instructing party is looking for alternative worker nodes if the primary node is not reachable. The second model based on the parallel execution of one task on more than one worker nodes. According to the underlying failure model

the instructing party can use the first result or can use more than one result value to vote for the most probable one. For the result values the methods mentioned in section 5.1 can be used to react to variable bandwidth.

As an alternative solution the instructing party can transmit more than one task to a worker node. If the worker could not transmit a result, he can process the next task and try to transmit the result values together.

6 Adaptation to different conditions

At application level we want to manage the connection and/or the available bandwidth. There are three different situations which should be considered:

1. No cell change: – The connection is probably stable and only the available bandwidth is assumed to vary.
2. Soft handover: – The new connection is established parallel to the current one. For a short time both connections exist. The new connection provides probably another bandwidth.
3. Hard handover: – The current connection is closed and afterwards a new connection is established. While the handover process is not finished the connection is lost and the new connection provides probably another bandwidth.

The required information can be directly measured or can be derived from lower layer parameters that in the case of mobile phone networks can be captured as status or monitor values from the modem.

6.1 Measurement of information

At the first glance, measurement of the available bandwidth seems to be an easy and straightforward way because the required values are directly detected. But this way raises the following questions which should be always kept in mind:

- How many time does one measurement need?
- How many measurement cycles are required to calculate a mean value?
- How long is the network state stable? (or) How long is the mean value valid?

In addition, there are the costs for transferring data over a public mobile network regarding available bandwidth, influence to other data transmissions and money. Based on this points an effective algorithm is needed in terms of short measurement duration and low network load and if possible a combination of usual data transfers and measurement.

In the past, different tools, such as Pathload [6,7], Pathchirp [10,11] or Spruce [12], were developed to measure parameters of the transmission path or single links within the path. All mentioned tools are specialized to measure the available bandwidth but

each of them uses its own algorithm. Each tool assumes that FIFO queuing is used at all routers along the transmission path, cross traffic packets have an infinitely small size and cross traffic average rate changes slowly and is constant while a single measurement cycle [12]. This tools were evaluated by Han, et al. [5].

In [9] some problems, for example the time variation of link capacity, are discussed which are especially relevant to end-to-end measurements within mobile phone networks, here EGPRS networks.

6.2 Derivation of information

Another approach is to derive information about the available bandwidth from status values captured from the modem. Therefore a special interface, the AT commands also known as the Hayes command set, was standardized [1, 3]. But which information is required? To answer this question, we can, for example, look for conditions that trigger the handover process. In [4, subclause 3.4] the strategy for an inter-cell handover is defined as follows:

> Intercell handover from the serving cell to a surrounding cell will normally occur either when the handover measurements show low RXLEV and/or RXQUAL on the current serving cell and a better RXLEV available from a surrounding cell, or when a surrounding cell allows communication with a lower TX power level. This typically indicates that an MS is on the border of the cell area. ...

The abbreviations used above are defined in [2]:

RXLEV – Received Signal Level	TX – Transmit
RXQUAL – Received Signal Quality	MS – Mobile Station

The received signal level and the signal quality can be requested using the command AT+CSQ [3, subclause 8.5]. The signal level is specified over the range from -51 dBm to -113 dBm in steps of -2 dBm. The signal quality is specified in eight steps as defined in [4] subclause 8.2.4. One problem with this command is, it is optional! First inquiries have shown that some modems do not support this command to the full extent. For example, the Hewlett Packard un2400 and the Huawei V100R001 report the received signal level but not the signal quality; the Motorola C24 reports both but instead of the BER the FER is used to characterize the signal quality.

Other AT commands that could be helpful within this context are:

- AT+COPS – Public land mobile network (PLMN) selection [3, subclause 7.3]
- AT+CPSB – Current packet switched bearer [3, subclause 7.29]
- AT+CPWC – Power class [3, subclause 8.29]
- AT+CBC – Battery charge [3, subclause 8.4]

The problem: The implementation of all these commands is also optional!

In contrast to modems, which are typically used within consumer products, industrial modems, for example from Sierra Wireless, Sixnet or Huawei, provide usually more network information about the serving and the neighbor cells. The problem here is that this AT commands are vendor specific!

Back to the original problem: What do status values of the modem reveal about the available bandwidth or the connection? All effects that occur within the radio frequency path influence direct or indirect the received signal level and/or the signal quality. The network itself react on this influence and changes its transmission parameters with the objective of being able to correct possible transmission errors on the receiver side. We can draw conclusions from the status values about possible reasons for variation of available bandwidth or connection status. On the other hand, effects what occur inside the network and influence the available bandwidth cannot be detected with status values of the modem.

7 Conclusions

Often the application must adapt their behavior to the current quality of mobile communication. A middleware for mobile systems has to establish awareness for mobility because controlling of available bandwidth and connection status is only meaningful if the system as a whole is able to react to. This can be done by passing network parameters through to the application or by providing callback events depending on the current situation.

On the over hand, if the middleware want to manage the mobile communication it has to know the current network status and the amount of available bandwidth. The amount of available bandwidth can, depending on the capabilities of the underlaying layers, be deduced from status values of the network or directly be measured. In the case of measurement the effectiveness, especially the accuracy of the captured values and their duration of validity, as well as the influence on the transfer medium and other data transfers have to be considered.

With a view to real-time requirements it is important to notice that the middleware is subject to unpredictability of the underlaying mobile network. Therefore, the middleware cannot guarantee the compliance with hard real-time requirements. Instead, the middleware should focus on providing soft real-time guarantees.

References

[1] 3GPP TS 27.005 V9.0.0: 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Use of Data Terminal Equipment - Data Circuit terminating Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS), December 2009.

[2] 3GPP TR 21.905 V10.2.0: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Vocabulary for 3GPP Specifications, March 2010.

[3] 3GPP TS 27.007 V10.0.0: 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; AT command set for User Equipment (UE), June 2010.

[4] 3GPP TS 45.008 V9.3.0: 3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Radio subsystem link control, May 2010.

[5] Young-Tae Han, Eun-Mi Lee, Hong-Shik Park, Ji-Yun Ryu, Chin-Chol Kim, and Yeong-Ro Lee. Experimental Evaluation of End-to-End Available Bandwidth Measurement Tools. In *APNOMS'09: Proceedings of the 12th Asia-Pacific network operations and management conference on Management enabling the future internet for changing business and new computing services*, pages 498–501, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Manish Jain and Constantinos Dovrolis. Pathload: a measurement tool for end-to-end available bandwidth. In *Proceedings of the 3rd Passive and Active Measurements Workshop*, Fort Collins CO, March 2002.

[7] Manish Jain and Constantinos Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation With TCP Throughput. *IEEE/ACM Trans. Netw.*, 11(4):537–549, 2003.

[8] Qusay H. Mahmoud, editor. *Middleware for Communications*. John Wiley & Sons, Ltd, 2004.

[9] Juan Andrés Negreira, Javier Pereira, Santiago Pérez, and Pablo Belzarena. End-to-end measurements over GPRS-EDGE networks. In *LANC '07: Proceedings of the 4th international IFIP/ACM Latin American conference on Networking*, pages 121–131, New York, NY, USA, 2007. ACM.

[10] Vinay J. Ribeiro, Rudolf H. Riedi, and Richard G. Baraniuk. Locating Available Bandwidth Bottlenecks. *IEEE Internet Computing*, 8:34–41, 2004.

[11] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Proceedings Workshop on Passive and Active Measurement PAM2003*, March 2003.

[12] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 39–44, New York, NY, USA, 2003. ACM.

Understanding Service Implementations Through Behavioral Examples

Michael Perscheid
michael.perscheid@hpi.uni-potsdam.de

The understanding of service implementations, with a special focus on internal details that constitute functionality, is an important aspect during the development of services. Actual run-time data supports the comprehension of service implementations like examples support the explanation of abstract concepts and principles. However, the required run-time analysis is often associated with an inconvenient overhead that renders current tools impractical for frequent use.

We propose a practical, lightweight, and incremental approach to dynamic analysis based on entry points that describe reproducible system behavior. By observing and enriching such concrete examples of behavioral paths, we investigate new perspectives on service implementations that improve comprehension of execution semantics. We are providing perspectives for exploring one execution path in detail, for comparing multiple paths with each other, and for understanding the internal behavior from the user's point of view.

1 Introduction

Developers of object-oriented software systems, including service implementations, spend a significant amount of time on program comprehension [4]. They require an in-depth understanding of the code base that they work on; ranging from the intended use of an interface to the collaboration of objects. Gaining an understanding of a program by reading source code alone is difficult as it is inherently abstract.

Run-time information supports developers in coping with this complexity. Collected run-time data reports on the effects of source code and thus helps understanding it. At run-time, the abstract gets concrete; variables refer to concrete objects and messages get bound to concrete methods. For example, a program's run-time helps to answer: "How is a particular method called?" or "How does the value of a variable change?"

Unfortunately, the overhead imposed by current tools renders them impractical for frequent use. This is mainly due to two issues; setting up an analysis tool usually requires a significant configuration effort and performing the required in-depth dynamic analysis is time-consuming. Both issues inhibit immediacy and thus discourage developers from using these tools frequently.

We propose a new approach to dynamic analysis that enables a feeling of immediacy that current tools are missing. Based on entry points that describe reproducible

behavior, we split the costs of dynamic analysis over multiple runs—an initial shallow analysis followed by detached in-depth on-demand refinements. For our implementation, we leverage test cases as such entry points, as they commonly satisfy the necessary requirements. Triggered by their execution, we observe and enrich behavioral examples to improve the comprehension of execution semantics. We develop the Path tool suite, which allows developers to explore a specific execution path in detail (PathFinder), to compare multiple paths with each other (PathMap), and to understand the internals from the user's point of view (PathTrace).

The remainder of this paper is organized as follows: Section 2 presents our approach to dynamic analysis that collects data exactly when needed. Section 3 describes the Path tool suite. Section 4 demonstrates how program comprehension is supported by our tools. Section 5 concludes and presents next steps.

2 Dynamic Service Analysis

Dynamic service analysis is a practical, lightweight, and incremental approach to dynamic analysis for understanding system and service implementations. It is build on reproducible entry points such as test cases that act as concrete examples of service behavior. These behavioral examples concrete the abstract entities of source code with meaningful information and so support program comprehension.

By executing entry points multiple times, we can split the dynamic analysis costs over multiple runs. Starting with an initial shallow analysis for navigating behavioral paths, developers get only a subset of all possible run-time data. When they require more detailed information such as object states, this additional data is collected in detached on-demand refinements by executing entry points multiple times. This approach enables a feeling of immediacy that current dynamic analysis concepts are missing.

For an easy setup of analysis tools, our approach can seamlessly be integrated into current development environments. Developers only have to define their subsystem of interest and a collection of entry points. While the first must be done with the declaration of packages only once, the latter can be automized if test cases or API examples are available. Furthermore, by continuously maintaining a coverage relationship between executed entry points and covered methods [12], it is possible to embed arbitrary methods into meaningful examples at all times.

Figure 1 summarizes our approach. First, reproducible entry points such as test cases are executed to produce behavioral example paths (Section 2.1). Second, dynamic analysis is applied to these examples for exploring a specific path in detail or comparing multiple paths with each other (Section 2.2). Finally, the meaning of entry points can be enhanced with links to requirements for recovering traceability information automatically (Section 2.3).

2.1 Entry Point Characteristics

Dynamic service analysis requires the ability to reproduce arbitrary points in a program execution. Therefore, we assume the existence of entry points that specify determinis-

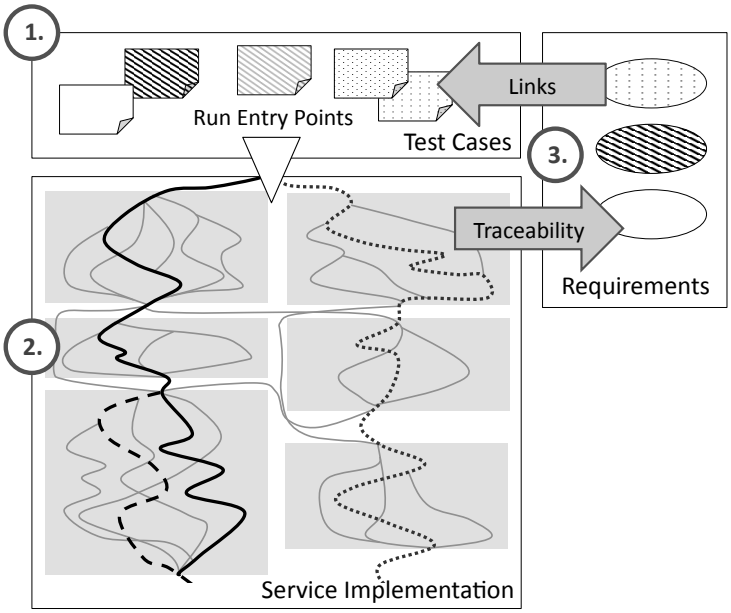


Figure 1: Dynamic service analysis allows for exploring behavioral paths through service implementations. Triggered by the run of entry points (here test cases) (1), we analyze specific or multiple paths (2) to support program comprehension. Furthermore, by linking entry points and requirements (3), we recover traceability data automatically.

tic program executions. For our implementation, we leverage test cases as such entry points, as they commonly satisfy this requirement [13]—a test case describes what the system is expected to do, its execution reveals how it is realized. Moreover, tests are meaningful behavioral examples of the system and its parts; they can be executed repeatedly, fast, and without any side effect before and after their execution; and they are an integral part of several development processes, especially agile processes.

Leveraging test cases as entry points is not a requirement for dynamic service analysis. Our tools work best if test coverage for the developed application is high, but resorts to manually specified entry points if no covering test is found. This, however, requires more knowledge about the system under observation than relying on test coverage: it is not always trivial to anticipate control flows leading to methods of interest.

2.2 Step-wise Run-time Analysis

Traditional approaches to dynamic analysis are time-consuming as they capture comprehensive information about the entire execution up-front. Low costs can be achieved by structuring program analysis according to user needs or, more specifically, dividing the analysis into multiple steps.

Step-wise run-time analysis [8] splits the analysis of a program's run-time over multiple runs: A high-level analysis followed by on-demand refinements. A first *shallow analysis* focuses on the information that is required for presenting an overview of a

program run. Further information about method arguments or instance variables are not recorded. As the developer identifies relevant details, such data is recorded on-demand in additional *refinement analysis* runs. The information required for program comprehension is arguably a subset of what a full analysis of a program execution can provide. While our approach entails multiple runs, the additional effort is kept to a minimum, especially when compared to a full analysis that has no knowledge of which data is relevant to the user. We reduce the costs by loading information only when the user identifies interest. This provides for quick access to relevant run-time information without collecting needless data.

The concept of step-wise run-time analysis is also adaptable to multiple execution paths. Instead of running only one entry point, several entry points can also be executed and analyzed one after another. Only required run-time information is collected and if necessary summarized with dynamic metrics. In this way, behavioral paths, different states, or coverage data can be compared with each other. Analogous to the dynamic analysis of a specific path, we collect only initial run-time information for the task at hand. When more detailed information is required, it can be refined in detached runs. Thus, dynamic analysis can also be divided into fine-granular and incremental steps for multiple entry points and their execution paths.

2.3 Link Entry Points, Get Traceability

Additional links between entry points and their primary objectives support program comprehension as developers can understand the reasons for exemplary system behavior. So far, entry points offer examples into system behavior but their real purpose is hidden in external requirements, the development history, or implementation artifacts. Especially, test cases were implemented with a particular reason in mind such as verifying a specific requirement. With the connection between entry points and requirements (or other development entities), the meaning of subsequently behavioral paths can be enhanced.

Links between entry points and requirements in combination with dynamic service analysis allow for recovering traceability information automatically [6]. Requirements traceability is considered to be important for software understanding as it supports answers for questions such as why a particular source code entity was implemented. Adapting the concepts of feature localization [2], annotated entry points are executed and behavioral paths are related to entry points and requirements in question. Afterwards a requirement is traced to a source code entity if it has been executed at least once in a specific entry point that is linked to this requirement. Depending on the coverage of entry points, large parts of the system can be traced and classified to requirements automatically.

Future work deals with the automatization of the manual linking step. We are working on the integration of acceptance test frameworks with dynamic service analysis. As some acceptance test frameworks store or explicitly represent the relation to tested requirements, this information can be reused for replacing the manual linking process. Consequently, we have a fully automatic requirements traceability approach from requirements via dynamic analysis of tests through to traced source code entities.

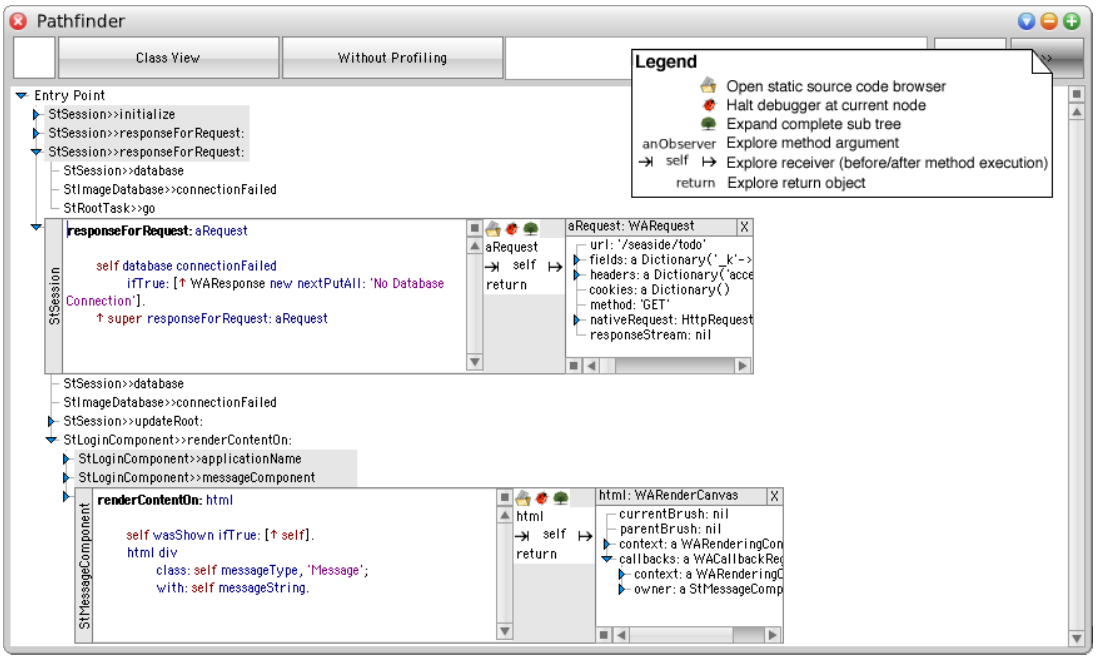


Figure 2: PathFinder is our interactive dynamic analysis tool that allows developers to explore a specific behavioral path through the service (ToDo Web application [9]).

3 An Overview of the Path Tool Suite

The *path tool suite* realizes the concepts of dynamic service analysis and supports the understanding of behavior. By means of examples, being triggered by reproducible entry points, our tools observe, analyze, and present behavioral *paths* for developers. At the same time, our tools distribute the costs of dynamic analysis across multiple runs and so enable a feeling of immediacy when program behavior is explored—a property that cannot be provided by fully dynamic analysis approaches.

Our tools are combined with each other in order to refine behavioral questions step by step and from different perspectives. PathFinder (Section 3.1) reveals one specific behavioral path, its state, and object interactions. PathMap (Section 3.2) summarizes run-time information with the help of several metrics and presents it inside the system's architecture. PathTrace (Section 3.3) connects the user's and developer's point of view as it traces requirements down to source code entities. At the end, all tools are supposed to answer behavioral questions for one or multiple execution paths from several points of view—a characteristic that is only minimal supported by current tools [5, 11].

3.1 PathFinder: Interactive Dynamic Views

PathFinder [8] is our dynamic analysis tool for interactively introspecting the behavior of one specific execution path. Based on a lightweight call graph representation used for navigation (shallow analysis), developers can state their points of interest and all further information is computed on demand by re-executing the chosen entry point

in background (refinement analysis). We distribute the overhead of dynamic analysis over multiple runs so that there is no need to collect all data at once. Thereby memory consumption and performance impacts are kept low. Thus, we have a practical approach for behavioral views that will be evaluated regarding its improvements for program comprehension in the near future.

Figure 2 presents PathFinder while a service request of our ToDo Web application [9] is processed. The internal behavior is revealed for this specific request and developers can follow what inside the application happens. If they are interested in further details, the request is executed again and more detailed information such as method arguments is collected and presented. Thus, behavioral reachability questions concerning only one execution path [5, 11], for instance, "what happens before executing the `renderContentOn:` method?" or "how does the request object change?", can be answered with the help of PathFinder. Its interactive dynamic views allow for navigating call and object trees in both forward and backward direction so that developers get insights into the execution and state history without any restrictions. Furthermore, almost all run-time information including the initial shallow analysis is provided in less than a second (less than 300 milliseconds for 95% of about 4.400 test cases [8]). These features should encourage developers to use PathFinder at least as often as they use tedious and time-consuming debugging strategies for program comprehension [5].

3.2 PathMap: What We Can Learn from Tests

PathMap summarizes multiple behavioral paths and merges static and dynamic views of a system under observation. In contrast to PathFinder, it has a stronger focus on test cases as entry points but the presented concepts are still independent of that fact. PathMap is integrated into a standard test runner and enhances the value of running test cases by analyzing their execution, rendering dynamic metrics, and suggesting meaningful entities, which can be further explored with the aforementioned PathFinder tool. We investigate new system perspectives that are intended to support several software engineering tasks such as guiding developers to potential locations for traced requirements, hot spots, or untested code. Moreover, we compare dynamic paths and reveal anomalies for more suitable fault localization or hints at design disharmonies.

Figure 3 illustrates the PathMap and the test quality analysis of the Seaside Web framework [1]. Seaside's system structure is rendered within a tree map layout where packages include classes which in turn include method entities. The entire Web framework (more than 3,700 methods) is presented in the space of a 500 pixel square, which can be explored interactively. Furthermore, the static view can be colorized with a static metric such as lines of code, complexity or as in this example the developer of a method. Having analyzed the execution of all test cases, PathMap darken method entities that has been covered by at least on entry point. As a consequence in this example, we are able to answer the questions "who has written insufficiently tested methods?" or "which subsystems need more attention during quality assurance?".

PathMap provides answers to behavioral reachability questions that deal with multiple execution paths [5, 11]. We support the following scenarios:

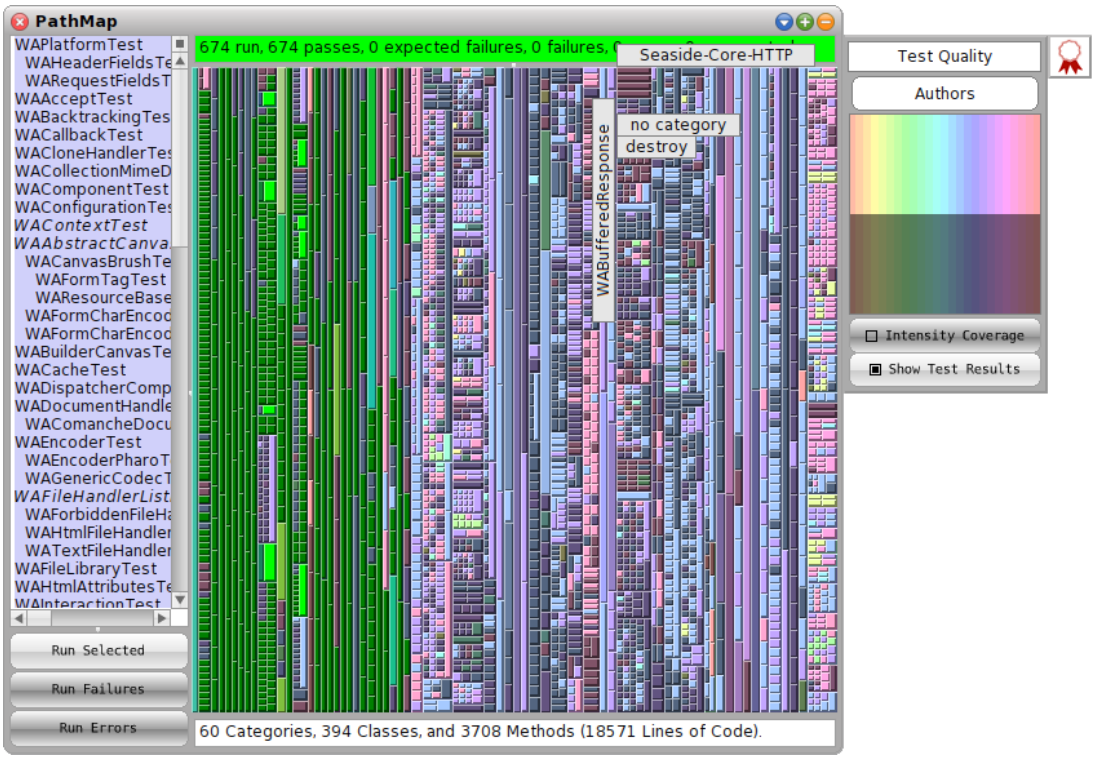


Figure 3: PathMap executes multiple entry points and summarizes the analyzed behavior within the system's architecture (Seaside Web framework [1] - Test coverage in relation to the developer of a method).

Test Quality Combining arbitrary static metrics with coverage information allows for assessing software quality in terms of application structure and testing activities.

Concerns Traceability Based on the concept of feature localization [2], we trace arbitrary concerns to covered entities and support developers in comprehending the system from different points of view.

Fault Localization With the comparison of passed and failed test cases [10], failures can be localized more closely to real causes.

Profiling Instead of looking at one execution profile, we summarize multiple executions and identify hot spots for processing time, number of objects, or method calls.

3.3 PathTrace: Understanding the User's Point of View

PathTrace is a semi-automated approach for the post-traceability of requirements that allows developers to comprehend the system from the user's point of view. Based on the concepts of feature localization, we manually link entry points with requirements, analyze their behavioral paths, and trace requirements to covered source code entities

Figure 4: AcceptIt allows for writing acceptance tests in a business-readable domain specific language so that requirement descriptions and related tests are one and the same. Based on such entities, PathTrace can automatically recover traceability data.

automatically [6]. Other development tools can use the gathered traceability information to improve understanding of user requests and their related implementation details. Thus, answers for typical software maintenance questions such as "how is this requirement implemented?" or "which source code entities are related to the user's failure report?" [11] are better supported.

To complete our approach to a fully automatic traceability technique, we are working on a new acceptance test framework called AcceptIt[1]. AcceptIt provides a way of merging requirements and tests into one executable source code entity. In other words, linking becomes redundant as links are implicitly available. Tests are described in a business-readable domain specific language (BR-DSL). This BR-DSL is mapped to ordinary library methods. In consequence, acceptance tests are executable as usual and readable (maybe also writable) by customers. The difference between an acceptance test and a scenario description becomes blurred.

Figure 4 presents the current state of AcceptIt with its BR-DSL, its language extension, and a part of the source code library (example taken from our ToDo Web application [9]). On the left side, the "login" scenario as part of a user story is defined and can be understood without any knowledge of the implementation—`given` declares preconditions, `when` describes actions, and `then` asserts a specific event or state. On the right side, the `given` step is mapped to the corresponding library method and its implementation. At a later time, libraries should offer a lot of generic mappings so that they are reusable in several scenarios.

AcceptIt introduces a new kind of entry point. On the one hand PathTrace is extended to be completely automatic on the other hand PathFinder and PathMap analyze entry points with clear objectives. So, developers have not only the answer to what a

[1]The idea is based on Cucumber `http://www.cukes.info`. Last accessed: October 8, 2010

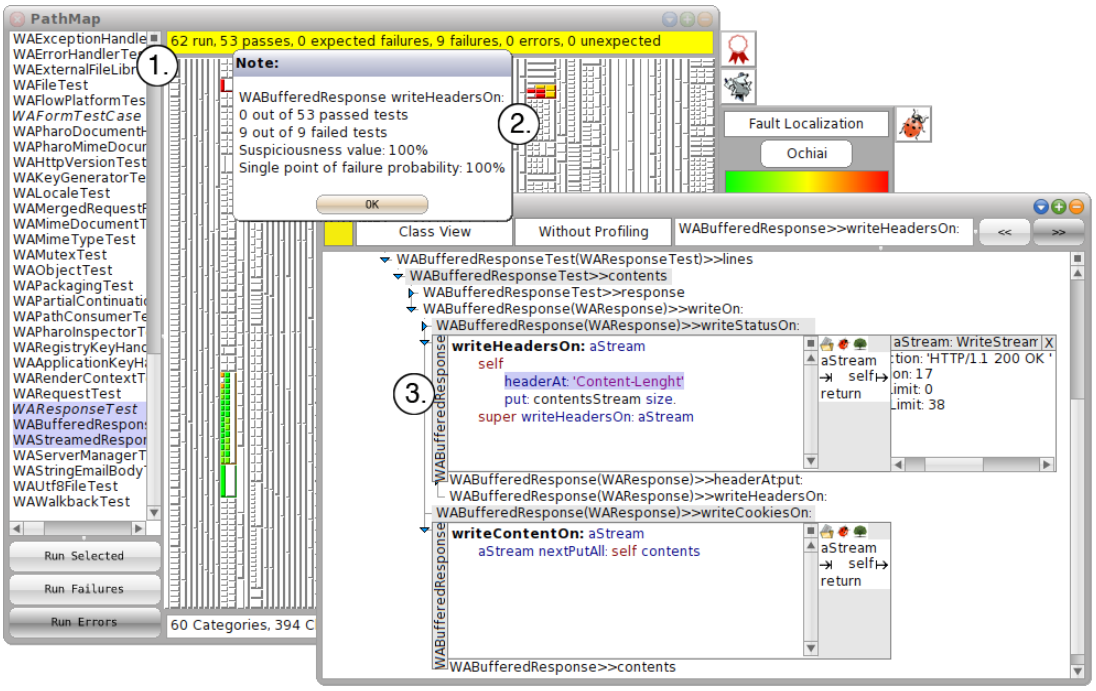


Figure 5: After selecting and running failing tests (1), PathMap compares test run results and colorizes suspicious methods (2). Subsequently, PathFinder allows for exploring a specific path to the cause of a failure (3).

test case describes and how it is realized, they have also its purpose. In the sense of program comprehension, we can enhance the meaning of behavioral paths.

4 Case Study: Fault Localization in Seaside

For demonstrating dynamic service analysis and our Path tool suite, we have done a case study about fault localization in the Seaside Web framework [1]. Since fault localization or in other words "debugging" usually requires much knowledge about system behavior, it is an ideal use case for presenting the benefits of our approach.

We have introduced a defect into Seaside's Web server—inside the header creation of buffered responses, we inserted a "Content-Lenght" typo. For that reason, service requests with buffered responses produced invalid results but streamed responses still worked correctly. Seaside's test suite answered with 9 failed and 53 passed test cases for all response tests. Starting the standard debugger at an failed test led to a violated assertion within the test itself. This means that the execution history was lost and the failure cause was not comprehensible. The assertion was thrown for the whole response object without any pointers to the invalid state. The typo was far away from the observable malfunction. Such a situation is the rule rather than the exception [14].

Figure 5 shows how the defect could be identified with the help of our Path tools. First, PathMap replaced the standard test runner, rendered the whole Seaside sys-

tem within a tree map, executed the response tests, analyzed the behavioral paths of passed and failed test cases with the "Ochiai" metric [10], and colorized the map with suspiciousness and confidence values. In short a method has a higher suspiciousness (hue towards red value) if it was executed in more failing than passing tests and a method has less confidence (brighter color) if it was not executed in all failing tests. Second, the computed values suggested only three methods in full red (100% suspiciousness and 100% confidence). All other methods were paler and not so hot. Third, for understanding all three methods, we opened PathFinder on a failed test as we expected that all three methods (same class) were related to each other. PathFinder showed that the methods `writeContentsOn:` and `writeHeadersOn:` were called in sequence within `writeOn:` (here, we ignored the third method as it was only a simple accessor). While using PathFinder, we navigated the execution history in both directions and with the help of on-demand refinements we verified assumptions concerning corrupted object states. For instance, the method argument `aStream` already included a valid response code. On closer examination we understood the implementation of both methods and found the failure's cause.

5 Summary and Next Steps

A current study [5] reveals that developers ask reachability questions, which are "searches across feasible paths through a program". The authors show that developers often failed to understand program behavior and modified the code relying on false assumptions. Especially, the lack of adequate tool support was a reason for the developers' problems with answering reachability questions.

Dynamic service analysis [7] and our Path tool suite enable developers to experience a feeling of immediacy when they explore program behavior. Run-time views support exploring run-time behavior and facilitate answering reachability questions such as: In what context is a particular method used or where is the cause of a failure? Our approach encourages frequent use of our tools and thus promotes the validation of assumptions rather than relying on guess work.

Besides an empirical evaluation for program comprehension in the near future, we are investigating how the concepts of dynamic service analysis can facilitate answers for behavioral reachability questions. In particular, we expect benefits by extending our PathFinder to a lightweight back-in time debugger and by summarizing sane behavior within dynamic contract layers [3]. For instance, we could debug test cases backwards in time, answer *why* questions, or verify generated assertions at run-time.

References

[1] S. Ducasse, A. Lienhard, and L. Renggli. Seaside: A Flexible Environment for Building Dynamic Web Applications. *IEEE Software*, 24(5):56–63, 2007.

[2] O. Greevy. *Enriching Reverse Engineering with Feature Analysis*. PhD thesis, University of Berne, 2007.

[3] R. Hirschfeld, M. Perscheid, C. Schubert, and M. Appeltauer. Dynamic Contract Layers. In *SAC '10: Proceedings of the 25th Symposium on Applied Computing*, pages 2169–2175. ACM, March 2010.

[4] A. J. Ko, R. DeLine, and G. Venolia. Information Needs in Collocated Software Development Teams. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 344–353. IEEE Computer Society, 2007.

[5] T.D. LaToza and B.A. Myers. Developers Ask Reachability Questions. In *ICSE '10: Proceedings of the 32nd International Conference on Software Engineering*, pages 185–194. ACM, 2010.

[6] M. Perscheid. Requirements Traceability in Service-oriented Computing. In *Proceedings of the Fall 2009 Workshop of the HPI Research School on Service-Oriented Systems Engineering*. Hasso-Plattner-Institut, October 2009.

[7] M. Perscheid. Dynamic Service Analysis. In *Proceedings of the Joint Workshop of the German Research Training Groups in Computer Science*, pages 204–205. m verlag mainz, May 2010.

[8] M. Perscheid, B. Steinert, R. Hirschfeld, F. Geller, and M. Haupt. Immediacy through Interactivity: Online Analysis of Run-time Behavior. In *WCRE '10: Proceedings of the 17th Working Conference on Reverse Engineering*, page to appear. IEEE, October 2010.

[9] M. Perscheid, D. Tibbe, M. Beck, S. Berger, P. Osburg, J. Eastman, M. Haupt, and R. Hirschfeld. *An Introduction to Seaside*. Software Architecture Group (Hasso-Plattner-Institut), 2008.

[10] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold. Lightweight Fault-Localization Using Multiple Coverage Types. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pages 56–66. IEEE, 2009.

[11] J. Sillito, G. C. Murphy, and K. De Volder. Asking and Answering Questions during a Programming Change Task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.

[12] B. Steinert, M. Haupt, R. Krahn, and R. Hirschfeld. Continuous Selective Testing. In *XP '10: Agile Processes in Software Engineering and Extreme Programming*, pages 132–146. Springer, 2010.

[13] B. Steinert, M. Perscheid, M. Beck, J. Lincke, and R. Hirschfeld. Debugging into Examples: Leveraging Tests for Program Comprehension. In *TestCom 2009: Proceedings of the 21st IFIP International Conference on Testing of Communicating Systems*. Springer-Verlag, November 2009.

[14] A. Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, 2006.

Modeling Browser-based Mashups by Means of Meaningful Choreographies

Emilian Pascalau
emilian.pascalau@hpi.uni-potsdam.de

1 Introduction

At the 5th Symposium on Future Trends in Service-Oriented Computing, Peter Lorenz in the talk "The SAP Vision on Cloud Computing"[1] was underling a set of characteristics that have a great impact on the current cloud based environment (*a different world* - as he called it): business models are changing, customer relationship is changing, software on demand, instant use. *"In cloud computing , on demand software is very different. Users - easy consumption. That's the story: Instant use, you want to have this you can use it. No big debate, no hand books, no I don't go to Waldorf for 4 month of training [...]. This is gone. We need to change. And we need to change to something which is simple to use software, instant but powerful. This is not dumb software. [...] The rest is I would say standard: mobile, everywhere, browsers, various devices all of those things."*[2]

These characteristics emphasize *deep architectural impact*, they *change the nature of application* and raise questions *how to design, model and execute this new type of applications*.

A set of requirements that this new type of applications should comply with can be distilled from here. A fundamental characteristics is *software on demand*. The user sees something and wants to use it, then he must be empowered with proper tools to be able to use the service by himself, as much as possible. No long and exhausting training, *instant use*. If the user has to do it by himself, it must be *user friendly*. The user should *not have to take care about technical issues*.

The Web 2.0, as it has been unveiled by O'Reilly [7], is an advance of technologies that are applied to build distributed software on the Internet, but also refers to a shift in people's perception of the Web and a paradigm shift in how they use it.

Mashups are a novel genre of Web applications that fosters innovation and creativity in a generative environment, which Zittrain denotes as the "future Internet" [15], and are probably "one of the more powerful capabilities coming out of the Web 2.0 wave" [11].

A series of approaches that tackle mashups, mostly technical approaches, have been defined. The widget/gadget based approaches although user friendly are too restrictive in functionality, e.g. functionality and data can not be composed / aggregated

[1] http://www.tele-task.de/archive/lecture/overview/5014/
[2] http://www.tele-task.de/archive/video/flash/10012/

Sent	Message Type	From	Subject	Deadline	Web Page
12-Oct-2010	conf. ann.	Iris Reinhartz-Berger	The program of the 3rd domain engineering workshop at ER'2010 – Nov 3rd, Vancouver	3-Nov-2010	web page
12-Oct-2010	conf. ann.	Iris Reinhartz-Berger	The program of the 3rd domain engineering workshop at ER'2010 – Nov 3rd, Vancouver	3-Nov-2010	web page

Figure 1: DbWorld excerpt

only under special assumptions and environments. Only a list of predefined services can be used. Thus the characteristics previously underlined are strongly restricted.

Our objective is to fulfill the characteristics underlined. A browser based approach it is desirable because as argued in [1] browser based mashups are *pure mashups*. Beside this the browser by nature facilitates the fulfillment of some of the characteristics emphasized: allows portability, run everywhere, services are accessed in an unique way, the fundamental interaction with the services is unique, no matter the service, the users already know how to interact with the services. What is still missing is an intelligent and easy way for the users to design, model, combine in an intelligent and simple way on demand services, data and behavior, and to execute these collaborations.

The main focus of this technical report is not about the execution of such applications as the execution has been discussed already (e.g. see [9]) but about the design of such applications. A use case example that complies completely with the described problem is the issue of searching and then storing conferences of interest in a calendar.

2 Use Case

For scientists in the filed of IT the DbWorld Mailing List is the well known place where they can search for an IT conference. A series of information is provided here, but most important is the subject, deadline and the web page of the event published (see Figure 1 for a small excerpt of the DbWorld service). From a technical perspective DBWorld does not provide an API to allow programmatically access and interrogation of the service. Thus with respect to current approaches this services is useless. Google Calendar is one of the most known Google Apps services. For Google Calendar the important information are the title of the event, the date and description (see Figure 2).

Opposed to the DbWorld services, Google provides for this service beside the regular form also an API to access the contents. However we are interested in an uniform way of dealing with all the services, and also the user should not be required to face technical issues (i.e. APIs).

The regular way to achieve this goal, of having the conferences stored in Google Calendar by their deadline, is manually. The user is required to maintain two open tabs in the browser; even though there might be several entries that comply with a search term, the user must deal with the events one by one as DBWorld does not provide built in search functionality, only manual search; it would be difficult to remember all the information about an event entry, the user will have to move between the two open tabs several times, in order to store only one event in the calendar.

Recorder/play like tools can't be used in this scenario either because the application must react according to user's behavior and according to specific search results.

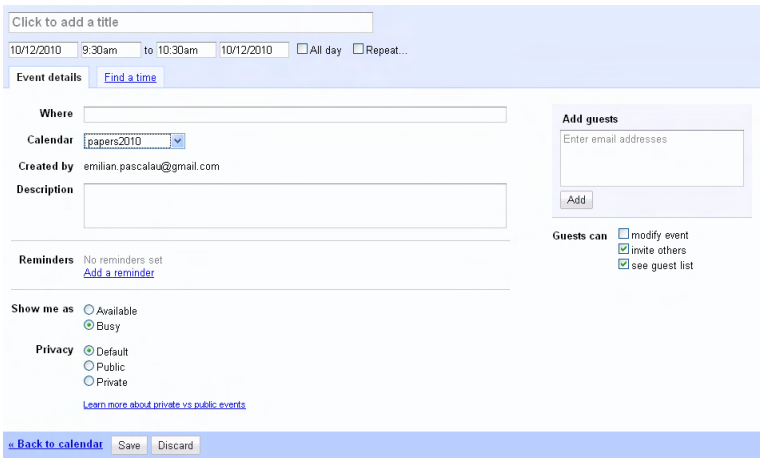


Figure 2: Google Calendar Event Creation

For example a particular search might return 5 entries or another one might return 2 entries. Next session will explain how this model can be model by means of processes and choreographies enriched with contextual information. These enriched processes and choreography models can be automatically transformed into rules. The resulting set of rules constitute the run-able application, the run-able mashup.

3 Enriching choreography models with contextual information

In order to achieve the desired outcome, to have stored a list of conferences in a calendar the user, must perform a collaboration between two services. As discussed in Section 2 between DbWorld and Google Calendar. The user himself/herself is part of the collaboration as he/she knows which is for example the subject of interest after which the search has to be performed. The user must interact with the services at least to search for the conferences. To automate this process of storing conferences information in a calendar, someone must be able to design and then execute such a collaboration. This collaboration must handle both user and machine behavior. We call this type of applications: mashups.

In [9] we have introduced and discussed how mashups can be executed by means of rules. According to [13] process choreography is used to define cooperation between process orchestrations. Moreover this collaboration is specified by collaboration rules. [9] argues that rule-based modeling and execution of mashups are nothing else but collaboration rules and the mashup itself is a *browser-based service choreography*.

[13] states that *"while domain-specific process choreography standards are important in their particular fields, they lack the flexibility to define new types of business-to-business collaborations that are important for supporting cooperation between companies in todaySs dynamic market environments. Therefore, new approaches for the definition and implementation of process choreographies are required"*.

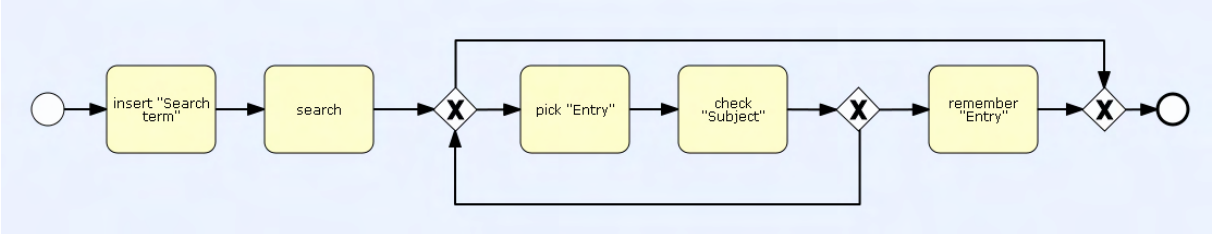


Figure 3: DbWorld Search Process - Variant 1

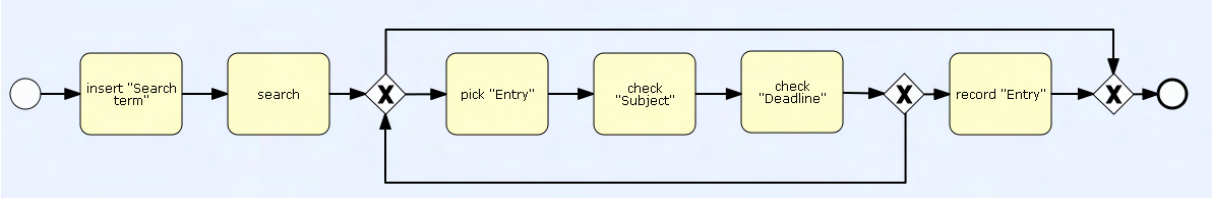


Figure 4: DbWorld Search Process - Variant 2

We believe that mashups are exactly such a particular type of choreographies that require flexibility, they require new means of modeling and execution. Browser based mashups are created by human users and more over their lifecycle requires the intervention of the user. The user is the one that powers the application, by interacting with systems, via events, via inputs etc.

Lately, context and context awareness has attracted a lot of attention (i.e. [2, 3]). Coutaz et al. argues in [2] that *context is key* in the development of new services. Furthermore explicit relationship between environment and adaption are the key factors that will unlock context-aware computing at a global scale [2].

Dey in [3] defines context as *any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*. Similarly for Coutaz et al. [2] the context is a *structured and unified view of the world in which the system operates*.

The use of contextual information in the field of business processes and workflows has been already addressed (see for instance [12, 14]).

An interesting aspect about the services in the form of web pages is that they allow the user to define its own process on how to use the service. Figures 3 and 4 exemplify this aspect. A user could search conferences only based on the subject (Figure 3) or both based on the subject and deadline (Figure 4). The process is quite intuitive, the user must insert a search term, which will be checked / compared with the subject, field. Every entry it is checked in this way. Each entry that matches the check it is remembered as its information must be stored in the calendar.

The process to create a basic Google Calendar event is depicted in Figure 5.

We use BPMN 2.0 [6] specification to represent our processes. The processes depicted (Figures 3, 4 and 5) both for the DbWorld and Google Calendar are based on

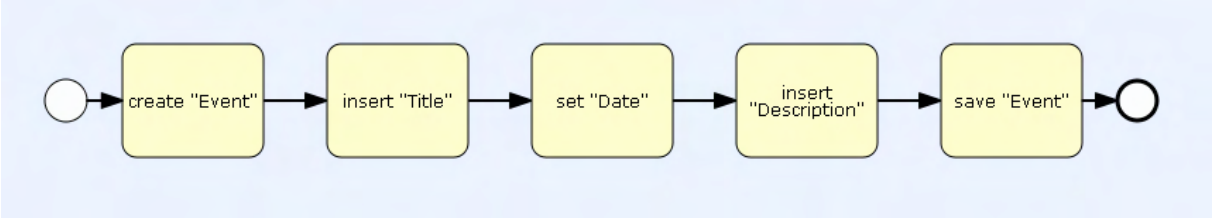


Figure 5: Create Event in Google Calendar Process

the Figures 1 and respectively 2.

As mentioned earlier we need to design a choreography to model the mashup. The design is based on the individual processes that we have already described.

In [10] we have argued that context gives concrete meaning to the processes. Context can be taken into account both at design time and also at run time. The approach we introduced in [10] proposes a solution based on a unified ontology that connects together context with process ontology thus facilitating a unified way of representation and reasoning.

To model the context in our models we use the idea introduced by Wieland in [14]. Wieland uses annotations to enrich BPEL models with contextual information.

Figure 6 depicts the simple choreography involved in this scenario. Basically every time a conference entry it is identified, then the information is stored as a calendar event in Google Calendar. Thus based on the processes we have already introduced in Figure 3 or Figure 4 DbWorld sends a message to Google Calendar service, containing the entry information. The information contained in the message has been clearly modeled via associated annotation. To be able to automatically store the information into the correct fields a mapping must be defined. Via annotation the mapping is straight forward.

Dey defines in [3] a set of major context categories: *identity*, *location*, *status* and *time*. These categories are recurrently reused, for example in [14]. In the scenario presented here, it is an obvious need for defining the location.

For the human user it is very easy to locate *Subject* field in the page and then to use that particular column to search for conferences. In addition the fact that the information is organized in a table is also very easy to observe for the human user. In the browser information is presented in an organized way, for example there is a specific structure used to define a table and to present this information as a table. Thus we take advantage of this knowledge and we define patterns to be able to map contextual concepts defined as annotation and then reason about the structure of the page to locate concepts in the page. This process of reasoning about contextual information and also about the processes modeled is a cognitive process. For an extensive discussion about cognitive processes one could refer to [5].

Figure 7 depicts a basic example where an element of a process has been annotated with contextual information. Recall that we use a unified representation for the concepts no matter that they represent context or process elements. To do so we take advantage of the approach we introduced in [10]. The `check "Subject"` element has

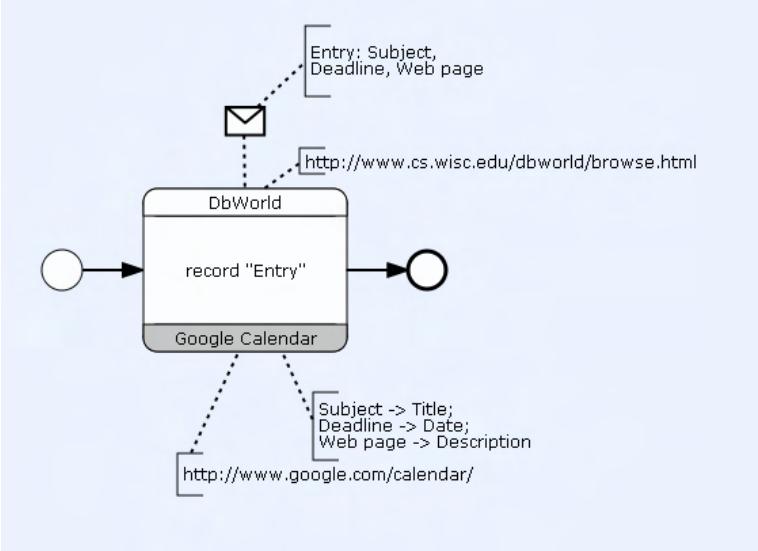


Figure 6: Simple DbWorld - Google Calendar Choreography

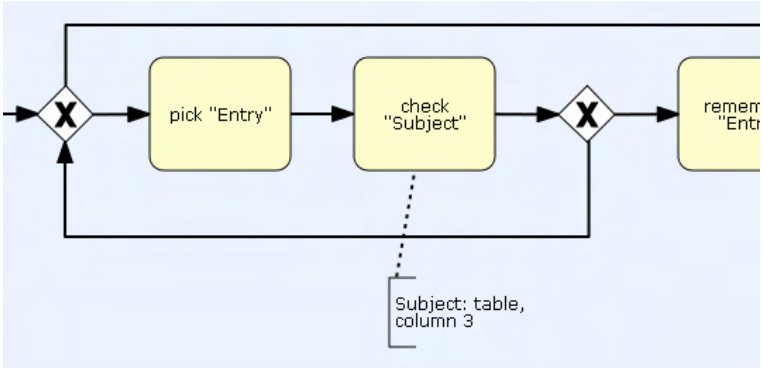


Figure 7: Element in DbWorld Process Annotated

been annotated with contextual information. This information refers to the `Subject` concept and identifies the location as being in a table, in the 3rd column.

The simple choreography depicted in Figure 6 already contains location information for the two services involved (e.g. http://www.cs.wisc.edu/dbworld/browse.html).

Via a cognitive process which uses the context enriched models as well as a set of predefined patterns (process that is performed without any other intervention from the user) the executable rule set that constitutes the run able mashup is created.

Figure 8 depicts an excerpt of such a rule. For a more detailed description about the run able mashup one can refer to [8, 9].

```
1
2
3  {
4  ...
5    eventExpression":{
6      "type":"click",
7      "target":{"variable":{"name":"$X"}}
8       },
9    "condition":[
10          {"description":{
11       "type":"HTMLButtonElement",
12       "binding":{"variable":{"name":"$X"}},
13                    "constraints":[
14                      {"propertyRestriction":{"property":"id",
15                         "operator":"EQ",
16                         "value":"mashSearchButton"
17                         }
18                      }
19                    ]
20                    }
21      },
22      {"description":{
23        "type":"INPUT",
24        "binding":{"variable":{"name":"$cSearch"}},
25                       "constraints":[
26                      {"propertyRestriction":{"property":"type",
27                        "operator":"EQ",
28                        "value":"text"
29                        }
30                      },
31                                {"propertyRestriction":{"property":"id",
32                        "operator":"EQ",
33                        "value":"mashSearchInput"
34                        }
35                      },
36                                    {"propertyBinding":{"property":"value",
37                        "variable":{"name":"$sValue"}
38                        }
39                      }
40                      ]
41                                       }
42      },
43          {"description":{
44         "type":"TD",
45         "binding":{"variable":{"name":"$dbEntry"}},
46                        "constraints":[
47                    {"propertyBinding":{"property":"firstChild",
48                      "variable":{"name":"$what"}
49                      }
50                    },
51                                    {"propertyBinding":{"property":"nextSibling",
52                      "variable":{"name":"$Z"}
53                      }
54                    }
55                    ]
56                                    }
57       },
58       ...
59       }
```

Figure 8: **Rule:** *Perform search in DBWorld*

4 Related work

The approach presented here is similar to the outcome of the Adaptive Service Grid (ASG)[3] platform. The characteristics of the ASG approach are discussed in [4]. In comparison with the ASG project where the main focus is to *discover and compose services to subprocesses using semantically specified data structures and services, based on domain ontologies using WSML* [4] here the services are already selected by the human user. In addition this approach strongly takes into account interaction with the human user, as the human user is the one who drives to some extent the application, the mashup. Here we are dealing with special type of environment, where the user must be able to define its software on demand. The user defines the behavior of the application via choreographies and processes enriched with contextual information.

5 Conclusions

This report has introduced a choreography based approach which has been enriched with contextual information in order to model and later directly execute user defined applications. A new type of applications is addressed here. Applications that are created on demand, by the users. These applications combine and aggregate data and behavior provided by arbitrarily chosen services. Once such an application has been modeled, it can be of course updated with new information but most important it can directly executed and can be shared / sold via a marketplace. The approach has of course its own limitations. One of these limitations is that the approach can access only information that is accessible via DOM; for example having an image that depicts the word "Search", but acts as a button, won't have no other meaning to the application, than being an actionable button.

Current implementation is able to execute mashups defined by means of run able rules.

References

[1] B. Bioernstad and C. Pautasso. Let it flow: Building Mashups with Data Processing Pipelines. In *Proc. of Mashups'07 International Workshop on Web APIs and Services Mashups at ICSOC'07*, number 4907 in LNCS, pages 15–28, 2007. `http://www.jopera.org/files/jopera_mashup07.pdf`.

[2] Joelle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[3] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[3] http://asg-platform.org/

[4] Dominik Kuropka, Anja Bog, and Mathias Weske. Semantic Enterprise Services Platform: Motivation, Potential, Functionality and Application Scenarios. In *Proceedings of the thenth IEEE international EDOC Enterprise Computing Conference. Hong Kong*, pages 253 – 261. IEEE, 2006.

[5] Allen Newell. *Unified Theories of Cognition*. Harvard Univeristy Press, 1994.

[6] OMG. Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0. http://www.omg.org/spec/BPMN/2.0, August 2009.

[7] T. O'Reilly. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. *Oreillynet.com*, September 2005. `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

[8] Emilian Pascalau and Adrian Giurca. A Lightweight Architecture of an ECA Rule Engine for Web Browsers. In *Proceedings of 5th Knowledge Engineering and Software Engineering, KESE 2009*, volume 486. CEUR-WS, 2009.

[9] Emilian Pascalau and Adrian Giurca. A Rule-based Approach of creating and executing Mashups. In S. Sharma C. Godart, N. Gronau and G. Canals, editors, *Proceedings of the 9th IFIP Conference on e-Business, e-Services, and e-Society, I3E 2009*, pages 82–95. Springer, 2009.

[10] Emilian Pascalau and Clemens Rath. Managing Business Process Variants at eBay. In Jan Mendling and Mathias Weske, editors, *Proceedings of the 2nd International Workshop on BPMN, BPMN2010*. Springer, 2010.

[11] Gene Phifer. End-User Mashups Demand Governance (But Not Too Much Governance). Technical report, Gartner Research, Sep 2008.

[12] M. Rosemann, J.C. Recker, and C. Flender. Contextualisation of Business Processes. *International Journal of Business Process Integration and Management*, 3(1):47–60, 2008.

[13] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag Berlin Heidelberg, 2007.

[14] Matthias Wieland, Oliver Kopp, Daniela Nicklas, and Frank Leymann. Towards Context-aware Workflows. In *Proceedings of the Workshops and Doctoral Consortium at CAISE07*, 2007.

[15] Jonathan Zittrain. *The Future of the Internet And How to Stop It*. Yale University Press New Haven and London, 2008.

Multiple Runtime Models and their Relations for Self-Management

Thomas Vogel
thomas.vogel@hpi.uni-potsdam.de

Software systems operating under highly dynamic and unpredictable conditions must be adaptive or even self-managing. Several approaches use *one* architectural model as a causally connected runtime representation of a managed system for monitoring, analysis and performing adaptation. These architectural models are often closely related to the system's implementation, thus at a low level of abstraction and as complex as the implementation. This impedes reusability and extensibility of autonomic managers working on these models. Moreover, the models often do not cover different concerns, like security or performance, and therefore they do not simultaneously support several self-management capabilities.

In contrast, we propose a model-driven approach that provides multiple causally connected runtime models reflecting the architecture at different levels of abstraction for adapting a managed system. Each runtime model focuses on a specific concern and abstracts from the underlying system and platform leveraging reusability and extensibility of managers. The different models are maintained automatically at runtime using model-driven engineering techniques that also reduce development efforts. Besides causally connected architectural models, other kinds of models are generally used at runtime. Based on the current state of the research field, we present a categorization of runtime models, conceivable relations between those models, and how runtime management using multiple models can benefit from *megamodel* concepts.

1 Introduction

Runtime adaptability or self-manageability is often required for software systems operating under highly dynamic and unpredictable conditions [14, 32]. Self-management is enabled through a feedback loop [11,32] that usually distinguishes between a *managed system* and an *autonomic manager*. The manager monitors and analyzes a managed system, and if changes are required, adaptations are planned and executed to the system. Separating a self-managing system in a manager and a managed system, reusability and extensibility of managers are potentially supported. However, appropriate representations of a running managed system for managers are required [4,22,42].

To represent a running systems, an architectural runtime model that is *causally connected* to a managed system is often used (cf. [4, 9, 20]). The causal connection, a concept originating from the research field of *computational reflection* [35], ensures that changes in the system are reflected in the model (monitoring), and that changes in the model are reflected in the system (adaptation).

Most approaches, like [12, 22, 42], use *one* runtime model. Typically, such models are rather specific to the managed systems' implementations, complex, at a low level of abstraction, and they cover different, but mostly one concern, like performance for self-optimization. In contrast, we argue that it is beneficial to simultaneously have more than one kind of architectural model at runtime as each of them can be independent of the system's platform and implementation, less complex, at a higher level of abstraction, and it can focus on a specific concern of interest. This supports the reusability of managers working on these models and can simplify the work of the managers.

Therefore, we proposed a model-driven approach that provides multiple architectural runtime models at different levels of abstraction as a basis for the feedback loop activities of monitoring, analyzing, planning and executing adaptations [50–54]. Model-driven engineering techniques, especially the incremental and bidirectional model synchronization, are employed to automatically maintain the runtime models and to reduce the development efforts for self-managing systems. While at the *Fall 2009 Workshop of the HPI Research School* a solution for monitoring and parameter adaptation has been presented [50], this time a solution for structural adaptation will be discussed.

Moreover, our approach of using multiple runtime models for adaptation raised questions and led to a discussion about coping with interdependent and related models at runtime [5]. Therefore, we proposed to use *megamodel* concepts [55]. Based on a categorization of runtime models, we demonstrated that multiple models are likely to be used simultaneously and how megamodel concepts can help in coping with them.

The rest of the report, which summarizes in particular our work published in [51,55], is structured as follows: Section 2 discusses structural adaptation based on abstract runtime models. Section 3 presents a categorization of runtime models and how megamodel concepts help in coping with multiple runtime models and their relations. Finally, the report concludes and gives an outlook on future work in Section 4.

2 Structural Adaptation and Abstract Runtime Models

The generic architecture of our proposed approach is depicted in Figure 1. A *Managed System* realizing the business logic provides *Sensors* and *Effectors* that are interfaces for observing and adapting the system, respectively. A *Source Model* as a runtime representation of the system is provided by these interfaces. This model is causally connected to the system, such that it can be directly used by *Autonomic Managers* to perform the feedback loop activities of *monitoring* and *analyzing* the system, and of *planning* and *executing* adaptations on the system if changes are required.

However, a source model is usually complex, related to the specific implementation and platform of a managed system, and thus, rather at a low level of abstraction. Instead of working on models, like the source model, that are oriented to the solution space of a system, autonomic managers should rather work on models providing views related to problem spaces. For example, a manager responsible for self-optimization is primarily interested in optimizing the system performance and not in details about the implementation or other concerns, like failures or security. Raising models from solution to problem spaces leverages the reusability of managers that focus on problem spaces shared by different managed systems.

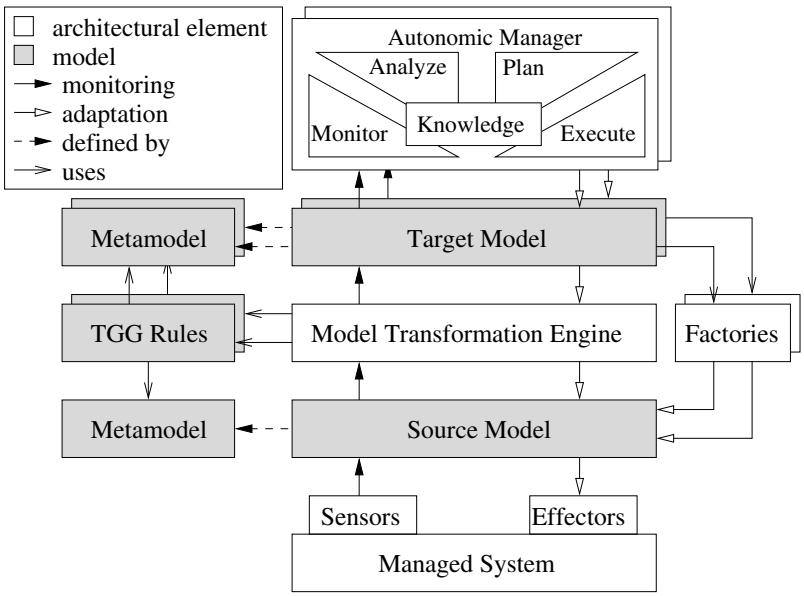


Figure 1: Generic Architecture

Therefore, several so called *Target Models* are derived from a source model at runtime. Each target model abstracts from the source model and potentially from the underlying system platform, and it provides a specific view on a system required for an addressed concern of interest, like performance in the case of self-optimization. Thus, target models are less complex than a source model and they are rather related to problem spaces.

Based on these different characteristics of source and target models as outlined in Table 1, managers preferably should use target models instead of a source model to perform the feedback loop activities. This requires that each target model is causally connected to the source model. Thus, changes in the source model are reflected in target models for monitoring, and vise versa for adaptation. Model-driven engineering (MDE) techniques are applied to maintain different target models at runtime and to realize causal connections between each target model and the source model. A *Model Transformation Engine* [23, 25] based on *Triple Graph Grammars* (TGGs) [46] incrementally synchronizes changes of the source model to the target model, and vice versa. This requires that each model conforms to potentially different user-defined *Metamodels* and that all models share the same meta-metamodel. *TGG Rules* specify declaratively at the level of metamodels how two models as instances of the corresponding metamodels are synchronized with each other.

Source Model	**Target Model**
Complex	Less Complex
Lower Level of Abstraction	Higher Level of Abstraction
Platform-Specific	Platform-Independent
Solution Space	Problem Space

Table 1: Characteristics of Source and Target Models

Synchronizing source model changes to target models for monitoring is not critical since target models are at higher levels of abstraction than a source model. During synchronization, concepts represented in a source model but not in a target model are simply discarded. This causes the intended abstraction. Recently, this monitoring approach [50, 52, 54] has been extended with structural adaptation capabilities.

Generally, there are two ways for adapting software. *Parameter adaptation* modifies variables of a program and *structural adaptation* changes the software architecture by adding, removing or replacing components and connections among components [38]. Using our approach for parameter adaptation requires that values of variables need to be synchronized from target models to a source model. As variables are generally of primitive data types, there is usually a bijective mapping between variables in the source and in the target models, which facilitates parameter adaptation as described in [50, 51]. Thus, there is no abstraction gap between source and target models regarding adaptable parameters. In contrast, we encountered and met at least three challenges for structural adaptation [51]:

(1) Refinement for Adaptation: Performing adaptations based on target models that are at higher levels of abstraction than a source model, changes performed by autonomic managers are also at a higher level of abstraction. The abstraction gap between a target and a source model can lead to a relation between both models that is only partial and not necessarily bijective. Consequently, a bidirectional synchronization between both models is hindered since information about how to refine abstract target model changes to the concrete source model could be missing. Such information has been discarded during monitoring to cause the intended abstraction of target models from the source model.

To overcome the refinement problem, *Factories* as depicted in Figure 1 are employed. If target model changes cannot be synchronized to a source model due to missing information, factories are invoked through a target model to perform these changes on the source model where the abstraction gap does not occur, followed by synchronizing these source model changes to the target models. Thus, factories are a pragmatic extension of the model transformation engine for the adaptation case if the abstraction gap between source and target models is too large. In MDE, this problem is generally discussed [29, 49] and in case of structural adaptations, it is similar to refining a software architecture [21, 40].

(2) Restrictions to Adaptation: Another challenge is the interface between a manager and a target model. The interface has to define the kind of changes that are allowed on an abstract target model and how they are performed. As a target model abstracts from a managed system, it can theoretically allow a variety of changes that however could not be executed on the system, for example due to stronger system constraints. Moreover, a target model can be changed by directly adding, removing or modifying model elements, associations among elements, or attribute values, or by invoking operations provided by elements.

This challenge has to be met for each target metamodel by specifying adaptation operators that determine a set of specific actions a manager can perform on elements of corresponding target metamodel instances. A similar solution for this challenge is applied in the *Rainbow* framework [22].

(3) Ordering of Adaptation Steps: Finally, the last challenge considers structural adaptations involving a set of atomic changes or steps that have to be synchronized at once from the target to the source model and even to the system. Rather than propagating each single target model change to the source model and system by triggering the model transformation engine, a set of changes should be propagated by one run of the engine. Usually, there exists dependencies among atomic steps, like a component should only be started when it is appropriately configured. Thus, the order of changes matters and the order of changes performed on a target model might differ from the order of performing corresponding changes to the source model or to the system. Not suitable orders might affect the consistency of a system.

Using our approach, three options are available to order adaptation steps. First, a manager can trigger the synchronization to the source model and managed system on demand between sets of atomic changes performed on a target model. All changes done before the triggering are executed on the system, which can be followed by further changes and triggers. Second, the design of TGG rules can influence the order of synchronizing model changes in one run. For example, a rule for synchronizing changes of a certain type can be designed in a way that it can only be applied if another rule and therefore a change of another type has been applied before. Thus, changes within one synchronization run can be coordinated by this option. Third, a set of source model changes performed by the model transformation engine is generally ordered for executing them on the system, which depends on the type of each change. At first, components that should be stopped are stopped, then connections and components are removed, new components are deployed, connections are created, parameter values are set, and finally, components are started. Thus, a basic ordering of adaptation steps is supported by the last option, while it is possible to determine more specific orders at design time by modeling appropriate transformation rules (cf. second option) or at runtime by triggering intermediate synchronizations (cf. first option).

Having met these challenges, our approach supports parameter and structural adaptation in addition to monitoring. The whole approach has been implemented on top of the *mKernel* infrastructure [10] that provides sensors and effectors for managing software systems being realized with *Enterprise Java Beans 3.0* (EJB)[1] technology for the *GlassFish v2*[2] application server. The metamodels, models and model-driven engineering techniques are based on the *Eclipse Modeling Framework* (EMF)[3], though the whole infrastructure can run decoupled from the *Eclipse* workbench.

Our approach of simultaneously using multiple runtime models for self-adaptive systems raised questions and led to a discussion about coping with these models at runtime since they are not independent from each other [5]. For example, any adaptation being triggered due to the performance state of a running system, which is reflected by one runtime model, might violate architectural constraints being reflected in a different model. Thus, there exists relations, like trade-offs or dependencies, between different concerns or models, which have to be considered for runtime management. In this context, we proposed concepts of *megamodels* as discussed in the following section.

[1] `http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html` (Sep 30, 2010)
[2] `https://glassfish.dev.java.net/` (Sep 30, 2010)
[3] `http://www.eclipse.org/modeling/emf/` (Sep 30, 2010)

3 Runtime Models, Relations and Megamodels

Based on the current state of the research field of runtime models, especially the past *Models@run.time* workshops [1], we present a categorization of runtime models. The categories demonstrate that multiple models are likely to be simultaneously employed at runtime since different kinds of models serve different purposes. The categorization as shown in Figure 2 supports the generality of our claim to use multiple models. Therefore, models are categorized according to their purposes and what they represent. Usually, runtime models (*M1*) of all categories are instances of metamodels (*M2*) that are specified by meta-metamodels (*M3*), which leverages typical model-driven engineering techniques, like model transformation, at runtime (cf. [47, 51, 54]).

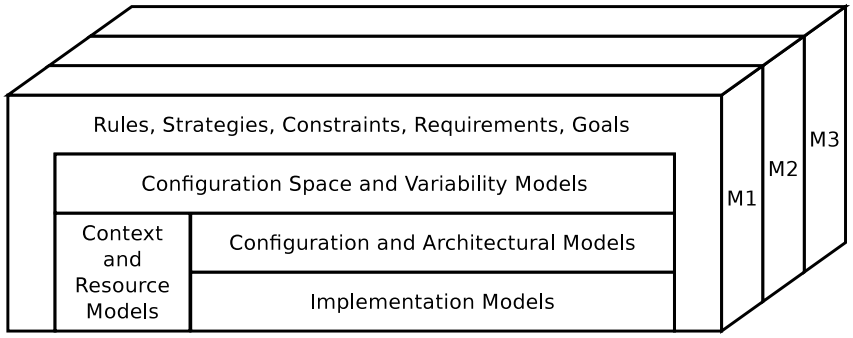


Figure 2: Categories of Runtime Models

Implementation Models describe the running system and they are similar to models used in the field of reflection that employs causally connected models based on the system's solution space (cf. [9]). Examples are models used in reflective programming languages [31, 34], or coupled to platforms like *CORBA* [15] or *EJB* [51, 54] (cf. *Source Model* in Section 2). Such models can be class or object diagrams and scenario-based sequence diagrams covering the interaction between objects or generally traces of a system [26, 31, 37]. Finally, behavioral models in the form of statecharts, state machines, or generally automatons are also used [27, 30, 36].

Configuration and Architectural Models also describe the running system, but at higher levels of abstraction than *Implementation Models*. Thus, they are often causally connected to the system. Such models are rather related to problem spaces by reflecting the current system configuration or architecture [22, 41, 42, 48, 51, 54]. Thus, these models are often similar to component diagrams, which are enhanced with elements or attributes to address non-functional properties as a basis for runtime analysis [22, 54]. At a even higher level of abstraction, process or workflow models are also feasible to provide a business-oriented view [44]. Moreover, model types of the *Implementation Models* category are also conceivable in this category, but at a higher level of abstraction. Overall, models of this category enable the self-awareness of a self-adaptive system and they are the basis for the feedback loop (cf. *Target Models* in Section 2).

Context and Resource Models describe the operational environment and context of a running system, which is especially required for systems that react to changes in their context. To represent a context, a variety of models can be used: semi-structured

tags and attributes, object-oriented or logic-based models [45], some form of variables, like key value pairs [41, 45], or even feature models [2]. Moreover, the operational environment consists of logical or physical resources a running system requires or currently uses for operation.

Configuration Space and Variability Models define potential variants of a running system, while *Configuration and Architectural Models* reflect the currently used variant. Thus, models of this category specify the system's configuration space and variability and they are used for finding adaptation points and options in a running system. Examples for such models are component type diagrams [24, 51], feature models originating from software product lines [13, 17, 41], or aspect models for *Configuration and Architectural Models* [19, 41].

Rules, Strategies, Constraints, Requirements and Goals may refer to any model of the other categories. Models in this category specify, among others, when and how a running system should be adapted. These models are reconfiguration strategies usually in some form of event-condition-action rules [2, 16, 22], or they are based on goals a running system should achieve and on utility functions guiding the adaptation [17, 41, 43]. Moreover, for runtime validation and verification, constraints on models of the other categories regarding functional and non-functional properties are employed. Constraints can be expressed in the *Object Constraint Language* (OCL), like in [28, 54], or formally in some form of *Linear Temporal Logic* (LTL), like in [27]. Though constraints can be seen as requirements that are checked at runtime, recently the idea of *requirements reflection* has emerged, which explicitly considers requirements as adaptive runtime entities that can be reflected in goal models, like KAOS [6].

The presented categories show the different roles of models at runtime and demonstrate that different kinds of models, also depending on the specific approach, are likely to be simultaneously employed. However, employing several models, these models are usually not isolated from each other, but they rather compose a network of models.

For example, Morin et al. [41] employ an architectural runtime model reflecting the running system, a feature model describing the system's variability, a context model describing the system's environment, and a so called reasoning model that can take the form of event-condition-action rules. These rules specify which feature should be (de-)activated on the architectural model depending on the context model. Thus, the employed models are related with each other and all of them are used for adaptation.

Another example is our approach presented in Section 2 using multiple target models that belong to the category of *Configuration and Architectural Models*. Each target model provides a view on the same managed system, but each one is focused on a different concern, like performance or failures. Adaptation of the system to optimize performance based on a performance target model can however interfere with other concerns covered by other target models. Thus, different concerns or their models are related with each other by means of trade-off or dependency relations. These relations have to be considered at runtime to balance competing concerns. Further kind of relations that are conceivable between runtime models are described in [55].

A similar issue of considering multiple models and relations between models occurs in the model-driven development of software. Starting from abstract models describing the requirements of a software, these models are systematically transformed and

refined to architectural, design, and implementation models until the source code level is reached [20, 39]. Thus, a multitude of development models and relations between those models exist and they have to be managed. Therefore, the idea of *megamodels* has emerged. Basically, a megamodel *is a model that contains models and relations between those models or between elements of those models* [3, 7, 8, 18].

Since existing approaches using models at runtime rather deal with multiple models and relations among those models in an ad-hoc manner, we proposed to use megamodel concepts at runtime to systematically address these issues [55]. Megamodel concepts can provide benefits for self-managing systems as megamodels are defined by metamodels, which leverages MDE techniques for managing models and relations.

Moreover, megamodels for the model-driven software development serve organizational and utilization purposes that should also be utilized at runtime. Organizational purposes are primarily about coping with the complexity of multiple models. Therefore, megamodels help in structuring and organizing different models together with their relations by storing and categorizing them. Likewise, megamodel concepts can be employed at runtime to organize and describe runtime models and their relations in the domain of self-managing systems since several related models can be simultaneously employed (cf. Section 2). This results in a kind of registry for models.

Utilization purposes of megamodels are about navigation and automation. Megamodels can be the basis for navigating through models by using relations between models. Thus, starting from a model and following relations, all related models can be reached in a model-driven manner. Navigating between models at runtime is essential for self-managing systems using multiple related models. Automation aims at increasing the efficiency by treating relations between models as executable units that take models as input and produce models as output. Therefore, a megamodel containing multiple of such units can be considered as an executable and automated process.

Thus, megamodels make relations between runtime models explicit and therefore, amenable for automated analyses. As an example, a megamodel can be used to automatically perform an impact analysis of model changes to other related models. Therefore, relations can be used to synchronize model changes to related models and these synchronized models are then analyzed to investigate the impact of the initial changes. This can be used at runtime to validate or verify a planned adaptation on different models before the managed system is actually adapted. This idea might be applicable to the different target models in our approach presented in Section 2.

4 Conclusion and Future Work

In this report our approach of simultaneously using multiple runtime models for adaptive and self-managing systems has been discussed. Our approach and the presented categorization of runtime models demonstrate that multiple runtime models are likely to be used simultaneously. Moreover, initial ideas of applying megamodel concepts to cope with multiple runtime models that are related with each other have been outlined.

As future work, the following points are considered. A solution for coordinating several autonomic managers working on different models is required to balance competing adaptations. Moreover, extending our approach to a distributed setting, which includes

mechanisms to synchronize distributed models, is currently work in progress. An initial prototype that incrementally synchronizes distributed models sharing the same metamodel has been realized on top of the *Open Message Queue*[4] middleware. This prototype must be enhanced with support for transactions and model versioning to ensure consistency. Moreover, the design of autonomic managers will be elaborated by investigating which runtime models can be used for which management tasks, and how the models and tasks can be structured. Therefore, our categorization of runtime models and the generic architecture for self-managing systems proposed by Kramer and Magee [33] might be the starting points. Furthermore, our categorization of runtime models should be refined by including other preliminary classification approaches, like [4, 9, 20]. Finally, when using runtime models as interfaces within the architecture of autonomic managers, and for monitoring and adapting managed systems, the semantics of runtime models and operations on these models must be clearly defined. Ontologies and work on model-driven interoperability form a basis for this issue.

References

[1] *Workshop on Models@run.time*. `http://www.comp.lancs.ac.uk/~bencomo/MRT/`, 2006-2009.

[2] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 89–98, 2009.

[3] M. Barbero, M. D. D. Fabro, and J. Bézivin. Traceability and Provenance Issues in Global Model Management. In *ECMDA-TW'07: Proc. of 3rd Workshop on Traceability*, pp. 47–55, 2007.

[4] N. Bencomo. On the Use of Software Models during Software Execution. In *Proc. of the ICSE Workshop on Modeling in Software Engineering*, pp. 62–67. IEEE, 2009.

[5] N. Bencomo, G. Blair, R. France, F. Munoz, and C. Jeanneret. 4th International Workshop on Models@run.time. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009*, vol. 6002 of *LNCS*, p. 119–123. Springer, 2010.

[6] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier. Requirements reflection: requirements as runtime entities. In *Proc. of the 32nd ACM/IEEE Intl. Conference on Software Engineering (ICSE)*, pp. 199–202. ACM, 2010.

[7] J. Bézivin, S. Gérard, P.-A. Muller, and L. Rioux. MDA components: Challenges and Opportunities. In *1st Intl. Workshop on Metamodelling for MDA*, pp. 23–41, 2003.

[8] J. Bézivin, F. Jouault, and P. Valduriez. On the Need for Megamodels. In *Proc. of the OOPSLA/GPCE Workshop on Best Practices for Model-Driven Software Development*, 2004.

[9] G. Blair, N. Bencomo, and R. B. France. Models@run.time. *IEEE Computer*, 42(10):22–27, 2009.

[10] J. Bruhn, C. Niklaus, T. Vogel, and G. Wirtz. Comprehensive support for management of Enterprise Applications. In *Proc. of the 6th ACS/IEEE Intl. Conference on Computer Systems and Applications*, pp. 755–762. IEEE, 2008.

[11] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Mueller, M. Pezze, and M. Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, vol. 5525 of *LNCS*, pp. 1–26. Springer, 2009.

[12] M. Caporuscio, A. D. Marco, and P. Inverardi. Model-based system reconfiguration for dynamic performance management. *Journal of Systems and Software*, 80(4):455 – 473, 2007.

[13] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *IEEE Computer*, 42(10):37–43, 2009.

[14] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, and et al. Software Engineering for Self-Adaptive Systems: A Research Road Map. In *Software Engineering for Self-Adaptive Systems*, vol. 5525 of *LNCS*, pp. 48–70. Springer, 2009.

[15] F. Costa, L. Provensi, and F. Vaz. Towards a More Effective Coupling of Reflection and Runtime Metamodels for Middleware. In *Proc. of 1st Intl. Workshop on Models@run.time*, 2006.

[16] J. Dubus and P. Merle. Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-based Systems. In *Proc. of 1st Intl. Workshop on Models@run.time*, 2006.

[17] A. Elkhodary, S. Malek, and N. Esfahani. On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 41–50, 2009.

[18] J.-M. Favre. Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of The Fidus Papyrus and of The Solarus. In *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, 2005.

[19] N. Ferry, V. Hourdin, S. Lavirotte, G. Rey, J.-Y. Tigli, and M. Riveill. Models at Runtime: Service for Device Composition and Adaptation. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 51–60, 2009.

[4] `https://mq.dev.java.net/` (Sep 30, 2010)

[20] R. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *Proc. of the ICSE Workshop on Future of Software Engineering (FOSE)*, pp. 37–54. IEEE, 2007.

[21] D. Garlan. Style-Based Refinement for Software Architecture. In *Joint Proc. of the 2nd Intl. Software Architecture Workshop and Intl. Workshop on Multiple Perspectives in Software Development*, pp. 72–75. ACM, 1996.

[22] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[23] H. Giese and S. Hildebrandt. Incremental Model Synchronization for Multiple Updates. In *Proc. of the 3rd Intl. Workshop on Graph and Model Transformation*. ACM, 2008.

[24] H. Giese, A. Seibel, and T. Vogel. A Model-Driven Configuration Management System for Advanced IT Service Management. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 61–70, 2009.

[25] H. Giese and R. Wagner. From Model Transformation to Incremental Bidirectional Model Synchronization. *Software and Systems Modeling*, 8(1), 2009.

[26] T. Gjerlufsen, M. Ingstrup, J. Wolff, and O. Olsen. Mirrors of Meaning: Supporting Inspectable Runtime Models. *IEEE Computer*, 42(10):61–68, 2009.

[27] H. J. Goldsby, B. H. Cheng, and J. Zhang. AMOEBA-RT: Run-Time Verification of Adaptive Software. In *Models in Software Engineering: Workshops and Symposia at MoDELS 2007*, vol. 5002 of *LNCS*, pp. 212–224. Springer, 2008.

[28] C. Hein, T. Ritter, and M. Wagner. System Monitoring using Constraint Checking as part of Model Based System Management. In *Proc. of 2nd Intl. Workshop on Models@run.time*, 2007.

[29] T. Hettel, M. J. Lawley, and K. Raymond. Model Synchronisation: Definitions for Round-Trip Engineering. In *Proc. of the 1st Intl. Conference on Model Transformation*, pp. 31–45, 2008.

[30] E. Höfig, P. H. Deussen, and H. Coskun. Statechart Interpretation on Resource Constrained Platforms: a Performance Analysis. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 99–108, 2009.

[31] F. Jouault, J. Bézivin, R. Chevrel, and J. Gray. Experiments in Run-Time Model Extraction. In *Proc. of 1st Intl. Workshop on Models@run.time*, 2006.

[32] J. Kephart and D. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[33] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *Proc. of the ICSE Workshop on Future of Software Engineering (FOSE)*, pp. 259–268. IEEE, 2007.

[34] A. Kuhn and T. Verwaest. FAME - A Polyglot Library for Metamodeling at Runtime. In *Proc. of the 3rd Intl. Workshop on Models@run.time*, pp. 57–66. Technical Report COMP-005-2008, Lancaster University, 2008.

[35] P. Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, 1987.

[36] S. Maoz. Model-Based Traces. In *Proc. of the 3rd Intl. Workshop on Models@run.time*, pp. 16–25. Technical Report COMP-005-2008, Lancaster University, 2008.

[37] S. Maoz. Using Model-Based Traces as Runtime Models. *IEEE Computer*, 42(10):28–36, 2009.

[38] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing Adaptive Software. *IEEE Computer*, 37(7):56–64, 2004.

[39] S. J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 2004.

[40] M. Moriconi, X. Qian, and R. Riemenschneider. Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, 1995.

[41] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg. Models@Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51, 2009.

[42] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based Runtime Software Evolution. In *Proc. of the 20th Intl. Conference on Software Engineering*, pp. 177–186. IEEE, 1998.

[43] A. J. Ramirez and B. H. Cheng. Evolving Models at Run Time to Address Functional and Non-Functional Adaptation Requirements. In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 31–40, 2009.

[44] M. Sanchez, I. Barrero, J. Villalobos, and D. Deridder. An Execution Platform for Extensible Runtime Models. In *Proc. of the 3rd Intl. Workshop on Models@run.time*, pp. 107–116. Technical Report COMP-005-2008, Lancaster University, 2008.

[45] D. Schneider and M. Becker. Runtime Models for Self-Adaptation in the Ambient Assisted Living Domain. In *Proc. of the 3rd Intl. Workshop on Models@run.time*, pp. 47–56. Technical Report COMP-005-2008, Lancaster University, 2008.

[46] A. Schürr. Specification of graph translators with triple graph grammars. In *Proc. of the 20th Intl. Workshop on Graph-Theoretic Concepts in Computer Science*, vol. 903 of *LNCS*, pp. 151–163. Spinger, 1994.

[47] H. Song, G. Huang, F. Chauvel, and Y. Sun. Applying MDE Tools at Runtime: Experiments upon Runtime Models. In *Proc. of the 5th Intl. Workshop on Models@run.time*, vol. 641 of *CEUR-WS.org*, pp. 25–36, 2010.

[48] H. Song, Y. Xiong, F. Chauvel, G. Huang, Z. Hu, and H. Mei. Generating Synchronization Engines between Running Systems and Their Model-Based Views. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009*, vol. 6002 of *LNCS*, pp. 140–154. Springer, 2010.

[49] P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and Systems Modeling*, 9(1):7–20, 2010.

[50] T. Vogel. Models at Runtime for Monitoring and Adapting Software Systems. Technical Report 31, Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering, Hasso Plattner Institute, University of Potsdam (Fall 2009 Workshop), 2010.

[51] T. Vogel and H. Giese. Adaptation and Abstract Runtime Models. In *Proc. of the 5th ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 39–48. ACM, 2010.

[52] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. Incremental Model Synchronization for Efficient Run-time Monitoring . In *Proc. of the 4th Intl. Workshop on Models@run.time*, vol. 509 of *CEUR-WS.org*, pp. 1–10, 2009.

[53] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In *Proc. of the 6th Intl. Conference on Autonomic Computing and Communications*, pp. 67–68. ACM, 2009.

[54] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. Incremental Model Synchronization for Efficient Run-Time Monitoring. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009*, vol. 6002 of *LNCS*, pp. 124–139. Springer, 2010.

[55] T. Vogel, A. Seibel, and H. Giese. Toward Megamodels at Runtime. In *Proc. of the 5th Intl. Workshop on Models@run.time*, vol. 641 of *CEUR-WS.org*, pp. 13–24, 2010.

Recent Developments in JCop – Context-oriented Concurrency Control and Compiler Optimization

Malte Appeltauer

Software Architecture Group
Hasso-Plattner-Institut
malte.appeltauer@hpi.uni-potsdam.de

Context-oriented Programming is a technique for dynamic adaptation of software systems. In the course of our research project we applied the approach to the Java programming language, adapted to specific properties of statically typed languages, and evaluated in several application domains including server-based and distributed applications. Our main goal is to provide programming language and tool support to meet the needs for dynamic adaptation of complex, distributed systems. This report presents our recent research results on context-oriented programming: a case-study to support context-dependent concurrency control and a implementation study on novel virtual machine support for dynamic invocation in Java 7.

1 Introduction

In object-oriented programming, functionality is encapsulated in methods that can be refined through inheritance mechanisms such as overriding. Inheritance supports behavior specification in fixed hierarchies. However, dynamic variations cannot be explicitly represented this way. Context-oriented Programming (COP) adds an orthogonal dimension to inheritance by addressing the explicit representation of *behavioral variations* and their *dynamic composition*. In COP, behavioral variations are functionality that should be executed in addition or instead of the base functionality. Behavioral variations are encapsulated into layers that can be de-/activated at runtime.

In the course of our research project we applied concepts of COP to the Java programming language. We evaluated this approach with several case studies, including mobile and ubiquitous computing [6], user interface applications [4], and distributed, server-based systems. Based on our experiences developing applications in these domains, we developed new layer declaration features and adaptation scopes. We implemented our concepts in prototypes and compiler-based language extensions [3, 6]. *JCop* [5], our current context-oriented Java language extension, fuses features of context- and aspect-oriented programming to support event-based COP and provides first-class context objects. Besides several applications, we implement an RMI (*remote method invocation*) framework supporting distributed programming with JCop.

In this paper we will report on our research activity in the past six month. Part of this work is to compare the expressiveness of JCop to alternative languages concerned with separation of concerns and dynamic adaptation. As an example, we introduce a DSL for concurrency control that we are currently implementing with JCop.

The JCop compiler translates source code to Java bytecode. This process contains a transformation of a JCop AST (*abstract syntax tree*) into a Java AST. JCop's dynamic adaptation constructs are henceforth internally represented by Java elements.

Since Java's support for dynamic adaptation has been limited until Java 6, a performance bottleneck of our compiler implementation is run-time maintenance and lookup of layers. We describe an implementation study, in which we use the upcoming *InvokeDynamic* bytecode instruction – to be supported by the standard Java virtual machine starting with the release of Java 7 – to implement layered method dispatch. We compare the resulting implementation approach with that of the existing JCop compiler.

The rest of the report is structured as follows. Section 2 presents our approach of context-oriented concurrency control. Section 3 describes Java's new feature InvokeDynamic and its application within the JCop compiler implementation. Section 4 summarizes the paper and discusses next steps.

2 Context-oriented Concurrency Control

Throughout this research project, we apply our new language mechanisms to several use cases to validate their usability. Currently, we are evaluating COP mechanisms for concurrency control[1]. With the wide-scale deployment of multi-core processors and the accompanying demand for parallel software, concurrency control receives increasing attention. Classic concurrency control guards critical resources with locking mechanisms [10] such as semaphores [9] and monitors [14]. However, handling these primitives in large systems can be difficult since they require a low-level implementation rather than policy specification. Some alternative approaches, such as *view-based concurrency control*, attempt to overcome this limitation.

2.1 View-based Concurrency

Literature describes approaches dedicated to provide a more abstract perspective on concurrency rules that, for example, support specification on object interface level. The *views* approach [8] introduces a new concept to access resources. A *view* declares both a partial object interface and a list of incompatible object views. A *partial object interface* lists a subset of the object's methods and fields; a thread must hold the given view to access the part of the object's interface protected by the view.

Concurrency control is provided by enforcing incompatibility specification of views. Two views are incompatible if two different threads cannot simultaneously hold both views on the same object. With that, views can statically detect many data races. Views have been implemented as domain-specific language (DSL) extension to Java.

[1] This is joint work with Alexander Schmidt

```
1  view read {
2    incompatible write, rezize;
3    size, capacity, array : readonly;
4
5
6
7    get(int i);
8    capacity();
9  }
10
11 public void someMethod() {
12   acquire (this@read)  {
13     someValue = get(somePos);
14   }
15 }
```

```
public layer Read
  without Write, Resize {
    public int getSize() {return size;}
    public int getCapacity() {return capacity;}
    public int[] getArray() {return array;}

    public int get(int i) {...}
    public int capacity() {...}
}

public void someMethod() {
  with(Read) {
    someValue = get(somePos);
  }
}
```

Figure 1: Implementations of partial resource interfaces using the views DSL [8] (*left*) and JCop (*right*).

Figure 1 (left) presents an example for a view `read` that provides access to three fields and a method `get`. To access `get`, one needs to acquire `read` for the dynamic extend of a method block.

2.2 JCop Approach

We are currently developing a JCop-based context-oriented concurrency control subsuming a dedicated views-like DSL. COP resembles most language features introduced by views, given a COP language that supports the specification of layer incompatibilities. Views, as units of modularization of partial interfaces, correspond to layers. Views contain class members that are accessible by a thread that acquires the view, much like partial methods and fields are activated with their enclosing layer. Hence, we are using layers as a means to realize a views-like resource representation. In addition to partial method definitions that adapt base methods at run-time, layers in JCop can also contain *layer-local* methods that are only accessible if their enclosing layer is active. An object that attempts to access such method outside an activation of the method's layers will cause a run-time exception. We are employing this features to define partial object interfaces. A view can define incompatibilities to other views using the `incompatible` keyword. The specification of layer (resource) incompatibilities can be expressed by feature descriptions [17]. A declaration of feature descriptions for layers has already been developed for the Lisp-based *ContexL* [7]; we will adopt this work and carry the ideas forward to JCop. The right listing of Figure 1 declares the layer `Read` to be incompatible with `Write` and `Resize` using the `without` keyword. The views DSL allows for restricting field access to `none`, `readonly`, and `readwrite`. In our JCop implementation, we only allow indirect control the field access through the accessor methods. For instance, to declare access to `size`, `capacity`, and `array` to be read only, the layer shown in Figure 1 (right) provides getter methods. Figure 1 compares a view- and layer-based implementation of a resource interface. It shows the declaration of a layer `Read` that excludes the joint activation with `Write` and `Resize`.

View allocation boils down to layer activation and can be expressed – with a slight

extension – by JCop's `with` statement. Like JCop, the statement `acquire(this@read)` in Figure 1 (left) acquires `read` for its dynamic extent. However, `read` is only activated for a specific object (`this`) within this control flow. In JCop, a layer `Read` would be activated for all objects within the dynamic extent. We will support a finer-grained layer composition mechanism that allows for instance specific layer activations within control flows.

At present, our context-oriented concurrency control implementation is not complete and requires some further refinements of JCop. The restriction of a layer activation to one thread at time – which is the core requirement of concurrency control – cannot be specified by JCop at present. One solution that we are pursuing in our ongoing work is to declare a layer to be `atomic`, meaning that its activation is mutual exclusive.

3 Compiler Optimization

JCop is implemented as a JastAdd [12] compiler extension, offering a convenient syntax extension and generating plain Java bytecode. JCop's layer-aware method dispatch performs a thread-local lookup of the appropriate partial method to be called in a composition object. This lookup is implemented by means of plain Java such as thread local objects, hash tables, and inheritance. The upcoming Java 7 release includes a powerful extension to the language's meta programming facilities: the *invoke dynamic* (ID) instruction[2] [15]. With ID and the accompanying Java compiler extensions, it is possible to generate method calls that are resolved at run-time by user-provided code. In other words, ID introduces the ability to specify arbitrary method lookup semantics – including changing late-bound methods.

In previous work [2], we conducted several micro-benchmarks to measure possible performance decreases of our implementation of layer-aware method lookup. Results showed a certain execution overhead of layer-aware lookup compared to a naïve Java implementation of the same behavior. Optimization possibilities of the internal Java representation generated by our compiler are restricted. Especially, thread-local access to layer composition is expensive even when caching mechanisms are used. Therefore, we investigated the applicability of Java's new dedicated support for dynamic invocation to our JCop compiler implementation. In the following we present a working implementation of layered method dispatch using ID as an alternative to the current JCop architecture. The implementation is a hand-made proof-of-concept style feasibility study; an adoption of the compiler generation has to be implemented in future work.

3.1 Invoke Dynamic Bytecode Instruction

The ID instruction, defined in Java specification request 292, supports the insertion of late-bound method calls in Java code that the VM resolves at run-time, using user-

[2] `http://jcp.org/en/jsr/detail?id=292`

provided resolution logic[3]. In Java source code, a method call of the form

```
1 InvokeDynamic.<T>message(arg1, arg2, arg3)
```

denotes sending a `message`, passing the given arguments, and interpreting the result as having the type `T`. No assumptions about the arguments are made; it is *not* the case that `arg1` is necessarily interpreted as the message receiver.

Each such dynamic call corresponds to precisely one `CallSite` object carrying information about the invocation. Most importantly, a `CallSite` instance stores the particular method to which the site is bound. Since that method is not known prior to run-time, the programmer is required to provide a so-called *bootstrap method* per class containing dynamic invocations. The VM will invoke the bootstrap method as it encounters dynamic calls for the first time during execution. Bootstrap methods are responsible for creating and returning the respective `CallSite` instances, and they can also bind those call sites to target methods. While the application is running, it is possible to re-bind call sites by assigning new target methods.

The method `messageImpl()` simply prints the passed `int` to standard output. The first parameter from the ID call, `this`, is implicitly used as the receiver of the virtual method call. Note that this does not have to be the case: had the ID call site been bound to a static method instead, it would have had to accept two parameters as given in the ID instruction in `go()`.

Even though the binding is established once the bootstrap method returns the initialized `CallSite`, this does not mean that it is fixed. It is always possible to send `setTarget()` to the `CallSite` to bind it to a new target method.

3.2 Implementing Layer-aware Method Lookup

JCop's method dispatch has been described in previous publications [1, 3]. In the following, we will sketch up that implementation and describe an ID-based implementation, and discuss the conceptual differences in both approaches. We show both implementations along the classic COP example of a class `Person` that provides a layered method `toString` for which a partial method is defined in the layer `Address` [13]. The JCop implementation of this example is shown in Figure 2, where the address layer is implemented as a member of `Person`[4].

3.2.1 JCop Mapping

During compilation, a JCop program's AST (abstract syntax tree) is transformed into a plain Java AST that contains additional auxiliary classes and methods. Since our experimental ID-based implementation only supports this core features, we only refer to core COP features that have been developed in our previous work on ContextJ [3] including layer and partial method definitions as class members and control-flow specific layer composition blocks. For illustration, Figure 2 shows a source code representation

[3]For details on the API that we describe here, we refer to `http://java.sun.com/javase/7/docs/api/java/dyn/package-summary.html`.

[4]JCop supports layer definition within classes and as top level module [5].

```
public class Person {
  private String name, address;
  public Person(String name, String addr) {
    this.name = name;
    this.address = addr;
  }
  public String getAddress() {
    return this.address;
  }
  public String toString() {
   return name;
 }
 layer Address {
   public String toString() {
      return proceed() + ", " + getAddress();
    }
  }
}
```

```
public class App {
  void m() {
    Person p = new Person("Bob");
     with(Address) {
       System.out.println(p.toString());
     }
  }
}
```

Figure 2: Person example in JCop.

```
public class Person {
  ...
  public String toString() {
    Composition c = Composition.getCurrent();
    return c.first().Person$$toString(this, c);
  }
  public String toString(Address l, Composition c) {
    String addr = ", " + getAddress();
    return c.next(l).Person$$toString(this, c) + addr;
  }
  public String toString$$base() {
    return getName();
  }
}

public class App {
  void m() {
    Person p = new Person("Bob");
    Composition comp = Composition.getCurrent();
    Composition old = comp.addLayer(Address);
    try {
      System.out.println(p.toString());
    }
    finally {
      Composition.setCurrent(old);
    }
  }
}
```

```
public class Layer {
  ...
  public String Person$$toString(Person tar,
                                 Composition c) {
    return tar.toString$$base();
  }

  public class ConceteLayer extends Layer {
    ...
    public String Person$$toString(Person tar,
                                   Composition c) {
      return c.next(this).Person$$toString(tar, c);
    }
  }

  public class Address extends ConcreteLayer {
    ...
    public String Person$$toString(Person tar,
                                   Composition c) {
      return tar.toString(this, c);
    }
  }

  public class Composition {
    ...
  }
```

Figure 3: Internal representation of layer-aware method dispatch in JCop.

of the generated byte code for our `Person` example. For brevity, the following listings do not present the `Composition` methods `getCurrent()` and `setCurrent(Composition)` that provide access to the thread local composition object, `first()` and `next()` that allow for its traversal, and `addLayer(Layer)` that adds a layer to a composition. Layers are transformed into classes; each application-specific layer is a subtype of `ConcreteLayer`, which in turn inherits from `Layer`. They provide *delegation methods* for their partial methods; the actual behavior of partial methods are implemented by *layer methods* which are defined in their corresponding class and called by their delegation methods. At run-time, a composition object holds references to activated layers. It is accessed upon layered method execution to decide to which of the method's variations execution should be delegated. The lookup algorithm is implemented as follows:

```
1  public class Person {
2    ...
3
4    public String toString() {
5     InvokeDynamic.<void>Person$$toString(this);
6    }
7    public String toString$$Address() {
8      InvokeDynamic.
9      <void>#"proceed:Person$$toString:Address"(this)
10     + ", " + getAddress();
11   }
12   public String toString$$base(){
13     return getName();
14   }
15 }
16
17 public class App {
18   void m() {
19     Person p = new Person("Bob");
20     with(Address);
21     System.out.println(p.toString());
22     without(Address);
23   }
24 }
25
26 public class Layer extends Layer {
27   ...
28 }
29
30 public class ConcreteLayer extends Layer {
31   ...
32 }
33
34 public class Address extends ConcreteLayer {
35   public MethodHandle[] getPartialMethods() {
36     ...
37   }
38   public String getName() {
39     ...
40   }
41 }
42
43
44
45
```

```
public class Composition {
  private static Hashtable<String, CallSite> cSites =
    new Hashtable<String, CallSite>();
  ...
  public static void with(Layer layer) {
    Composition.getCurrent().addLayer(layer);
    updateTarget(layer);
    updateProceedChain(next(layer));
  }
  public static void updateTarget(Layer l) {
    for(MethodHandle handle : l.getHandles()) {
      CallSite cs = cSites.get(handle.toString());
      if(cs != null) {
        MethodType handleType = handle.type();
        MethodType rType =
        MethodType.methodType(handleType);
        String mName = cs.name() + "_" + name;
        MethodHandle target = MethodHandles.lookup().
        findVirtual(cs.callerClass(), mName, rType);
        cs.setTarget(target);
      }
    }
  }
  private static updateProceedChain(Layer layer){
    // implementation similar to updateTarget();
  }
  private static CallSite createCallSite(Class c,
                    String name, MethodType type) {
    CallSite cs = new CallSite(c, name, type);
    MethodHandle h = MethodHandles.lookup()
     .findVirtual(c, name, type.dropParameterTypes(0,1));
    callSites.put(h.toString(), cs);
    Layer first = Composition.getCurrent().first();
    updateTarget(first.getName(), h);
    return cs;
  }
  public static void register(Class c) {
    MethodType bt =
      MethodType.methodType(CallSite.class, Class.class,
        String.class, MethodType.class);
    MethodHandle bm = MethodHandles.lookup().findStatic(
    Composition.class, "createCallSite", bt));
    Linkage.registerBootstrapMethod(c, bm);
  }
}
```

Figure 4: Representation of layer-aware method dispatch using INVOKEDYNAMIC.

- If no layer is active, a composition only consists of one `Layer` element denoting the base layer. For each layered method `m` declared in class `C`, `Layer` provides a delegation method that simply calls `m`'s base definition (which is declared in its corresponding class).
- If the first layer in the composition provides a partial method for `m`, it contains a delegation method that calls the partial method representation in `C`.
- If the current layer does not provide a partial method, the composition list must be traversed. Since the current layer does not override `m`'s delegation method, the definition of the super class `ConcreteLayer` is executed, which simply delegates the call to the next layer of the composition.

Layer activation is implemented by two static methods that replace the `with`/`without` blocks and allow to add and remove items from the list.

3.2.2 INVOKEDYNAMIC Implementation

In the JCop implementation, every execution of a layered method performs the aforementioned layer-aware method dispatch. Thus, behavioral variations are adapted as

late as possible. This design decision allows for the definition of flexible, “late bound” lookup mechanisms such as event-based layer activation [5].

However, this strategy might cause performance penalties if methods are frequently executed with the same composition chain. Furthermore, this late-bound lookup is not required to support basic COP features. Alternatively, the lookup table can be manipulated on composition change, for which we can employ ID. We developed an ID-based version of layer lookup for Java for which we adapted our JCop implementation. Figure 4 presents the source code of our running example using ID. We modified JCop lookup logic as follows:

- Layered methods delegate to an ID object, instead of performing a composition lookup. On first execution, a bootstrap method creates a `CallSite` object that is stored in a thread-local hash map in `Composition`.
- Layers provide a `MethodHandle` list for their partial methods instead of delegation methods. This list can be collected and generated at compile time using static analysis.
- Composition statements (`with`/`without`) trigger recomposition in terms of updating call site target objects. First, all partial method call sites defined by layers included in the new composition are collected. The required call sites are gathered using the hash table of `Composition`, where the `MethodHandle` objects of the layers are used as keys. Finally, a new `MethodHandle` pointing to the partial method definition (methods with suffix ‘_*<layer name>*’) is set as target for the call sites.
- The `proceed` pseudo method is also implemented by ID calls and points to the next partial method of the composition. Their target objects are redirected by calling the methods `with` and `without`.

The complete ID-based COP implementation consists of some more classes that are not shown here for brevity. The application-specific implementation presented in Figure 4 contains boilerplate code, such as defining layer classes and method handles for their partial methods, dynamic invocation statements and explicit layer (de)activation statements that developers should not have to care about. However, such code could be eventually be generated by a JCop compiler.

4 Summary and Next Steps

Context-oriented programming is an approach for dynamic adaptation. In previous work, we developed JCop, a Java language extension providing COP’s core features and additional support for declarative, event-based adaptation. In this report, we describe a case-study implementing context-oriented concurrency control and a compiler optimization employing a novel Java virtual machine support for dynamic invocation. Next steps will focus on tool support and evaluation of JCop.

Language Evaluation In addition to our case studies, we will apply JCop to a large scenario including implementation of new application features, refactorings, and testing. We will validate this application development process with respect to modularity metrics and code comprehension.

Formal Semantics We will provide a JCop language specification and a formal semantics. Related work already describes a semantics for *cj*, a minimal COP language for the *delMDSoc* environment [11, 16]. Cj's features are similar to JCop's core features but its declarative adaptation mechanism and first-class context objects have not been considered, yet.

Enhanced Tool Support We will enhance JCop's tool support. COP and other approaches to multi dimensional separation of concerns allow for more concise and declarative specifications of dynamic crosscutting concerns than pure object-orientation. For instance, COP extends object-oriented method dispatch to consider context information in addition to the method's signature and receiver. This new dimension requires development tools that support comprehension of program behavior and source code navigation.

References

[1] Malte Appeltauer. ContextJ – Context-oriented Programming for Java. In *Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering*, number 27. Hasso-Plattner-Institut, Potsdam, Germany, 2009.

[2] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, Jens Lincke, and Michael Perscheid. A Comparison of Context-oriented Programming Languages. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–6, New York, NY, USA, 2009. ACM Press.

[3] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, and Hidehiko Masuhara. ContextJ - Context-oriented Programming for Java. *Computer Software of The Japan Society for Software Science and Technology*, 1:20, 2010.

[4] Malte Appeltauer, Robert Hirschfeld, and Hidehiko Masuhara. Improving the Development of Context-dependent Java Applications with ContextJ. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–5, New York, NY, USA, 2009. ACM Press.

[5] Malte Appeltauer, Robert Hirschfeld, Hidehiko Masuhara, Michael Haupt, and Kazunori Kawauchi. Event-specific Software Composition in Context-oriented Programming. In *Proceedings of International Conference on Software Composition*, Lecture Notes in Computer Science, pages 50–65, Berlin, Heidelberg, Germany, 2010. Springer-Verlag.

[6] Malte Appeltauer, Robert Hirschfeld, and Tobias Rho. Dedicated Programming Support for Context-aware Ubiquitous Applications. In *UBICOMM 2008: Proceedings of the 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 38–43, Washington, DC, USA, 2008. IEEE Computer Society Press.

[7] Pascal Costanza and Theo D'Hondt. Feature Descriptions for Context-oriented Programming. In Steffen Thiel and Klaus Pohl, editors, *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*, pages 9–14. Lero Int. Science Centre, University of Limerick, Ireland, 2008.

[8] Brian Demsky and Patrick Lam. Views: Object-Inspired Concurrency Control. In Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel, editors, *ICSE 2010, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 395–404. ACM, 2010.

[9] Edsger W. Dijkstra. Cooperating sequential processes. Technical Report EWD-123, Technical University Eindhoven, Eindhoven, the Netherlands, 1965.

[10] Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, September 1965.

[11] Michael Haupt and Hans Schippers. A Machine Model for Aspect-Oriented Programming. In Erik Ernst, editor, *ECOOP 2007, 21st European Conference on Object-Oriented Programming*, volume 4609 of *Lecture Notes in Computer Science*, pages 501–524, Berlin, Heidelberg, Germany, August 2007. Springer-Verlag.

[12] Görel Hedin and Eva Magnusson. JastAdd: An Aspect-oriented Compiler Construction System. *Science of Computer Programming*, 47(1):37–58, 2003.

[13] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3):125–151, March-April 2008.

[14] Charles A. R. Hoare. Monitors: an operating system structuring concept. *Communications of the ACM*, 17(10):549–557, 1974.

[15] John R. Rose. Bytecodes meet combinators: invokedynamic on the jvm. In *VMIL'09: Proceedings of the Third Workshop on Virtual Machines and Intermediate Languages*, pages 1–11, New York, NY, USA, 2009. ACM.

[16] Hans Schippers, Dirk Janssens, Michael Haupt, and Robert Hirschfeld. Delegation-based Semantics for Modularizing Crosscutting Concerns. *ACM SIGPLAN Notices*, 43(10):525–542, 2008.

[17] Arie van Deursen and Paul Klint. Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, 10:2002, 2001.

Towards Service-Oriented, Standards- and Image-Based Styling of 3D Geovirtual Environments

Dieter Hildebrandt
dieter.hildebrandt@hpi.uni-potsdam.de

The server-side, service-oriented rendering of massive *3D geovirtual environments* (3DGeoVEs) has the potential to enable users with lightweight clients to access high quality, image-based visual representations of the environments without having to download massive amounts of geodata. *Image post-processing* (IPP) is a well-known concept that allows for decoupling the process of generating an image from applying enhancement processes to already generated images. Moreover, several visual effects are implemented most efficiently and effectively as image post-processes. However, so far no proposals exist for providing image post-processing of 2D images of projective views of visual representations of 3DGeoVE in a SOA and based on standards.

In this paper, we investigate how IPP functionality and the functionality of styling of visual representations of 3DGeoVE based on IPP can be provided in a SOA based on standards. First, we introduce the concepts of IPP and styling. Then, we present an analysis of different characteristics of styling and IPP and design dimensions relevant when building a system for styling and IPP. From the analysis, we derive a set of requirements for a concept and system providing IPP and styling functionality based on IPP. We present a preliminary concept for a system meeting the identified requirements and report on initial implementation and evaluation results.

1 Introduction

The server-side, service-oriented rendering of massive *3D geovirtual environments* (3DGeoVEs) has the potential to enable users with lightweight clients to access high quality, image-based visual representations of the environments without having to download massive amounts of geodata. *Image post-processing* (IPP) [2] is a well-known concept that allows for decoupling the process of generating an image from applying enhancement processes to already generated images. Moreover, several visual effects are implemented most efficiently and effectively as image post-processes. In a *service-oriented architecture* (SOA), providing the functionality of image post-processors as dedicated, loosely coupled services as an application of the principle of separation of concerns offers several advantages. These services easily can be reused and recomposed with further services to form different distributed applications, the increased modularity has the potential to improve the maintainability and flexibility of the resulting systems, and existing applications and services can be extended to take advantage

of image post-processing functionality without having to implement the functionality themselves. Providing this functionality based on open standards such as approved by the W3C and the *Open Geospatial Consortium* (OGC) [7]) facilitates building effective, open and interoperable systems. However, so far no proposals exist for providing image post-processing of 2D images of projective views of visual representations of 3DGeoVE in a SOA and based on standards.

In this paper, we investigate how IPP functionality and the functionality of styling of visual representations of 3DGeoVE based on IPP can be provided in a SOA based on standards. First, we introduce the concepts of IPP and styling. Then, we present an analysis of different characteristics of styling and IPP and design dimensions relevant when building a system for styling and IPP. From the analysis, we derive a set of requirements for a concept and system providing IPP and styling functionality based on IPP. We present a preliminary concept for a system meeting the identified requirements and report on initial implementation and evaluation results.

This paper is structured as follows. In Section 2, we introduce the fundamentals of IPP and styling. We present an analysis of characteristics and design dimensions of IPP and styling and a set of derived requirements in Section 3. The preliminary design of the concept is presented in Section 4, initial implementation and evaluation results in Section 5. Section 6 concludes this paper with a summary, conclusions and next steps.

2 Fundamentals

2.1 Image Post-Processing

Digital image processing encompasses processes whose inputs and outputs are images and, in addition, encompasses processes that extract attributes from images, up to and including the recognition of individual objects [9]. In the context of computer graphics, performing image processing after rendering is called *image post-processing* (*IPP*) [2]. We define *IPP effect* as a unit of image post-processing with a specific functionality and purpose. IPP effects can be efficiently implemented on current graphics processing units (GPUs) [2]. The concept of IPP can be applied for different applications. In this paper, we focus on the use of IPP for styling 2D images of projective views of 3DGeoVE. On the contrary, in general, styling can be implemented without the use of IPP.

As input and output for IPP, 2D images representing G-buffer [23] can be used. For a projective view of a visual representation of a 3DGeoVE, a G-buffer encodes per pixel a specific type of information such as color, depth, normal, and material properties. Figure 1 depicts example G-buffer containing different, exemplary types of information.

In this context, several characteristics and benefits of IPP are important. IPP allows for decoupling the process of generating an image from applying enhancement processes to already generated images. Moreover, several visual effects are implemented most efficiently and effectively as image post-processes. The complexity of representing a perspective view in an image and applying IPP to the image depends

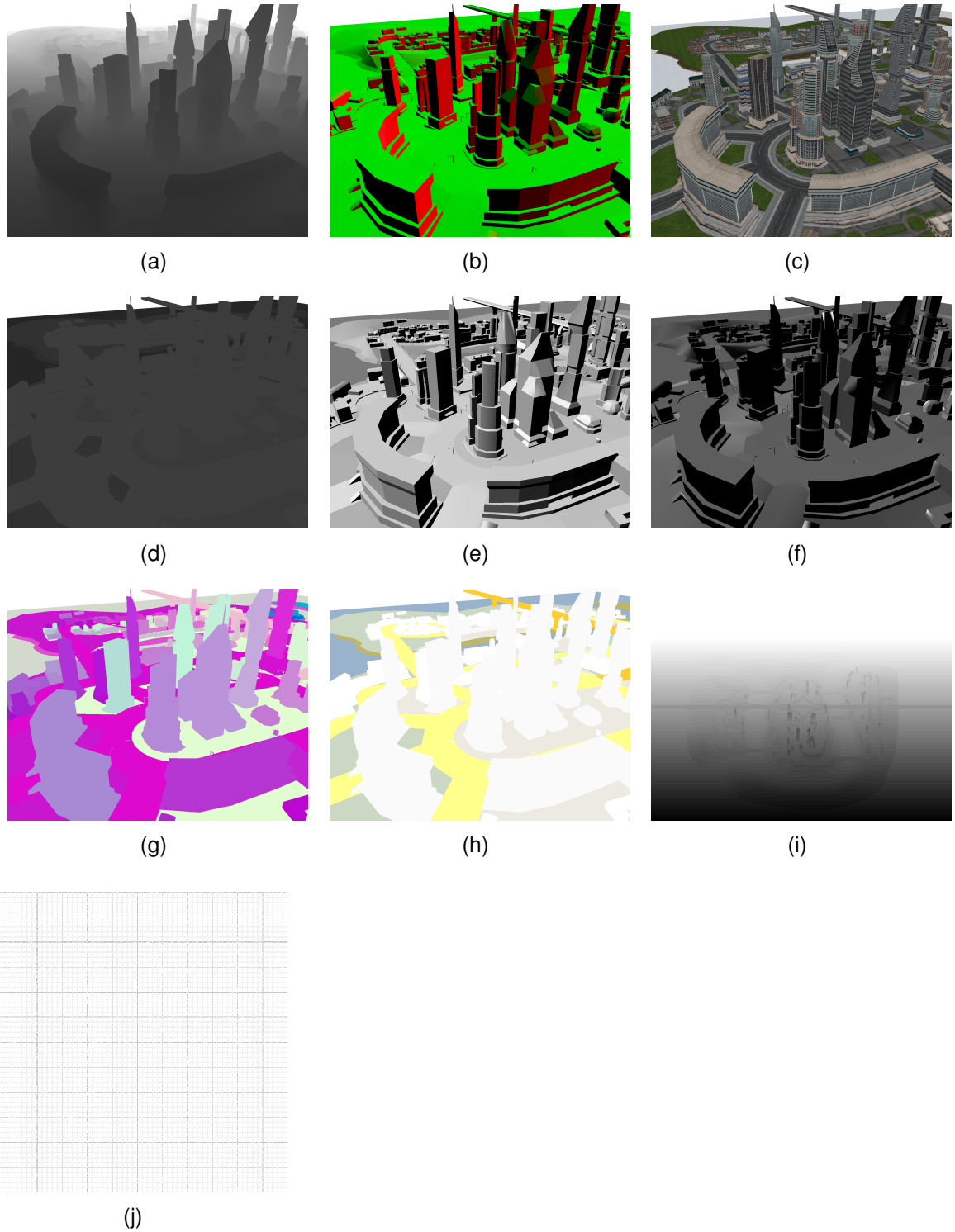


Figure 1: Examples of different types of information encoded per pixel in a G-buffer: (a) depth, (b) normal, (c) albedo (color texture), (d) ambient lighting, (e) diffuse lighting, (f) specular lighting, (g) object ID, (h) colorCode (encoding semantics of objects with a color), (k) shadowMap (depth of scene as seen from a light source), and (i) projective texture (a texture to be applied to the scene via projective texturing).

only on the image dimensions and not the complexity of the scene [16]. Furthermore, images as inputs and outputs for IPP are conceptually simple, robust, commonly used and supported. Additionally, image formats exist (e.g., JPEG, PNG) that are standardized, commonly used and supported, and storage and processing efficient. Using the G-buffer concept [23], multiple information layers of a 3D model (e.g., 3D position, normal, color, and object ID of surface elements) can be encoded into 2D images. Thus, images can be used as an alternative representation for 3D models that sample 3D models in a discreet and multidimensional way and reduce the diversity and heterogeneity of their original representations (e.g., points, triangles, NURBS, voxel) to a simpler, unified representation.

2.2 Styling

According to the OGC, the importance of the visual portrayal of geographic data cannot be overemphasized. The skill that goes into portraying data (whether it be geographic or tabular) is what transforms raw information into an explanatory or decision-support tool. Fine-grained control of the graphical representation of data is a fundamental requirement for any professional mapping community [19, 21]. In this context, styling can be defined as the mapping of data to geometry and/or appearance attributes.

The are efforts from the OGC to standardize the styling of 2D and 3D portrayal of geospatial data in a SOA. *Symbology Encoding (SE)* [21] represents a language for defining rendering parameters for specific features and coverages. SE describes the symbolizer (line, polygon, point, text, raster) to use for rendering the feature geometry, which appearance parameters to consider, and for which scale this styling is applicable. The *Styled Layer Descriptor (SLD) Profile for WMS* [19] allows for user-defined styling. Together with the *GetMap* request an SE-encoded SLD description is transmitted inline or as URL reference. Therefore, the *Web Map Service* (WMS) [8] interface is extended for retrieving the feature types of a layer. In contrast to styling the 2D portrayal from a WMS, until now no OGC standard exists that defines the styling of 3D portrayal from a *Web 3D Service* (W3DS) [24] and the *Web View Service* (WVS) [12, 13]. Nevertheless, Haist et al. [14] and Neubauer et al. [22] propose separate extensions for the SLD and SE for 3D portrayal.

Within the visualization pipeline [11], functionality for styling can be conceptually located in the filtering, mapping, rendering or even after the rendering stage. Widely-used is styling in the mapping stage (*M-styling*). The already introduced concepts of SLD and SE for 2D and 3D portrayal in the context of standardization apply styling in the mapping stage. However, styling can also applied in or even after the rendering stage (*R-styling*). For styling 2D images of projective views, IPP can be applied in or after the rendering stage.

3 Analysis

In this Section, we analyze different characteristics of styling and IPP and design dimensions relevant when building a system for styling and IPP. From the analysis, we

derive a set of requirements for a concept and system providing IPP and styling functionality based on IPP.

Granularity of IPP Effects On the lowest level, images can be processed by iterating in one pass over all pixel of the target image and applying an algorithm for calculating for each pixel the resulting value from the input images (e.g., alpha blending of two input images). On a higher level, specific effects can only be calculated by combining several basic passes and using outputs of passes as inputs for other passes (e.g., image abstraction by structure adaptive filtering [18]).

Programmable vs. Configurable When the processing of IPP is offered as a service, it can be programmable and/or configurable. The processing is *programmable* if a service consumer can provide the service with an algorithmic description of the effect that service provider is supposed to execute. The focus is on offering processing as a service. The *Web Coverage Processing Service* (WCPS) [4] is an example of such a service that offers processing of coverages. The processing is *configurable* if a service provider already implements and offers one more IPP effects. The service consumer can choose from the offered effects and can configure their behavior. In this case, the service provider offers IPP effect implementations as well as processing as a service.

IPP Effect Composition (Service Interface) Services that provide the functionality of IPP and IPP-based styling must provide an interface to service consumers that allows accessing this functionality. The interface must allow specifying which IPP effects in what configuration are to be applied to what geospatial data and what output images are required.

For specifying how input images are transformed by a set of IPP effects into output images, the concept of *data flow graphs* (DFG) [27] is commonly used. Nodes represent IPP effects and directed edges the flow of data between the nodes. The resulting graphs are *directed acyclic graphs* (DAG) with the tendency to form a tree with the inputs as leafs and the result as root node. This concept is used in commercial software products such as *Adobe Pixel Bender* [1] and *Apple Quartz Composer* [3].

For styling 2D portrayal in the mapping stage (M-styling), the OGC proposes the concepts of SLD and SE. In essence, these concepts allow a service consumer to specify which symbolizer from a predefined list in what configuration is to be applied to which features (selected as layers of features with a set of predefined selection operators). As a simpler alternative, image generating services (such as the WMS) allow choosing a style from a predefined list of styles for each layer that is selected for portrayal.

Semantic-based and Selective Styling and Application of IPP Efficient and effective communication of geospatial information in 3DGeoVEs typically requires that for subsets of the geospatial data specific, adequate representations are choosen. The mapping of geospatial data to visual representations is accomplished by styling. It follows that, generally, effective styling requires that different styling can be applied to

different subsets of the geospatial data. As an example, to guide the viewers gaze to a focus area in a visual representation, the focus area could be visually highlighted and represented in detail (e.g., by applying photorealistic rendering techniques) whereas the surrounding context area would be represented abstracted and with less detail (e.g., by selectively applying non-photorealistic rendering techniques).

Geospatial data is commonly organized in *features* and collections of features called *layers*. When using a semantic data model such as *CityGML* [10], features are enriched with semantic information. Layers of features or individual features can be selected explicitly or rule-based (e.g., all features within a buffer region from a road or with a specific semantics or thematic attribute) and styled individually. For selecting features from collections of features, the OGC proposes a dedicated filter language [28]. When applying IPP for styling, additional methods for selection become viable. Applying IPP to the whole scene or on spatial regions across feature and layer boundaries can be accomplished efficiently and effectively.

Architecture From an architectural perspective, in a SOA, the functionality of IPP and IPP-based styling can be located in three places:

1. Integrated with the service that provides images as input to the IPP (e.g., WVS),

2. Provided as a dedicated service that receives input images from other services or the calling service consumer and provides the service consumer with output images, or

3. Integrated with the interaction service [15] that a human user directly interacts with.

From the analysis we derive the following set of requirements for a concept and system that provides IPP and styling based on IPP in a SOA based on standards:

- Standardization: The service interface and employed data models and encodings in the interface should adhere to or build on existing open standards.

- Granularity of IPP Effects: Service consumers can specify IPP on low-level (e.g., single-pass IPP effect) and high-level granularities (e.g., multi-pass IPP effects).

- Programmable vs. Configurable: Service providers can provide IPP effect implementations and their processing to service consumers. Service consumers can algorithmically specify IPP effects, and transmit the specification to the service provider for execution.

- Architecture: The IPP functionality must be accessible as a dedicated service and integrated in a image rendering service (i.e., the WVS). Integrating IPP in the interaction service is not recommended unless unavoidable to keep the interaction service as lightweight as possible and the IPP functionality reusable as a service.

- Selective Application of IPP: Subsets of geodata for styling must be selectable on the level of parts of space and features, individual features, layer, whole scene.
- IPP Effect Composition (Service Interface): Services providing IPP functionality must provide an interface based on data flow graphs. Services providing functionality for styling based on IPP must provide an interface that is based on SLD and SE.

4 Design

In this Section, we briefly sketch the preliminary concept for services providing the functionality of IPP and IPP-based styling.

A service consumer can specify IPP with a data flow graph (DFG) expressed in XML. A DFG is directly suitable for being processed by the service provider. For specifying IPP-based styling, the current proposals for SLD and SE are extended to support the specific characteristics of IPP-based styling. The SLD/SE-based styling description is more abstract than the DFG-based description, while the latter is more expressive. We assume that the SLD/SE-based styling description can be mapped to a DFG-based description that then can be directly processed.

Service providers can offer a predefined set of IPP effect implementations that can be referenced in DFG-based and SLD/SE-based IPP descriptions. Additionally, service providers allow users to provide their own executable code for IPP effects. We plan to evaluate both the open standards *OpenGL Shading Language* (GLSL) and *OpenCL* for this purpose.

The IPP and IPP-based styling is implemented as a library that is used for implementing a dedicated service and for integration in the WVS. The dedicated service is based on the OGC's generic *Web Processing Service* (WPS) [25] proposal.

We have identified a first collection of general-purpose, reusable IPP effects that enable the composition of web view services for 3DGeoVEs operating on G-buffer images. The collection includes:

- 3D image synthesis effects generate the base G-buffer.
- Shadow mapping synthesize effects generate and apply shadow maps [29] in screen-space.
- Ambient occlusion synthesis effects generate approximated ambient light intensities in screen-space (SSAO, [26]).
- Non-photorealistic image processing effects emphasizes edges and remove detail from surfaces [2, 18].
- Depth-of-field effects infiltrate an artificial object-based focus area in the view [2].
- Projective texture effects superimpose projective textures on the given view [2].
- Highlighting effects emphasize the silhouette of selected objects.

- Generic G-buffer and image converter, blending, and convolution operators.

5 Implementation and Evaluation

In this Section, we briefly report on the initial implementation and evaluation of the concept for services providing the functionality of IPP and IPP-based styling.

As a basis, we implemented a library for specifying and executing IPP based on data flow graphs in C++. Second, we implemented a set of exemplary low-level and high-level IPP effects in C++, OpenGL and GLSL. A list of the implemented IPP effects is presented in Figure 2.

We implemented ambient occlusion synthesis effects that generate approximated ambient light intensities in screen-space (SSAO, [26]). For evaluation, we implemented two different techniques: [17, 20] and [5, 6].

We implemented non-photorealistic image processing effects that emphasize edges and remove detail from surfaces [18]. For evaluation, we implemented two different techqniues. The first NPR technique “NPR1” is based on image abstraction by structure adaptive filtering [18], and requires only one color image as input.

The second NPR technique, “NPR2 ColorBlend and EdgeEnhance”, requires as input two color images, depth, and diffuse lighting. The two color images are intended to be both appearance representations of the 3D scene. However, the first one is expected to be a more abstract representation than the second one. In our implementation and evaluation, the first color image represents a color coding of semantics of the scene objects. Each pixel in the image is assigned a color that is based on the semantics of the object that the pixel belongs to. We used a color scheme that was inspired by common 2D web mapping applications. For the second color image, in our evaluation, we used the output of the first technique NPR1 based on image abstraction. The two color code images are then blended (via mix) based on the distance of each fragment from the virtual camera. Optionally in this blend, the second image is modulated by the first image to preserve its effect. The effect of the blending is that the more detailed representation dominates fragments near the virtual camera while, optionally, the color coding is still visible. This representation is gradually blended with the more abstract representation that dominates fragments farther from the camera. Subsequently, the diffuse lighting is applied to the new appearance image that results from the blend. Finally, edges in the image are detected using discontinuities in the depth and color code images. Found edges are enhanced by darkening respective fragments.

For evaluating the implementation, we performed experiments. We build a graph specifying a complex IPP (Figure 3) using the implemented IPP effects. As input to the graph, the ten G-buffers depicted in Figure 1 are used. Intermediate G-buffer processing results of the data flow graph are presented in Figure 4. The final result (color buffer) is displayed in Figure 5. Figure 2 reports on the processing time for the individual effects. The experiment is performed on a Core2Duo E6600 with 2.4Ghz, nVidia GTX 260. Measurements were performed on G-buffer resolutions of 800x600 and averaging the timings of 5.000 processing calls. The timings indicate that even complex

Category	Effect	Effect Mode/Subtype/Config	Avg Processing Time (ms) Single	Accum	Accum.	View Independ.	Number of Passes
Integration	Composite	Depth and ObjectId Selector	0,26		0,26	x	1
	Blend	Add	0,22		0,22	x	1
		Hardlight	0,22			x	1
Abstraction	NPR1	Image Abstraction	16,80		16,80	o	6
	NPR2	ColorBlend and EdgeEnhance	2,77		2,77	o	1-2
Photorealism	SSAO	Mittring (half res)	3,88	4,74	10,05	o	3
		Mittring (full res)	5,60			o	3
		Bavoil (s=8, d=16)	9,72	12,71		o	3
		Bavoil (s=8, d=16, halfres)	5,30			o	3
		Bavoil (s=24, d=32)	32,63			o	3
		Bavoil (s=3, d=4)	3,20			o	3
	ShadowMapping	Hard Shadows	0,35		1,59	o	2
		PCSS	2,84			o	
Focus and Context	Highlight	Halo (r=2)	1,07	1,57	0,92	-	3
		Halo (r=7)	1,40			-	3
		Halo (r=20)	2,26			-	3
		Background Grey	0,27			x	1
		Brightness	0,30			x	1
		ColorOverlay	0,26			x	1
	Depth of Field		1,01		1,01	-	5
Augmentation	ProjectiveTexturing		0,40		0,40	x	1
Low Level Processing	ColorAdjust		0,22		0,22	-	1
	Convolution	Gauss (r=3)	0,64	1,06	0,85	x	2
		Gauss (r=10)	1,48			x	2
		Box (r=3)	0,75	1,19		x	2
		Box (r=10)	1,64			x	2
		Unsharp Masking (Laplace)	0,27	0,31		x	1
		Unsharp Masking (Sobel)	0,34			x	1
Environmental	Fog	*not implemented yet*					
	Rain	*not implemented yet*					
	Water	*not implemented yet*					

Figure 2: Overview of the implemented IPP effects, their average processing time (measured on Core2Duo E6600 with 2.4Ghz, nVidia GTX 260, resolution 800x600), view independence characteristic, and number of passes.

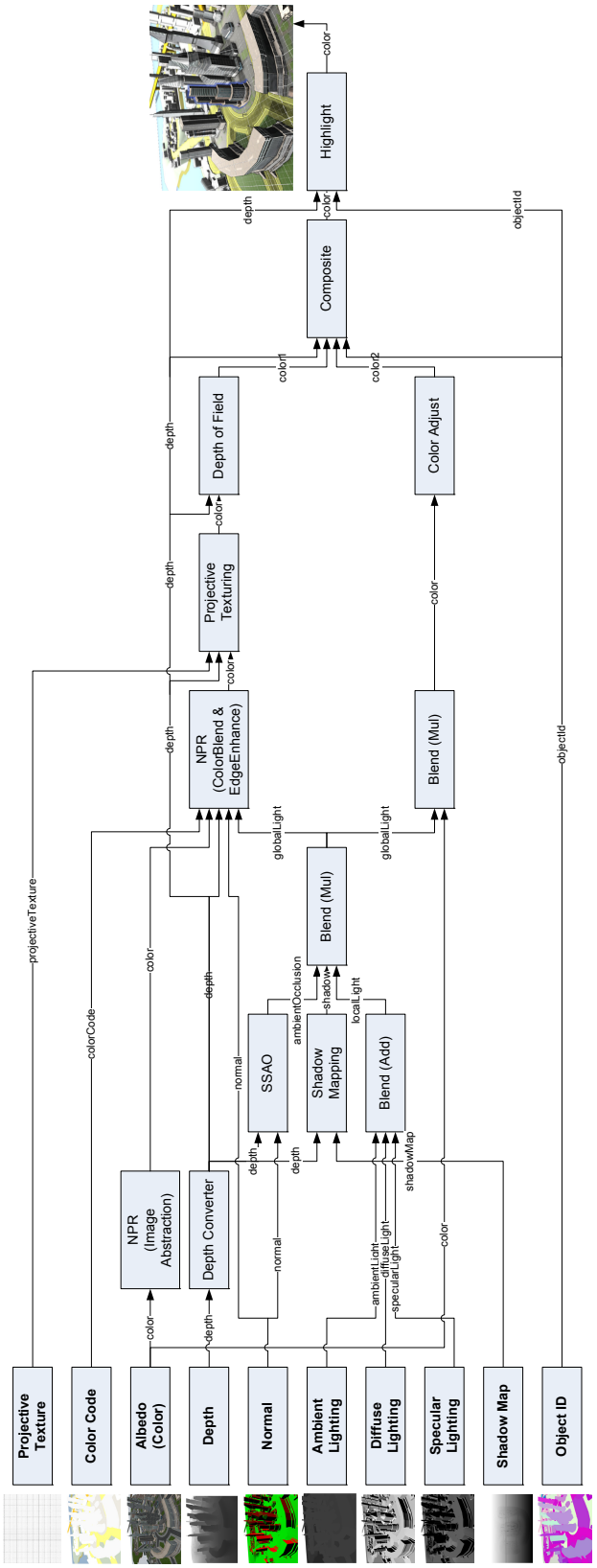


Figure 3: Data flow graph of a complex IPP effect composition using the implemented IPP effects build for the evaluation experiment.

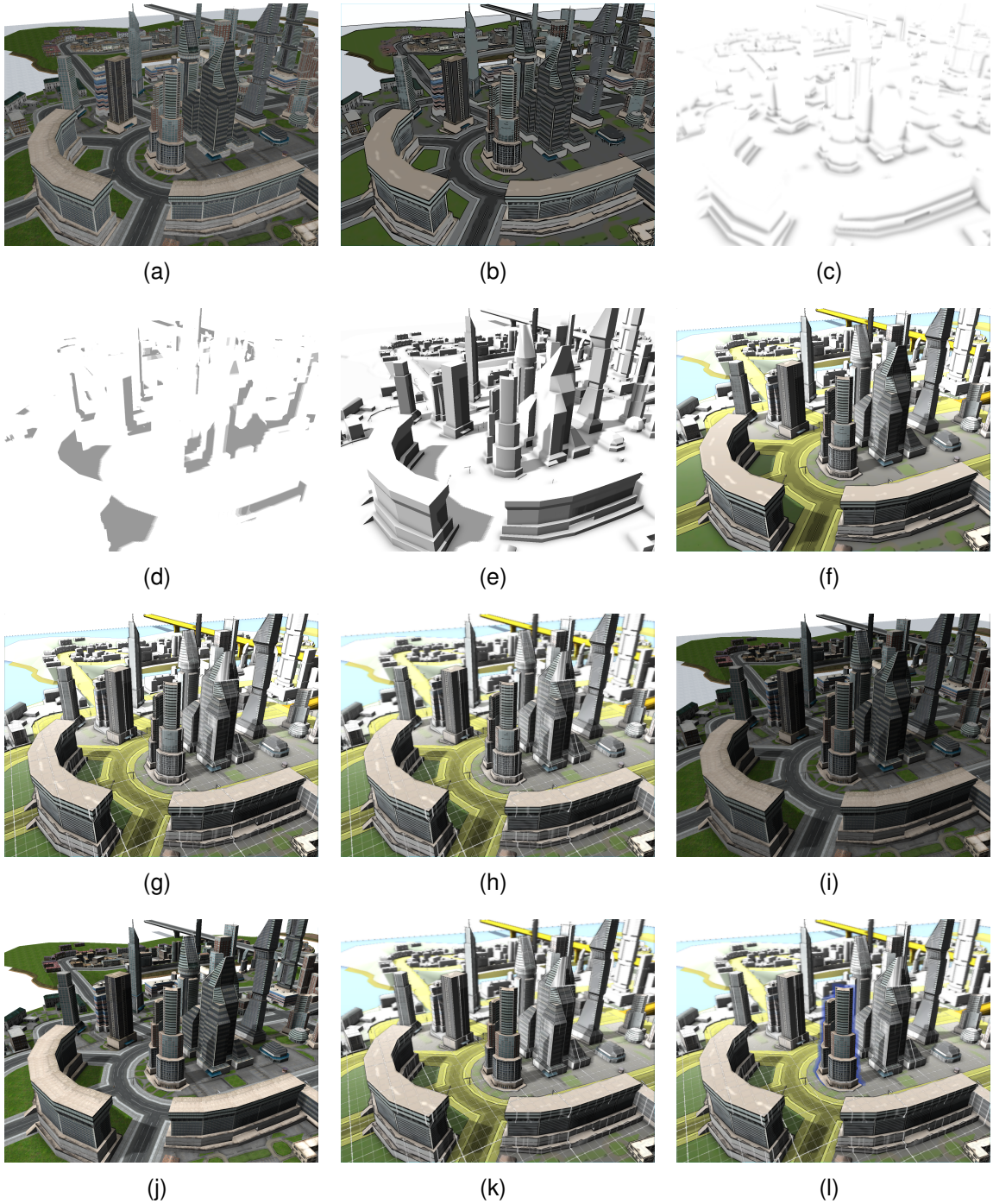


Figure 4: Intermediate G-buffer processing results of the data flow graph. (a) color as the most important input appearance data, (b) output of the NPR (Image Abstraction) node, (c) output of the SSAO node, (d) output of the Shadow Mapping node, (e) output of the first Blend (Mul) node, (f) output of the NPR (ColorBlend and EdgeEnhance) node, (g) output of the Projective Texturing node, (h) output of the Depth of Field node, (i) output of the second Blend (Mul) node, (j) output of the Color Adjust node, (k) output of the Composite node, (l) output of the Highlight node.

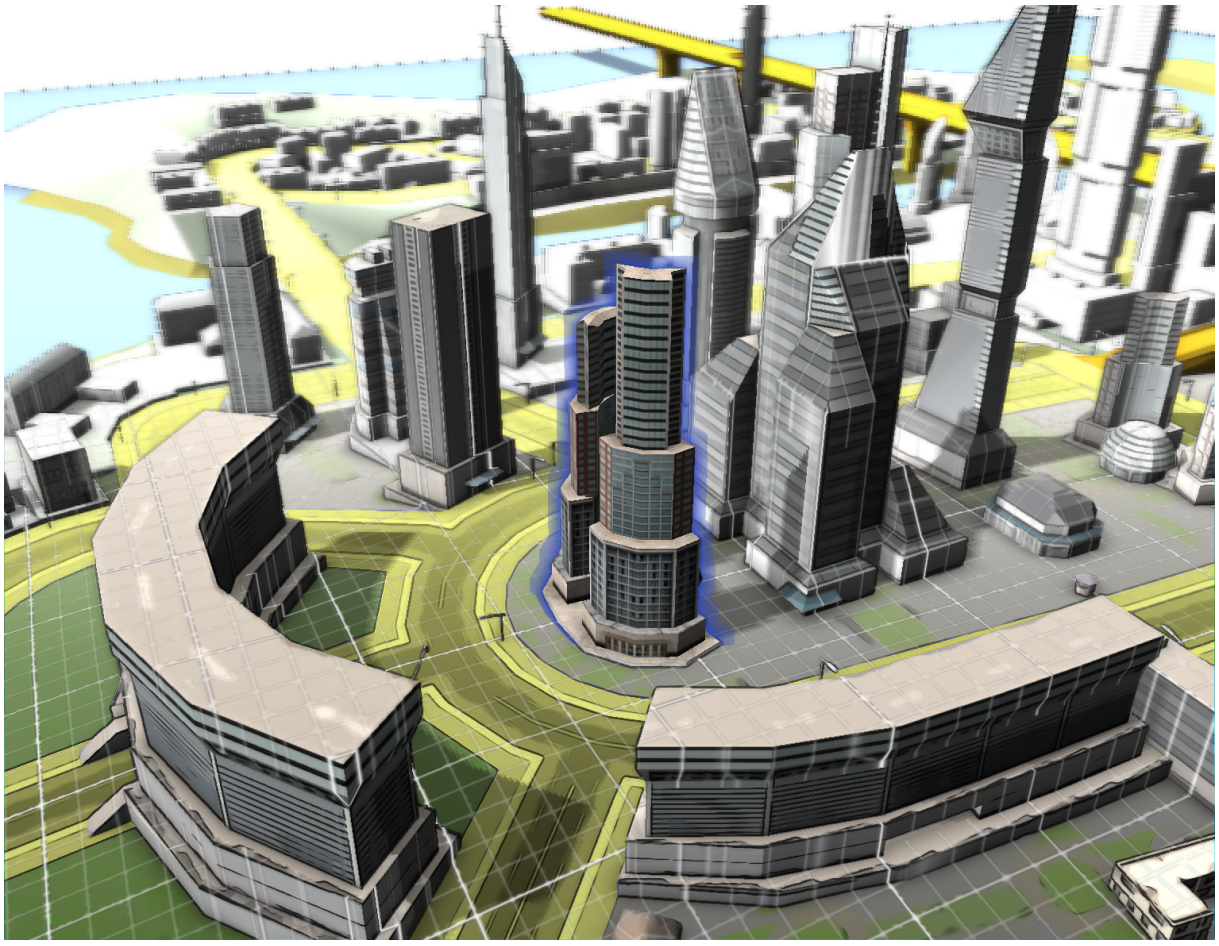


Figure 5: Final result (color buffer) of processing the data flow graph build for the evaluation experiment.

IPP-based styling can be performed at interactive frame rates on current consumer hardware. As expected, IPP effects have the tendency to take more time the more complex they are (in terms of algorithmic and memory access complexity per pixel) and the more passes they require. Surprisingly, the SSAO (screen-space ambient occlusion) effect turned out to be comparatively costly with respect to its contribution to the final result.

6 Summary, Conclusions, and Next Steps

In this paper, we investigated how IPP functionality and the functionality of styling of visual representations of 3DGeoVE based on IPP can be provided in a SOA based on standards. We introduced the concepts of IPP and styling and presented an analysis of different characteristics of styling and IPP and design dimensions relevant when building a system for styling and IPP. From the analysis, we derived a set of requirements for a concept and system providing IPP and styling functionality based on IPP. We presented a preliminary concept for a system meeting the identified requirements and reported on an initial implementation and evaluation results.

The first results indicate that IPP functionality and the functionality of styling of visual representations of 3DGeoVE based on IPP can be provided in a SOA based on standards. Furthermore, IPP-based styling promises to offer powerful ways of styling visual representations with interactive frame rates decoupled from the process of image generation.

My next steps include extending the SLD/SE for IPP, integrating IPP and IPP-based styling in the WVS and offering it as a dedicated service, and allowing service consumers to specify IPP effects by providing executable code.

References

[1] Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. *Adobe Pixel Bender Developer's Guide*, 2010.

[2] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, third edition, 2008.

[3] Apple Inc., 1 Infinite Loop, Cupertino, CA 95014. *Apple Quartz Composer User Guide*, July 2007.

[4] Peter Baumann, editor. *OGC Web Coverage Processing Service (WCPS), Version 0.0.3*. Open Geospatial Consortium Inc., July 2006.

[5] Louis Bavoil and Miguel Sainz. Image-Space Horizon-Based Ambient Occlusion. In Wolfgang Engel, editor, *ShaderX7 - Advanced Rendering Techniques*, pages 425–444. Charles River Media, 2009.

[6] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-Space Horizon-Based Ambient Occlusion. In *SIGGRAPH 2008 Talks*. ACM, New York, NY, USA, 2008.

[7] Open Geospatial Consortium. Open Geospatial Consortium (OGC) Website. URL, `http://www.opengeospatial.org/`, 2009. Accessed 16.10.2010.

[8] Jeff de la Beaujardiere, editor. *OpenGIS Web Map Server Implementation Specification, Version 1.3.0*. Open Geospatial Consortium Inc., March 2006.

[9] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall International, Upper Saddle River, N.J., third edition, 2008.

[10] Gerhard Gröger, Thomas H. Kolbe, Angela Czerwinski, and Claus Nagel, editors. *OpenGIS City Geography Markup Language (CityGML) Encoding Standard, Version 1.0.0*. Open Geospatial Consortium Inc., August 2008.

[11] R.B. Haber and D. A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In B. Shriver, G. M. Nielson, and L. Rosenblum, editors, *Visualization in Scientific Computing*, pages 74–93, Los Alamitos, 1990. IEEE Computer Society Press.

[12] Benjamin Hagedorn, Dieter Hildebrandt, and Jürgen Döllner. Towards Advanced and Interactive Web Perspective View Services. In *Developments in 3D Geo-Information Sciences*, Lecture Notes in Geoinformation and Cartography, Berlin, Heidelberg, November 2009. Springer.

[13] Benjamin Hagedorn, Dieter Hildebrandt, and Jürgen Döllner. *Web View Service Discussion Paper, Version 0.6.0*. Open Geospatial Consortium Inc., February 2010.

[14] Jörg Haist, Hugo Miguel Figueiredo Ramos, and Thorsten Reitz. Symbology Encoding for 3D GIS - An Approach to Extending 3D City Model Visualization to GIS Visualization. In *Urban Data Management Symposium*, pages 121–131, October 2007.

[15] Dieter Hildebrandt and Jürgen Döllner. Service-oriented, standards-based 3D geovisualization: Potential and challenges. *Journal on Computers, Environment and Urban Systems*, 34(6):484–495, November 2010. GeoVisualization and the Digital City - Special issue of the International Cartographic Association Commission on GeoVisualization.

[16] Dieter Hildebrandt, Benjamin Hagedorn, and Jürgen Döllner. Image-Based, Interactive Visualization of Complex 3D Geovirtual Environments on Lightweight Devices. In *7th International Symposium on LBS and Telecartography*, 2010.

[17] Vladimir Kajalin. Screen-Space Ambient Occlusion. In Wolfgang Engel, editor, *ShaderX7 - Advanced Rendering Techniques*, pages 413–424. Charles River Media, 2009.

[18] Jan Eric Kyprianidis and Jürgen Döllner. Image Abstraction by Structure Adaptive Filtering. In Ik Soo Lim and Wen Tang, editors, *EG UK Theory and Practice of Computer Graphics*, pages 51–58. Eurographics Association, 2008.

[19] Markus Lupp, editor. *Styled Layer Descriptor Profile of the Web Map Service Implementation Specification, Version 1.1.0*. Open Geospatial Consortium Inc., June 2007.

[20] Martin Mittring. Finding next gen: CryEngine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[21] Markus Müller, editor. *Symbology Encoding Implementation Specification, Version 1.1.0*. Open Geospatial Consortium Inc., July 2006.

[22] Steffen Neubauer and Alexander Zipf. Suggestions for Extending the OGC Styled Layer Descriptor (SLD) Specification into the third Dimension - An Analysis of possible Visualization Rules for 3D City Models. In *Urban Data Management Symposium*, Stuttgart, Germany, October 2007.

[23] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. *SIGGRAPH Computer Graphics*, 24(4):197–206, 1990.

[24] Arne Schilling and Thomas H. Kolbe, editors. *Draft for Candidate OpenGIS Web 3D Service Interface Standard, Version 0.4.0*. Open Geospatial Consortium Inc., 2010.

[25] Peter Schut, editor. *OpenGIS Web Processing Service, Version 1.0.0*. Open Geospatial Consortium Inc., June 2007.

[26] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 73–80, New York, NY, USA, 2007. ACM.

[27] Michael A Shantzis. A Model for Efficient and Flexible Image Computing. In *SIGGRAPH 94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, volume 28, pages 147–154, New York, NY, USA, 1994. ACM Press.

[28] Peter Vretanos, editor. *OpenGIS Filter Encoding 2.0 Encoding Standard, Version 2.0.0*. Open Geospatial Consortium Inc., March 2010.

[29] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, volume 12 of *SIGGRAPH '78*, pages 270–274, New York, NY, USA, 1978. ACM.

Modeling and Verification of Self-Adaptive Service-Oriented Systems

Basil Becker
basil.becker@hpi.uni-potsdam.de

Service-oriented architecture enables more flexible IT solutions. At the level of the architecture the runtime binding of service contracts, starting new component instances and terminating components result in a dynamic assembly and runtime reconfiguration of complex open IT landscapes. As new services contracts can be added at runtime as well, the dynamics goes even further and permit that the IT landscapes evolves at the level of its components as well as service contracts. In this report I will shortly outline which problems concerning modeling and verification arise in the domain service-oriented systems and identify requirements which an approach that targets these problems has to fulfill. Further I will give a rough overview of the state of the art for the formal verification and modeling of service-oriented systems.

1 Introduction

Service-oriented architecture enables more flexible IT solutions. At the level of the architecture the runtime binding of service contracts, starting new component instances and terminating components result in a dynamic assembly and runtime reconfiguration of complex open IT landscapes. As new services contracts can be added at runtime as well, the dynamics goes even further and permit that the IT landscapes evolves at the level of its components as well as service contracts. In the service-oriented approach *orchestration* describes a collaboration with a single dedicated coordinator that enacts the collaboration between the other parties. The *choreography* interaction scheme in contrast support the free interplay of different roles within a collaboration.

The service-oriented approach in contrast to standard component-based architectural models employs collaborations describing the interaction of multiple roles in form of service contracts (cf. [1,8]). Current approaches for modeling service-oriented architectures, however, do either only support scenarios where the dynamics and evolution are restricted to static collaborations with fixed service contracts [1] or an appropriate rigorous formal underpinning for the model for the conceptually supported dynamics is missing. While orchestration is often described by business processes or activity diagrams [22, 26, 27, 29], for choreography it is less clear which kind of behavioral description is best suited [29].

Further, service-oriented systems differ from classical component-based systems as they reside in so called open-world setting [3]. The term open-world setting or open-world assumption expresses the observation that the assumption of a known

and unchanging border between system and environment is no longer valid in todays software systems.

Proposals for the formal verification of service-oriented architectures are even more restricted and do only support scenarios where the evolution has been so restricted that checking a bounded formal model is sufficient [12].

Therefore, the current proposals for modeling and verification do not support the beforehand outlined dynamics and evolution of IT landscapes with orchestration and choreography but only checking specific configurations.

Given a fixed system configuration you can of course use tests to detect compatibility problems.

Systems can be distinguished concerning the degree and the type of dynamism they allow. Typically one distinguishes static, top-down and bottom-up style systems (cf. [10]). In a static system the system's structure – i.e. which components exist in the system and how are the components connected to each other – does not change at run-time. Adaptive systems – i.e. systems that change their structure at run-time – can be further separated into top-down and bottom-up adaptation. In a system that follows the top-down adaptation approach, few components determine in which way other components, contained in the systems, have to change. These changes can range from mode changes, internal to a component, to changes to the system's structure, including the addition and removal of components. Further top-down adaptive systems can be organized in several layers, which form a hierarchy. In contrast to a top-down adaptive system a bottom-up adaptive systems does not have a small set of special component, that are responsible for applying changes, but each single component itself has the capabilities to change it's mode and establish and terminate connections to other components. In a bottom-up adaptive system the desired behavior emerges from the behaviors of the system's components.

Going back to service-oriented systems one easily notes that the concepts of top-down and bottom-up adaptive systems appear in this architectural style, too. In the literature on service-oriented systems the top-down approach is called orchestration and the bottom-up approach is called choreography.

Following for service-oriented systems we have two major influences: the open world assumption and the relationship to adaptive systems. These two influences lead to the observation, that, in general, a system can have an infinite number of structural configurations. This is because in an adaptive system the structure is not fixed and can be changed almost arbitrarily. Further, the open world assumption states that at any time new component-types can be introduced into the system and contained types can be removed. In the following we will use the term *service landscape* for the set of all available service- and component-types and *landscape configuration* for an instance of a specific service landscape.

Given such a complex scenario it can be easily seen that standard verification and validation techniques – i.e. simulation, testing, monolithic formal verification – fail due to the systems' complexity. While in case of in-house service-oriented architectures testing can thus provide some coverage in case the dynamics and evolution of IT landscapes is restricted to the tested cases, for more dynamic scenarios the high if not unbounded number of possible configurations results in a low coverage. Furthermore,

in case of more advanced scenarios such as cross-organizational service-oriented architectures, digital ecosystems [19] or ultra-large-scale systems [20], no overarching governance exists and thus the open and dynamic character of these systems prevents that all possible combinations of components and service contracts can be systematically tested before they could become active. Thus testing is only applicable to service-oriented systems if the landscape configuration can be fixed for some reasons, otherwise the test coverage will be much too low.

Similar arguments hold for simulation. A simulation run can only cover one specific execution trace of the system and hence the results of the simulation can hardly be generalised for the whole system. Formal verification – as a monolithic approach – also is not applicable to the class of systems we are interested in, as to the best of our knowledge, no technique exists that can cope with the additional complexity introduced by the infinite number of possible structural configurations.

In my thesis I will concentrate on the aspect of service choreographies. The functional correctness of choreographies is especially important because often choreographies are used for the communication between two companies and according to [29] any failure in the communication has direct influence to the companies' operational business. Further, with ultra-large-scale systems [20] services-oriented architectures will become more important in the domain of safety-critical systems. For this domain verification is strictly required. Nevertheless, the approach I will develop in my thesis is also applicable to service orchestrations.

2 Requirements

Given the above description we have to develop a technique that is *applicable* to service-oriented systems and that is *scalable*. A technique is considered scalable if it is able to verify – where verification includes testing, simulation and formal verification – a potentially extremely large landscape configuration. However, recalling the fact that landscape configurations are not fixed in a service oriented system, we also have to verify landscape configurations that are reachable from a given one.

In order to be applicable to service-oriented systems an approach has to be able to cope with the fact that no consistent, global view of the system is available at any time. Further new systems constituents can be introduced into the system uncoordinatedly. Both facts result in situations where no one is aware of neither the currently valid service landscape nor the landscape configuration representing the system's current state. However in order to test whether, e.g., a component works properly the developer has to know with which other components the new one can possible interact.

2.1 Modeling

Verification of software systems always requires a rigorous modeling beforehand the verification can start. Beside the system the specification the system has to fulfill have to be modelled, too. Thus, every verification approach requires a suitable modeling approach that provides the information needed by the verification technique.

For the modeling aspects the requirements that have to be met by appropriate modeling approaches are similar to the requirements that have to hold for the verification. A modeling approach has to be scalable, i.e. it has to be possible to express potentially large landscapes and landscape configurations. Further, an approach also has to be applicable to the modeling of service-oriented systems. Applicability of a modeling approach is especially important if the system gets modified after it has been deployed. Again the non-existent consistent view of the system's state and thus the improper knowledge of the system's constituents is the limiting factor. If a modeling approach requires a complete model of the system to extend the system's model, such an approach will not be applicable to service-oriented systems.

3 State of the Art

Modeling using roles and focusing on collaborations rather than components is not new: Since the 1970s the OOram Software Engineering method [23] has been developed which provides a clear distinction between roles and objects and separates different collaborations in form of role models. The idea of contracts, which has been introduced in [15], also already supports a number of participants and in addition results in some *contract obligations* the classes that take over the role of the participants have to fulfill. Also a less clear historical connection between roles/collaborations and design pattern [14] exists, which is reflected today by the fact that design patterns can be modeled in UML using collaborations. The use of collaborations for the modeling of services has been proposed by several authors (cf. [8, 25]) as well as all proposals for a UML Profile and Meta-Model for Services [1, 7, 18]. In [25] static but hierarchic UML collaborations and the distinction between the collaboration and the collaboration use are presented. However, the authors omit the definition of the roles' behavior. An approach not using UML that overcomes this limitation is presented in [8] which uses sequence diagrams for potentially incomplete early behavior specifications. The UML Profile [1] is conceptually similar to [25]. It further extends [25] also supporting behavior specifications for the different roles.

UML class diagrams for the structure and graph transformations for the behavior modeling are also employed in [4] to model service-oriented architectures, but in contrast to our approach services are not modeled as collaborations. We can conclude that none of the modeling concepts supports dynamic collaborations as addressed in this work.

The use of UML collaborations for the modeling of services has been proposed by several authors (cf. [1, 25]). In [8] also collaborations have been used but not UML collaborations. However, none of the three modeling concepts supports dynamic collaborations. In [25] static but hierarchic collaborations and the distinction between the collaboration and the collaboration use are presented. Further the authors omit the definition of the roles' behavior. This has been done in [8] but only partially - Broy et al. use sequence diagrams for the behavior specification. [1] is similar to [25] but extends [25] with behavior specifications for the different roles.

All the presented collaboration concepts could be seen as advancements of the

idea of contracts, which has been introduced in [15]. A contract consists of a number of participants each of them having some *contract obligations* to fulfill. Contracts also support the idea of roles that have to be mapped to classes. Again contracts only support a constant number of participants and do not provide support for adding or removing participants to contracts at run-time.

The OOram Software Engineering method (cf. [23]), which has been developed since the 1970s, already used the distinction between roles and objects. Whereas roles and objects are often counted to the object oriented paradigm, OOram assigned each role a specific behavior and used behavior synthesis to derive the final behavior. Obviously the behavior synthesis is a hard task and can only be generalized for a restricted set of problems

To our best knowledge no work exists which especially addresses the problem to verify dynamic collaborations, however, a number of related approaches for the verification of service-oriented systems exist. Model checking has been employed to check business process models with varying number of active process instances. In [11], for example, standard BPEL models are enriched by resource allocation behavior to ensure the correct detection of deadlocks and safety violations for web services compositions under resource constraints. In [9] an approach dedicated to the compositional verification of middleware based software architectures is presented. The verification of a software architecture is divided into the verification of properties, which hold for the middleware and those, which hold for the complete architecture. However the approach does not cover structural dynamics and is restricted to finite state systems.

For systems with structural dynamics like our earlier work [6] some work has been published, which does not cover dynamic collaborations to their full extent: An approach which has been successfully applied to verify service-oriented systems [4] is the one of Varró et al. It transforms visual models based on graph theory into a model-checker specific input [28]. A more direct approach is GROOVE [24] by Rensink where the checking works directly with the graphs and graph transformations. DynAlloy [13] extends Alloy [16] in such a way that changing structures can be modeled and analyzed. For operations and required properties in form of logical formulae it can be checked whether given properties are operational invariants of the system. In [17] a petri net variant is employed for the modeling and verification of some issues of an intelligent transportation system and it is suggested to use classical model checking techniques. Real-Time Maude [21], which is based on rewriting logics, is the only approach we are aware of covering structural changes as well as time. The tool supports the simulation of a single behavior of the system as well as bounded model checking of the complete state space, if it is finite. However, all these approaches do not fully cover the problem as they require an initial configuration and only support finite state systems (or systems for which an abstracted finite state model of moderate size exist).

There are only first attempts that address the verification of infinite state systems with dynamic structure: In [2] graph transformation systems are transformed into a finite structure, called Petri graph which consists of a graph and a Petri net, each of which can be analyzed with existing tools for the analysis of Petri nets. For infinite systems, the authors suggest an approximation. The approach is not appropriate for the verification of the coordination of autonomous vehicles even without time, because

it requires an initial configuration and the formalism is rather restricted, e.g., rules must not delete anything. Partner graph grammars are employed in [5] to check topological properties of the platoon building. The partner abstraction is employed to compute over approximations of the set of reachable configurations using abstract interpretation. However, the supported partner graph grammars restrict not only the model but also the properties, which can be addressed a priori.

4 Goals of the Thesis

In my thesis I want to achieve the following goals: First, I want to develop a formal modeling technique that allows the exact specification of service-oriented systems without suffering from restrictions concerning applicability or scalability. Thus the specification technique to develop should be able to cope with the special requirements that have to be met, when developing service-oriented systems, such as reuse of already existing parts and reduced knowledge of development activities going on in parallel. Further, the formal modeling technique will be mapped to the standardized modeling language SoaML by the Object Management Group (OMG).

Second I want to deliver a reasoning scheme for compositional reasoning about service-oriented systems, modeled using the above modeling and specification technique. I will give some general restrictions the specified systems have to meet, to be suited for the reasoning scheme. The benefit of the developed compositional approach is that only partial knowledge of the current landscape configuration and service landscape is required. Further, it is possible to reuse verification results, which significantly reduces the verification task's complexity.

Third, I give a concrete way for the development of service-oriented systems that conforms to both the modeling and specification technique and the compositional reasoning scheme. I will introduce a verification technique that is able to verify possible infinite state rule based systems and that can be used together with the reasoning scheme. However, the compositional reasoning scheme is independent of the developed verification technique. Any other suited technique can be used as well.

References

[1] Jim Amsden, Pete Rivett, Kolk Henk, Fred Cummins, Jishnu Mukerji, Antoine Lonjon, Cory Casanave, and Irv Badr. *UML Profile and Metamodel for Services*, June 2007. http://www.omg.org/docs/ad/07-06-03.pdf.

[2] Paolo Baldan, Andrea Corradini, and Barbara König. A Static Analysis Technique for Graph Transformation Systems. In *Proc. CONCUR*, volume 2154 of *LNCS*, pages 381–395. Springer, 2001.

[3] Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Toward Open-World Software: Issue and Challenges. *Computer*, 39(10):36–43, 2006.

[4] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Daniel Varro. Modeling and Validation of Service-Oriented Architectures: Application vs. Style. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 68–77, New York, NY, USA, 2003. ACM.

[5] Jörg Bauer and Reinhard Wilhelm. Static Analysis of Dynamic Communication Systems by Partner Abstraction. In *Proceedings of the 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007*, volume 4634 of *Lecture Notes in Computer Science*, pages 249–264. Springer Berlin / Heidelberg, 2007.

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

[7] Gorka Benguria, Philippe Desfray, Bob Covington, Arne J. Berre, Stephan Roser, Christian Hahn, Michael Pantazoglou, Dumitru Roman, Miahi Moldovan, James Odell, and Andreas Ditze. *UML Profile and Metamodel for Services - for Heterogeneous Architectures (UPMS-HA)*, June 2007. http://www.omg.org/docs/ad/2007-06-02.pdf.

[8] Manfred Broy, Ingolf Krüger, and Michael Meisinger. A formal model of services. *ACM Trans. Softw. Eng. Methodol.*, 16(1):5, 2007.

[9] Mauro Caporuscio, Paola Inverardi, and Patrizio Pelliccione. Compositional Verification of Middelware-Based Software Architecture Descriptions. In *Proceedings of the 19th International Conference on Software Engineering ESEC04*, 2004.

[10] Betty H. Cheng, Rogerio 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, Raffaela 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. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[11] Howard Foster, Wolfgang Emmerich, Jeff Kramer, Jeff Magee, David Rosenblum, and Sebastian Uchitel. Model checking service compositions under resource constraints. In Ivica Crnkovic and Antonia Bertolino, editors, *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007 Dubrovnik, Croatia, September 3-7, 2007*, pages 225–234. ACM, 2007.

[12] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. LTSA-WS: a tool for model-based verification of web service compositions and choreography. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 771–774, New York, NY, USA, 2006. ACM.

[13] Marcelo Fabian Frias, Juan Pablo Galeotti, Carlos Lopez Pombo, and Nazareno Aguirre. DynAlloy: Upgrading Alloy with actions. In *Proc. of International Conference of Software Engineering*, pages 442–451. ACM, 2005.

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[15] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay. Contracts: specifying behavioral compositions in object-oriented systems. In Norman Meyrowitz, editor, *Object-oriented Programming Systems, Languages and Applications (OOPSLA/ECOOP'90), Ottawa, Canada, October 21-25 1990*, volume 25 of *ACM SIGPLAN notices*, pages 169–180, New York, NY, USA, October 1990. ACM.

[16] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[17] Fabrice Kordon. Mastering Complexity in Formal Analysis of Complex Systems: Some Issues and Strategies Applied to Intelligent Transport Systems. *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, 00:420–427, 2007.

[18] Ingolf Krüger and Vina Ermagan. A UML2 Profile for Service Modeling. In Gregor Engels, Bill Opdyke, Douglas Schmidt, and Frank Weil, editors, *Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, volume 4735 of *Lecture Notes in Computer Science*, Nashville, TN, USA, 2007. Springer Berlin / Heidelberg.

[19] David G. Messerschmitt and Clemens Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. The MIT Press, 2005.

[20] Linda Northrop, Peter H. Feiler, Richard P. Gabriel, Rick Linger, Tom Longstaff, Rick Kazman, Markus Klein, and Douglas Schmidt. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

[21] Peter C. Olveczky and Jose Meseguer. Specification and Analysis of Real-Time Systems Using Real-time Maude. In Tiziana Margaria and Michel Wermelinger, editors, *Proceedings on Fundamental Approaches to Software Engineering (FASE2004)*, volume 2984 of *Lecture Notes in Computer Science*. Spinger-Verlag Heidelberg, 2004.

[22] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36:46–52, 2003.

[23] Trygve Reenskaug, Per Wold, and Odd Arild Lehne. *Working With Objects - The OOram Software Engineering Method*. Manning Publications, 1996.

[24] Arend Rensink. Towards Model Checking Graph Grammars. In Michael Leuschel, S. Gruner, and S. Lo Presti, editors, *Workshop on Automated Verification of Critical Systems (AVoCS)*, Technical Report DSSE–TR–2003–2, pages 150–160. University of Southampton, 2003.

[25] Richard Torbjorn Sanders, Humberto Nicolas Castejon, Frank Kraemer, and Rolv Braek. Using UML 2.0 Collaborations for Compositional Service Specification. In *Model Driven Engineering Languages and Systems*, volume 3713 of *Lecture Notes in Computer Science*, pages 460–475. Springer Berlin / Heidelberg, 2005.

[26] Wil M.P. van der Aalst, Lachlan Aldred, Marlon Dumas, and Arthur H.M. ter Hofstede. Design and Implementation of the YAWL System. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering*, volume 3084 of *Lecture Notes in Computer Science*, pages 281–305. Springer Berlin / Heidelberg, 2004.

[27] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[28] Daniel Varro. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, May 2004.

[29] Mathias Weske. *Business Process Management*. Springer, 2007.

Parsing Behavior: The Hierarchical Nature of Concurrent Systems

Artem Polyvyanyy
Artem.Polyvyanyy@hpi.uni-potsdam.de

Behavioral models are the conceptual models that capture operational principles of real-world or designed systems. A behavioral model defines the state space of a system and the way the system can operate within its state space. A concurrent system allows for several threads of computation to execute simultaneously in the system. Parsing is a technique for discovering the structure of a behavioral model. The result of a parsing is a hierarchical decomposition of a model into logically independent units of behavior. In this paper, we report on two parsing techniques applicable for two different types of behavioral models. Sect. 1 discusses a technique for parsing workflow graphs, whereas Sect. 2 is devoted to parsing ordering relations. Finally, in Sect. 3, we sketch how these two parsing techniques can be related to provide a solution to the problem of structuring unstructured acyclic control flow specifications of concurrent systems under the behavioral equivalence notion which preserves the level of observable concurrency in the resulting structured model.

1 Parsing Workflow Graphs

Concurrent systems are often modeled using some kind of a directed flow graph, which we call a *workflow graph*, e.g., these are systems modeled in BPMN, EPC, UML activity diagrams, Petri nets, etc. A workflow graph can be parsed into a hierarchy of subgraphs with a single entry and single exit (SESE fragments, or fragments). Such a fragment can be addressed as a logically independent part of a concurrent system, in which the semantics of the fragment must be clarified based on the semantics of the respective modeling language. The result of the parsing procedure is a parse tree, which is the containment hierarchy of all fragments of a workflow graph.

The Refined Process Structure Tree (RPST) is a technique for workflow graph parsing which has various applications, e.g., translation between process languages, control-flow and data-flow analysis, process comparison and merging, process abstraction, process comprehension, model layout, and pattern application in process modeling. The RPST has a number of desirable properties: The resulting parse tree is unique and modular, where modular means that a local change in the workflow graph only results in a local change of the parse tree. Furthermore, it is finer grained than any known alternative approach and it can be computed in linear time. Finally, the RPST of a workflow graph is the set of its canonical fragments, where a fragment is said to be canonical if it does not overlap on the set of edges with any other fragment of the graph.

In [17], we proposed an alternative way to compute the RPST that is simpler than the one developed originally [20]. In particular, the computation is reduced to constructing the *tree of the triconnected components* [5, 18] of a workflow graph in the special case when every node has at most one incoming or at most one outgoing edge.

A triconnected graph is a graph such that if any two nodes are removed from the graph, the resulting graph stays connected. A pair of nodes whose removal renders the graph disconnected is called a separation pair. Triconnected components of a graph are again graphs, smaller than the given one, that describe all separation pairs of the graph. Each triconnected component belongs to one out of four structural classes: A *trivial* (T) component consists of a single edge. A *polygon* (P) component represents a sequence of components. A *bond* (B) stands for a collection of components that share a common separation pair. Any other component is a *rigid* (R) component.

In this report, we only sketch the simplified procedure for construction of the RPST, whereas for the details we refer the reader to [17]. The simplified procedure for computing the RPST of a workflow graph can be summarized as follows: First, we normalize a workflow graph by splitting nodes that have more than one incoming and more than one outgoing edge into two nodes. We then compute the RPST of the normalized workflow graph, which coincides with its tree of the triconnected components, cf., Sect. 3.1 in [17]. Finally, we project the RPST of the normalized workflow graph onto the original graph and obtain its RPST.

Figure 1(a) shows a workflow graph and its triconnected components. Triconnected components are defined by dotted boxes, i.e., a triconnected component is composed of edges that are inside or cross the boundaries of the corresponding box. The workflow graph in Figure 1(a) is composed of two non-trivial triconnected components: P1 and B1. Note that names of components hint at their structural class. Figure 1(b) shows the tree of the triconnected components of the graph in Figure 1(a), which is an alternative representation of all triconnected components of a graph. Each node of the tree represents a triconnected component that is composed of components that are its descendants in the tree.

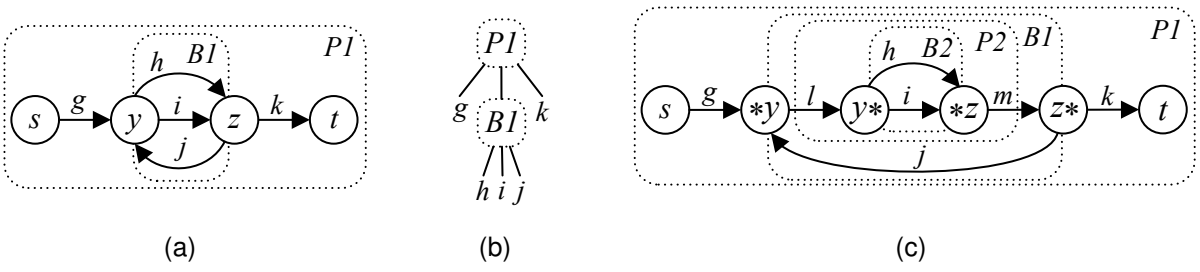


Figure 1: (a) A workflow graph and its triconnected component subgraphs, (b) the tree of the triconnected components of (a), and (c) the normalized version of (a) and its triconnected component subgraphs

Figure 1 demonstrates the concept of node-splitting. If the splitting is applied to node y of the graph in Figure 1(a), it results in the new graph given in Figure 1(c) with three fresh elements: nodes $*y$ and $y*$, and edge l. After splitting nodes y and z in the graph in Figure 1(a), the graph in Figure 1(c) is a normalized version of the graph in

Figure 1(a).

After normalization, the simplified algorithm for constructing the RPST proceeds by computing the tree of the triconnected components of the normalized graph. This tree coincides with the RPST of the normalized graph, cf., [17]. Next, this tree must be projected onto the original graph by deleting all the edges introduced during node-splittings. The deletion of the edges may result in fragments which have a single child fragment. This means that two different fragments of the normalized graph project onto the same fragment of the original graph. We thus clean the tree by deleting redundant occurrences of such fragments. The final stage of the algorithm for computing the RPST of the workflow graph in Figure 1(a) is exemplified in Figure 2.

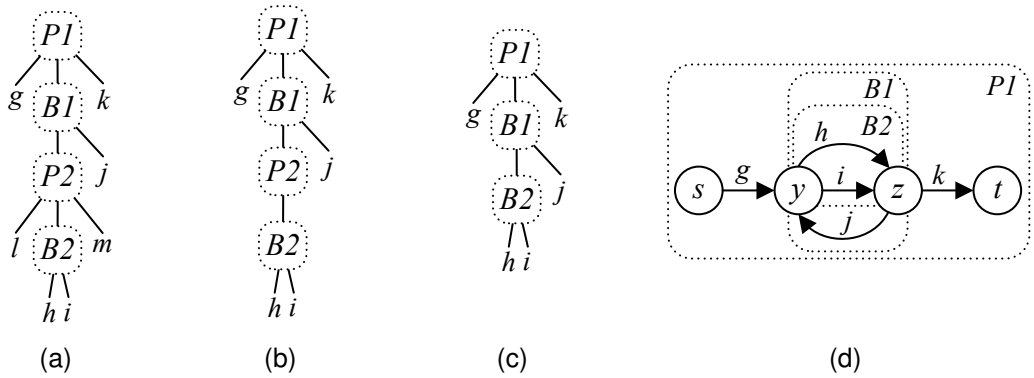


Figure 2: (a) The tree of the triconnected components of the workflow graph in Figure 1(c), (b) the tree from (a) without the fresh edges l and m, (c) the RPST of the workflow graph in Figure 1(a), and (d) the workflow graph from Figure 1(a) and its canonical fragments

The tree of the triconnected components of the normalized graph, cf., Figure 1(c), consists of four triconnected components: $P1$, $B1$, $P2$, and $B2$. Figure 2(a) shows the corresponding tree of the triconnected components. One can see the RPST without trivial fragments that correspond to the fresh edges l and m in Figure 2(b). Observe that $P2$ now specifies the same set of edges as $B2$. Therefore, we omit $P2$, which is redundant, to obtain the tree given in Figure 2(c). This tree is the RPST of the original graph that is given in Figure 1(a). Finally, Figure 2(d) visualizes the graph again together with its canonical fragments. In comparison with the triconnected decomposition shown in Figure 1(a) and Figure 1(b), by following the described procedure we additionally discovered canonical fragment $B2$. $P1$, $B1$, and $B2$ are all the canonical fragments of the workflow graph. For the proof of the fact that resulting tree is indeed the RPST of the original graph we refer the reader to [16].

2 Parsing Ordering Relations

Concurrent systems can be described with the help of ordering relations between pairs of tasks or pairs of occurrences of tasks. There exist different notions of ordering relations, e.g., unfolding relations, cf., [4, 11, 12], behavioral profile [21], relations of the

α mining algorithm [19], etc. These relations are, essentially, behavioral abstractions that capture core behavioral characteristics of a system at different levels of detail. Examples for such behavioral characteristics are causality, conflict, and concurrency. In this section, we discuss a technique of parsing ordering relations that can be applied to any given notion of ordering relations. The parsing decomposes ordering relations into *clans*, each with clear behavioral characteristics specific to the employed notion of ordering relations. To make parsing possible, we give a structural characterization to ordering relations, i.e., ordering relations are treated as a generalization of a directed graph.

The adjacency array representation of a directed graph $D = (V, E)$ is a coloring of a set $E_2(V) = \{(v_1, v_2) \mid v_1, v_2 \in V,\ v_1 \neq v_2\}$, where $E \subseteq E_2(V)$, with two colors, e.g., 0 and 1. Therefore, an adjacency array of a directed graph can be given by an indicator function $I_E : E_2(V) \to \{0, 1\}$. The notion of a *two-structure* is a generalization of the notion of a graph [3]. A two-structure allows an arbitrary coloring of the set $E_2(V)$. A *two-structure* is an ordered pair $S = (N, R)$ such that N is a nonempty finite set of nodes, and R is an equivalence relation on $E_2(N)$.

A two-structure can be seen as a complete directed graph with labeled (colored) edges, where $\alpha : E_2(N) \to C$ is a coloring function corresponding to the edge classes such that $e_1\ R\ e_2$, if and only if $\alpha(e_1) = \alpha(e_2)$; C is a set of colors. Observe that a coloring function α is not unique, as the choice of colors can be arbitrary.

Given the ordering relations, we treat them as a two-structure where nodes are tasks, over which relations are defined, and colors of edges encode different types of relations. An equivalence class of the equivalence relation of such a two-structure represents all ordering relations of the same type, e.g., causality.

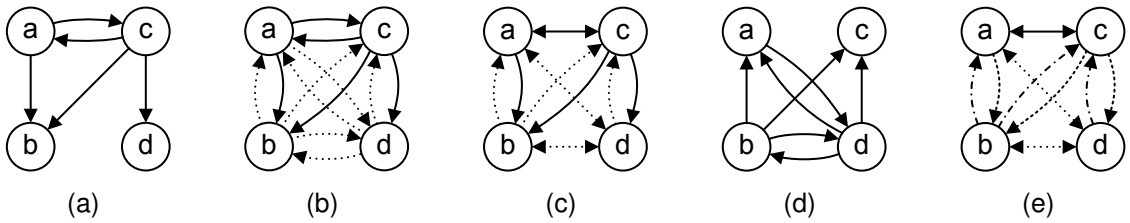


Figure 3: (a),(d) Directed graphs, and (b),(c),(e) two-structures

A directed graph that is defined by the pair (V, E), where $V = \{a, b, c, d\}$ and $E = \{(a, c), (c, a), (a, b), (c, b), (c, d)\}$, is shown in Figure 3(a), whereas Figure 3(b) presents one of the possible corresponding two-structures (N, R). The two-structure has two equivalence classes of edges, where one class contains edges E (drawn with solid edges) and the other one contains edges $E_2(N) \setminus E$ (drawn with dotted edges). Figure 3(c) shows the same two-structure using a simplified notation, i.e., symmetric edges are drawn as two-sided arrows. Notice that the correspondence between the two-structure and the graph is rather arbitrary, as one can also accept the two-structure as such that corresponds to the graph in Figure 3(d) by exchanging the roles of its equivalence classes. Alternatively, one can define a correspondence by using larger sets of colors, e.g., the 2-structure given in Figure 3(e) uses four equivalence classes.

One of the central notions of the theory of two-structures is the notion of a clan. Let $S = (N, R)$ be a two-structure. A node $n \in N$ *distinguishes* nodes $m, k \in N$, if and only if (n, m) and (n, k) are of different colors or (m, n) and (k, n) are of different colors. A *clan* of a two-structure $S = (N, R)$ is a set $X \subseteq N$, such that for all $x, y \in X$ and for all $z \in N \setminus X$ holds $(z, x)\ R\ (z, y)$ and $(x, z)\ R\ (y, z)$.

Let $S = (N, R)$ with $|N| > 1$ and P be a partition of $E_2(N)$ induced by R. It follows immediately that $\emptyset$, N, and the singletons $\{n\}$, $n \in N$, are clans of S. These clans are the *trivial* clans of S. S is complete, if and only if $|P| = 1$. S is linear, if and only if $|P| = 2$ and there exists a linear order $(n_1, \ldots, n_{|N|})$ of elements of N, such that the edges $\{(n_i, n_j) \mid i < j\}$ form an equivalence class of R and the edges $\{(n_j, n_i) \mid i < j\}$ form an equivalence class of R. S is *primitive*, if and only if it contains at least three nodes and all clans in S are trivial.

Construction principles of a two-structure are defined by its decomposition into factors and a quotient that gives the relations between the factors. Let $S = (N, R)$ be a two-structure. A partition $\chi = \{X_1, \ldots, X_k\}$ of N into nonempty clans is a *factorization* of S. The *quotient* of S by a factorization χ is a two-structure $S/\chi = (\chi, R_\chi)$, where $(X_1, Y_1)\ R_\chi\ (X_2, Y_2)$, if and only if $(x_1, y_1)\ R\ (x_2, y_2)$ for some $x_i \in X_i$, $y_i \in Y_i$, $X_i, Y_i \in \chi$. A *decomposition* $(S_{X_1}, \ldots, S_{X_k}; S/\chi)$ of S consists of the factors S_{X_i} with respect to a factorization $\chi = \{X_1, \ldots, X_k\}$ and the quotient S/χ.

A nonempty clan X of S is *prime*, if and only if for all clans Y of S holds that X and Y do not overlap. We denote by $\mathcal{C}(S)$ the set of all clans of S. We denote by $\mathcal{P}(S)$ the set of all prime clans of S. A prime clan is *maximal*, if it is maximal with respect to inclusion among *proper* prime clans of S, where a clan is proper if it is a proper subset of N. We denote by $\mathcal{P}_{max}(S)$ the set of all maximal prime clans of S; if $|N| = 1$, then $\mathcal{P}_{max}(S) = \{N\}$.

The maximal prime clans $\mathcal{P}_{max}(S)$ of a two-structure S form a partition of N, i.e., the domain of each two-structure can be partitioned by the domains of its maximal prime clans. For each two-structure S, the quotient $S/\mathcal{P}_{max}(S)$ is either primitive, or complete, or linear, cf., [3].

By iteratively discovering maximal prime clans and deriving the quotient for each factor that corresponds to an element of the decomposition one builds a hierarchy of quotients. Such a hierarchy is unique for a given two-structure and can be seen as its structural characterization.

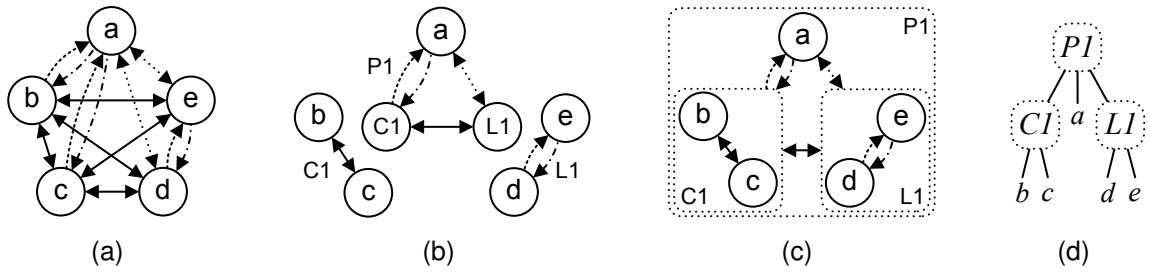
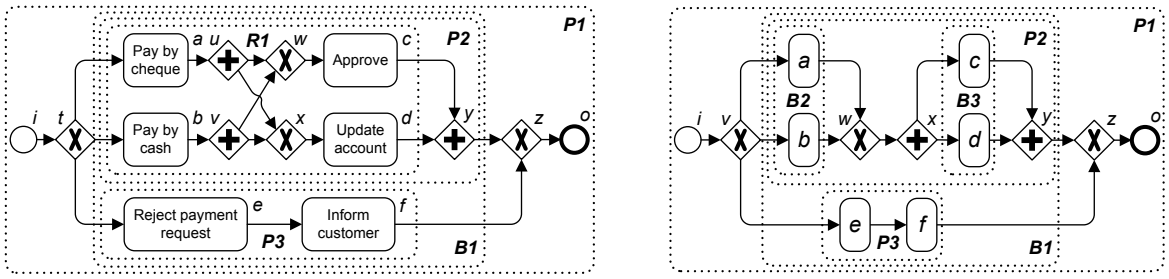


Figure 4: (a) A two-structure, (b) clans of (a), and (c),(d) the hierarchy of clans of (a)

Figure 4 exemplifies the decomposition of a two-structure. Figure 4(a) shows a two-structure which is composed of five nodes and has four equivalence classes on edges. Partition $\chi = \{\{a\}, \{b, c\}, \{d, e\}\}$ is factorization of this two-structure. Two-structures induced by subsets of nodes $\{a\}$, $\{b, c\}$, and $\{d, e\}$ are, respectively, a trivial, a complete, and a linear clan of the original two-structure. Clan $P1$ (of class primitive), cf., Figure 4(b), is the quotient of the two-structure by factorization χ; observe that clan names hint at their class. Finally, Figure 4(c) organizes clans in a hierarchy; each quotient and each nontrivial clan is enclosed in a dotted box with rounded corners, whereas containment of boxes represents the parent-child relation of quotients and clans. Figure 4(d) shows a tree representation of the decomposition.

3 Structuring Acyclic Concurrent Systems

Concurrent systems modeled as graphs can have almost any topology. However, it is often preferable that they follow some structure. In this respect, a well-known property of concurrent systems is that of *(well-)structuredness* [6], meaning that for every node with multiple outgoing arcs (a *split*), there is a corresponding node with multiple incoming arcs (a *join*), such that the set of nodes between the split and the join form a SESE fragment. For example, Figure 5(a) shows an unstructured system, while Figure 5(b) shows an equivalent structured system. Note that Figure 5(b) uses short-names for tasks (a, b, c . . .), which appear next to each task in Figure 5(a). We assume a simple modeling language, i.e., a concurrent system is composed of tasks, events, gateways, and sequence flow edges. We allow exclusive and parallel gateways. Our modeling language can be seen as a basic subset of BPMN.



(a) (b)

Figure 5: (a) Unstructured concurrent system and (b) its equivalent structured version

This section sketches the main idea of the solution to the problem of automatically transforming acyclic concurrent systems, whereas the details can be found in [15]. The motivations for such a transformation are manifold. Firstly, it has been empirically shown that structured models are easier to comprehend and less error-prone than unstructured ones [8]. Thus, a transformation from an unstructured to a structured system can be used as a refactoring technique to increase model understandability. Secondly,

a number of existing analysis techniques only work for structured systems [2, 7]. By transforming unstructured models into structured ones, we can extend the applicability of these techniques to a larger class of models. Thirdly, a transformation from unstructured to structured models can be used to implement converters from graph-oriented process modeling languages to structured process modeling languages, e.g., transforming from BPMN models to BPEL executable code.

As mentioned above, the problem of structuring concurrent systems is relevant in the context of designing BPMN-to-BPEL transformations. However, BPMN-to-BPEL transformations, such as [14], treat rigids as black-boxes that are translated using BPEL links or event handlers, rather than seeking to structure them. A large body of work on flowcharts and GOTO program transformation [13] has addressed the problem of structuring rigid fragments composed of exclusive gateways. In some cases, these transformations introduce additional boolean variables in order to encode part of the control flow, while in other cases they require certain nodes to be duplicated. In [6], the authors show that not all acyclic rigids composed of parallel gateways can be structured. They do so by providing one counter-example, but do not give a full characterization of the class of models that can be structured nor do they define any automated transformation. Instead, they explore some causes of unstructuredness. In a similar vein, [9] presents a taxonomy of unstructuredness in process models, covering cyclic and acyclic rigids. However, the taxonomy is incomplete, i.e., it does not cover all possible cases of models that can be structured. Also, the authors do not define an automated structuring algorithm.

The RPST of a well-structured system contains no rigid fragments. If one could transform each rigid fragment into an equivalent structured fragment, the entire model could be structured by traversing the RPST bottom-up and replacing each rigid by its equivalent structured fragment. Observe that in Figure 5, the only rigid fragment R1 in Figure 5(a) is replaced by an equivalent polygon fragment P2 in Figure 5(b).

Our goal is that the structured system preserves the level of observable concurrency of the equivalent unstructured system, i.e., we require that both systems are fully concurrent bisimilar [1]. The core idea of the structuring method proposed in [15] is to compute the ordering relations, in particular the unfolding relations [4, 11, 12], of every rigid fragment, and to synthesize a structured fragment from these ordering relations (if such a structured fragment exists). To this end, the unfolding relations are computed on the alternative representation of a system, viz. its complete prefix unfolding [11]. An unfolding is a “compact” representation of all concurrent runs (instance subgraphs) of a system. A complete prefix unfolding is a part of the unfolding that contains information about all states that are reachable by the system.

For the technical details on computing unfolding relations we refer the reader to [15]. A two-structure in Figure 6(a) shows the unfolding relations computed for fragment R1 of the system in Figure 5(a). The equivalence relation contains four equivalence classes that represent causality, inverse causality, conflict, and concurrency relations. Figure 6(a) must be read as follows: Solid edges represent the conflict relation ($\{(a, b), (b, a)\}$). Dotted edges stand for the concurrency relation ($\{(c, d), (d, c)\}$). The causality relation is encoded by dash dotted lines ($\{(a, c), (a, d), (b, c), (b, d)\}$). Finally, the inverse causality

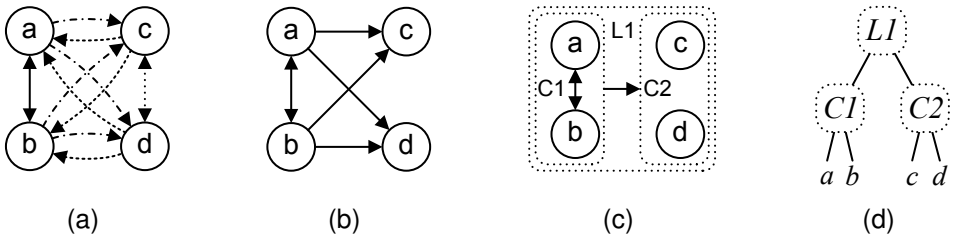


Figure 6: Ordering relations of systems in Figure 5(a) and Figure 5(b) given (a) as a two-structure and (b) as a directed graph. (c) Modular decomposition of (b) and (d) its tree representation.

relation is given by dashed lines ($\{(c,a),(d,a),(c,b),(d,b)\}$). Because of the nature of unfolding relations, the corresponding two-structure can always be represented by an equivalent directed graph, cf., Figure 6(b). We call such a graph the *ordering relations graph*. In this graph, two-sided arrows hint at conflict, absence of an edge between a pair of nodes signals for concurrency, and a directed edge stands for causality.

The important observation in the context of the structuring problem is that the system in Figure 5(b) also exposes unfolding relations that can be represented by the ordering relations graph in Figure 6(b) [15]. However, the well-structured system is not given, rather it needs to be synthesized from the ordering relations graph. To this end, we employ the technique for parsing ordering relations, cf., Sect. 2. Decomposition of a directed graph into clans is known as modular decomposition and can be accomplished in linear time [10]. Figure 6(c) shows the decomposition, whereas Figure 6(d) gives its tree representation.

Finally, we conclude that there exists an equivalent well-structured system, if and only if decomposition of the ordering relations graph of the unstructured system contains no primitive clan, cf., [15]. A complete clan can be represented as a bond fragment as all nodes of a complete clan are pairwise in the same ordering relation. If this relation is the conflict relation, then one can construct a bond with exclusive gateways; in the case of the concurrency relation, on the other hand, one can construct a bond with parallel gateways. A linear clan can be represented by a polygon fragment in the resulting well-structured fragment. Therefore, in order to construct a well-structured fragment, one needs to traverse the hierarchy of clans bottom-up and synthesize a bond fragment for each complete clan and a polygon fragment for each linear clan. For instance, complete clan C1 in Figure 6(d) corresponds to bond B2 in Figure 5(b), C2 corresponds to B3, and $L1$ corresponds to P2.

4 Conclusion

In this report, we have discussed two techniques for parsing two different representations of concurrent systems. Parsing can be used to learn the hierarchical structure of a concurrent system. First, we sketched the simplified algorithm for computing the Refined Process Structure Tree—a technique for workflow graph parsing. Second, we discussed a technique that can be used to parse concurrent systems specified as ordering relations

between pairs of tasks or pairs of occurrences of tasks. Finally, we showed how these two techniques relate to each other in a solution to the problem of structuring acyclic concurrent systems.

References

[1] Eike Best, Raymond R. Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent bisimulations in petri nets. *Acta Informatica*, 28(3):231–264, 1991.

[2] Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *BPM*, volume 5701 of *LNCS*, pages 64–79. Springer, 2009.

[3] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Theory of 2-structures, Part I: Clans, basic subclasses, and morphisms. *Theoretical Computer Science (TCS)*, 70(3):277–303, 1990.

[4] Joost Engelfriet. Branching processes of petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[5] John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing (SIAMCOMP)*, 2(3):135–158, 1973.

[6] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *CAiSE*, volume 1789 of *LNCS*, pages 431–445. Springer, 2000.

[7] Manuel Laguna and Johan Marklund. *Business Process Modeling, Simulation, and Design*. Prentice Hall, 2005.

[8] Ralf Laue and Jan Mendling. The impact of structuredness on error probability of process models. In *UNISCON*, volume 5 of *LNBIP*, pages 585–590. Springer, 2008.

[9] Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. In *BPM*, volume 3649 of *LNCS*, pages 268–284. Springer, 2005.

[10] Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.

[11] Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design (FMSD)*, 6(1):45–65, 1995.

[12] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, volume 70 of *LNCS*, pages 266–284. Springer, 1979.

[13] G. Oulsnam. Unravelling unstructured programs. *The Computer Journal (CJ)*, 25(3):379–387, 1982.

[14] Chun Ouyang, Marlon Dumas, W. M. P. van der Aalst, Arthur H. M. ter Hofstede, and Jan Mendling. From business process models to process-oriented software systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(1), 2009.

[15] Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. In *BPM*, volume 6336 of *LNCS*, pages 276–293. Springer, 2010.

[16] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. Technical Report RZ 3745, IBM, 2009.

[17] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In *WS-FM*, 2010. to appear.

[18] Robert Endre Tarjan and Jacobo Valdes. Prime subprogram parsing of a program. In *POPL*, pages 95–105. ACM, 1980.

[19] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(9):1128–1142, 2004.

[20] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data and Knowledge Engineering (DKE)*, 68(9):793–818, 2009.

[21] Matthias Weidlich, Artem Polyvyanyy, Jan Mendling, and Mathias Weske. Efficient computation of causal behavioural profiles using structural decomposition. In *Petri Nets*, volume 6128 of *LNCS*, pages 63–83. Springer, 2010.

Categorization and Use of Identity Trust

Ivonne Thomas

ivonne.thomas@hpi.uni-potsdam.de

This report summarizes my PhD activities of the past year. Creating a first outline of the thesis revealed some missing parts, which I aimed to fill during these past months. In this report, I am giving a more detailed description of what we call Attribute Verification Context Classes and show how we use them to better match an identity providers haves with the needs of a relying party by considering not only the trustworthiness of the identity provider as a whole, but also for single attributes an identity provider can assert. This allows us to aggregate identity attributes from various sources to fulfill a relying party's policy, herewith providing a more flexible use of digital identities.

1 Introduction

In open environments such as Service-oriented architectures or the Web, participants as users, service providers and service consumer often do not know each other, but nevertheless require information from each other to perform meaningful transactions. Just think of an online store, which requires personal information as our name, address and credit card number to deliver goods and to hold us liable in case anything bad happens. Identity assurance is the degree of confidence another party, such as the online shop, can have in the belief that our identity in the digital world actually matches with our Òreal-lifeÓ identity. Depending on what our identity data is used for, a relying party can have different requirements for the degree of required assurance. In order to achieve a certain assurance level, the relying party usually implements and enforces adequate verification processes. However, with the emerge of open identity management system, the problem of identity assurance is not an isolated problem of the relying party anymore, but has moved to the open world of the Internet. In open identity management systems, identity information is not necessarily hold by the party that is using it. Instead, designated services, so called identity providers, hold identity information of users for the purpose of provisioning it to parties, that are willing to rely on it.

Current approaches for identity assurance foster a model that assesses identity providers as a whole, meaning an identity provider can conduct an independent audit, that assesses all processes, used technologies and protection mechanisms in place that have an influence on the trustworthiness on the statements of this identity provider. However, this approach has some limitations.

1.1 Limitations of current assurance frameworks

Different trust levels for the same attribute – Referring to the identity as a whole makes it hard to reflect trust requirements of specific attributes. An identity provider could for example manage self-asserted attributes besides verified attributes to meet different trust requirements of relying parties. In fact, in blogs and forum discussions, users often prefer using pseudonyms rather than using their real life identity.

Change of trust levels over time – Also, using existing assurance frameworks, it is hard to reflect possible changes of a user's identity trust level over time. As identity proofing processes are cost-intensive and time-consuming due to the effort required to verify a user's identity attributes, a verification of an attribute might not be desired as long as a user is not involved in transactions that demand a higher trust level. Therefore a user might decide to register with an identity provider without proper identity proofing, having for example his/her name self-asserted and getting involved in the identity proofing only upon concrete requirement. This requires a different trust level per user and does not allow to rate an identity provider as a whole.

Diversity of identity providers – Furthermore, identity providers are inherently different due to their affiliation with an organization or institution and might be suitable for asserting certain identity attributes only to a limited extent. For example, a banking identity provider will be in particular suitable to assert that a user can pay for a certain service, but might have weak records of the user's status as a student while for a university's identity provider it would probably be the opposite.

In my research, I am developing a more flexible solution to express trust requirements for identity attributes issued by various identity providers. The following Section 2 shows an example scenario to demonstrate the research focus. Afterwards Section 3 depicts briefly the underlying two layered trust model and describes our categorization of identity trust and how we use this information in policies and attribute requests to allow a more flexible choice of identity providers.

2 An Example Scenario

Let's have a look at the following scenario. The scenario has three different identity providers and an identity consumer. The identity providers issue identity information as attributes conveyed in security tokens as well as meta information about these attributes. The identity consumers, so called relying parties, have a policy, in which they state which attributes they require. Current technologies as OpenID or Information Card only offer the possibility to express (a) which attributes are required and (b) whether any identity provider or a specific one should be used. Using Web Service Technologies possibilities are a bit wider: WS SecurityPolicy allows to state a list of required attributes *per* issuer (identity provider).

In our scenario, we go further and aim at expressing attribute needs with assurance requirements as in the following examples:

- The relying party requires an attribute *name* from the user who proved his name

by registering *in-person* at a *federated* identity provider.

- The relying party requires a *verified eMail* address issued by *any* identity provider and the *name* and *age* of the user issued by an *authorized eID Service* (electronic ID card).

- The relying party requires a *verified student attestation* from an identity provider with an *ICAM [1] trust level of at least 3*.

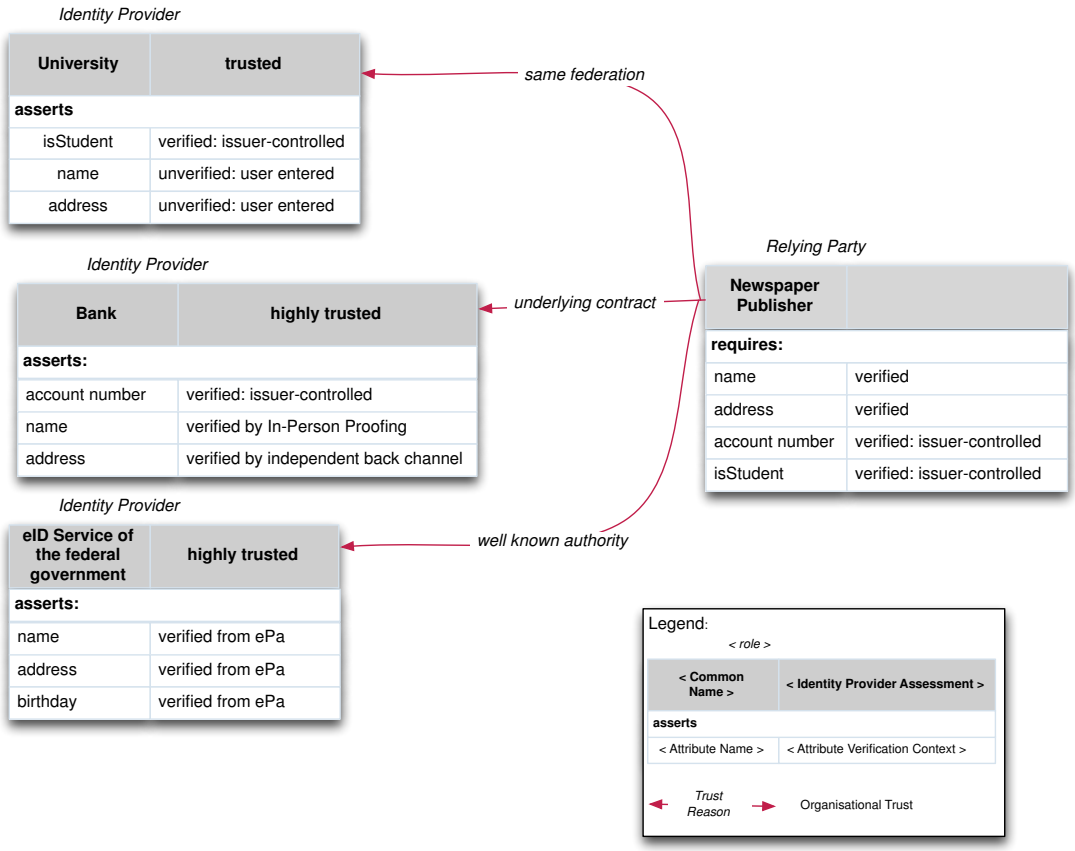


Figure 1: A motivating scenario.

In the scenario, we have different assessments for the parties taking the role of the identity provider as "highly trusted" or "trusted" that are based on the relying party's assessment. And we have different attribute verification classes that contain meta information about the verification of the attribute. Based on this information, the relying party formulates its policy and a client resolves the possible sources for retrieving the identity attributes. Hereby it is possible (and maybe even necessary) to request these attributes from multiple sources and to aggregate them for the authorization step.

3 Two Layered Trust Model

As can be seen also in the example above, we distinguish two different layers of trust in our trust model: *organizational trust* and *identity trust*. First, a trust relationship is required between the service provider and the identity provider in order to trust the correctness of the assertions (= organizational trust) and second, for a concrete transaction, the service provider has to decide whether the identity-based information in the assertions are sufficient to reach a certain trust level which is required to perform the request (identity trust).

Our two layered trust model comprises the following elements: facts, attributes, attribute verification contexts, identity provider, relying parties, organizational trust, organizational trust level (identity provider trust level), identity trust as well as the concept of a knowledge base.

An *identity provider* is an agent which provisions attributes to independent *relying parties* who are willing to rely on them. A *relying party* is a trusting agent that is consuming identity data.

A *fact* is a statement about an *identity provider*, such as "*Identity Provider* I_1 *can assert attributes* A_1 *and* A_2". A *trusted fact* is a known fact, that is trusted by the relying party. The set of facts that a relying party knows constitutes its *knowledge base* about an identity provider.

Organizational Trust describes the relationship between an identity provider and a relying party and is characterized by an *organizational trust level*. *Identity Trust* describes the believe into the identity of a subject and its behavior. An identity of a subject is reflected by several digital identities in the digital world. Each digital identity comprises a set of attributes that characterizes a subject's identity. An *attribute* has a well-known identifier and a value, that is hold by an identity provider on behalf of a subject. An *Attribute Verification Context* is the justification that an attribute can be trusted.

3.1 Categorization of Identity Trust

As described above, Identity Trust refers to the trust an entity such as a service provider has into the identity of a subject and its behavior. As described in previous reports, identity trust comprises different aspects, such as (a) trust into the authentication process and the subject-to-account mapping, (b) trust into the token and (c) trust into a subject's attributes. In this report, we focus on the trust into the subject's attributes and provide a classification (ontology) of Attribute Verification Context Classes that represent how an attribute has been verified by the identity provider. Figure 2 shows our ontology to describe various verification methods and their applicability to identity attributes. Our attribute verification context classes denote general verification schemes that can be applied to several attributes, but might be implemented in different ways. For example, the verification of an attribute by an independent back channel can be done for eMail addresses by sending an email to the claimed address with a verification link in it. The same scheme can also be used to verify a bank account. In this case a small amount of money (1 cent) is usually transferred to the claimed account

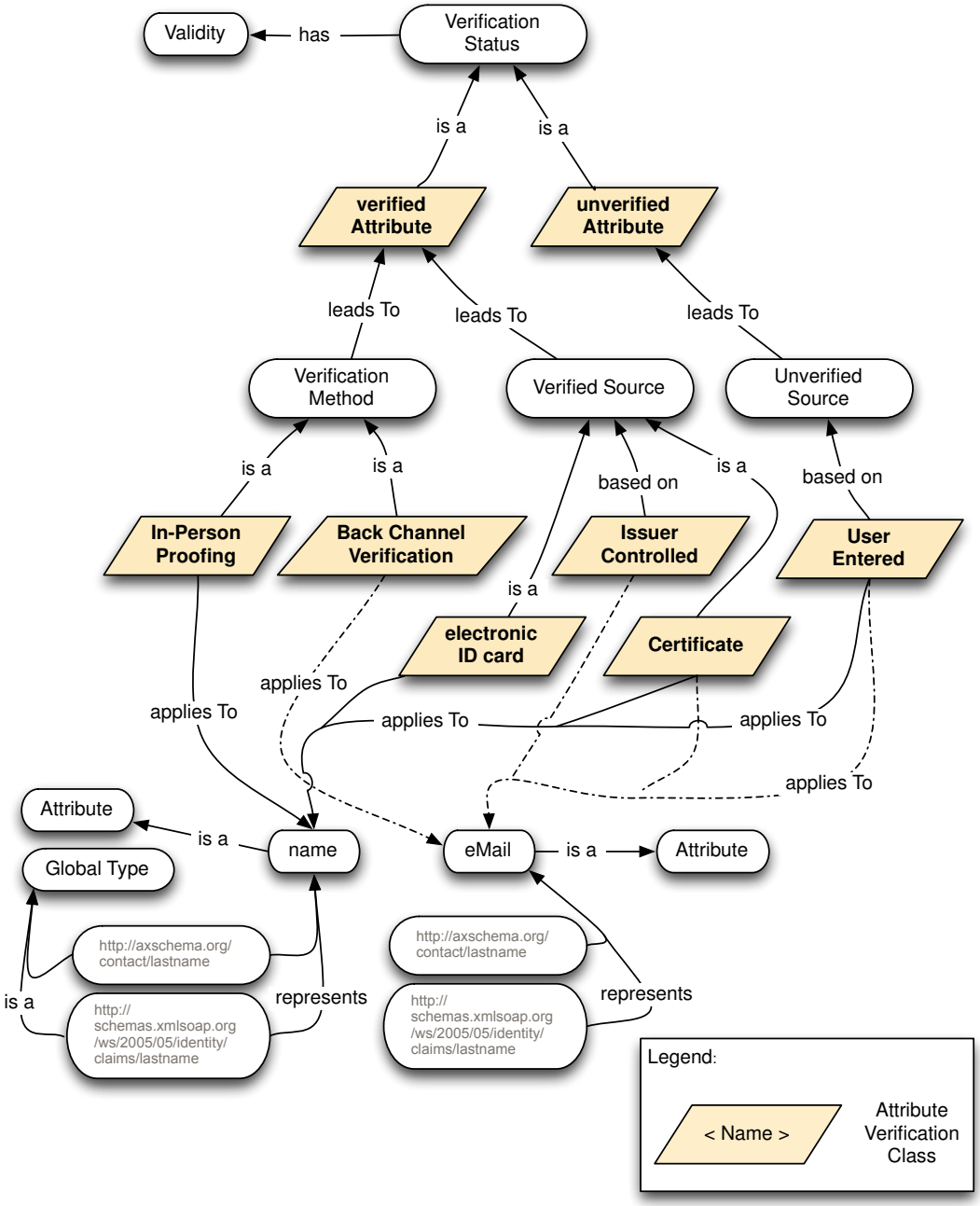


Figure 2: Identity Trust Ontology.

with a password in the transaction data, that the user needs to enter later on to prove that s/he is the owner of the account.

For each verification context class, we state the attributes that are eligible to be verified in the given manner. However, for the sake of readability, figure 2 only shows the suitable verification context classes for the two attribute *email* and *lastname*. A complete assignment for a common set of globally know attributes is given in table 2 below.

3.2 Verification Classes for Identity Attributes

This sections describes each of the attribute verification context classes depicted in our identity trust ontology in figure 2 in more detail. The list is a proposal based on experience with existing assurance frameworks and can be extended and adapted to match the needs of a given use case.

In-Person proofing is an attribute verification class that is also used in most assurance frameworks. As required evidence for the attribute value, the verifying authority has to ensure that the applicant is in possession of a primary Government ID document that bears a photographic image of the holder and that this image matches with that of the applicant. Furthermore it has to be ensured that the presented document appears to be a genuine document properly issued by the claimed issuing authority and valid at the time of application.

Back-channel proofing denotes a verification method by which a claimed attribute value is proven by sending some information via another communication channel than the one used to claim the value.

Issuer-controlled is a verification class that can be used when the holder of the identity information is equal to the creator of the identity information or is in control of this identity information, as for example in case of email providers for email addresses or a company in case of affiliation claims .

User-entered denotes a verification class that is used when identity data is entered by the user or received from a user-like source without any further verification.

Proof by electronic ID card This verification class is used when an electronic ID card has been read by service approved by the government (in Germany: eID Service Providers)

Proof by certificate This verification class is used when a certificate has been presented by the applicant and it has been ensured that the presented certificate was valid at the time of application.

Verified attribute is a a very high level attribute context class that subclasses all context classes that are based on a verification method or a verified source.

Unverified attribute is also a very high level attribute context class that denotes that an attribute has not been verified by an accepted verification method nor issued by an accepted verified source.

3.2.1 Assigning Identity Attributes To Verification Methods

The following table assigns attributes to the attribute context classes defined in the previous section.

Verification Context Class	**applies to**
In-Person Proofing	given name, family name, gender, date of birth, address
Back-channel Proofing	eMail, telephone number, address, credit card number (given name), (family name)
Issuer-controlled	eMail, telephone number, credit card number, (address)
User-entered	eMail, given name, family name, gender, date of birth, telephone number
Proof by electronic ID card	given name, family name, gender, date of birth, address
Proof by certificate	given name, family name, email, (etc.)

Table 2: Globally known Attributes and Applicable Attribute Verification Context Classes

4 Formalisation and Implementation

We formalized and implemented our model using horn clauses as an effective way to represent the knowledge base. As said above the set of trusted facts constitutes a relying party's knowledge base about its trusted identity providers and their ability to assert attributes with a certain attribute verification context. Given this formalization, we can easily reason over this knowledge base and express policies.

4.1 Formalization

We formalize this knowledge base in the following way:

fact1:= *Attribut*(A)	A *is an attribut*
fact2:= *IdentityProvider*(I)	I *is an identity provider*
fact3:= *IdPTrustLevel*(T)	T *is a trust level of an identity provider reflecting the organizational trust relationship.*
fact4:= *attributeVerificationContext*(V)	V *is an attribute verification context class*
fact5:= *applies*(V,A)	*Attribute verification context class* V *can be applied to Attribute* A
rule1:=*corresponds*(V_1, V_2)	*Attribute verification class* V_1 *equates to attribute verification class* V_2
fact6:= *hasTrustLevel*(I, T)	*Identity provider* I *has trust level* T
fact7:= *federatedIdP*(I)	*Identity provider* I *is a federated identity provider*
fact8:= *trustedIdP*(I)	*Identity provider* I *has been marked trusted by the relying party*

rule2:= *isTrusted*(I) :=*trustedIdP*(I) $\vee$ *hasTrustLevel*(I, T) $\vee$ *federatedIdP*(I)	*Identity provider* I *is trusted if* I *is marked trusted by the relying party* or if I has a trust level of at least T or if I is a federated identity provider
fact9:= *assert*(I, A, V)	*Identity provider* I *can assert attribute* A *with verification level* V

4.1.1 Revisiting the Example Scenario

Revisiting the example scenario from Section 1, we can express the example requirements from the Section 1 in the following way:

- Given *Attribut(name)*, *Attribut(isStudent)*, *attributeVerificationContext(In-Person Proofing)*, *attributeVerificationContext(verified)*, *corresponds(In-Person Proofing, verified)*, etc.:
- The relying party requires an attribute *name* from the user who proved his name by registering *in-person* at a *federated* identity provider.

  assert(I*, name, In-Person Proofing)* $\wedge$ *federatedIdP*(X) = $true$
- The relying party requires a *verified student attestation* from an identity provider with an *ICAM trust level of at least 3*.

  assert(I*, isStudent, verified)* $\wedge$ *hasTrustLevel*(X*,ICAM3)* = $true$

4.2 Implementation

We use Prolog as a functional programming language to reason over the relying party's knowledge base and to match requirements of the relying party with the possible sources.

Given our model and formalization, we can express identity provider characteristics and relying party's requirements in an easy and extendable way. In order to support a complete web service and web-based scenario, we extend existing web and web service technologies as OpenID, SAML and WS-Security Policy by mechanisms to

1. express a relying party's requirements as policy, e.g. as WS-Security Policy or inside the $<$object-tag$>$ using Information Cards
2. choose an identity provider from a set of possible identity providers
3. formulate a request for identity attributes with a certain verification context to the prospective identity providers
4. assert requested attributes with a certain verification context
5. aggregate identity information from multiple sources and pass it to the authorization component of the relying party

Parts of this implementation have for example been described in [2].

5 Conclusion and ongoing work

The need to trust on information received from a foreign party is inherent to open identity management systems. If a relying party has to rely on identity information received from a foreign party, the need for assurance that the information is reliable is a natural requirement prior to using it. Unfortunately, existing identity assurance frameworks assess identity providers mostly as a whole which leads to the situation, that identity attributes are mostly requested from a specific trusted identity provider if the trust requirements are high or from any identity provider, if there are no trust requirements. In our model, we aim at providing trust information on the level of identity attributes, especially about the verification process, and to use this information in policies and attribute requests to allow a more flexible choice of identity providers as well as an aggregation of identity attributes from multiple sources. Therefore, we defined Attribute Verification Context Classes that describe these differences of attribute trust in on ontology that is easily extendable and adaptable to a particular use case. We formalized our model, in a way, which allows us to express the haves and needs of identity providers and relying parties and to match them by not choosing only one identity source, but allowing a dynamic aggregation of identity information from multiple sources.

Evaluation As a proof of concept we are currently implementing our approach using different technologies such as OpenID and web services, hereby extending them to meet our needs. We also work on setting up a small simulation environment to compare the flexibility that we achieve with our approach with current solutions and to provide an evaluation for our concepts.

6 List of Latest Publications

2010

- Ivonne Thomas, Christoph Meinel: *Conceptual Evolution of Identity Management From Domain-based Identity Management Systems to Open Identity Management Models*, Book Chapter In Digital Identity and Access Management: Technologies and Frameworks, IGI Global, To be released: 2011. (In Review)

- Ivonne Thomas, Christoph Meinel: *Identity Assurance In Open Networks* Book Chapter In Strategic and Practical Approaches for Information Security Governance: Technologies and Applied Solutions, IGI Global, To be released: 2011. (In Review)

- Ivonne Thomas and Christoph Meinel: *An Identity Provider to manage Reliable Digital Identities for SOA and the Web* In Proceedings of the 2010 ACM 9th Symposium on Identity and Trust on the Internet (IDTrust 2010), Gaithersburg, USA, April 13 - 15, 2010.

- Michael Menzel, Robert Warschofsky, Ivonne Thomas, Christian Willems and Christoph Meinel: *The Service Security Lab: A Model-Driven Platform to Compose and Explore Service Security in the Cloud.* In Proceedings of the 2010 IEEE World Congress of Services at ICWS/SCC, pp.115-122, Miami, USA, Juli 2010.

2009

- Martin Wolf, Ivonne Thomas, Michael Menzel, and Christoph Meinel: *A Message Meta Model for Federated Authentication in Service-oriented Infrastructures* In Proceedings of the 2009 IEEE International Conference on Service-Oriented Computing and Applications (Los Alamitos, CA, USA, 2009).
- Ivonne Thomas and Christoph Meinel: *Enhancing Claim-Based Identity Management by Adding a Credibility Level to the Notion of Claims* In Proceedings of the 2009 IEEE International Conference on Services Computing (SCC-09) (Bangalore, India, Sept 21 - 25).
- Michael Menzel, Ivonne Thomas, Benjamin Schueler, Maxim Schnjakin, and Christoph Meinel: *Security Requirements Specification in Process-aware Information Systems*, Highlights of the Information Security Solutions Europe (ISSE) 2009 Conference, vol. , Vieweg-Verlag, 2009.
- Regina N. Hebig, Christoph Meinel, Michael Menzel, Ivonne Thomas and Robert Warschofsky: *A Web Service Architecture for Decentralised Identity- and Attribute-based Access Control*, In Proceedings of the 2009 IEEE International Conference on Web Services (ICWS-09)(L.A., USA, July, 2009).
- Uwe Kylau, Ivonne Thomas, Michael Menzel, and Christoph Meinel: *Trust Requirements in Identity Federation Topologies* In Proceedings of the 2009 IEEE International Conference on Advanced Information Networking and Applications (AINA-09)(Bradford, UK, May 26 - 29, 2009).
- Michael Menzel, Ivonne Thomas, and Christoph Meinel *Security Requirements Specification in Service-oriented Business Process Management* In Proceedings of the 2009 IEEE International Dependability Conference (ARES-09) (Fukuoka, Japan, March 16 - 19, 2009).

References

[1] ISIMC. Identity, Credential and Access Management. http://www.idmanagement.gov, September 2010.

[2] Ivonne Thomas and Christoph Meinel. An identity provider to manage reliable digital identities for soa and the web. In *IDTRUST '10: Proceedings of the 9th Symposium on Identity and Trust on the Internet*, pages 26–36, New York, NY, USA, 2010. ACM.

Enabling Reputation Interoperability through Semantic Technologies

Rehab Alnemr
rehab.alnemr@hpi.uni-potsdam.de

In this technical report I show my contribution in designing and implementing a semantic artifact for reputation representation that agrees with the theoretical and social formation and processing of reputation information. This report also illustrates how the research pieces that i have worked on fit together as well as the last step to finalize my PhD.

1 Introduction

Reputation is a complex concept that has a major role in fields like social sciences, economics as well as computer science. Representing it as a simple form of property-rating or a vector of ratings strips it from its original notion and postulation. It does not also facilitate the derivation of meaningful conclusions from it. This work presents a semantic model for the representation of reputation as a complex object; Reputation Object (RO). The model facilitates reputation interoperability and portability using semantic technology. In previous work, the line of argumentation went through:

- showing why representing simple rating is not enough, how can a design of a reputation object help in capturing the social formation of reputation information, and explaining the formal model [3] [5] [1] [4],
- tools and applications of this design [14] [8] [2],
- how it is implemented and integrated in several domains [4] [10] [13].

In this report I continue with explaining the details of the ontology and the implemented library, and also introduce the latest use cases; reputation management approach for a reasoning agent-based systems called Rule Responder [11] (accepted in the upcoming RuleML2010 Conference[1] [10] and a reputation service for a cloud service provider selection [13] (accepted in the upcoming CIKM ClouDB2010 workshop[2]). Recent publications include: [2] [13] [10] [4] and one paper in review [12]. In the conclusion and future work section, the last step in the thesis is explained.

[1]The 4th International Web Rule Symposium: Research Based and Industry Focused
[2]The Second International ACM Workshop on Cloud Data Management

2 Reputation Object: Model and Ontology

The issues raised in previous work led to the development of the model described in this section. Here, I describe the model and its benefits abstracted/detached from the semantic technologies used. This object is constructed to hold a profile of the behavior or performance of an entity in several contexts. For example, in the e-market domain a seller's reputation object reflects his expected performance and rating in several criteria such as *product-quality, payment-methods*, and *delivery*. Each criterion in this list has a numerical value, a string, or a reference to an object value describing the evaluation of this particular criterion. Moreover, a set of criteria (e.g. price, payment method) can be aggregated to be represented by one context (i.e. financial). Aggregating a set of criteria to a single context can be done to enhance the usability of the reputation object (i.e. if it is visible to users or agents) and also to ease the ontology matching process when comparing between two reputation objects (i.e. to relate a criterion's meaning like a *"payment method"* to its general domain or topic which is *"financial"*).

The Reputation Object (RO), however, is more than a flat list. The model structure (Figure 1) contains a description of how this value is collected (e.g. by community ratings or monitoring service), the computation function (for this criterion) used to aggregate - or recompute - the values each time a new one is entered, and a history list (previous values dated back to a certain time slot). This enables the destination system to map its perception (or its reputation computation function) to the one used in computing this value (i.e. a "very-good" value in system A can be "good" value in system B). The reputation object in this case is seen as a profile of the entity's expected performance which is constructed using different information sources. The degree of visibility for these criteria to the community's users (i.e. how many criteria presented for users in a web site) depends on the community (i.e. a web site can limit the number of criteria for usability reasons). The model therefore achieves several goals:

- the reputation of an entity is more meaningful because it is associated with the context in which it was earned
- automation of criteria assignment is possible by declaring a relevant resource as a criterion (ex. `URI1 is_a _:criterion`)
- one can easily extend these criteria list dynamically by adding to the list of contexts/criteria in the reputation objects

The goal of the work is to have a standard way to represent the reputation of one entity to be understood by any other entity in a different system or domain. Therefore, the aim is to embed more information within the reputation statements with an explanation (or the semantics) of how to interpret it. This model is generic enough to be used in any domain, but also can be domain-specific by incorporating information (i.e. its contexts, criteria list, and quality processes) that is specific to this domain. In a service oriented environment (SOA), a service registry can use combined sources for service's quality assessment (which leads to building the reputation object) such as service description, invocation analysis, history, rating, meta-data, and elements in Service Level Agreements (SLAs).

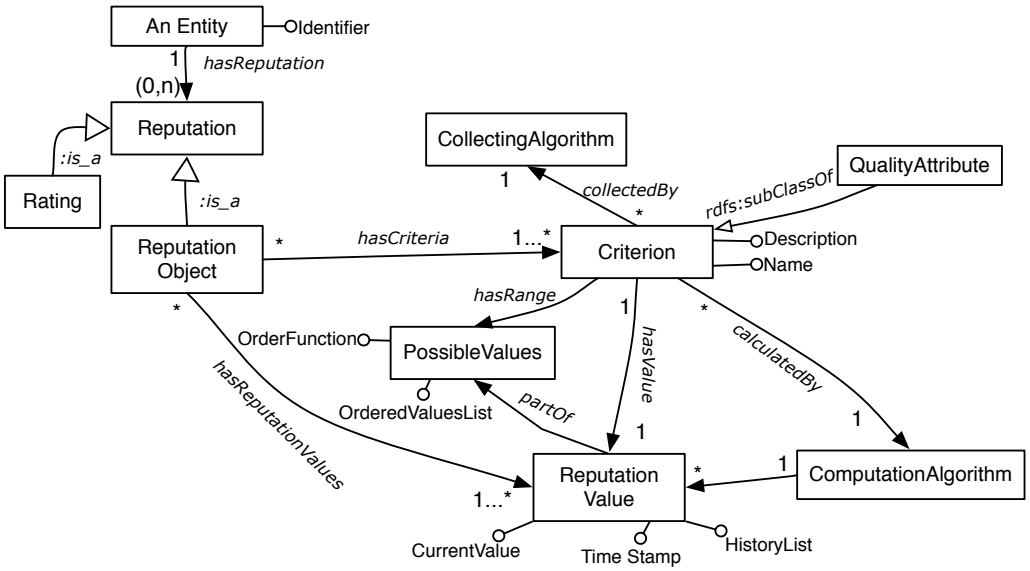


Figure 1: Reputation Object Model

One of the benefits of using such model is that regardless the domain that using it, there will be enough information for better decision making. In SOA, having a profile about a service performance (i.e. its reputation object) facilitates customized service selection. The same applies for domains like e-markets (selecting a seller based on a consumer's preferences), in cloud environment (selecting a cloud provider based on the company's customized priorities), in agent-based communities (constructing trust relationships by gossiping about agents' reputation), and in SLA-breach management (identifying violation-prone services at service selection phase [8]). Not to mention the future vision of being able to exchange reputation information between related communities such as eBay and Amazon, credit cards databases and C2C money transfer systems, social networks, etc.

3 Model development using Semantic Technologies

Achieving the goals described in the previous section requires a technology that provides common data representation framework as well as a way to connect concepts with their definitions. Semantic Web is developed with a main objective of facilitating data integration, enhancing information usage by connecting it to its definitions and context. [6] RDF is used as a mechanism for data integration across applications and the web.[3] Therefore, it was only to be expected that Semantic Web is the technology of choice to achieve reputation portability and interoperability. Developing the model using semantic web technologies achieves:

- seamless interaction between agents of different domains
- the goal of exchanging reputation information (and knowledge) and its meaning

[3]RDF Vocabulary Description Language: `http://www.w3.org/TR/rdf-schema/`

- reputation interoperability
- the development of context-aware reputation
- customized service-provider selection
- understandability and reusability of the embedded reputation information

3.1 A Simple View: Reputation Objects in RDF Graphs

A reputation statement usually describes the target of the statement, the topic of evaluation, and the value of this evaluation (i.e a judgment or a result of monitoring process). When talking about someone's reputation most of the time one would describe it in a set of such statements. These type of statements correspond to the RDF statements (or triple) form of: `<subject, predicate, object>`, where the reputation statement in this case is: `<target, context, value>`. The same as an RDF graph which is a set of RDF triples, the set of reputation statements therefore form a reputation RDF graph. Lets assume that we are rating a seller in an e-market identified by `<foaf:Person rdf:nodeID="Bob">` then a simple description of his reputation can be viewed as declaring the statements in table 1. If Bob's *servie-quality, delivery*, and *payment* are identified by URIs and evaluated by the literal values *0.87* and *"very good"*, this table corresponds to the RDF graph instance shown in figure 2 where the edges represent the `context`. This is a snippet of the reputation object instance that describes Bob's reputation in different criteria:

```
RO={<Bob,quality,0.87>,<Bob,delivery,‘‘very good‘‘ >,<Bob,payment,gr:MasterCard>}
```

Target	Criterion	Value
Bob	Service Quality	0.87
Bob	Delivery	"very good"
Bob	Payment	purl.org/goodrelations/v1/MasterCard

Table 1: Reputation Statements about Bob

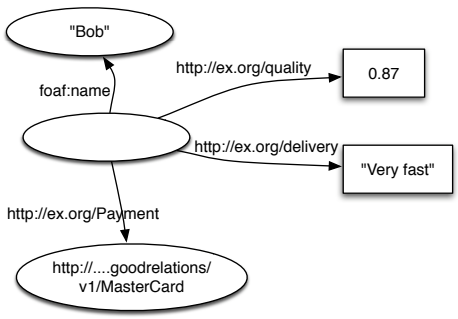


Figure 2: A graph describing part of Bob's reputation

For the same person "Bob" who is identified by a given URI, more statements can be asserted about him and easily *merged* to the graph representing his reputation if the

predicate is a new criterion. If a new statement has an equivalent criterion, then the reputation value (object) is aggregated (or recomputed, according to the computation function) to produce a new current value for this criterion.

3.2 Reputation Expressiveness via Reputation Object Ontology

The next step was to formalize and develop the model components and concepts using a proper technology. RDFS (RDF Schema) can be used to describe the model classes (ex. Reputation,ReputationObject,Context,Criterion, etc.) and properties (ex. reputationValue,hasCriterion, etc.). However, after developing the schema and testing it using some use cases, we found that the model needs to be described using more expressive method. Restrictions and axioms of the model should be incorporated in its description such as: how is the reputation value obtained, can a criterion refer to another concept (criterion matching) in other platforms, how to aggregate values of this concept if a new evaluation value is entered, can a set of criterion be aggregated in one context, how many reputation objects can an entity have, can the reputation object be extended, cardinality, inverse relationships, influencing factors, etc.. In such case, ontologies are used to provide such level of expressiveness. We have developed an OWL ontology to represent an entity's (foaf:Agent) reputation object. Tables 2 and 3 have a description of the classes and their properties. A ReputationObject has:

1. hasCriteria: one or multiple instances of class Criterion or QualityAttribute (for a service, the criterion describing service reputation is referred to as a quality attribute). The criterion is collected using a CollectingAlgorithm and hasValue ReputationValue.

2. hasReputationValues: each criterion instance has a ReputationValue (which includes the currentValue, its time stamp, and a simple list of its previous values called historyList) that in turn has the range of values defined in PossibleValues. It describes the data type that the criterion can have or a specific set of values (literals or resources URI) evaluating this criterion (e.g. a set of integers $\{1, 2, 3, 4\}$ describing 4 trust levels or a set of Strings $\{''good'','' bad'','' excellent''\}$ describing a user opinion). Each time a criterion is being evaluated (i.e. a new entry value for this criterion), a new currentValue is calculated using the ComputationAlgorithm which is the reputation computation function used with this criterion such as *sum, average, etc.*.

Since it is not always the case to identify intuitively what the highest reputation value is -among the defined possible value set, for instance-, the PossibleValues class has an orderedList that is ordered from the relatively highest reputation value to the lowest (e.g.$\{''excellent'','' good'','' bad''\}$). It has also the possibility to define a comparison and ordering function; OrderFunction. For example: if the criterion is a student grade-float number from 1 to 4 representing the GPA- the function is "greater than" when it is American GPA (4 is the highest) and the function is "less than" when it is German GPA (1 is the highest). In the presented ontology, the pattern *OWL-List* [7] is used to retain the order of the list. The ontology makes use of other vocabulary such as OWL, RDFS,

FOAF, XSD, and can integrate with vocabulary such as Trust or RDF Review to describe one criterion in a reputation object. Using this ontology, the object representing the reputation can be transferred within a domain or to another domain without negotiating on its format or semantics. The advances in ontology matching techniques ensures the matching between the criteria of a reputation object in one platform to be used in another platform.

Classes	Description
Reputation	An abstract Reputation
Rating	A single Rating representing an entity's reputation, refers to other ways of representing reputation, subclassof:Reputation and has one owner
ReputationObject	A Reputation Object of an entity related to multiple criteria, subclassof:Reputation and has one owner
ReputationValue	contains the current value of the criterion along with its past values if they exist
PossibleValues	A set of possible values that a criterion in a reputation object can have (literals or resources) which can be a static predefined set or a general range
Context	A reputation context that represent multiple criteria
Criterion	A reputation criterion to be evaluated and saved in a reputation object
QualityAttribute	A reputation criterion regarding quality measuers
Algorithm	A methodological method, entry point, or an engine
ComputationAlgorithm	The method used to compute or aggregate several reputation values (i.e. reputation function)
CollectingAlgorithm	The engine or method used to collect the value of a reputation criterion

Table 2: Reputation Object Model Ontology Classes

3.3 Implementation

We used Protégé-OWL tools[4] in the development of the ontology. Currently, the ontology is being tested for stability using the default reasoning engines and adjusted accordingly. Implementation for *reading, writing, and processing* an RO along with the selection method (given a consumer priority list) was developed in Java using Jena-API[5] which facilitates the integration of the model in any system on the implementation layer.

[4]Protege OWL: http://protege.stanford.edu/overview/protege-owl.html
[5]Jena framework: http://jena.sourceforge.net/

Properties	domain:	range:
hasReputation	foaf:Agent	ReputationObject, Rating
hasCriteria	ReputationObject	Criterion or QualityAttribute
hasReputationValue	Criterion or QualityAttribute	ReputationValues
historyList	ReputationValue, list of the past values for a criterion in a particular time slot	Collection of PossibleValue
currentValue	ReputationValue, describes the current value of a criterion	PossibleValues
hasRange	Criterion or QualityAttribute	PossibleValues
orderedValuesList	PossibleValues, describes the order of the possible values for a criterion to be able to compare between 2 values	OWLList
orderFunction	PossibleValues, describes the comparison function (i.e. between two given reputation values) and is used as an alternative to order a dynamic set of possible values if a static list is not given	Algorithm
calculatedBy	Criterion or QualityAttribute	ComputationAlgorithm
collectedBy	Criterion or QualityAttribute	CollectingAlgorithm
hasRatingValue	Rating	type:literal

Table 3: Reputation Object Model Ontology Properties

In order to test the use of ROs in a reasoning environment, we have been working on integrating ROs in the Rule Responder system[6]. For declarative processing of the semantic reputation objects we make use of rules. Reputation objects are attached to the rule-based services (agents). Rule Responder allows to deploy distributed rule inference services running a local rule engine such as Prova or Drools on an enterprise service bus. [11] The rule services, which can act as multi-agents, can communicate with each other using Reaction RuleML[7] as a standard rule interchange format. The reputation values can be used in the agent's rule logic, e.g. to implement access policies, information dissemination rules or decision management strategies. For instance, an agent might reveal more information to a trusted requestor or might internally prioritize incoming requests from other agents according to the details in their reputation objects. That is, an agent in Rule Responder can manage reputation locally in its knowledge base, but can also communicate reputation objects to other agents. For the implementation of the application scenarios in the following section (4.1) we used the

[6]Rule-Responder: http://responder.ruleml.org
[7]RuleML Initiative: http://ruleml.org/

Prova Semantic Web rule engine[8] in Rule Responder, which supports using Semantic Web ontologies as type systems and allows queries to RDF data. [9]

4 Applications

As explained before, the model can be used in several domains and it was shown in previous publications how it can be beneficial to these domains. Here, I show our latest work on integrating the RO model in the two presented domains.

4.1 Reputation Management in Rule Responder

We developed a new architectural design artifact for a reputation management system which is distributed on the (Semantic) Web. This work is recently published in [10]. In this paper we introduced a reputation management system based on distributed rule agents, which uses Semantic Web rules for implementing the reputation management functionalities as rule agents and which uses Semantic Web ontologies for representing simple or complex multi-dimensional reputations. For the architecture we presented a distributed *Reputation Processing Network (RPN)* consisting of *Reputation Processing Agents (RPAs)* that have two different roles:

1. *Reputation Authority Agents (RAAs)*: act as reputation scoring services for the reputee entities whose Reputation Objects are being considered or calculated in the agents' rule-based Reputation Computation Services (RCSs). A RCS runs a rule engine which accesses different sources of reputation (input) data from the reputers about an entity and evaluates a RO based on its declarative rule-based computational algorithms and contextual information available at the time of computation (described in the RO by the `Criterion` and its `PossibleValues` along with its `ComputationAlgorithm`, `orderedList`, and `OrderFunction`).

2. *Reputation Management Agents (RMAs)*: -aka reputation trust center- provide reputation management functionalities. A RMA manages the local RAAs providing control of their life cycle in particular and also ensuring goals such as fairness. It might act as a Reputation Service Provider (RSP) which aggregates reputations from the reputation scores of local RAAs. Based on the final calculated reputation, it might also perform actions, e.g. compute trust worthiness, make automated decisions, or trigger reactions. It also manages the communication with the reputors collecting data about entities from them, generates reputation data inputs for the reputation scoring; and distributes the data to the RAAs. It might also act as central point of communication for the real reputee entities (e.g. persons) giving them legitimate control over their reputation and allowing entities to governance their reputations. RMAs can act as a single point of entry to the managed sets of local RAAs, which allows for efficient implementation of various mechanisms of making sure the RAAs functionalities are not abused (security

[8]Prova Engine: `http://www.prova.ws`

mechanisms) and making sure privacy of entities, the reputation input data, and computed reputation objects is respected (privacy & information hiding mechanisms). For instance, an RMA can disclose abstracted aggregate reputation objects, such as trustworthiness levels of local entities, to authorized parties without revealing private reputation scores or local data about the entities.

As an example to evaluate a person using his/her RO, a resource authorization agent, called $SocialActivities$, for computing the number of friends, implements a private rule $friend$ for deriving all friends of a person using a SPARQL query to access all friends from a local `foaf` document. A public rule $numberOfFriends$ computes the number of friends using this private rules. Other agents can query this public rule via a `rcvMsg` reaction rule, which gives access only to authorized agents and only to all it's public rules. Privacy is ensured by not reveling persons' friends to other agents.

```
% private rule for collecting persons' friends
friend(Person, Friend) :-
 SparqlQuery = ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
                 PREFIX ex: <http://ns.example.org/#>
                 SELECT ?person ?friend
                 FROM <friends.n3>
                 WHERE {?person foaf:knows ?friend .} ',
 sparql_select(SparqlQuery,person(Person),friend(Friend)).
% public rule for computing number of friends
@public(numberOfFriends(?,?))
numberOfFriends(Person, Number) :-
  count(friend(Person,Friend),Number).
rcvMsg(CID,esb,Agent,acl_query-ref, Query) :-
 authorized(Agent), % check if requesting agent is authorized
 derive(Query) [public(Query)] % derive query if public guard is true
 sendMsg(CID,esb,Agent,acl_inform-ref,Query). % send back result
```

Another RAA, called $Driving$, computes driving ratings with respect to the driving skill level of a person. The shown rule $drivingSkill$ is part of a set of the agent's rules, which derive personal driving ratings with respect the years of driving experience and years without accidents. The input data is coming from a local relational database which is queried by SQL in the private rule $yearsOfDrivingExperience$.

```
% public rule for computing number of friends
@public(drivingSkill(?,?))
drivingSkill(Person,Rating):-
   Rating = 8, % level is 8
   yearsOfDrivingExperience(Person,DrivingYears),
   DrivingYears > 5, % if driving experience is higher than 5 years
   yeasWithoutAccident(Person,AccidentFreeYears),
   AccidentFreeYears > 3. % and no accidents within 3 years
yearsOfDrivingExperience(Person,Years) :-
 ...,
 sql_select(DB,person,[years,Years]). % access data from local database
```

The following rule of a reputation management agent (RMA) allows external agents to ask for a person's reputation object. Both RAAs are queried by the RMA in two parallel running sub-conversations. The RMA then creates a (complex) reputation object for a person from the resulting two reputation criteria received from the RAAs. All details about the reputation computing algorithm and the reputation input data is not revealed to the external requesting agent, thus ensuring privacy of person's data.

```
rcvMsg(CID,esb,Agent,acl_query-ref, reputation(Person,ReputationObject) :-
  % query RAA Driving in new sub-conversation 1
  sendMsg(Sub-CID1,esb,"Driving",acl_query-ref,drivingSkill(Person,Rating)),
  % in parallel query RAA Rating in sub-cid 2
  sendMsg(Sub-CID2,esb,"SocialActivities",acl_query-ref,numberOfFriends(Person, Number)),
  rcvMsg(Sub-CID1,esb,"Driving",acl_inform-ref, ReputationCriteria1),  % receive reputation criteria 1
  rcvMsg(Sub-CID2,esb,"SocialActivities",acl_inform-ref, ReputationCriteria2),% receive reputation criteria 2
  % create reputation object for a person
  createReputationObject(ReputationObject,Person,ReputationCriteria1,ReputationCriteria2),
  % send back reputation object to requesting agent
  sendMsg(CID,esb,Agent, acl_inform-ref, reputation(Person, ReputationObject)).
```

4.2 A Reputation Object Service in a Cloud Architecture

In [13] we described an architecture where the selection of a cloud service provider is based on both the reputation of the provider and consumer's priorities. Using the RO model, the provider's reputation is represented as the collection of his reputation regarding several quality parameters. The consumer who uses his services is required to rate him in some detailed way (e.g his service availability, price, response time, reliability, technical support, etc.). From the detailed profile, the reputation service is able to cross reference the quality parameters required by the consumer (retrieved from his priority list) and the performance parameters extracted from the providers' reputation objects. The priority list contains the quality parameters ordered from the consumer's most important parameter to the least important one. Once the service receives the list of potential providers, it refines it based on the providers' reputation and the consumer priority list. For example, if a consumer cares about `"quality"` more than `"price"`, the service returns the providers that were rated as having the highest quality. The refining process in the reputation service is based on a simple algorithm where:

```
get priority P1 from the ConsumerList
For all Providers_reputation in the ProviderList
   select the providers with the highest P1 values
   save in FilterList1
get priority P2 from the ConsumerList
For all Providers_reputation in FilterList1
   select the providers with the highest P2 values
   save in FilterList2
```

Finally, the consumer receives the filtered list produced by a trade off between the best QoS parameters and the user requirements. The consumer therefore takes the decision on selecting his provider. In this work, we illustrate how the RO model can be used as an effective tool; to better choose a service provider in the cloud environment based on the embedded information in the reputation objects and based also on the consumer customized priorities.

5 Conclusion and Next Steps

The line of this work for the past years focused on bringing the way we perceive reputation in our social communities to the computerized reputation systems. This included

analyzing the problems of abstracting reputation values to simple rating formats and isolating reputation values in their domain of creation only. This was followed by working on developing a model to enhance the use of reputation by re-formatting its representation into an object that embeds more information along with their semantics. To see the value of using such model, I worked with several colleagues to integrate the model in their domain of work and observe how it can be used and the benefits of using it. The next step is to conduct a user study to confirm that the developed Reputation Object model aligns with the social view of reputation.

References

[1] Rehab Alnemr, Justus Bross, and Christoph Meinel. Constructing a context-aware service-oriented reputation model using attention allocation points. *Proceedings of the IEEE International Conference on Service Computing (SCC 2009)*, pages 451– 457, 2009.

[2] Rehab Alnemr, Stefan Koenig, T. Eymann, and C. Meinel. Enabling usage control through reputation objects: A discussion on e-commerce and the internet of services environments. In *in the special issue of Trust and Trust Management, Journal of Theoretical and Applied Electronic Commerce Research*, 2010.

[3] Rehab Alnemr and Christoph Meinel. Getting more from reputation systems: A context-aware reputation framework based on trust centers and agent lists. *Computing in the Global Information Technology, International Multi-Conference*, 2008.

[4] Rehab Alnemr, Adrian Paschke, and Christoph Meinel. Enabling reputation interoperability through semantic technologies. In *ACM International Conference on Semantic Systems*. ACM, 2010.

[5] Rehab Alnemr, Matthias Quasthoff, and Christoph Meinel. *Taking Trust Management to the Next Level*. Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, pp. 796-816, 2009.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American Magazine*, May 17, 2001.

[7] Nicholas Drummond, Alan Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai Wang, and Julian Sedenberg. Putting owl in order: Patterns for sequences in owl. In *OWL Experiences and Directions (OWLEd 2006)*, Athens Georgia, 2006.

[8] Irfan Ul Haq, Rehab Alnemr, Adrian Paschke, Erich Schikuta, Harold Boley, and Christoph Meinel. Distributed trust management for validating SLA choreographies. *Proc. Workshop SLAs in Grids (in conjunction with Grid'09), CoreGRID Springer series, Banff, Canada*, October 2009.

[9] Adrian Paschke. A typed hybrid description logic programming language with polymorphic order-sorted DL-Typed unification for semantic web type systems. In *OWLED*, 2006.

[10] Adrian Paschke, Rehab Alnemr, and Christoph Meinel. Rule responder distributed reputation management system for the semantic web. In *RuleML-2010 Conference in Washington*, 2010.

[11] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Larry Craig. Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web. In *ICPW*, pages 17–28, 2007.

[12] Maxim Schnjakin, Rehab Alnemr, and Christoph Meinel. Security and high-availability layer for cloud storage (in review). 2010.

[13] Maxim Schnjakin, Rehab Alnemr, and Christoph Meinel. Contract-based cloud architecture. In *The Second International ACM Workshop on Cloud Data Management*, October, 2010.

[14] Kia Teymourian, Rehab Alnemr, Olga Streibel, Adrian Paschke, and Christoph Meinel. Towards semantic event-driven systems. In *IEEE 3rd Int. Conference on New Technologies, mobility, and Security*, December 2009.

A Proactive Service Registry With Enriched Service Descriptions

Mohammed AbuJarour

Information Systems Group
Hasso-Plattner-Institut
mohammed.abujarour@hpi.uni-potsdam.de

Although service descriptions play a crucial role in Service-oriented Computing (SOC), it has been observed that service providers release poor service descriptions about their web services. This lack of rich service descriptions reduces the (re)usability of web services and limits the role of service registries. In this work, we propose a novel approach and platform to alleviate this problem and investigate the benefits of information integration in SOC, where information about web services is gathered from multiple sources, e.g., service providers, consumers, invocations, etc., and integrated in rich universal service descriptions that enable our proactive service registry, Depot.

1 The Role of Service Descriptions

In the triangular SOA operational model (cf. Fig. 1), service providers represent the *single* source of information about web services that is provided during the publish phase. Typically, this information is technical-oriented and provided in `XML` in the form of service description, such as, `WSDL`, `WADL`, etc. This description is then used by service consumers in several tasks, such as service discovery, service invocation, service composition, service replacement, etc. During service discovery, the requirements of service consumers are matched against the published specifications of services. To invoke a web service, service consumers need to know how to call the service, which inputs it expects, and which outputs it returns, etc. When a service fails achieving its task, it should be replaced by an other service that achieves the same task. This replacement depends on the descriptions of the considered services. Moreover, composing several services into a composite service requires enough knowledge about the composed services and their interfaces to match their inputs/outputs, perform required transformations, provide missing information, etc.

In spite of their crucial role in Service-oriented Computing (SOC), researchers have identified several limitations in service descriptions, regarding their sources, management, formats, etc. An important factor, that highlights the limitations of service descriptions, is the “Internet of Services” (IoS) vision [13]. This vision is based on the concept of *business services* that represent an abstraction of the IT web services. Business services are basically concerned with end-to-end delivery of an added value and outcome. These requirements need much information about the considered services rather than the technical information provided by service providers in the form of

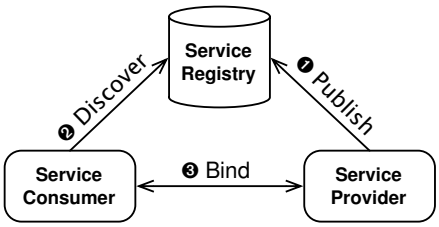


Figure 1: The triangular SOA operational model

service descriptions. In [13], the authors stated that ". . . there is definitely the need for more than the technical description of a web service interface".

One of the main effects of the aforementioned limitations in service descriptions is the limited (re)usability of the offered web services, especially in open environments, such as the Internet. To increase the (re)usability of the offered web services, several approaches have been proposed to enrich service descriptions. Typically, a single source of information is used to enrich service descriptions, e.g., service providers, service consumers, or domain experts (cf. Sec. 2). In our approach, we propose an information integration approach, where information about web services is gathered from several sources and integrated into rich universal service descriptions.

The remainder of this report is organized as follows. In Sec. 2, we introduce the research context and give further details about the addressed research problem. Then we explain our proposed approach in Sec. 3. Further implementation details are given in Sec. 4. We highlight the significant related research papers in Sec. 5. Finally, we conclude and summarize our next steps in Sec. 6.

2 Research Context and Research Problem

The available tools that enable service providers deploy their systems as web services and the popular Software-as-a-Service and Cloud Computing trends have helped increasing the number of public web services. However, this easiness in service creation has complicated the problem of service (re)usability because such services, usually, lack rich description artifacts that are vital in service discovery, service invocation, service composition, service replacement, etc.

Service providers tend to focus on the implementation aspects of their services rather than giving rich service descriptions. Usually, service descriptions appear in the form of comments and notes by service developers [12]. Such descriptions are technical-oriented and not suitable for non-IT people. In the Internet of Services (IoS) initiative, this fact is reflected in the new concept of *business services* that abstract the technical web services to fit the background of their intended users, namely, business people [5].

A common approach followed by service providers to offer public web services is to provide a list of services with a textual description attached to each service to explain its functionality. Moreover, service providers usually offer a *try* option, where one can invoke their services directly via their web interfaces. Such options typi-

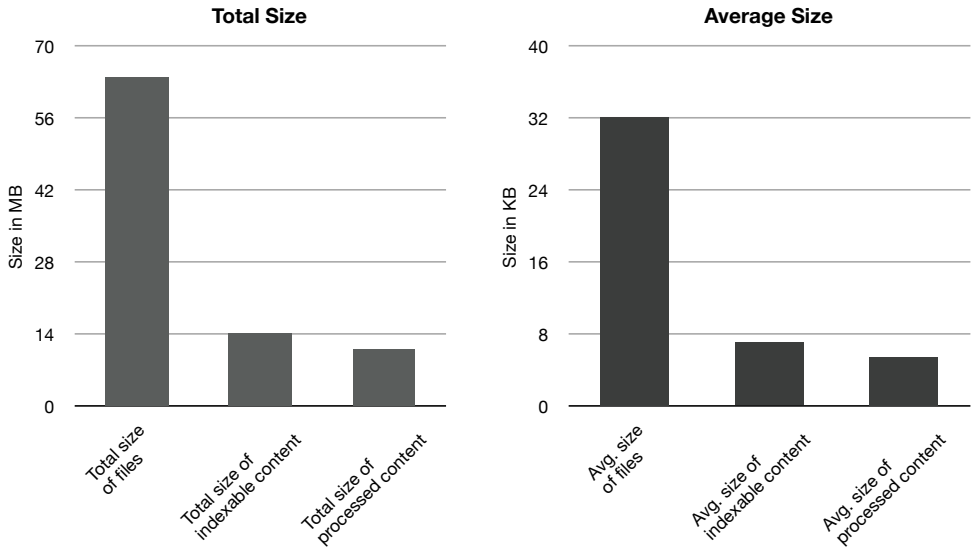
cally include web forms to collect input parameters from service consumers. Much of the information provided on such web pages is not provided in the description (e.g., WSDL) of the corresponding web service. We view such HTML pages and forms as rich sources of information and descriptions about web services. For instance, Amazon.com provides several public web services, e.g., Amazon Relational Database Service (Amazon RDS). Typically, Amazon provides a detailed description for each of their Web Services in HTML. For example, Amazon RDS is described in details at: https://aws.amazon.com/rds. Much of this information, such as the fact that it is based on MySQL, is not provided in its WSDL description available at: https://rds.amazonaws.com/doc/2010-01-01/AmazonRDSv2.wsdl. In our approach, we extract the information that service providers release about their offered web services on their websites and generate richer service descriptions from this information.

In particular we focus on the following research problems in our research:

Passive service registries: In the traditional SOA operational model (cf. Fig. 1), a service registry acts passively. It reacts to register-requests from service providers and discovery-requests from service consumers. This role is typically removed in practice, but the removal of this role violates the basic principles of loose-coupling and dynamic-binding of SOC [11]. Although its role is vital, we believe that service registries can do more, especially in enterprise applications.

Single source of information: Typically, service providers represent the main source of information about their web services. Although they know much about their web services (e.g., source code), it has been observed that they release poor service descriptions [12]. Several proposals in research assume that service providers give specific information in service descriptions, such as Non-Functional Properties [15]. Another common source of information about web services is service consumers. Typically, service consumers rate web services and provide further information about their quality. Domain experts represent another source of information about web services, e.g., Chiu et al. [7]. However, an integrated view for service descriptions is still missing.

Lack of rich service descriptions: This problem has been highlighted by several researchers, e.g., [10], where ontologies are proposed to express semantically-rich service descriptions (i.e. Semantic Web Services). In our research, we target web services released before Semantic Web Services and those that do not conform to Semantic Web Services, such as Amazon web services. We investigated the value of service descriptions with respect to service discovery using Chen Wu's collection of public web services that was collected using search engines over the Internet [17]. This collection has around 2000 WSDL files for public web services. In Fig. 2(a), we summarize relevant statistics about these files w.r.t service discovery, where we evaluate the ratio of information used in service discovery to the entire provided information. The first bar shows the total size of all WSDL files ($\sim$ 64 MB). The second bar shows the total size of all indexable content extracted from these WSDL files. Indexable content includes:



(a) Total Size (b) Average Size

Figure 2: The total and average size of WSDL files, their indexable, and processed content

documentation, service name, porttype name, operation name, message names, and message types. The total size of the indexable content extracted from this collection is around 14 MB. Indexing such a collection to perform service discovery involves a processing step using stopwords removal and stemming to eliminate insignificant terms. The total size of the processed content of this collection is around 11 MB. The size of processed content is about 17% of the total size of their WSDL files, only. In Fig. 2(b), we show the same statistics using the average size of files, indexable content, and processed content, respectively. Also, this statistics shows that the average size of processed content is only 17.3% compared to the average size of all WSDL files in the collection.

In our research, we aim at enriching service descriptions, maximizing the benefits of existing descriptions, and integrating all available resources to provide the highest precision for service discovery and selection. Our research statement is summarized in this question: How to enrich, integrate, and manage service descriptions efficiently and what are the benefits of enriching service descriptions in SOC?

3 Depot at a Glance

Our proactive service registry is called “Depot”. The architecture of Depot is shown in Fig. 3. Depot uses a focused crawler (component 1) to collect public web services from the websites of their providers. The collected web services are validated, parsed, and annotated with information, which is gathered from the websites of their providers using the WSDL and information parser (component 2). This gathered information is

stored in a service database. Service consumers can discover web services through the web service explorer (component 3) that provides a personalized result list of web services based on the profiles of service consumers through the personalization filter (component 4). Moreover, service consumers can invoke the selected web services using the web service executor (component 5), where HTML forms are provided to collect service's input parameters from service consumers. This feature enables Depot to gather further information about the invoked web services and the invoking service consumers. This information is gathered using the invocation analyzer (component 6) and stored as service metadata. Further service metadata can be provided explicitly by service consumers in the form of service annotations through the community annotation handler (component 7). Depot uses the collected information about web services, service providers, and consumers to notify consumers about new web services that might be relevant to their business through the web service recommender (component 8).

In our initial prototype of Depot [1], we introduced (among others) a focused crawler and a preliminary web service executor. In [3], we have extended this prototype by introducing the WSDL and information parser (component 2) and the web service explorer (component 3). We give further details about these components in Sec. 4. The remaining components will be added incrementally.

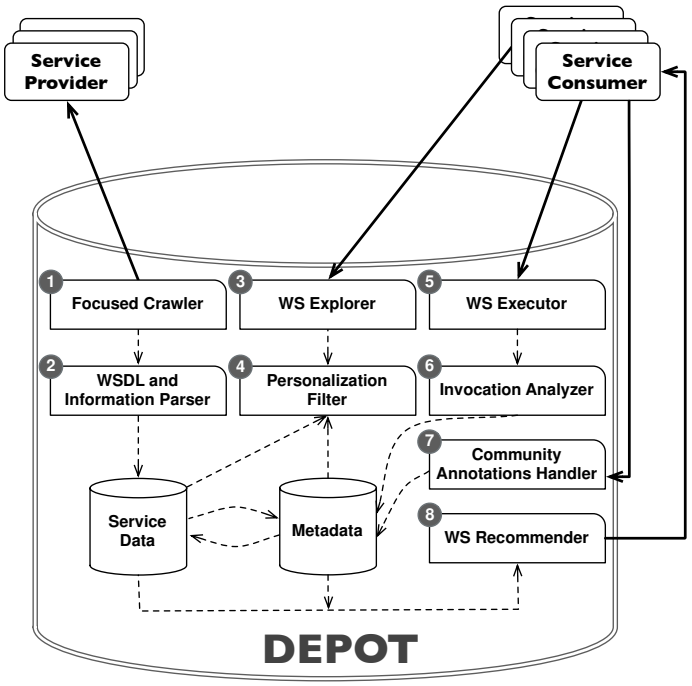


Figure 3: The architecture of Depot

As an information integration environment, Depot uses three different sources of information about its managed web services, namely, service providers, service consumers, and invocation analysis. Internally, this information is divided into service data – e.g., the location of its WSDL – and service metadata – e.g.,number of invocations of a web service. Depot performs several tasks proactively. It employs web crawling techniques to collect public web services automatically. It returns personalized lists of web services to service consumers that reflects their requirements and behavior. Moreover, it recommends new web services to their potentially interested consumers based on their profiles, proactively.

To handle the challenge of inadequate criteria for service discovery and selection, Depot uses several sources of information to enrich service descriptions. These

sources are service providers, consumers and invocation analysis.

Service providers: Along with the technical service descriptions (published in service registries) that service providers release about their web services, they give additional textual descriptions (on their own websites) to explain their functionalities. Typically, such textual descriptions do not appear in their counterpart technical service descriptions. Depot collects technical service descriptions for web services and annotates them with textual descriptions extracted from the websites of their providers automatically.

Service consumers: Feedback and ratings are the most common types of information that service consumers provide about the web services they use. The goal of such information is to assess the quality of the used web service(s), e.g., response time, cost, performance, security, etc. This information is also used to evaluate the reputation of service providers. We extend this approach in Depot by allowing service consumers to annotate the web services they used to enrich their descriptions. Users' annotations can be simple tags or detailed descriptions about the behavior of the invoked web service.

Invocation analysis: Depot provides interfaces where consumers can use web services directly. Web forms are used to collect inputs from users [1]. This feature gives Depot the opportunity to analyze such service invocations and learn more about the invoked web services. Several things can be extracted from such analysis: performance and quality, caching, classification of service consumers, tagging of web services, and the context where web services are invoked.

4 Implementation Details

In Sec. 3, we gave a general overview of our approach to a proactive service registry with enriched service descriptions. In this section, we give further implementation details about the following components: Focused crawler, WSDL and information parser, WS explorer, WS executor, and invocation analyzer.

4.1 Focused Crawler and WSDL and Information Parser

To collect public web services, we employ web crawling techniques to the web. We have implemented a focused crawler that targets XML and HTML resources only. XML files are potential candidates for web service descriptions, e.g., WSDL, whereas HTML files are potential places to find further information about the collected web services. Our focused crawler is based on Heritrix crawling framework [9]. We developed a set of decision rules to accept XML and HTML resources and reject all other types. The collected resources that conform to our decision rules are stored in an archive file.

Additionally, we developed a WSDL parser that verifies whether the collected resources by Heritrix are valid web services or not. Web services are validated using WSDL4J. Valid web services are registered and stored in Depot. Moreover, we developed an information parser that extracts further information (annotations) about the collected web services from the crawled HTML pages. This parser collects two types of

annotations, namely, textual service descriptions and tags. Textual service descriptions are typically given by service providers on their web pages in HTML. We collect these textual descriptions from the parts of the HTML pages where the corresponding web services are referenced. The entire contents of an HTML page, where a collected web service is referenced, are used to generate tags that describe the service. For further details, please refer to [3].

4.2 Web Service Explorer

Depot uses the information about web services, service providers, and service consumers to provide an enhanced service exploration and discovery for service consumers. Four types of service exploration are provided by Depot:

1. Browse by provider: This type of service exploration enables service consumers to find relevant web services from specific service providers. For instance, service consumers prefer to use web services from service providers with high reputation or well-known providers.

2. Full-text search: This type requires basic knowledge in the application domain to choose “good” keywords, e.g., address normalization, credit card validation, etc.

3. Browse by category: The increasing complexity of web services and their driving business needs makes finding “good” keywords for full-text search a difficult task. For such cases, Depot provides web service browsing based on categories. Collected web service are automatically classified in several application domains, e.g., education, finance, entertainment, etc. This classification is based on the enriched descriptions of web services.

4. Browse by tag cloud: For a quick way of exploring common web services, regardless of their providers or categories, Depot provides a tag cloud that enables service consumers to browse through common tags attached to web services. Part of these tags are automatically generated from websites of service providers during service crawling through WSDL and information parser (component 2) or from invocation analysis using invocation analyzer (component 6). Additional tags are provided by service consumers in the form of community annotations.

4.3 Web Service Executor and Invocation Analyzer

Depot provides two variants of the web service executor component: an API for enterprise applications and a web application. We introduced an API-based web service executor in our proposed “Diamond SOA Operational Model” [2]. The web-based executor enables users to invoke web services using a web form that collects input parameters to call the required web service. We use schema definitions of data types from the WSDL files to organize form fields in a convenient way. Additionally, each field in the web form is associated with annotations that we generate from invocation analysis.

Invocation analysis is an additional source of information about web services. This source is instance-based, where the actual service invocations are used to generate additional information about the invoked web services. Several things can be learnt from such analysis, e.g., performance and quality measures, identifying categories of service consumers, tagging web services – as we show next.

In Sec. 4.1, we introduced our approach to crawl public web services and annotate them with textual descriptions and tags based on the content provided by their providers on their own websites in HTML. However, the generated tags have two main limitations: i) Provider's content dependency, ii) Fixed tags. The invocation analyzer component of Depot targets these limitations in *data web services* by extracting further information about web services based on their actual invocations, such as dynamic tags. The implementation of our dynamic tag generator for *data* web services includes two components: *invoker* and *tagger*. The invoker has the task of invoking the considered web services and managing their requests and responses. As an input, the invoker takes the target web service (e.g., WSDL) and its input parameters (if any). The expected output of this component is service response (e.g., SOAP). The tagger has the task of processing the returned responses to generate tags based on their delivered contents. The invoker notifies the tagger whenever it receives a successful service response. After that, the tagger applies a set of decision rules and performs a set of processing steps to generate dynamic tags from the corresponding service response. For each successful service response, tagger decides whether it is *dynamic* or not. If the considered response is dynamic, then tagger verifies that it carries a *new result*. Then, tagger *extracts contents* from the received response and applies a set of *content processing* steps including stopwords removal and stemming. Finally, terms in the processed content are *ranked* such that candidate tags can be *selected*.

5 Related Work

Most existing service registries and repositories are based on UDDI, ebXML, or a mix of both: *Centrasite* is a UDDI service registry that is limited to the web services inside a single organization [6]. Sun's service registry is based on ebXML 3.0 with added support for UDDI 3.0 [14]. IBM's WebSphere Registry and repository mainly manages services' metadata that is gathered from all available resources, such as UDDI registries [8]. Most of these registries use service providers as a single source of information.

The limitations in the traditional SOA operational model have been highlighted by several researchers. In [11], the authors showed that the triangular model is not used widely in practice because of the limited role of service registries. Another approach to achieve active web service registries was introduced in [16]. The authors use RSS feeds to announce changes in the registered services to interested service consumers. The information provided by such feeds is generated by service providers, who tend to focus on the technical parts of their services. Moreover, such a service registry cannot force service providers to notify them about any updates so that they can add RSS feeds to announce the corresponding changes.

The main reason behind the highlighted limitations of the triangular model in the

aforementioned work is the lack of rich service descriptions [10]. Therefore, researchers have proposed several approaches to gather information about services to handle the problem of poor service descriptions. In [4], the authors use a specialized crawler to collect web services from multiple UDDI registries. Although the idea of using crawlers to collect web services is innovative, restricting it to UDDI registries does not give the maximum benefit of web crawling, as such an approach is still limited to what service providers announce during service registration.

An other web service crawler has been introduced by the EU project Seekda (http://www.seekda.eu). In this recent project, the authors use a specialized crawler to collect public web services over the web, and present them in a web 2.0 environment, which allows users to annotate, tag, and use them. We extend this approach by annotating the collected web services with automatically extracted descriptions and generated tags. Additionally, we enrich service descriptions with metadata extracted from service invocation analysis.

6 Summary and Roadmap

In this report, we introduced an approach to increase the (re)usability of public web services by enriching their poor descriptions through a proactive service registry, called "Depot". It uses three sources of information about web services: service providers, service consumers, and invocation analysis. Depot achieves three features proactively, namely, crawling of web services, personalization of web service discovery, and recommendation of web services.

We have already implemented the components of focused crawler, WSDL and information parser, web service explorer, web service executor, and part of the invocation analyzer. Our next steps include an implementation of the remaining components to verify our approach, in addition to extensive evaluation of the entire system.

References

[1] Mohammed AbuJarour, Mircea Craculeac, Falko Menge, Tobias Vogel, and Jan-Felix Schwarz. Posr: A Comprehensive System for Aggregating and Using Web Services. In *SERVICES '09*, pages 139–146, LA, CA, USA, 2009. IEEE Computer Society.

[2] Mohammed AbuJarour and Felix Naumann. Towards a Diamond SOA Operational Model. In *SOCA '10: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, Perth, Australia, 2010. IEEE Computer Society. To appear: http://bit.ly/bMgqXJ.

[3] Mohammed AbuJarour, Felix Naumann, and Mircea Craculeac. Collecting, Annotating, and Classifying Public Web Services . In *Proceedings of the 18th International Conference on Cooperative Information Systems*, Crete, Greece, 2010. Springer. To appear: http://bit.ly/bI6aUL.

[4] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating Web Services on The World Wide Web. In *WWW '08*, pages 795–804, NY, USA, 2008. ACM.

[5] Jorge Cardoso, Konrad Voigt, and Matthias Winkler. Service Engineering for the Internet of Services. In *ICEIS*, pages 15–27. Springer, 2008.

[6] Centrasite Community. Centrasite Registry. `http://www.centrasite.org`.

[7] David Chiu, Sagar Deshpande, Gagan Agrawal, and Rongxing Li. A dynamic approach toward qos-aware service workflow composition. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 655–662, Washington, DC, USA, 2009. IEEE Computer Society.

[8] IBM. WebSphere Service Registry and Repository. `http://www.ibm.com/software/integration/wsrr`.

[9] Internet Archive. Heritrix Web Crawler Project. `http://crawler.archive.org`.

[10] Dominik Kuropka, Peter Tröger, Steffen Staab, and Matthias Weske. *Semantic Service Provisioning*. Springer, Germany, 2008.

[11] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards recovering the broken SOA triangle: a software engineering perspective. In *IW-SOSWE '07*, pages 22–28, New York, NY, USA, 2007. ACM.

[12] Marta Sabou, Chris Wroe, Carole Goble, and Gilad Mishne. Learning Domain Ontologies for Web Service Descriptions: An Experiment In Bioinformatics. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 190–198, New York, NY, USA, 2005. ACM.

[13] SAP Research. Unified Service Description Language. `http://www.internet-of-services.com/uploads/media/USDL-Information-Sheet.pdf`, 2009.

[14] SUN Microsystems. SUN's Service Registry. `http://www.sun.com/products/soa/registry`.

[15] Ioan Toma, Dumitru Roman, Dieter Fensel, Brahmanada Sapkota, and Juan Miguel Gomez. A multi-criteria service ranking approach based on non-functional properties rules evaluation. In *Proceedings of the International Conference on Service-oriented Computing*, 2007.

[16] Martin Treiber and Schahram Dustdar. Active Web Service Registries. *IEEE Internet Computing*, 11(5):66–71, 2007.

[17] Chen Wu. A Public Web Services Collection for Service Discovery. 2010. INEX 2010 Web Service Discovery Track.

Towards Automated Analysis and Visualization of Distributed and Service-based Software Systems

Martin Beck

Computer Graphics Systems Group
Hasso-Plattner-Institut
martin.beck@hpi.uni-potsdam.de

Distributed software systems gain more and more importance due to a paradigm shift in software systems and applications, evolving from single-chip solutions to multi-tiered web-based applications. For a single developer, it becomes increasingly difficult to cope with the complexity of such software systems. Hence, dedicated software development tools are required.

This report reflects on the first six months of an ongoing research work at the HPI Research School that aims at providing novel automated analysis and visualizing techniques for the interactive exploration of static structures and behavior of distributed and service-based software systems.

1 Introduction

Distributed software systems gain more and more attention due to the rise of service-based software, which enables reliable, configurable and scalable IT solutions. Consequently, software developers are confronted with the necessity to efficiently develop, maintain and thus understand this category of systems. However, developers generally cannot cope with the inherent complexity of such systems, leading to a severe increase in development and maintenance costs as well as project risks. To improve program comprehension and to gain insights into the distributed system's architecture, dedicated software development tools are required. For example, a web-based shop application may be faced with timeouts in one out of a hundred uses, and developers have to investigate the reason for this behavior.

In this research project, we strive to provide a novel technique for automated analysis and visualization of distributed systems that extracts, analyzes and visualizes the messages exchanged in the system. We target existing systems used in production. Thus, our technique cannot be used to forward engineer distributed systems. Instead, live systems can be visualized and explored to understand their architecture and behavior. To extract necessary data, we not only trace messages, but also instrument each component itself. Thereby, actions taken by the system upon incoming messages can be considered for analysis.

This report is structured as follows. The following section presents related work. Sections 3 and 4 describe our first ideas for a data extraction and analysis process that especially targets our visualization context. The subsequent section elaborates on visualization techniques we are proposing. Finally, section 6 summarizes this report and gives an overview of the future tasks for the next half year.

2 Related Work

The visualization of software, its structure, behavior and evolution has long been a topic in research [5]. However, distributed systems have received little attention in the software visualization literature [3].

For dynamic analysis of distributed software systems, the runtime data needs to be collected. This can include different strategies. For example source code instrumentation [9], Java Virtual Machine profiling [13] or Aspect-Oriented Programming techniques [2] have been used to gather behavioral information from distributed systems.

To our best knowledge, there exists only few literature about the visualization of a distributed software system's structure, which goes beyond leveraging UML diagrams [12]. Zhou et al. use quaternary fat-tree networks and adjacency matrices for visualizing extreme-scale supercomputers [18]. The data jewelry box by Yamaguchi and Itoh depicts the hierarchical structure of distributed processes [17]. Allcock et al. have designed GridMapper, which simply maps the real global positions of grid nodes onto a virtual earth globe [1].

The visualization of behavior has been exploited more than the visualization of the mostly intangible structure of distributed software systems. De Pauw et al. have presented a visualization tool that supports understanding of service-oriented architectures by enhancing UML sequence diagrams [4]. Moe and Sandahl used interactive scatter plots to visualize execution traces from distributed systems [11].

Our first idea for an interactive layout idea is based on a magnet-metaphor for graph visualization introduced by Spritzer and Freitas [16]. To handle massive message data in our visualization, we leverage the Information Mural presented by Jerding and Stasko [8].

3 Data Acquisition

To successfully analyze and visualize existing distributed software systems used in production, the extraction of message and software component traces requires special care. Due to the higher complexity of these systems as compared to single-chip applications, issues such as the monitoring software's performance impact of and the central collection of data play a major role.

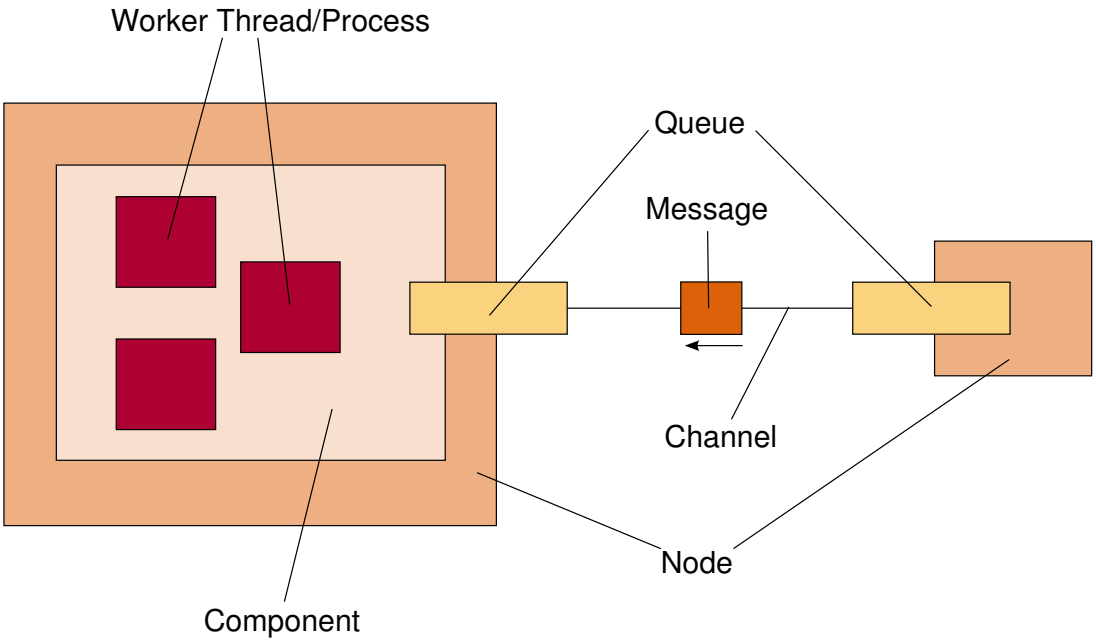


Figure 1: Low-level building blocks of a distributed software system from a developer's perspective.

3.1 Extracting Structural Information

A distributed software system's observable structure is typically made of multiple components deployed on several nodes. These components are comprised of processes and threads. Fig. 1 illustrates this. Instead of trying to extract this information via system introspection or deployment configuration analysis, we plan to use gathered behavioral data described in the next section. Presumed this data set is large enough, it should result in a more accurate extracted structure as compared to the actually used parts of the system. Due to the analysis of production systems, enough data should be present. However, system parts inactive during the extraction phase cannot be identified this way.

3.2 Extracting Behavioral Information

The directly observable behavior of a distributed software system basically consists of the messages exchanged between its components and the process-internal call-graphs of these components. While we plan to use existing solutions for the component traces [15], partially developed in our group, the way of gathering message traces is still subject to research. With regard to the analysis step, component-specific message tracers could ease later message categorization. For example, a Java RMI component tracer would be able to log invoked target operations and, thereby, marking a message as method invocation and specify its content. However, besides tremendous development efforts, this requires additional configuration and deployment overhead possibly preventing potential system analysts from employing our technique. The alternative is to simply trace each message on the network protocol level and lay the burden to rea-

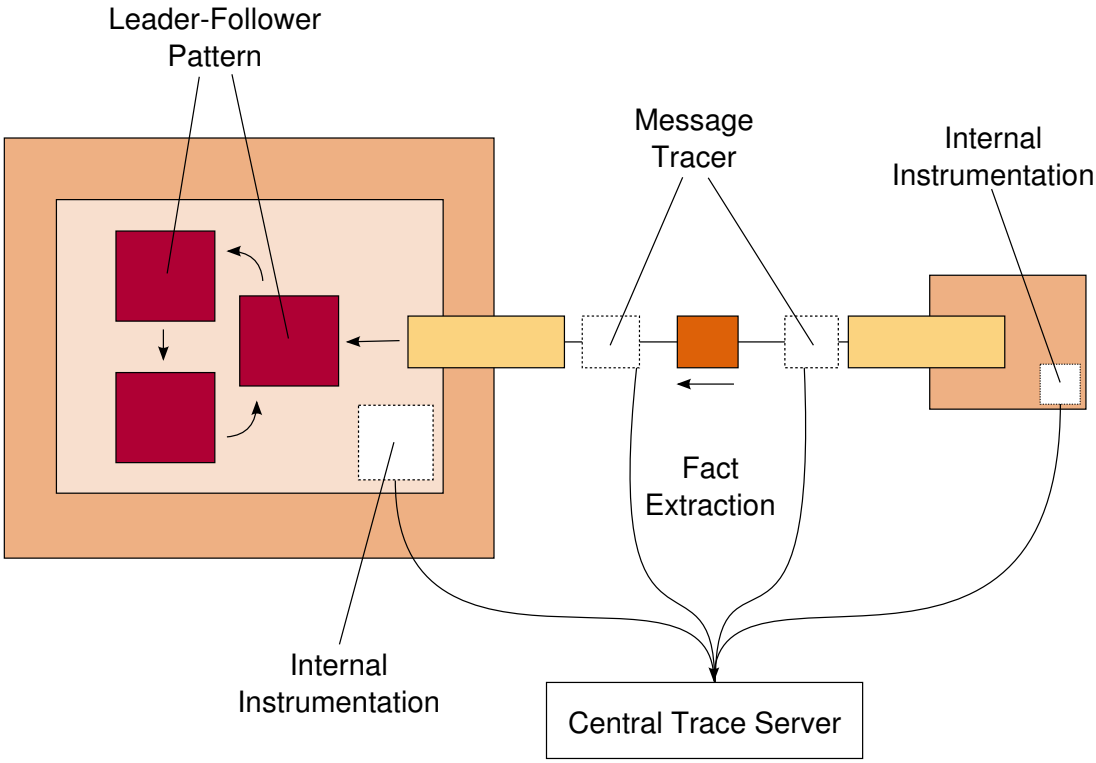


Figure 2: Instrumentation of a distributed software system. Each component is instrumented with a software tracer and each messages is logged. Both data are eventually sent to a central storage server.

son about the message's purpose on the analysis step. Fig. 2 illustrates our planned instrumentation strategy.

3.3 Data Collection

To examine gathered extracted data later, a central storage solution is required. First, each component stores traced events such as message arrivals or function calls itself. Second, this buffered data is transferred to a central server automatically as also shown in Fig. 2. Finally, the server orders gathered data chronologically. A problem in this setup are unsynchronized system clocks on the different nodes of the system. However, this has been handled in literature earlier [6, 10].

Furthermore, we expect a tremendous amount of data. Tracing function calls within each component creates large data sets on its own. As we focus on live production systems and possibly large time frames, massive message traces between these components add another dimension. Handling the efficient transmission of this kind of data over the network and its storage for later analysis is one of our research questions. In-memory databases may be necessary to fulfill the required tasks [7].

4 Analysis

The analysis phase tries to automatically recognize software development patterns and infer causal relationships between the messages. Communication profiles of components and nodes are identified and used as hints for the visualization step. For example, nodes with similar behavior can be clustered.

To support analysis, we model distributed software systems from a system architect's perspective. Low-level building blocks such as messages, queues and channels enable precise description of actual system structures and their behavior. Processes and worker threads provide a suitable abstraction of a component's inner workings, omitting unnecessary details. Fig. 1 depicts this model. Extracted structural and behavioral data is transformed into an instance of this model, enabling the recognition of more high-level architecture patterns. For example, the Leader-Followers pattern [14] depicted in Fig. 2 describes a solution to provide high through-put for processing incoming events by introducing a ring of worker threads.

The model itself can be refined recursively by repeating analysis of the extracted system facts and reusing already identified structures. Thereby, hierarchical structures are identified step by step this way. For example, a database component or node can be the incoming message queue of a larger web-service itself. Vice versa, a web-service consisting of multiple nodes and components may play the role of a database system. Eventually, this recursive refinement leads to a high-level abstraction of the distributed software system.

After its establishment, the analysis step uses this high-level model to reason about possible causal relationships between incoming and outgoing messages. Messages can be traced on their way through receiving queues to the responsible worker thread. Every action taken by this thread such as sending messages is considered to be a direct consequence of the incoming message. We seek to model this thread behavior to improve the visualization.

Statistical analysis of message trace data aids to categorize messages by their types and reveals communication profiles of the components. For example, command sessions typically use small and high-frequency messages while data streaming tries to maximize packet sizes for better through-put.

5 Visualization

Besides automated analysis, visualization is the second main part of our research on distributed and service-based software systems. We aim at leveraging the results gathered from the analysis phase to enhance current visualization techniques. A special goal is to integrate structural and dynamic information in one unified view. Existing tools for behavior visualization generally layout components in columns. However, to support developers to create a mental model of a system, it is generally not sufficient to understand structural aspects or dynamic aspects in isolation. For this reason, an approach is required that combines both information sources. Interactivity aids to adapt the visual representation to the mental model, leading to improved comprehension.

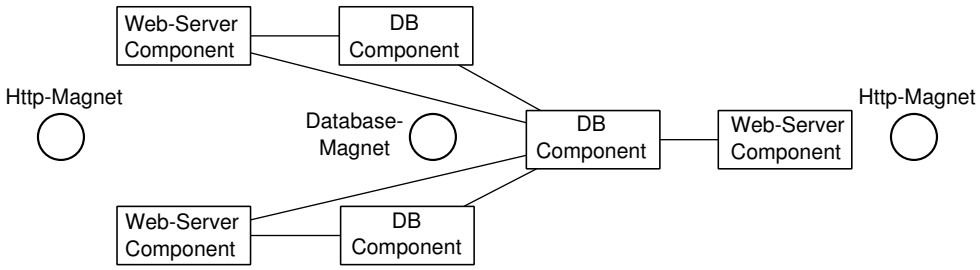


Figure 3: Layout concept using a magnet metaphor. Multiple magnets attract different kinds of nodes, enabling users to adjust the layout interactively.

In the next section, we first describe an idea for visualizing the structural aspects of distributed software systems. Afterwards, the subsequent section elaborates on visual integration possibilities for the system's behavior.

5.1 Structure Visualization

To visualize the structure of a distributed system, we plan to compute a dynamic 2D graph layout for the participating components. An initial layout can be automatically derived using the data gathered in the analysis step. Especially, recognized message patterns help the algorithm to control the layout. To further improve the layout, we investigate a magnet metaphor to visualize the component communication graph. This would allow users to interactively adjust the layout to their needs.

Fig. 3 illustrates this idea. Multiple magnets attract nodes with regards to their attributes, e.g., nodes sending and receiving HTTP messages or database queries. These magnets can be placed interactively and configured with constraints such as multiple attributes the attracted nodes have to adhere to. Furthermore, the same magnet configuration can be used more than once, thus allowing similar subgraphs to be positioned distinctively.

Further research topics include using Level-Of-Detail techniques for an interactive hierarchy exploration, visualization of participating threads and processes, and how to depict recognized development patterns.

5.2 Behavior Visualization

One of the main behavioral aspects of distributed software systems is the communication between the system's components. Distributed systems used in production typically emit millions of messages within a given time frame. Therefore, special care has to be taken to avoid visual clutter.

Fig. 4 illustrates an idea to overcome this. The straight connection line between two nodes separates the available space into two regions, one for each message direction. Incoming and outgoing messages are drawn as bars orthogonal to the separator line in chronological order on their respective side. As the connection line between two components may be rotated in the layout, increasing time is represented by increasing

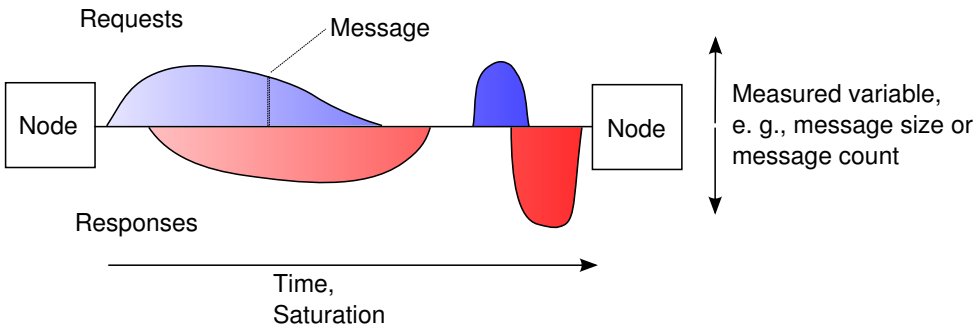


Figure 4: Possible rendering of traced messages exchanged between two nodes. Increasing saturation represents increasing time, while request and response message directions are differentiated by color. Additional information can be coded in the amplitude.

saturation of the message bar colors. Furthermore, incoming and outgoing message bars use different colors to provide a visual distinction regardless of the direction. The height of each bar represents the chosen measurement variable such as message size or transfer time. Due to space constraints, multiple messages can be accumulated in one bar. However, zooming can be used to analyze individual messages.

Nevertheless, behavioral visualization is not limited to message sends. We have to find ways to visualize addition and removal of nodes, queue activity or worker thread models. Ideally, these techniques can be embedded into the structural visualization. This combination would allow developers to mentally connect visualized data to an imaginary spatial model.

6 Summary & Future Work

We have presented an approach for program comprehension of distributed software systems. First, the system's behavior is traced by instrumenting the software components and intercepting the messages exchanged. Second, a model representing the structure and the behavior of the system is inferred from the gathered data. Next, an analysis step uses the high-level abstractions in the model and the original extracted data to reveal communication profiles, identify outlier and cluster similar behaving components. Finally, the results of the previous steps are visualized allowing users to interactively explore the available data.

In the next half year, we want to concentrate on the data extraction and model deduction. This includes to specify the model for a distributed system from a developer's perspective more precisely. The data extraction itself is a mostly covered topic [3], but nevertheless it needs to be implemented in some way by ourselves. Using this extraction implementation and the model specification, we will analyze and model existing example applications. We hope to find these applications within the industry. Furthermore, those industry partners can help us to align our research questions with concrete problems developers have with their distributed and service-based software systems.

References

[1] William Allcock, Joseph Bester, John Bresnahan, Ian Foster, Jarek Gawor, Joseph A. Insley, Joseph M. Link, and Michael E. Papka. Gridmapper: A tool for visualizing the behavior of large-scale distributed systems. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 179, Washington, DC, USA, 2002. IEEE Computer Society.

[2] Lionel C. Briand, Yvan Labiche, and Johanne Leduc. Toward the reverse engineering of uml sequence diagrams for distributed java software. *IEEE Transactions on Software Engineering*, 32:642–663, 2006.

[3] Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, and Rainer Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009.

[4] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. F. Morar. Web services navigator: visualizing the execution of web services. *IBM Systems Journal*, 44(4):821–845, 2005.

[5] Stefan Diehl. *Software Visualization. Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, Berlin, 2007.

[6] Colin Fidge. Logical time in distributed computing systems. *Computer*, 24(8):28–33, 1991.

[7] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering*, 4:509–516, 1992.

[8] Dean F. Jerding and John T. Stasko. The information mural: A technique for displaying and navigating large information spaces. *IEEE Transactions on Visualization and Computer Graphics*, 4:257–271, 1998.

[9] David Kortenkamp, Reid Simmons, Tod Milam, and Joaquín L. Fernández. A suite of tools for debugging distributed autonomous systems. *Form. Methods Syst. Des.*, 24(2):157–188, 2004.

[10] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[11] Johan Moe. Understanding distributed systems via execution trace data. In *In International Workshop on Program Comprehension*, pages 60–67. Society Press, 2001.

[12] Object Management Group. The unified modeling language uml. `http://www.uml.org`, retrieved October 9th 2010.

[13] Maher Salah and Spiros Mancoridis. Toward an environment for comprehending distributed systems. In *Proceedings of the Working Conference on Reverse Engineering*, pages 238–247. IEEE Computer Society, 2003.

[14] Douglas C. Schmidt, Hans Rohnert, Michael Stal, and Dieter Schultz. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[15] Software Diagnostics Developer Edition. `http://www.softwarediagnostics.com/`, retrieved 14 Jul 2010.

[16] Andre Suslik Spritzer and Carla Freitas. A physics-based approach for interactive manipulation of graph visualizations. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 271–278. ACM, 2008.

[17] Yumi Yamaguchi and Takayuki Itoh. Visualization of distributed processes using "data jewelry box" algorithm. *Computer Graphics International Conference*, 0:162–169, 2003.

[18] Cheng Zhou, Kenneth L. Summers, and Thomas P. Caudell. Graph visualization for the analysis of the structure and dynamics of extreme-scale supercomputers. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 143–149, New York, NY, USA, 2003. ACM.

Towards Efficient Camera Interaction in Service-based 3D Geovirtual Environments

Jan Klimke
jan.klimke@hpi.uni-potsdam.de

Service based geovisualization helps to enable high quality 3D visualization with minimal requirements for clients regarding computational power or rendering capabilities. While interaction with 2D maps is quite well understood, interaction in 3D is more challenging due to its additional degrees of freedom that have to be controlled. So called smart interaction techniques can help a user to master this 3D complexity using standard 2D input devices. To do so, additional information about the 3D environment, which is used for visualization, is needed. Such data is not per se available in many of the applications of service-based 3D visualization. This report proposes to encapsulate parts of the interaction process into service components that can be deployed independently from client applications. This lowers the complexity of client implementations, that are possibly running on a small handheld devices, so data access and computation necessary for interaction can be externalized to faster systems, which may also have a faster, more reliable network connection. Such a decoupled paradigm enables the deployment of service-based visualization applications depending on the client device's capabilities regarding computational power or data access (access rights or networking bandwidth).

1 Introduction

Since "a 3D world is only as useful as the user's ability to get around and interact with the information within it" [12] interaction components should play a major role in development of systems using geoinformation for 3D visualization. In a 3D virtual environment camera manipulation is the primary interaction since the specification of camera parameters defines what is visible to a user and therefore defines a user's context. Six degrees of freedom (namely 3D camera position (3 DOF) and 3D orientation (3 DOF)) have to be controlled to specify the position and orientation of a virtual camera. These additional variables, compared to interaction with 2D user interfaces, make camera interaction in 3D geovirtual environments (3D GeoVEs) a complex task for users. While the capabilities of standard input devices for today's applications, such as mouse, keyboard or touch sensitive surfaces, provide adequate means for controlling 2D user interfaces (such as map-based applications or conventional window-based ones), they do not allow a direct manipulation of all the parameters that are necessary

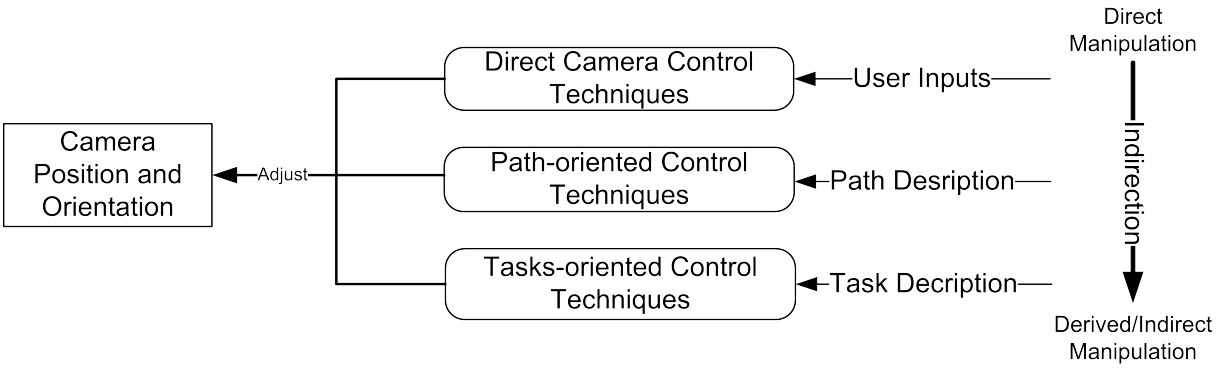


Figure 1: Classification of camera interaction techniques regarding their indirection from direct manipulation of camera parameters.

for 3D positioning and orientation of a camera view in virtual 3D space. Hence, an intelligent mapping from user input to camera configurations is needed. Approaches for smart (assisting) camera control for 3D GeoVEs can use semantics of the underlying spatial model (e.g., 3D city model) to provide higher-level interaction to users, which reduces the number of feedback cycles needed to position and orient the virtual camera. Additionally such techniques try to avoid distracting or disorienting views of a virtual camera by applying constraints to its parameters.

Service-based portrayal for 3D GeoVEs allows for thin client applications by hiding the complexity of geodata handling, processing and large parts of the rendering from clients [8]. In this way, also clients with constraint capabilities regarding data access, memory, data processing, rendering and input (e.g., mobile phones as client devices) can be used for running 3D geovisualization applications. To support the arefore mentioned smart interaction techniques for camera interactions, such clients need service-side support for computation of camera view parameters and camera paths. To improve efficiency of interaction, camera navigation techniques have to be designed and selected with special regard to such limitations [4].

In this report a concept for separating the camera control process from the service-based portrayal itself is described. Therefore we provide a concept of how to decompose a camera control process into single, possibly distributed service components, each performing a specific task in the interaction process. Well defined interfaces for each service component facilitate reuse of existing functionality and can additionally serve for a more efficient prototyping of interaction techniques. The concept for a service-based support for interaction applications is a first step in my research towards future applications for SOA-driven 3D geovisualization.

The remainder of this report is organized as follows: Section 2 will give a short overview of work that motivates this paper. Afterwards, in Section 3 our approach for distributing camera control functionality is presented. Section 4 will summarize this paper and gives an outline of future research activities towards future service-based 3D geovisualization systems.

2 Related Work

Smart interaction techniques, also called assisting navigation techniques, help a user to perform a task and avoiding disorienting or distracting camera positions and orientations. There are several types of camera control techniques regarding the level of indirection from user input to camera parameters (Fig. 1). The notion of camera task decribes an intented of camera movement and position. In general, users can be supported better, if a higher level specification of desired camera tasks is possible.

Semantics contained in virtual 3D city models, as type of geovirtual environment, can be used to evaluate user input and derive camera control tasks [7]. A defined camera control task (e.g., "show me that building", "guide me along this road") allows for generation of a camera path with respect to quality criterions, e.g., collision avoidance, retaining a user's orientation inside the 3D GeoVE, perceived smoothness of camera animation, or keeping certain points of interest visible. To enable comprehensible camera paths, approaches exist that are using basic physical models for camera control. For example inertia effects like acceleration and deceleration lead to a perceived better camera motion [2].

Assisting camera control techniques, which use semantics of the underlying model, involve additional requirements regarding data management and computational power since geodata is typically massive, heterogeneous and distributed. Especially thin clients are restricted in hardware (mobile devices like phones) or in software (thin, browser-based clients). This is a problem when 3D geovisualization applications are running on such devices. Nevertheless, through the high abstraction level from the user input, assisting camera navigation techniques seem to be promising especially for use cases that demand for a relatively low number of interaction cycles to reach a specific goal, e.g. performing a desired camera animation. A low count of interaction cycles between user and application is especially favorable for service-based systems because of the inevitable network delays during requests, which slow down service-based applications. Since mobile devices tend to have a possible unreliable network connection, this effect plays an important role if end-user applications are running on such devices.

Handling geodata for visualization is a complex task that demands for special high performance hardware (e.g., graphics adaptors, large main memory, CPU) to produce high quality visualizations. The heterogenity of hardware components and the financial and configuration effort to deploy those hamper the implementation of such high quality visualizations. Since geodata itself is often not freely accessible, another point concerning geovisualization applications is data access and rights management. Owners of geodata do not want to grant full access to their data, but might want to enable it's usage for visualization. Using the paradigm of service orientation for geovisualization applications can hide the complexity of rendering or suffisticated rigths management from client applications by encapsulating steps of the visualization pipeline. The data management encoding and processing part of the service chain is already sufficiently standardized by the Open Geospatial Consortium (*OGC*) (i.e., Web Feature Service [13] for data access, CITYGML [6] or GML [10] for data encoding, and Web Processing Service [11] for data processing).

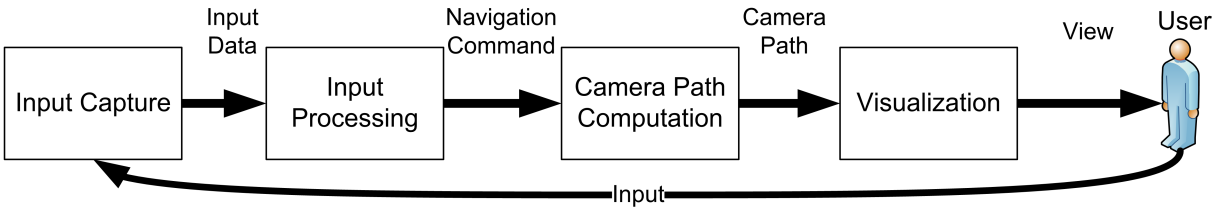


Figure 2: Separation of a camera interaction process into tasks together with the results generated by each step.

For portrayal of geodata, a 2D portrayal service has been specified by the OGC and is widely adopted. This Web Map Service (WMS) [3] creates map images from geodata. Currently, there is no corresponding OGC standard for 3D portrayal. Two active approaches towards OGC standards for 3D portrayal services are currently existing in the OGC community: The Web 3D Service (W3DS) [1] and the Web View Service (WVS) [8]. The approaches differ in the type of data they provide. While a W3DS provides display elements in a computer graphic coordinate system, a WVS provides layers of scene views encoded as images. This leads to different requirements for presentation clients needed to use such services. While a W3DS demands for 3D rendering capabilities on the client side for image synthesis, this is not necessary for WVS clients since, because this service performs server-side image synthesis.

Supporting navigation in 3D GeoVEs has not been a central issue during the development of current portrayal standard candidates. The W3DS draft standard does not include any facilities to support camera control on the server side, because the client is expected to handle all issues concerning user interaction and rendering. The WVS standard draft defines an optional `GetCamera`-Operation that returns a good camera definition for a set of 2D pixels. Here, the definition of what is a good camera perspective for the actual application is defined by the WVS implementation. So a very fundamental support for camera control is included in this standard, and can be used by thin clients for exploration of 3D GeoVEs .

3 The Camera Interaction Process

To support camera control for service-based visualization environments the process of interaction needs to be analyzed and decomposed into actions that can be performed by conceptually independent components. A clear definition of an interaction process in service-based visualization systems helps to identify the interfaces and data for communication of services. The components of such an interaction support system can be distributed. Depending on the capabilities and demands of client devices and applications, the distribution of the components could be dynamically adjusted.

We divide four tasks for camera control (Fig. 2). The input capturing task of an interaction process is done by the client device running an application. A variety of sensors can deliver input values for 3D camera control techniques, e.g., (series of) touch events for tangible surfaces, ui button events or keyboard events. Further, mobile often

include additional sensor hardware, such as GPS receivers, gyroscopes or accelerometers. So they are able to capture different information that describes a user's current context. This information can be used to influence the interaction with a 3D application to support a user by adjusting to his current contex, especially with regard to mobile applications.

Raw input data can be preprocessed to provide a more high level input for the following steps. The input preprocessing step can involve, for example, smoothing input values, recognition of shapes from series of positions. The result of this step is a *navigation command* that encodes a camera task to perform by the animation of virtual camera.

The camera path computation step executes a navigation command by creating a camera path, which consist of one or more sets of parameters defining a virtual camera's position and orientation. The path computation itself may use additional data sources, for example, to provide collision avoiding camera paths. Therefore it has to use the same geometries that is currently used for visualization to be able to perform the necessary computations to comply with the data that is currently viewed.

The visualization step concludes a camera interaction loop. It applies the generated camera path to the image synthesis stange. Image generation itself may be done using geovisualization services or implemented independently at client side. A camera service itself is not bound to thin clients. It may also be used by thick oder medium clients to provide smart camera paths.

4 Service Support for Camera Interaction

The single steps of a camera interaction cycle presented above can be supported using service instances. Which types of services can be used and how they could communicate is depicted in Fig. 3. As described in the previous section, navigation commands are recognized from user inputs. The necessary operations for command recognition can possibly involve additional data to provide commands that rely on semantics of objects included in the current scene view.

As shown in Figure 1, a user's inputs do not have to influence the parameters of the virtual camera directly. They can also describe a higher-level task to be executed by the camera. Tasks to perform are, e.g., following a route, inspect a building, or taking an overview position for parts of the scene. To allow this kind of indirection from user inputs, a command recognition service can be used to extract navigation commands. Each type of navigation command can have specific parameters, e.g., a "move to" command can have a feature identifier as target description.

Conceptually a 3D camera service is able to compute a camera position or animation for executing one type of navigation command. Therefore a registry for such camera service is introduced which manages metadata of camera services and therefore helps to find the right service-endpoint that is able to execute the command which was recognized from user input. Camera navigation techniques compute camera parameters, respectively camera animations, according to criteria, defined by the type and implementation of a technique. Examples for such criteria are:

Collision Avoidance A navigation technique may be capable of creating a camera path for animation that does not intersect with other scene geometry. There are several strategies for collision avoidance for camera path computation. Collision avoidance can be either guaranteed, best-effort or there can also be no collision avoidance at all.

Orientation Camera navigation techniques can implement orientation preserving features. For example, a technique may try to optimize the visibility of landmarks to provide orientation support to users.

Task Task specific techniques compute camera paths that are specialized in performing a task such as object inspection, overview tours our routing the camera to a defined endpoint.

3D Camera services encapsulate camera navigation techniques. They use navigation commands, which have been computed in preceding steps, alongside with command specific parameters, e.g., current camera position or a point in space as target for the camera animation, and return a camera path, which consist of one or sets of camera parameters that describe at least a camera's position and orientation. The path computation itself may use additional data, either originating from the visualization service (e.g., for image-based techniques such as the one presented by McCrae et al. [9]) or from a WFS to provide certain quality features such as, for example, collision avoidance or a guaranteed visibility of certain points of interest.

5 Summary and Outlook

This report presented a first step towards service support for camera interaction in service-based 3D geovisualization systems. Here our central point is the definition of a service-supported interaction process using different processing stages.

The future research towards more user friendly service-based 3D geovisualization applications is seen in the following areas:

Definition of quality criterions for camera animations The definition of criterions for camera positions and animations is not trivial. Quality criterions might depend on several factors defined by the type of application using these camera positions, the context of a user and especially on the current task to be performed by a 3D visualization system. To judge the quality of camera positions and respectively animations, a computable definition of the quality of a scene view is necessary. A numerical description of the quality of a scene view may then be optimized by a technique that computes camera paths. A promising approach could be, for example, using methods from visual analytics in 3D city models, as presented by Engel and Döllner [5] in connection with image analysis to rate a specific scene view regarding the quality criteria.

Camera navigation techniques New approaches for creating the camera animation itself will help to generate camera animation paths that are flexible enough to include a variety of parameters into the camera path computations. For example,

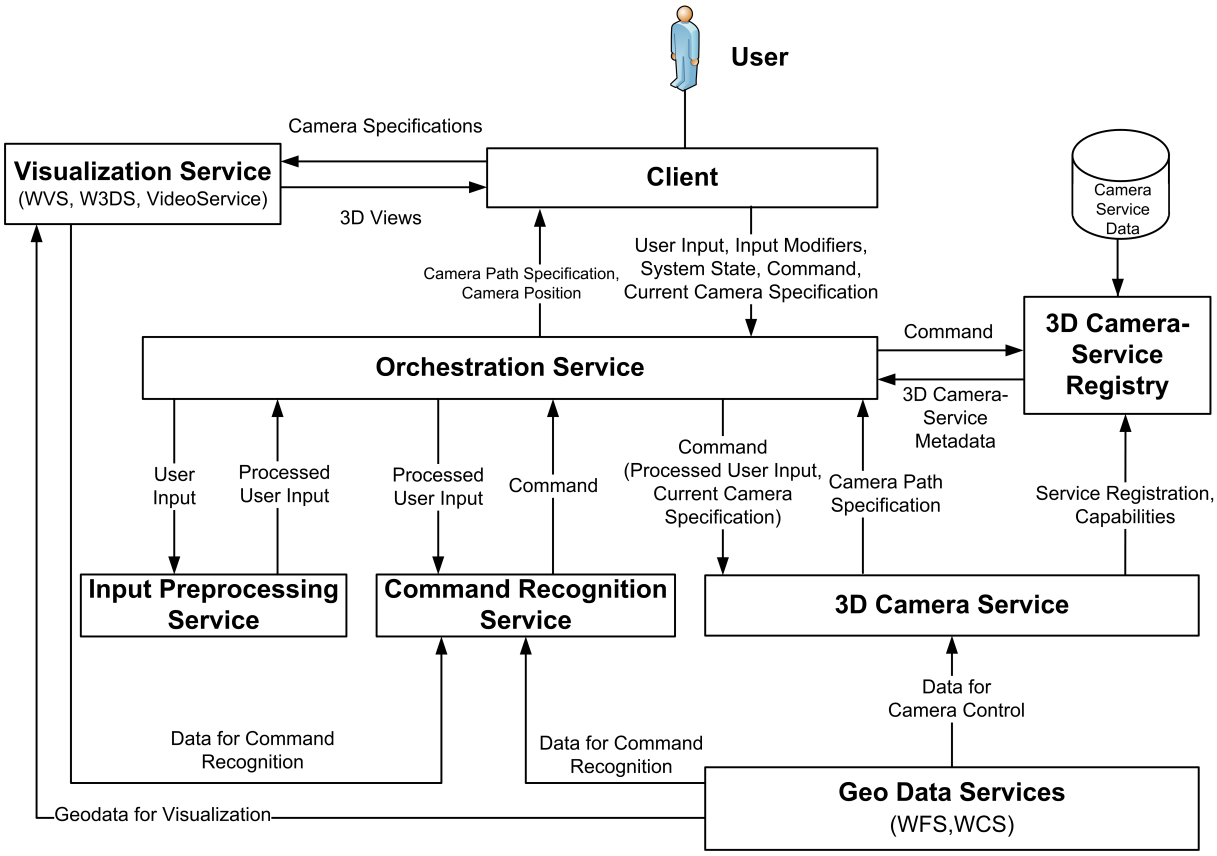


Figure 3: Abstract component architecture and data flow of a geovisualization system using service-based camera navigation

to provide the mentioned context-sensitive camera animations for mobile applications, the available parameters, such as user position, device orientation and acceleration, influence the camera animation. One promising approach for such calculation is using a physics engine to simulate forces that affect the camera and to create a camera animation. Here, questions for my further research may include the type of phsical forces (e.g., spring forces, force fields, flow simulation etc.) that can be used to create a camera animation.

Visual feedback for camera navigation Visualization for current or future actions a camera animation will include is essential to keep a user oriented and also to communicate recognized navigation commands to a user of a service-based visualization system. The research question here is how to introduce meta elements into the visualization that communicate clearly the navigation intention that a camera animation executes. Especially in the case of indoor visualization this is an open question in the reasearch community.

Definition of camera control intentions In which way can a user describe complex camera navigation tasks using conventional input devices. This is important to allow reliable recognition of navigation commands and to provide task oriented navigation techniques. Here the central question is, how can semantic data,

which is available when using geodata for visualization, be used to derive such intentions in 3D from actions that are performed on the 2D camera view plane.

Camera services Definition, assessment and implementation of the service-based camera control remains a challenging task. Especially performance considerations together with an analysis of the application potentials and restrictions will show how such a system can be applied. The definition of exchange formats and camera service metainformation for the services introduced in Section 4 is another part of work to be done to enable a loosely coupled system and a camera service registry.

Multi-touch input devices The mapping of the input of tangible displays, which are increasingly used in end-user hardware, to camera navigation intentions is a question that will be in focus. The goal to achieve highly interactive visualizations of massive 3D geodata on small devices demands for more research in thios specific direction.

References

[1] 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. Springer, 2008.

[2] H. Buchholz, J. Bohnet, and J. Döllner. Smart and Physically-Based Navigation in 3D Geovirtual Environments. In *9th Int. Conf. on Information Visualisation (IV'05)*, pages 629–635. IEEE, 2005.

[3] J. de la Beaujardiere. OGC Web Map Service Interface, 2004.

[4] F. Decle and M. Hachet. A Study of Direct Versus Planned 3D Camera Manipulation on Touch-basedMobile Phones. In *Proc. of the 11th Int. Conf. on Human-Computer Interaction with Mobile Devices and Services - MobileHCI '09*, page 1, New York, USA, 2009. ACM Press.

[5] J. Engel and J. Döllner. Approaches towards visual 3d analysis for digital landscapes and its applications. *Digital Landscape Architecture Proceedings 2009*, pages 33–41, 2009.

[6] G. Gröger, T. H. Kolbe, A. Czerwinski, and C. Nagel. OpenGIS City Geography Markup Language (CityGML) Encoding Standard Version 1.0.0, 2008.

[7] B. Hagedorn and J. Döllner. Sketch-Based Navigation in 3D Virtual Environments. In *Proc. of the 9th Int. Symp. on Smart Graphics*, volume 5166 of *Lecture Notes in Computer Science*, pages 239–246, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] 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.

[9] J. McCrae, I. Mordatch, M. Glueck, and A. Khan. Multiscale 3D Navigation. *Proc. of the 2009 Symp. on Interactive 3D Graphics and Games - I3D '09*, page 7, 2009.

[10] C. Portele. OpenGIS Geography Markup Language (GML) Encoding Standard, Juli 2007.

[11] P. Schut. OGC Web Processing Service, 2007.

[12] D.S. Tan, G.G. Robertson, and M Czerwinski. Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, volume pp, pages 418–425. ACM New York, USA, 2001.

[13] P.A. Vretanos. OGC Web Feature Service Implementation Specification, 2005.

Towards Synchronization of Partitioned Applications

Felix Geller

Software Architecture Group
Hasso-Plattner-Institut
felix.geller@hpi.uni-potsdam.de

The Web has become an established platform for serving advanced applications. The common partitioning of Web applications into server- and client-side, and ubiquitous access to the Internet faciliate online collaboration. While the division entails strong benefits, such as simplified release management, multiple client instances raise issues with regard to synchronization that remain a difficult task for software developers. More specifically, if clients share or collaborate on common resources these need to be synchronized among clients, as well as the server. A stateless protocol and possible delay due to offline use or packet loss further complicate the developers' task to ensure a coherent application state.

There is a wide range of frameworks to ease various aspects of Web application development, such as database access, control flow and markup generation. However, there is few programming platform support for synchronizing application state across multiple instances. As a first step, we present related work and highlight the connection to our goal of facilitating the synchronization of applications.

1 Introduction

Current Web technologies and pervasive access to the Internet enable ubiquitous access to advanced user applications. Software providers can build on evolved language runtimes and rendering systems to produce comprehensive applications. In addition, they benefit from simplified release management due to a popular client platform and server-side installations. The established partitioning of Web applications into server- and client-side entails strong benefits but remains a difficult task for software developers. The distinction between server and client involves different tool sets and programming languages for a single integrated application and communication among components is aggravated by the use of a stateless protocol.

More recently, distributed teams or the use of smart mobile devices yield multiple client-side instances of a single application. Common examples are online word processing applications that enable collaboration among a distributed team, or email clients that run on regular desktop computers as well as mobile devices. Information needs to be synchronized across instances to ensure a coherent application state. Changes to a text document need to be propagated to other team members and the

current state of an email folder should be consistent across individual application instances.

Figure 1 is an abstract depiction of the described scenario and related approaches to software deployment. The first column (1) displays a more traditional scenario, where a user accesses a single application instance on a local computer. In contrast, Web application instances are usually partitioned into a client- and server-side component (2). This usually means that either resources are mirrored between the client and server, or that changes to the resource are published to the server, which holds the primary version of the respective resource. The third column (3) depicts the described scenario of partitioned applications with multiple client instances. This is applicable to distributed teams or the use of mobile devices that access Web application instances. There are multiple instances of the client-side component whose application state is based on shared resources and should be coherent among all instances.

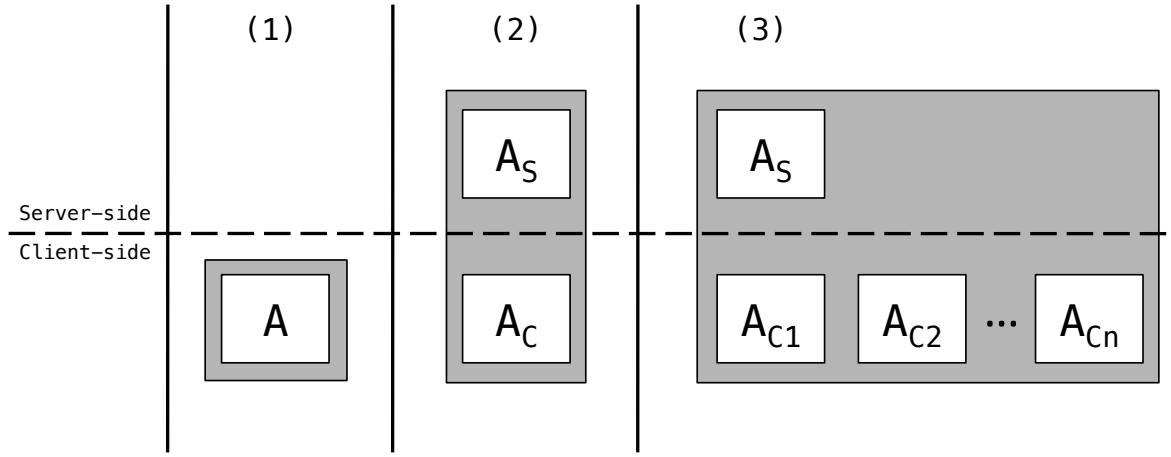


Figure 1: Synchronization of partitioned applications

Most programming platforms today provide only limited support for software developers to cope with the difficulty of concurrent connected application instances. Synchronization of application instances across the Web is often aggravated by significant delay due to offline time or lost packets as a result of poor connections. Delay of synchronization implies a higher chance of deviation of application state and therewith more difficult conflict resolution. Moreover, finding efficient and appropriate algorithms for synchronization is a challenging task depending on the application domain.

We suggest to support application development for the described scenario, where synchronization of multiple instances is required. Inherent platform support for synchronizing multiple partitioned application instances should reduce the complexity of building applications that support collaboration among teams or can be accessed concurrently from different devices.

Platform support should encompass appropriate abstractions for managing computation across application partitions and communication among components to synchronize application state. Furthermore, providing inherent platform support offers the

possibilty to automate and optimize tasks that currently are left as manual work to the software developers. Automated conflict resolution and reduction of communication payload are possible examples of where platform support may facilitate application development.

2 Related Work

Many techniques exist that are related to our scenario. In this section we present an excerpt of related work and highlight the connection to our goal of synchronizing application instances. While existing techniques offer partial solutions to providing inherent platform support, to the best of our knowledge, there is no previous work that combines all aspects and supports our scenario in its entirety.

First, we describe a small set of programming languages that contain related language-level abstractions or were designed with a similar goal in mind. Closely related are application frameworks that support distributed or partitioned applications and aim to facilitate the work of software developers. We continue to present techniques that promise to be assistant when augmenting programming platforms with support for efficient and automated synchronization of application instances: Orthogonal persistence for applications and optimistic replication of shared resources.

2.1 Programming Languages

Erlang [1] and Clojure [10] are examples for languages that provide relevant abstractions for performing parallel and possibly distributed computation. For example, Erlang features concise control structures for scheduling and handling light-weight threads, while Clojure introduces abstractions to faciliate concurrency issues by separating the notions of identity and computation of values. Both languages aim to ease application development for parallel computation by offering suitable abstractions but, to the best of our knowledge, have no inherent support for synchronization of application state across instances.

The programming language Acute [15] is an example for a set of languages that address issues related to distributed computation. Acute examines type-related issues resulting from communication among components in a distributed setting. For example, the language aims to support interaction among components that are based on different versions of shared modules. Mace [11] is an extension to the C++ programming language that provides language-level support for developing distributed applications. It includes facilities to ease event handling across distributed components and support for verifying the correctness of applications by model-checking. Both approaches aim to support specification of communication among distributed applications but seem to include no facilities for synchronization.

The Newspeak programming platform aims to implement the vision of “Objects as Software Services” [6] where entire applications are synchronized across instances. Currently, the platform features the programming language Newspeak [7] that was designed to facilitate synchronization, for example by abandoning a global namespace.

However, the features related to synchronization are currently limited and not supported in regular application development.

The programming language Hop [14] supports a two layer programming model for Web applications: the graphical user interface is managed by the client's Web browser, while computation intensive logic resides on the server. The language supports developing such partitioned applications as a single unit and features transparent communication among both application components via various event notification mechanisms. While a holistic approach to Web application development is closely related to our scenario, Hop does not address Web applications with multiple client-side instances and a single corresponding server instance. Thus, there is currently no inherent support for synchronization of application state across multiple instances.

2.2 Application Frameworks

There is a wide variety of frameworks that support Web application development. Popular frameworks[1] facilitate the separation of application components according to the Model-View-Controller pattern and ease access to databases or the generation of HTML entities via domain specific languages. Other frameworks[2] aim to reduce the complexity that results from using a stateless protocol by managing control-flow of Web applications via continuations. However, none address the issue of synchronizing multiple application instances.

The Jini system [17] is an infrastructure that aims to support "network-centric" design of distributed applications. It is built on the Java language system and allows to divide computation among loosely coupled entities that communicate via proxies. The protocol uses the features of the Java language and imposes no additional interface definition language, as is the case in other popular distributed system. Thus communication across application components is consistent with internal communication. While the system offers insights on issues related to distributed systems, there seems to be no explicit support for synchronization of components.

Croquet [16] is a framework that supports development of interactive multi-user applications that support collaboration in a 3D space. The system implements the TeaTime protocol for inter-component communication and content synchronization via replicated objects. The protocol is a direct extension to the message passing model of the Squeak Smalltalk platform and depends on synchronous communication. More recently, the Hedgehog architecture[3] introduces a router that acts as a single point of change for coordinating message propagation across replicated instances. There is only rudimentary support for resolving conflicts caused by deviation in application state. However, divergence of resources of Web application is often caused by delay as described in the first section and imposes a significant challenge in our scenario.

[1] Examples are: Django and Ruby on Rails, available at http://www.djangoproject.com/ and http://www.rubyonrails.org respectively, last accessed on October 8th, 2010.

[2] Examples are: Seaside and the Racket Web server, available at http://www.seaside.st/ and http://docs.racket-lang.org/Web-server-internal respectively, last accessed on October 8th, 2010.

[3] "Croquet - Hedgehog", http://www.opencroquet.org/images/e/ee/2005_Hedgehog_Architecture.pdf, last accessed on October 8th, 2010.

2.3 Orthogonal Persistence

Orthogonal persistence [5] for applications means that each value in an application may automatically be persisted, thus enabling resumption of application state at a later point in time. The goal of orthogonal persistence for application development is related in terms of efficiently managing the evolution of application state without impact on program performance. Value changes of persisted objects need to be propagated to a persistent store whithout affecting regular execution. Similarly, changes to shared resources or application state should be propagated among synchronized instances without imposing performance overhead. However, in contrast to most persistent application systems, our scenario involves delay that may cause significant differences between instances.

There exists research on supporting orthogonal persistence for the Java programming language [2–4] that documents design decisions, as well as related work that highlights general points of critique for orthogonal persistence [8]. Considering both perspectives should be assistant to adding efficient run-time platform support for synchronization of application instances.

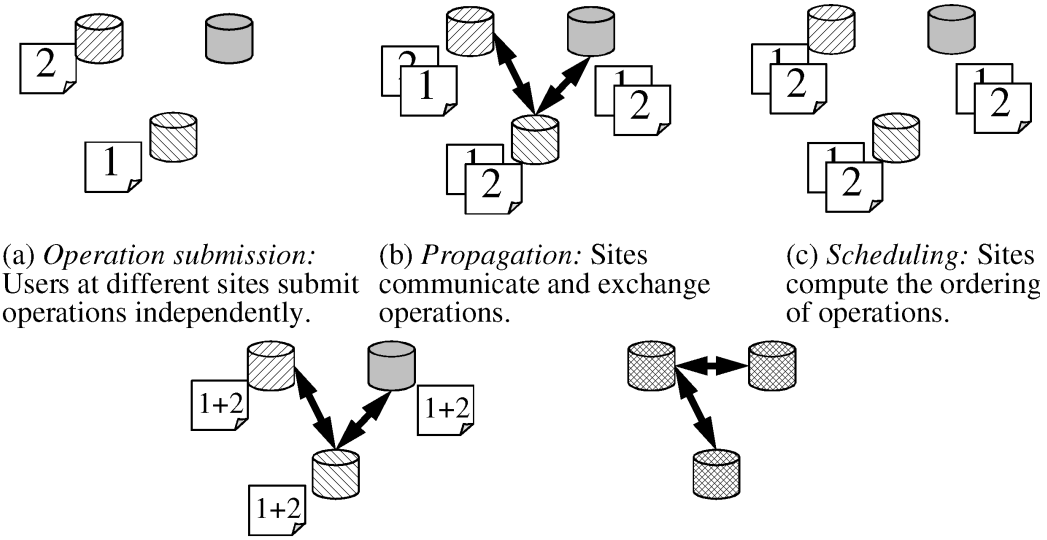
2.4 Optimistic Replication

Data replication enables collaboration on shared resources and is often used to gain benefits through redundancy. Optimistic replication refers to redundant data management where replicas might deviate for short periods of time, allowing for temporary inconsistencies among copies. This relaxation enables certain improvements with respect to performance and facilitates collaboration where users are able to contribute changes independently. On the other hand, diverging resource state can lead to conflicts and thus resolving differing versions is a common concern for the optimistic approach. However, optimistic replication provides assistant insights with respect to our scenario where application instances might deviate due to delay.

Saito and Shapiro provide a comprehensive survey [13] of related research, as well as, point out existing techniques that are implemented in existing products. In Figure 2 they summarize the individual stages and actions that are commonly part of optimistic replication approaches. They list techniques for each stage and point out their applicability for different scenarios. Based on this comprehensive analysis, we are able to evaluate individual techniques with respect to our scenario.

With respect to submission of operations, in our scenario, it should be possible to immediately execute operations locally and independently of related instances. More specifically, operations should be performed without delay resulting from re-ordering operations (for example, to resolve differences caused by concurrent operations on other instances). If a client works offline, state deviation is imminent. But rather than deferring local actions to be coordinated pessimistically to prevent conflicts, platform support should allow immediate execution and faciliate conflict resolution.

The concept of operational transformations [9] supports these requirements: Operations are immediately executed locally and remote operations are transformed to resolve conflicts if applicable. Moreover, part of the research on operational transfor-



(a) *Operation submission:* Users at different sites submit operations independently.

(b) *Propagation:* Sites communicate and exchange operations.

(c) *Scheduling:* Sites compute the ordering of operations.

(d) *Conflict resolution:* Sites detect conflicts and transform offending operations to produce results intended by users.

(e) *Commitment:* Sites agree on the final ordering and reconciliation result. Their changes become permanent.

Figure 2: Elements of optimistic replication and their roles [13]

mations aims to support remote collaboration in high-latency networks and thus may be applicable to our described scenario that involves delay between synchronization points. Existing extensions to the original work, such as synchronization of a file system based on operational transformations [12], provide further support. Issues that arise when applying the concept to synchronizing application state are dependent on the domain of the application. For example, the identification of operations and possible rewriting rules to resolve conflicts pose challenges in our scenario of synchronizing applications.

3 Summary and Outlook

The related research encompasses different approaches none of which are geared solely towards our scenario of supporting the synchronization of partitioned applications. While partial solutions are presented and various techniques may be applicable to parts of our scenario, we briefly summarize our scenario and highlight next steps to further investigate platform support for application synchronization.

We propose to augment programming platforms to include support for synchronizing application state across multiple instances. Inherent platform support can help to shift the difficulty of partitioning an integrated application and supporting synchronization of multiple instances. More specifically, we suggest considering two connected problems that arise when developers synchronize multiple concurrent application instances:

1. Finding appropriate abstractions to control the synchronization of state across multiple instances of an integrated application. The interfaces for communication among components should be consistent with intra-component communication and not be obscured by additional protocols. While well-defined protocols ensure communication among loosely related applications, they impose additional complexity when the software architect is aware and in control of the connections between components. Nevertheless, inter-component communication should be explicit. More specifically, the software developer should be aware of component boundaries and respective consequences. In summary, software developers should retain an appropriate level of control over the distribution of where logic and state is handled.

2. Platform support to enable efficient automated application synchronization while handling delay. Augmenting platforms with support for application synchronization enables automation and optimization of tasks that currently remain with the software developer. Optimizations should aim to reduce the payload and required frequency of communication among partitions of an application. Moreover, automated synchronization should be supported by appropriate techniques for conflict management, especially in the light of delay and resulting deviations in application state.

In summary, we suggest to reduce the complexity of dealing with partitioned applications with multiple client-side instances by offering inherent platform support. While the main goal is to alleviate program development, such inherent support also raises questions in regard of security and creates room for improvement. For example, by further utilization of information that becomes available by synchronizing multiple application instances, such as the identification of corrupt application components. We plan to investigate these questions further as platform support is available for experimentation.

References

[1] Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007.

[2] M. P. Atkinson, L. Daynès, M. J. Jordan, T. Printezis, and S. Spence. An Orthogonally Persistent Java. *SIGMOD Rec.*, 25(4):68–75, 1996.

[3] Malcolm P. Atkinson, Mick J. Jordan, Laurent Daynès, and Susan Spence. Design Issues for Persistent Java: A Type-Safe, Object-Oriented, Orthogonally Persistent System. In Richard C. H. Connor and Scott Nettles, editors, *Workshop on Persistent Object Systems*, pages 33–47, 1996.

[4] Malcom Atkinson and Mick Jordan. Providing Orthogonal Persistence for Java. *ECOOP'98*, pages 383–395, 1998.

[5] Malcom P. Atkinson, Peter J. Bailey, Kenneth J. Chisholm, Paul W. Cockshott, and Ronald Morrison. An Approach to Persistent Programming. *The computer journal*, 26(4):360, 1983.

[6] Gilad Bracha. Objects as Software Services. Whitepaper Available Online at: http://bracha.org/objectsAsSoftwareServices.pdf, August 2006.

[7] Gilad Bracha. Newspeak Programming Language Draft Specification Version 0.05. 2009.

[8] Tim Cooper and Michael Wise. Critique of Orthogonal Persistence. In *Proceedings of the 5th International Workshop on Object Orientation in Operating Systems*, pages 122–126. IEEE Computer Society, 1996.

[9] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, SIGMOD '89, pages 399–407, New York, NY, USA, 1989. ACM.

[10] Stuart Halloway. *Programming Clojure*. Pragmatic Bookshelf, 2009.

[11] Charles Edwin Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin M. Vahdat. Mace: Language Support for Building Distributed Systems. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 179–188, New York, NY, USA, 2007. ACM.

[12] Pascal Molli, Gérald Oster, Hala Skaf-Molli, and Abdessamad Imine. Safe Generic Data Synchronizer. Research Report A03-R-062, LORIA – INRIA Lorraine, May 2003.

[13] Yasushi Saito and Marc Shapiro. Optimistic Replication. *ACM Computing Surveys*, 37(1):42–81, March 2005.

[14] Manuel Serrano, Erick Gallesio, and Florian Loitsch. Hop, a Language for Programming the Web 2.0. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 975–985, New York, NY, USA, 2006. ACM.

[15] Peter Sewell, James J. Leifer, Keith Wansbrough, Francesco Zappa Nardelli, Mair Allen-Williams, Pierre Habouzit, and Viktor Vafeiadis. Acute: High-Level Programming Language Design for Distributed Computation. *Journal of Functional Programming*, 17(4-5):547–612, 2007.

[16] David A. Smith, Alan Kay, Andreas Raab, and David P. Reed. Croquet–A Collaboration System Architecture. In *Proceedings of the First Conference on Creating, Connecting, and Collaborating through Computing (C503)*, pages 2–9, 2003.

[17] Jim Waldo. The JINI Architecture for Network-Centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
45	978-3-86956-128-8	**Survey on Healthcare IT systems: Standards, Regulations and Security**	Christian Neuhaus, Andreas Polze, Mohammad M. R. Chowdhuryy
44	978-3-86956-113-4	**Virtualisierung und Cloud Computing: Konzepte, Technologiestudie, Marktübersicht**	Christoph Meinel, Christian Willems, Sebastian Roschke, Maxim Schnjakin
43	978-3-86956-110-3	**SOA-Security 2010 : Symposium für Sicherheit in Service-orientierten Architekturen ; 28. / 29. Oktober 2010 am Hasso-Plattner-Institut**	Christoph Meinel, Ivonne Thomas, Robert Warschofsky et al.
42	978-3-86956-114-1	**Proceedings of the Fall 2010 Future SOC Lab Day**	Hrsg. von Christoph Meinel, Andreas Polze, Alexander Zeier et al.
41	978-3-86956-108-0	**The effect of tangible media on individuals in business process modeling: A controlled experiment**	Alexander Lübbe
40	978-3-86956-106-6	**Selected Papers of the International Workshop on Smalltalk Technologies (IWST'10)**	Hrsg. von Michael Haupt, Robert Hirschfeld
39	978-3-86956-092-2	**Dritter Deutscher IPv6 Gipfel 2010**	Hrsg. von Christoph Meinel und Harald Sack
38	978-3-86956-081-6	**Extracting Structured Information from Wikipedia Articles to Populate Infoboxes**	Dustin Lange, Christoph Böhm, Felix Naumann
37	978-3-86956-078-6	**Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Leen Lambers
36	978-3-86956-065-6	**Pattern Matching for an Object-oriented and Dynamically Typed Programming Language**	Felix Geller, Robert Hirschfeld, Gilad Bracha
35	978-3-86956-054-0	**Business Process Model Abstraction : Theory and Practice**	Sergey Smirnov, Hajo A. Reijers, Thijs Nugteren, Mathias Weske
34	978-3-86956-048-9	**Efficient and exact computation of inclusion dependencies for data integration**	Jana Bauckmann, Ulf Leser, Felix Naumann
33	978-3-86956-043-4	**Proceedings of the 9th Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS '10)**	Hrsg. von Bram Adams, Michael Haupt, Daniel Lohmann
32	978-3-86956-037-3	**STG Decomposition: Internal Communication for SI Implementability**	Dominic Wist, Mark Schaefer, Walter Vogler, Ralf Wollowski
31	978-3-86956-036-6	**Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
30	978-3-86956-009-0	**Action Patterns in Business Process Models**	Sergey Smirnov, Matthias Weidlich, Jan Mendling, Mathias Weske

ISBN 978-3-86956-129-5
ISSN 1613-5652