

Proceedings of the 8th Ph. D. Retreat of the HPI Research School on Service-oriented Systems Engineering

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

Technische Berichte Nr. 95

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



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

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

**Proceedings of the 8th Ph. D. Retreat
of the HPI Research School
on Service-oriented Systems Engineering**

Bibliografische Information der Deutschen Nationalbibliothek

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

Universitätsverlag Potsdam 2015

<http://verlag.ub.uni-potsdam.de/>

Am Neuen Palais 10, 14469 Potsdam

Tel.: +49 (0)331 977 2533 / Fax: 2292

E-Mail: verlag@uni-potsdam.de

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

ISSN (print) 1613-5652

ISSN (online) 2191-1665

Das Manuskript ist urheberrechtlich geschützt.

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam

URL <http://publishup.uni-potsdam.de/opus4-ubp/frontdoor/index/index/docId/7230>

URN <urn:nbn:de:kobv:517-opus4-72302>

<http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-72302>

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:

ISBN 978-3-86956-320-6

Contents

Testbed Automation for Network Security and Security Analytics	1
<i>Aragats Amirkhanyan</i>	
Data-Centric Business Process Improvement	11
<i>Ekaterina Bazhenova</i>	
History Assisted View Authoring for 3D Models	23
<i>Tim Chen</i>	
Service-Oriented Integration and Processing of Massive 3D Point Clouds . .	31
<i>Sören Discher</i>	
Consciousness in Artificial Agents – A Theory	43
<i>Fahad Khalid</i>	
Implementing an Object-Constraint Extension Without VM Support	59
<i>Tim Felgentreff</i>	
Comparing the Layout Stability of Treemap Algorithms	71
<i>Sebastian Hahn</i>	
Generating Dynamic Dependability Models from Call Traces	81
<i>Lena Herscheid</i>	
Physical Motion Displays	93
<i>Alexandra Ion</i>	
Profiling the Web of Data	101
<i>Anja Jentsch</i>	
Enterprise Simulations based on Value Driver Trees	111
<i>Stefan Klauck</i>	

Contents

Proprioceptive Interaction	123
<i>Pedro Lopes</i>	
Question Answering for Biomedicine	135
<i>Mariana Neves</i>	
Adaptive Just-in-time Value Class Optimization	151
<i>Tobias Pape</i>	
Processing Over Encrypted Data: Between Theory and Practice	163
<i>Eyad Saleh</i>	
Migrate Highly-Available Applications to Non-HA-Infrastructure	181
<i>Daniel Richter</i>	
Leveraging Programmers' Skills: Interleaving of Modification and Use in Data-driven Tool Development	191
<i>Marcel Taeumel</i>	
Omniscient Debugging in Database Applications	203
<i>Arian Treffer</i>	
Learning Deep Semantic Feature for Cross-modal Representation	215
<i>Cheng Wang</i>	

Testbed Automation for Network Security and Security Analytics

Aragats Amirkhanyan

Internet Technologies and Systems
Hasso-Plattner-Institut
aragats.amirkhanyan@hpi.de

The report summarizes my work in the past semester in the IT security team of the chair “Internet Technologies and Systems”.

The testbed is an important part of any research work in the area of network security and security analytics. Common testbeds are data and test network environments. The absence and the need to create a new testbed make an overhead of research work and, as a result, this overhead slows down research progress. Therefore, the issue of automation of the testbed creation is a crucial goal to accelerate research progress. In this report I show a method, which I used for solving a problem of the lack of data for the certain research project in the security analytics area.

The goal of the report is to show a method and identify research challenges in solving the problem of testbed data automation for network security and security analytics.

1 Introduction

When researching in network security and security analytics areas, it is very important to have an appropriate testbed. The appropriate testbed includes appropriate data in needed amount and an appropriate test network environment. But often researchers do not have one of them or both of them.

In case with data, the lack of data is a result of the fact that mostly the huge amount of data belongs to industrial companies, but they are usually not able to share them because of security and privacy reasons. The limitation with data makes data for security research is more inaccessible. And even if there is possibility to get such information, this information should be anonymized before exporting for analysis. Therefore, researchers have to simulate scenarios and generate data. And it means that we have new challenges, such as how to simulate a scenario, how to generate data, how to design a scenario based on modeling user behavior.

In case with a test network environment, researchers need to create and configure a test network environment manually. Even if the installation is manual, it is still a challenge, because it is needed to set up a test network environment of enterprise or

campus level. It means that we have to set up different complex systems, for instance, a domain controller.

Any test network has its own specifics. In case with research in the area of network security and security analytics, this specifics means that often needed to have a potentially vulnerable environment and that means that old and vulnerable software applications must be installed. This restriction complicates the ability to use many existing IT automation systems, such as Chef¹ and Puppet².

There are several ways to generate data. One of them is to generate synthetic data. This method implies that you know about the structure of data, the correlation between data and you roughly know what the result of analysis will be on the generated data. The advantage of this method is that you do not need to have a test network environment and you can generate any data if you know the structure of them. If the structure is quite simple then this way is more preferable. But if the structure of data and the correlation between data are very complex and varies, then this approach becomes complicated, because in this case, you have to implement algorithms for each type of data and for each correlation.

Another approach to generate needed data is based on the simulation of some activities needed to produce this data [1]. This approach is more complex than the first one in case of simple data, because you need to set up a test network environment before applying the simulation. Despite the fact that this approach is more complex, it is more flexible and it can be applied for generation data with complex structure, because we do not need to care about the structure of data or the correlation between data. We need to care only about the simulation of specific activities. Also, this approach is suitable for real-time research, for example, for generating data for real-time intrusion detection systems.

Within the IT security team of the chair “Internet Technologies and Systems” we also face with problems of the lack of testbeds. We are trying to solve them by automation the process of the testbed creation. In the second section of the report, I show how we solve the problem of the lack of data by the simulation of user behavior.

2 Preparing testbed data to analytics

In the past semester the IT security team of the chair “Internet Technologies and Systems” worked on the project “Machine Learning for Security Analytics powered by SAP HANA”. Within this project the team aimed to implement and test the machine learning approach for security analytics based on SAP HANA. Under this approach

¹Chef. <http://www.getchef.com>, accessed December 16, 2014.

²Puppet. <http://puppetlabs.com>, accessed December 16, 2014.

the team focused on the analysis of user events, particularly login and logout events. To test the machine learning analytics module, we have generated several simple brute-force attacks. Also, there was the idea to try the machine learning approach powered on SAP HANA for security analytics with more sophisticated cases. But the main problem of trying the approach with sophisticated cases is the lack of data. As I mentioned in the introduction section, mostly only industrial companies have such data in needed amount, but they do not share them because of security and privacy reasons. For example, login and logout events contain information about users and this information cannot be passed to third parties. I was faced with the task to set up a test network environment and implement a tool to simulate particular user scenarios, user behavior, and as a result, generate necessary data for further analysis. This simulation tool must provide capability to simulate normal and abnormal scenarios to be able to analyze user behavior by the machine learning approach to detect anomalies.

Challenges to create such testbed are setting up the domain controller of enterprise or campus level, creating large amount of users, generating random user activities and designing a model of user behavior, especially abnormal behavior.

2.1 Network architecture

For the first challenge, we started with designing and describing the test network. Our test network contains the domain controller (DC) with installed Windows Server 2012, the wiki and database servers with Windows Server 2003 and four client computers with Windows 7 Professional 64-bit. To complete the installation, we created four user accounts. All the above information is presented below in the form of compact lists.

Description of the network:

- 4 client computers with Windows 7 Professional 64-bit
- Domain controller with Windows Server 2012
- Wiki server with Windows Server 2003
- Database server with Windows Server 2003

Users:

- Ivanov
- Petrov
- Smirnov
- Admin

wiki server. This behavior is abnormal but not suspicious, because other users use it every day. The second abnormal behavior is that the user Petrov uses the database server but in the normal scenario no one uses it. This user behavior is more suspicious and could be determined as an attack. The abnormal scenario is illustrated in Figure 2. The straight solid lines show normal user behavior, the curve solid line from Ivanov to the wiki server shows the first abnormal behavior and the curve solid line from Petrov to the database server shows the second abnormal behavior that is suspicious and could be determined as an attack.

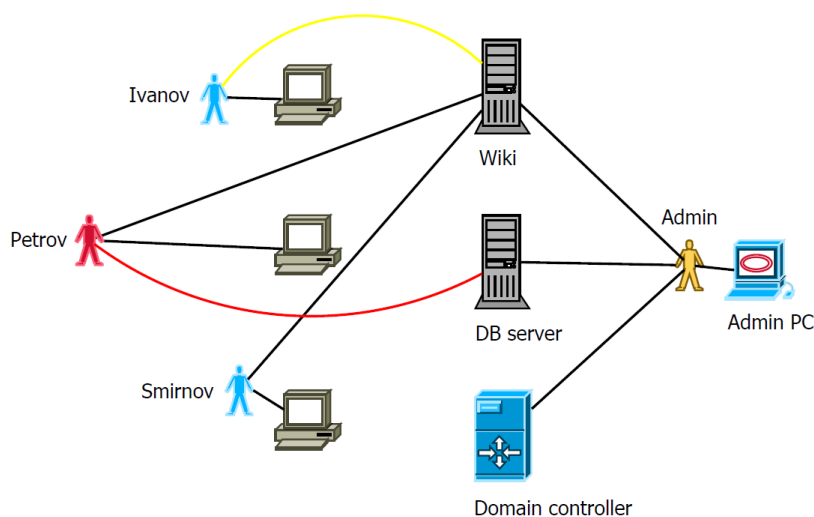


Figure 2: Abnormal scenario

2.3 Implementation

The first part of the implementation is the deployment and the configuration of the test network environment. To set up the network, we used the Future SOC Lab³ with installed VMware ESXi. Firstly, we created the network to isolate our virtual machines from others hosted on the same ESXi server. Afterwards, all machines were deployed and configured as described in the section 2.1. We used Windows Server 2012 for the domain controller, Windows Server 2003 for the wiki and database servers and Windows 7 Professional 64-bit for client computers.

The architecture of the simulator is illustrated in Figure 3. This simulation tool is implemented on the python programming language and uses the virtual network

³Future SOC Lab. <http://hpi.de/en/research/future-soc-lab.html>, accessed December 16, 2014.

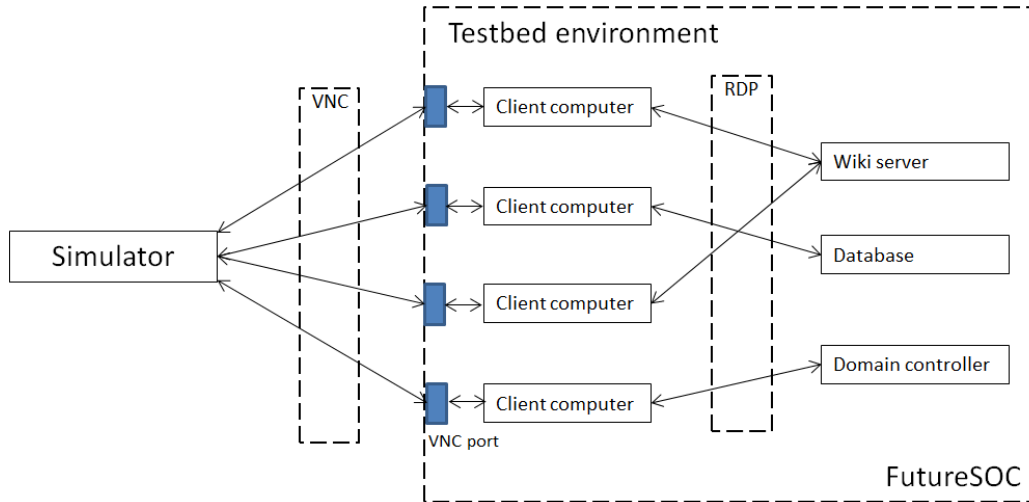


Figure 3: Simulator's architecture

Table 1: Actions and commands

Action	Command
Open the logon window	['ctrl-alt-del', 'alt-w', 'right', 'right', 'right', 'right', 'enter']
Enter username, password and press the enter key	[':' + kwargs['username'], 'tab', ':' + kwargs['password'], 'enter']
Log off	['lsuper-r', ':shutdown /l /f', 'enter']
Run the powershell script to open the RDP connection	['cd /', 'powershell rdp.ps1']

computing (VNC) protocol to connect to virtual machines as the ESXi server supports the VNC protocol. But to use the VNC protocol, it is needed to enable VNC on the ESXi server, specify the VNC port for each virtual machine and support the VNC protocol on the application side. To use VNC in the simulator, we used the vncdotool library [3]. This library also can take a screenshot of the remote desktop. We used this screenshot functionality for checking a state of the virtual machine by comparing image histograms with predefined states. Once we determined the current state of the virtual machine, we can specify the activities to be undertaken in each case according to the scenario description, such as sending the *ctrl+alt+delete* command to the virtual machine, entering the username and the password, opening the RDP connection and others. The RDP connection from the client computer to the database and wiki servers is implemented by invoking the PowerShell script hosted on the client computer. This script can accept parameters and it means that we can use the

script to establish the connection with any server by specifying parameters, such as the IP address, the username and the password. In the Table 1 you can see examples of relations between actions that must be undertaken according to the scenario and commands passed to VM to perform it.

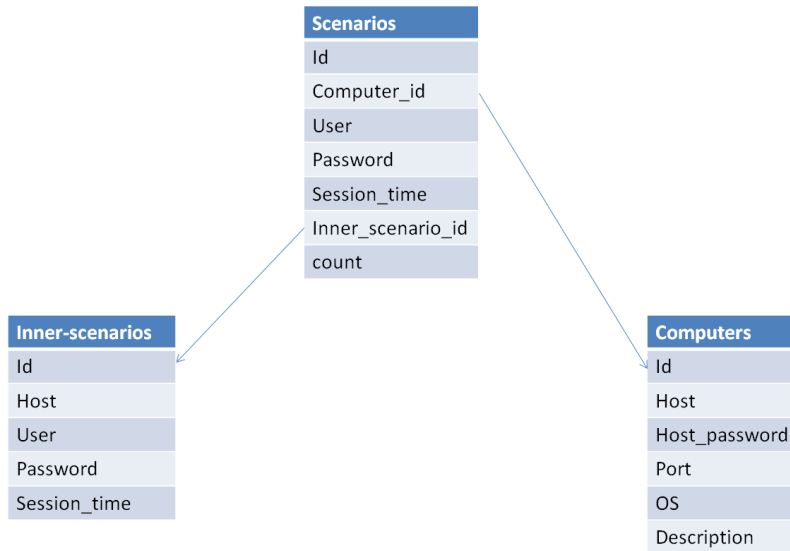


Figure 4: Scenario relations

Level	Date and Time	Source	Event...	Task Category
Information	9/4/2014 11:18:15 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:18:14 PM	Security-Auditing	4634	Logoff
Information	9/4/2014 11:18:14 PM	Security-Auditing	4624	Logon
Information	9/4/2014 11:18:14 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:18:13 PM	Security-Auditing	4624	Logon
Information	9/4/2014 11:18:13 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:17:47 PM	Security-Auditing	4624	Logon
Information	9/4/2014 11:17:47 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:17:37 PM	Security-Auditing	4634	Logoff
Information	9/4/2014 11:17:35 PM	Security-Auditing	4624	Logon
Information	9/4/2014 11:17:35 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:17:35 PM	Security-Auditing	4769	Kerberos Service Ticket Operations
Information	9/4/2014 11:17:27 PM	Security-Auditing	4634	Logoff
Information	9/4/2014 11:17:27 PM	Security-Auditing	4624	Logon
Information	9/4/2014 11:17:27 PM	Security-Auditing	4672	Special Logon
Information	9/4/2014 11:17:18 PM	Security-Auditing	4634	Logoff

Figure 5: Active Directory's logs

In this paragraph, I show how we described the scenario of user behavior. The scenario descriptions are stored in *csv* files. There are three *csv* files. The first of them is called *computers.csv*. It describes all computers that are involved in scenarios and it contains the information about the computer identifier, the IP address or the host, the VNC port, the VNC password and the type of operation system. The second file is called *scenarios.csv*. It describes the main user activity, which is the connection to the client computer. The file contains the computer identifier referring to the identifier in the *computer.csv* file, the username, the password, the session time, the count of sessions and the identifier referring to the identifier in the *inner-scenarios.csv* file. The third file called *inner-scenarios.csv* describes the actions of users after logging into the client computer. For example, it can describe the connection to the wiki server, database server, domain controller or even the connection to another client computer. The file contains the identifier, the host, the username, the password and the session time. There are two sets of *csv* files. The first set is used to perform the normal scenario and the second is used to perform the abnormal scenario, respectively. The described relations between *csv* files are illustrated in Figure 4.

I have successfully used the simulator to simulate simple normal and abnormal scenarios. The normal scenario takes about 3.5 hours with 41 login and logout events into client computers and 25 RDP connections to the wiki and database servers. The abnormal scenario takes about 5.5 hours with 45 login and logout events to client computers and 34 RDP connections to the wiki and database servers. The result of running the simulator is set of logs in the active directory of the domain controller. This log dataset is presented in Figure 5.

3 Conclusion

I have presented a method of solving a problem of testbed automation for research in the area of network security and security analytics regarding the problem of the lack of data. My proposal is based on the simulation of user behavior. To prove the concept of the method, I provided the implementation of the idea and, additionally, I presented the network architecture used for the simulation, the description of simulated scenarios and the architecture of the implementation. As result, and I have successfully used the simulation tool to generate necessary data for research and presented the example of dataset produced by the simulator.

As future work, I plan to try the simulator with large amount of users, with the network of enterprise or campus level and random user behavior.

References

- [1] S. U. A. Garg V. Sankaranarayanan and K. Kwiat. "USim: A User Behavior Simulation Framework for Training and Testing IDSes in GUI Based Systems". In: 39th Annual Simulation Symposium, 2006.
- [2] H. K. Emilie Lundin Barse and E. Jonsson. "Synthesizing Test Data for Fraud Detection Systems". In: 19th Annual Computer Security Applications Conference, 2003.
- [3] M. Sibson. "VNCdotool. <https://github.com/sibson/vncdotool>". In: 2014.

Data-Centric Business Process Improvement

A Data-Centric Approach for Business Process Improvement Based on Decision Theory

Ekaterina Bazhenova

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

Business process management and improvement is crucial for the enterprises of today. Nowadays, a great number of technologies are presented to public such as SaaS, PaaS, in-memory computing etc., which can potentially improve business processes. But an effective and quick application and adaptation of these technologies for business processes is a big challenge in an organization with its specific goals and constraints. Traditional approaches for business process improvement are based on activity flows, not considering data of business processes. This report provides an ongoing work on an integrated method which will allow to consider both activities and data into the scheme of business process improvement. At the present time, we narrow down the scope of work to the improvement of decisions subprocesses in process models and provide its analysis according to the techniques from decision theory.

1 Introduction

The prerequisite of successful existence of the enterprise of today is effective business process management. In consequence of technological progress in the last decades, organizations have received not only vast opportunities for the optimization of business processes, but also daunting challenges with regards to applying these innovations in real businesses. With that, the question of how to re-organize the business process in order to use new technologies, represents the challenge of business process redesign which “is often not approached in a systematic way, but rather considered as a purely creative activity” [3].

The majority of existing approaches to business process redesign are activity-centric and they do not consider process model data. However, data-centric approach to modeling business operations and processes “has been evidenced in both academic and industrial researches where it not only provides higher level of flexibility of workflow enactment and evolution, but also facilitates the process of business transformations” [9].

Other factors, which influence the application of business process management in enterprises are the instability of markets and the necessity of making decisions under the conditions of risk and uncertainty. Even a simple business process, such as scheduling meetings at an enterprise, can have different execution outcomes depending on, for example, the time preferences of customers. Due to technological development, centralized, calendar-oriented software for scheduling meetings is available, which can potentially improve the business process of time management [4]. However, the methodology of redesigning such a business process, considering both the internal structure of the process, and uncertainties of the external environment, does not exist.

The above mentioned factors served as the prerequisite for the development of a methodology for data-centric business process improvement based on the application of decision theory, which we present in this report. Our fundamental contribution is a presentation of the integrated methodology for the identification of patterns for redesign in process models, redesign guidelines and introduction of process indicators which will allow the effectiveness of the redesigned models to be monitored.

The remainder of the report is structured as follows. The related work is provided in Section 2. In Section 3 the notions of process models, data and the foundations of decision theory used in our approach are presented. In Section 4 we introduce a special kind of process model, a decision subprocess, which serves as a redesign pattern. Additionally, we present a transformation rule for improvement of such a process model. In Section 5 we discuss the planned future work. Finally, the report is concluded.

2 State-of-the-Art

In contrast to the topic of process modeling, process redesign has not received so much attention from the scientific community [3]. A fundamental approach for business-process re-design based on best practices of successful redesign heuristics was presented in 2005 in [11]. In this report the authors are introducing best practices, which can support the technical challenge of the business process re-design challenge in four dimensions: time, cost, quality and flexibility. This approach was applied, for example, in the healthcare domain for the reduction of throughput and service times of medical management processes, as described in [6]. As well, a number of different automation platforms supporting business process re-design were presented to the public, such as a framework based on Petri-nets [13] or, for example, software based on process mining techniques [7].

However, the above mentioned approaches are based on traditional activity flows and most of them do not consider data or business artifacts presented in the models. In our work we suggest an integrated approach which considers both activities, and the data of process models. Similar work was presented in IBM's artifact-centric process modeling approach [5]. Also, the artifact-based approach was developed at Eindhoven University of Technology in cooperation with a Dutch consultancy company [12]. However, the above mentioned approaches provide company-specific redesign patterns. In contrast, our goal is to provide a generic hybrid scheme for business process re-engineering, based on the application of techniques from decision theory.

3 Foundations

The generic scheme of our approach for business process redesign is presented in Figure 1, the detailed version of which can be found in our paper [1]. The first step is to identify if the initial process model P contains patterns for redesign. The example of such a pattern, a decision subprocess, is presented in Section 4. If it is detected that the process model contains such patterns, the transformation of the process model is implemented as the second step of the redesign scheme, which will be explained in detail in Section 4. This transformation yields, as an outcome, an improved process model P' . To verify the effectiveness of the transformation, the third step of the redesign scheme simulates the execution of the improved process model P' with the usage of the key performance indicators, their development is planned for future work.

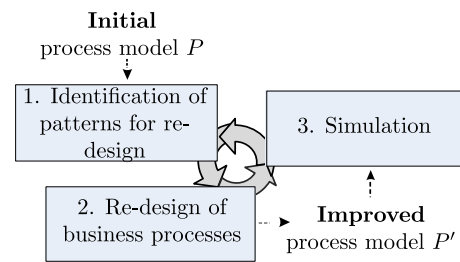


Figure 1: Scheme for business process improvement

Depending on the simulation results, a conclusion is made, either to accept the improved process model P' and start using it in the enterprise, or to conduct further improvements of the process model. Such a decision can be done, for example, by a business analyst or higher management.

3.1 Process Model and Data

The input and output for our redesign scheme are process models, which can be viewed as blueprints for a set of process instances with a similar structure [14].

Definition 2.1 (Process Model). $P = (N, E, D, F, R, \psi, \gamma)$ is a *process model* if it consists of a finite non-empty set N of nodes, and a finite set E of edges. Herewith, $N = N_A \cup N_E \cup N_G$ is a union of the mutually disjoint nonempty sets N_A (a set of activities), N_E (a set of events), and N_G (a set of gateways). With that, E is a set of directed edges between nodes, such that $E \subseteq N \times N$, representing control flow. Further, F is a set of edges representing data flow relations: $F \subseteq (N_A \times D) \cup (D \times N_A)$. R is a set of resources assigned to activities. $\psi : N_A \rightarrow R$ is a function assigning to each activity a corresponding resource. $\gamma : N_G \rightarrow \{\text{xor}, \text{and}\}$ is a function assigning to each gateway a corresponding control flow construct. \diamond

In Definition 2.1, we take into account the resources which are involved in the execution of a business process. It is also assumed in the definition that the activities of process models operate on an integrated set D of data nodes, which represent application data, created, modified, and deleted during the execution of a process model. The term data flow refers to dependencies between process activities and data.

In our work we use the distinction of process data into data classes and data nodes (see Figure 2), which can be viewed as analogous to the object-oriented programming paradigm.

Data class, used in a process model, serves as an abstract data type, which describes the properties of *data nodes*. The data nodes can be viewed as instances of the data classes at the modeling level. Data nodes are associated

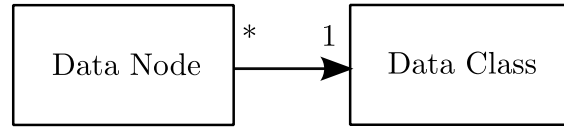


Figure 2: Relations between data entities

with exactly one data class in a process model, in a way that the particular values of data class properties are assigned to the data node associated with it.

Definition 2.2 (Data Class). *Data class* $D_c = (\text{name}, S, Q_c)$ is a tuple, where:

- name is a constant which serves as a unique identifier for the data class D_c ;
- S is a finite non-empty set of data states;
- Q_c is a finite set of attributes, which are properties representing data fields containing values of an arbitrary type. \diamond

Definition 2.3 (Data Node). Let D_c be a data class, used in a process model. A tuple $D_n = (\text{name}, s, \delta, \tau, \varphi, Q)$ is a *data node*, related to the corresponding data class D_c , with the following parameters:

- name is a constant labeling data node D_n , which serves as a reference to the corresponding data class D_c ;
- $s \in S$ is a variable reflecting the state assigned to D_n , where S is the set of data states of D_c ;

- $\delta : D_n \longrightarrow \{\text{singlinst}, \text{multinst}\}$ is a function indicating if the data is a collection (singlinst) or not (multinst);
- $\tau : D_n \longrightarrow \{\text{input}; \text{output}; \text{default}\}$ is a function indicating if D_n is an input data node (existed before the start of the process), output data node (will exist after termination of the process) or none of these (default);
- $\varphi : D_n \longrightarrow R$ is a function indicating the resource allocated for D_n ;
- $Q \in Q_c$ is a set of attributes assigned to D_n , where Q_c is a set of attributes of D_c . ◇

To be precise, we assume that the resource of a process model, allocated for the data node, is the same as the resource allocated to the activity of a process model, which accesses this data node. Thus, the value of the function ψ (from Definition 2.1), mapping the activity a to the resource R , is equal to the value of the function φ (from Definition 2.3), mapping the data node D_n , with which a is in a data flow relation, to the same resource R . More specifically, $\varphi(D_n) = \psi(a)$, where $a \in N_A$, and $(a, D_n) \in F \vee (D_n, a) \in F$. Also, as it can be seen from Definition 2.3, the set of attributes Q store the context data relevant to the business process, i. e. the particular characteristics of the data class.

3.2 Definitions from Decision Theory

As it was mentioned in the introduction, many business processes face uncertainties of the business environment and decision theory is a tool which is focused on dealing with such challenges. Below we provide the notions used in our approach, with regards to the foundations of decision theory [8, 10].

The core setting of decision theory is an occurrence of a subject *decision maker* whose aim is to make an optimal choice between a set of n *alternatives*: $X = \{x_i\}$, $i = 1, \dots, n$, with a possible *outcome* event O . The main assumption is that any realization of the alternatives resulting from a decision can be compared, which is described by the *preference relations* of the decision makers, represented by the \succsim sign.

Definition 2.4 (Preference Relation). A *preference relation* \succsim is a subset of the cartesian product $X \times X$, that satisfies two principles :

1. *Completeness*. $\forall x_i, x_j \in X: x_i \succsim x_j$, or $x_j \succsim x_i$.
2. *Transitivity*. $\forall x_i, x_j, x_k \in X: \text{if } x_i \succsim x_j \text{ and } x_j \succsim x_k \text{ then } x_i \succsim x_k$. ◇

Definition 2.5 (Lottery). A *lottery* L is a finite vector (p_1, \dots, p_n) , where p_i is the probability that the alternative $x_i \in X$ will be realized, such that $\sum_{i=1}^n p_i = 1, p_i \geq 0$. ◇

Another assumption of decision theory is that a decision maker is making a choice in a rational way, which is expressed by a *utility function* assigned to the decision maker.

Definition 2.6 (Utility Function). A *utility function* u is a function which assigns a real number to any given choice of the alternatives, $u : X \rightarrow \mathbb{R}$ where \mathbb{R} is a set of real numbers. A utility function u is said to represent a *preference relation* \succsim if and only if $\forall x_i \in X, \forall x_j \in X, u(x_i) \geq u(x_j) \Leftrightarrow x_i \succsim x_j$ \diamond

The value of the utility function is a *payoff*. For comparing the alternatives in a decision making process, a notion of *expected payoff* is used:

Definition 2.7 (Expected Payoff of the Lottery). An *expected payoff* E of the lottery is the average of payoffs which the decision maker gets from the assumed realization of the alternative, weighted by the probability of such a realization: $E(L) := \sum_{i=1}^n p_i u(x_i)$ \diamond

In terms of the introduced definitions, the assumption of rational behavior is the following: the goal of each decision maker is to maximize the expected payoff of the lottery.

4 Decision Subprocess Improvement

Searching for ways to improve business processes led us to consider the typical challenges of the business environment, such as making decisions under conditions of risks and uncertainties. In order to provide an effective mechanism for dealing with the uncertainties in business environment, in this section we provide a mapping between the decision theory and business process management, and devise how to use it for the business process redesign.

4.1 Process Model as a Decision Subprocess

The notions of decision theory, presented in Section 3, provides the premise for defining a special kind of business process models, which we refer to as *decision subprocesses*.

The generic structure of a decision subprocess is shown in Figure 3. The decision subprocess represents a process model, the internal logic of which is hidden inside

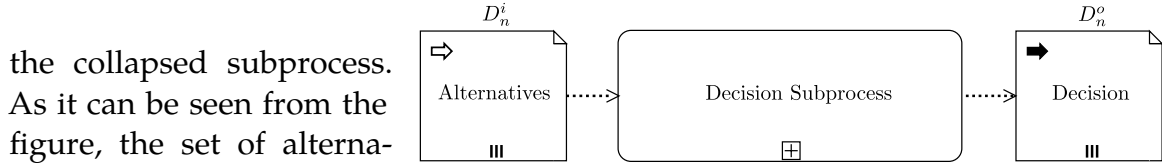


Figure 3: Structure of a decision subprocess

the collapsed subprocess. As it can be seen from the figure, the set of alternatives in the decision subprocess is presented as the collection input data node D_n^i , and the final decision is presented as the collection output data node D_n^o , so that $\tau^i = \{\text{input}\}$, $\tau^o = \{\text{output}\}$. The decision subprocess should reflect the process of decision making, therefore it is assumed that the data represented by the output data node “Decision” is a subset of the data represented by the input data node “Alternatives”.

Based on the above mentioned considerations, the decision subprocesses can be defined formally, based on following conditions. Let P be a process model, which consists of K data nodes, including the input data node D_n^i and the output data node D_n^o , which are bound to J data classes.

Condition 1. Set of alternatives is represented by the set of attributes Q^i of the input data node D_n^i .

Condition 2. Final decision is represented by the set of attributes Q^o of the output data node D_n^o . The set of attributes Q_c^o of data class D_c^o , which is assigned to the *output* data node D_n^o , is a subset of the set of attributes Q_c^i of data class D_c^i , which is assigned to the *input* data node D_n^i : $Q_c^o \subseteq Q_c^i$.

Condition 3. Decision makers are represented by a function φ , indicating resources allocated for data nodes D_n (see Definition 2.3).

Condition 4. Decision making process consists of decision makers choosing alternatives, so that each set of attributes Q of data class D_c assigned correspondingly to any data node D_n is a subset of the set of attributes Q_c^i of data class D_c^i assigned to the input data node D_n^i : $\forall Q_c : Q_c \subseteq Q_c^i$.

Definition 2.8 (Decision subprocess). If a given process model P satisfies conditions 1–4, then such a process model represents a *decision subprocess*. \diamond

4.2 Scheme of Business Process Improvement

The introduction of the decision subprocess enables us to suggest an approach for the improvement of such a process model. Below we present the detailed approach, which consists of three consequent phases, corresponding to three stages of the scheme for business process improvement (see Figure 1):

S1 (a). Analysis of Business Process Model The business process improvement scheme is launched when a business analyst of the enterprise decides that the current business process is not efficient.

S1 (b). Detection of Decision Subprocess It is identified if the current process model P represents a decision subprocess, according to Definition 2.8.

S2 (a). Definition of Payoff Function The improvement of the internal structure of the decision subprocess (i.e., the collapsed subprocess in Figure 3) can be done by the application of the decision theory methods. The persons, or other resources, involved in the execution of the decision subprocess, can be viewed as decision makers. Additionally, according to the assumption of rational behavior of decision makers, their goal is to maximize the *expected payoff* for the decision subprocess. Therefore, the assigned goal of this stage is to set the payoff function of the decision subprocess. The example of the payoff function could be the time saved by participants, to agree on the decision.

S2 (b). Optimization of Decision Subprocess In such a way, we reduced the challenge of business process improvement to the task of maximizing the expected payoff for the decision subprocess. To solve this task, we propose the following transformation, which consists of two steps:

1. All the data classes D_c of the decision subprocess are consolidated into one data class D'_c . Such transformation preserves the business context of the process model, as, according to Condition 4 of Definition 2.8, each set of attributes of any data class in a decision subprocess is a subset of the set of attributes of the data class assigned to the input data node.
2. The access management of resources is changed in such a way that within the decision subprocess, all the resources should have access to all the data nodes assigned to the consolidated data class D'_c .

The output of such a transformation is a process model P' , which is different from the initial process model P only in a way, that it contains a set of K data nodes D'_n , all of which are assigned to one consolidated data class $D'_c = (\text{name}', S', Q'_c)$, where

- name' reflects the consolidated nature of the data class, name' can be assigned by a business analyst;
- $S' = \{S_j\}, j = 1, \dots, J$ is the set of states retrieved as a maximal subset of the sets of states of data classes $D_c^j, j = 1, \dots, J$ assigned to the initial process model P ;
- Q'_c is the consolidated set of attributes retrieved as a maximal subset of the sets of attributes for all data classes in the initial process model P .

The data nodes D'_n of the transformed model P' are different from the corresponding data nodes D_n of the initial model P only in a way, that the value of the parameter

name' for each data node D'_n is equal to the value of the corresponding parameter of the consolidated data class D'_c .

S3. Simulation of Redesigned Process Model In order to assess the efficiency of the transformation, we plan to develop a set of indicators and conduct a simulation of the process model for estimating the values of these indicators. This is the final step of the improvement scheme which is planned for future work.

To conclude, in this section we presented an integrated methodology for the identification of decision subprocesses in process models and its redesign guidelines. An introduction of process indicators which will allow the effectiveness of the redesigned models to be monitored, is not covered in this report and is planned for future. We demonstrated the applicability of our approach to business process improvement with a use case, which incorporates the decision making process, in our paper [2].

5 Planned Work

In this report we presented a methodology for detecting and improving decision subprocesses in process models. However, the step "Optimization of Decision Subprocess" (see Section 4.2) at the current moment operates with the most abstract definition of the decision subprocess (see Definition 2.8), which does not take into account the specifics of the decision subprocess. For example, the ways of representation of preferences of decision makers can differ from voting to executive decision. However, the differentiation of the decision subprocesses into different types might help to concretize the approach of maximizing the expected payoff of the subprocess. In order to investigate this hypothesis, the following activities are planned:

1. Classification of the Types of Decision Subprocesses This step implies a thorough review and analysis of interdisciplinary literature on decision and business process management theories. All decision subprocess in process model differ from each other in terms of structuredness, representation of the information, number of decision makers etc. We plan to make an extended classification of possible types of decision subprocesses.

2. Detection of Decision Subprocesses Types Based on the classifications of types of decision subprocesses, we plan to derive its type automatically from the process model. For example, the use case of scheduling the meeting of enterprises, based on

its process model structure (several roles, non-shared access to data objects), might have a type “group decision, anonymous voting”.

3. Guidelines on Changing a Decision Subprocess Type In order to improve a given decision subprocess, we plan to generate the recommendations on changing the type of the decision subprocess. Such transformation should increase the expected payoff of the process. For example, the improvement of scheduling process can be done by granting access for the decision makers to the data produced by each participant. Then the decision makers can estimate the payoff of their choices more precisely which might result in reducing the time spent on the decision making process and the time spent on the execution of the whole scheduling process. In such a case, it can be recommended for a business user to change the type of the process from “group decision, anonymous voting” to “group decision, non-anonymous voting”. Our goal is to provide such recommendations automatically.

4. Evaluation of Redesigned Decision Subprocess In order to assess the efficiency of changing the type of the decision subprocess, we plan to develop a set of indicators or non-functional parameters and to conduct a simulation of the process model for estimating the values of these indicators. This is the final step of the improvement scheme, and, depending on the results of the simulation, the conclusion is made, either to accept the changing of the type of the decision subprocess, or to conduct further improvements.

6 Conclusion

In this report we provided an approach for business process improvement, according to the scheme, consisting of the identification of specific patterns in process models and the redesigning of these models in order to increase its efficiency.

As a specific pattern for potential business process improvement, we presented a decision subprocess, which incorporates the mapping of decision theory and the business process model at the modeling level. We introduced an approach for improving the internal structure of the decision subprocess by introducing and maximizing the payoff function.

At the current moment, the transformation step of the improvement scheme stays quite context-dependent and heuristic. To overcome this issue, we came up with a plan of making the classification of different types of decision subprocesses and deriving the type of the decision subprocess from the process model automatically. We plan to provide the guidelines to a business user on how to change the type of the decision subprocess in order to increase its expected payoff.

References

- [1] E. Bazhenova. "Support of Decision Tasks in Business Process Management". In: *Proceedings of the 2014 Central European Workshop on Services and their Composition*. ZEUS '14. 2014.
- [2] E. Bazhenova and M. Weske. "A Data-Centric Approach for Business Process Improvement Based on Decision Theory". English. In: *Proceedings of the 15th International Conference on Business Process Modeling, Development and Support (BPMDS 2014): Enterprise, Business-Process and Information Systems Modeling*. Volume 175. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2014, pages 242–256. DOI: 10.1007/978-3-662-43745-2_17.
- [3] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013, pages I–XXVII, 1–399.
- [4] E. Ephrati, G. Zlotkin, and J. S. Rosenschein. "Meet your Destiny: A Non-Manipulable Meeting Scheduler." In: CSCW. Edited by J. B. Smith, F. D. Smith, and T. W. Malone. ACM, 1994, pages 359–371.
- [5] R. Hull. "Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges". In: *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*. OTM '08. Monterrey, Mexico: Springer-Verlag, 2008, pages 1152–1163. DOI: 10.1007/978-3-540-88873-4_17.
- [6] M. H. Jansen-Vullers and H. A. Reijers. "Business Process Redesign in Healthcare: Towards a Structured Approach". In: *INFOR: Information Systems and Operational Research* 43.4 (2005), pages 321–339.
- [7] M. M. Essam and S. L. Mansar. "Towards a Software Framework for Automatic Business Process Redesign". In: *ACEEE International Journal on Communication* 2.1 (Mar. 2011). Edited by D. V. V. Das, page 6.
- [8] J. V. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
- [9] K. Ngamakeur, S. Yongchareon, and C. Liu. "A framework for realizing artifact-centric business processes in service-oriented architecture". In: *Proceedings of the 17th international conference on Database Systems for Advanced Applications - Volume Part I. DASFAA'12*. Busan, South Korea: Springer-Verlag, 2012, pages 63–78. DOI: 10.1007/978-3-642-29038-1_7.
- [10] J. Pratt, H. Raiffa, and R. Schlaifer. *Introduction to statistical decision theory*. Cambridge, Mass. [u.a.]: MIT Press, 1995. XIX, 875.

- [11] H. A. Reijers and S. Liman Mansar. "Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics". In: *Omega* 33.4 (2005), pages 283–306.
- [12] H. A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Edited by G. Goos, J. Hartmanis, and J. van Leeuwen. Lecture Notes in Computer Science 2617. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.
- [13] W. M. P. Van der Aalst and K. M. van Hee. "Framework for business process redesign". In: *Proceedings of the 4th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'95)*. WET-ICE '95. Washington, DC, USA: IEEE Computer Society, 1995, pages 36–.
- [14] M. Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012, pages I–XV, 1–403.

History Assisted View Authoring for 3D Models

Tim Chen

HCI Group
Hasso-Plattner-Institut
tim.chen@hpi.de

3D modelers often wish to showcase their models and associated workflows for review, tutorial, and visualization purposes. This may consist of generating static viewpoints across versions of the model, or authoring animated fly-throughs. Unfortunately, manually creating such views is often tedious and few automatic methods are designed to interactively assist the modelers on view authoring process. We present a new view authoring system that supports automatic creation of informative view points, view paths, and view surfaces, allowing authors to create interactive summaries and viewpoint collages of a model. The key concept of our implementation is to analyze the model's workflow history, to infer important regions of the model and representative viewpoints of those areas. An evaluation indicated that the viewpoints generated by our algorithm are comparable to those manually selected by the modeler. In addition, participants of a user study found our system easy to use and provides an effective alternative for authoring viewpoint summaries.

1 Introduction

Recent years have witnessed significant progress in 3D acquisition and modeling. These advancements have resulted in a variety of 3D model libraries, and an abundance of users wishing to understand proper modeling processes. It has thus become desirable to be able to rapidly summarize and display 3D models [6] and their associated workflows [4]. This can be achieved through a collection of static viewpoints or animated fly-throughs across versions of the model. However, manually creating such views can be a long and tedious process.

As a result, automatic viewpoint selection [5, 9] has been an active area of research in the Computer Graphics community. However, the associated algorithms that have been developed typically consider only the final geometric models. Such algorithms may ignore features of the model that an author spent a particular amount of effort on during their authoring process. Furthermore, automatic techniques that rely solely on geometric features may not produce subjectively preferable views.

Alternatively, in the HCI literature, techniques for navigating along constrained view paths and view surfaces to obtain effective overviews of 3D models have been

investigated [1]. However, these systems require predefined constrained views. The authoring of such views, either manually or automatically, remains an open problem.

Recent work has demonstrated the utility of instrumenting a 3D modeling environment for visualizing editing operations and model differences [4]. Guided by a set of observations and interviews, we hypothesize that enhancing such instrumentation to also include the camera history could be useful for authoring viewpoint summaries. In particular, original workflows used to create a model may be indicative of the important parts of the models and the viewpoints from which these important parts should be viewed. Thus, by capturing these workflows in-situ, effective views can be derived from the workflows in addition to the final geometric models.

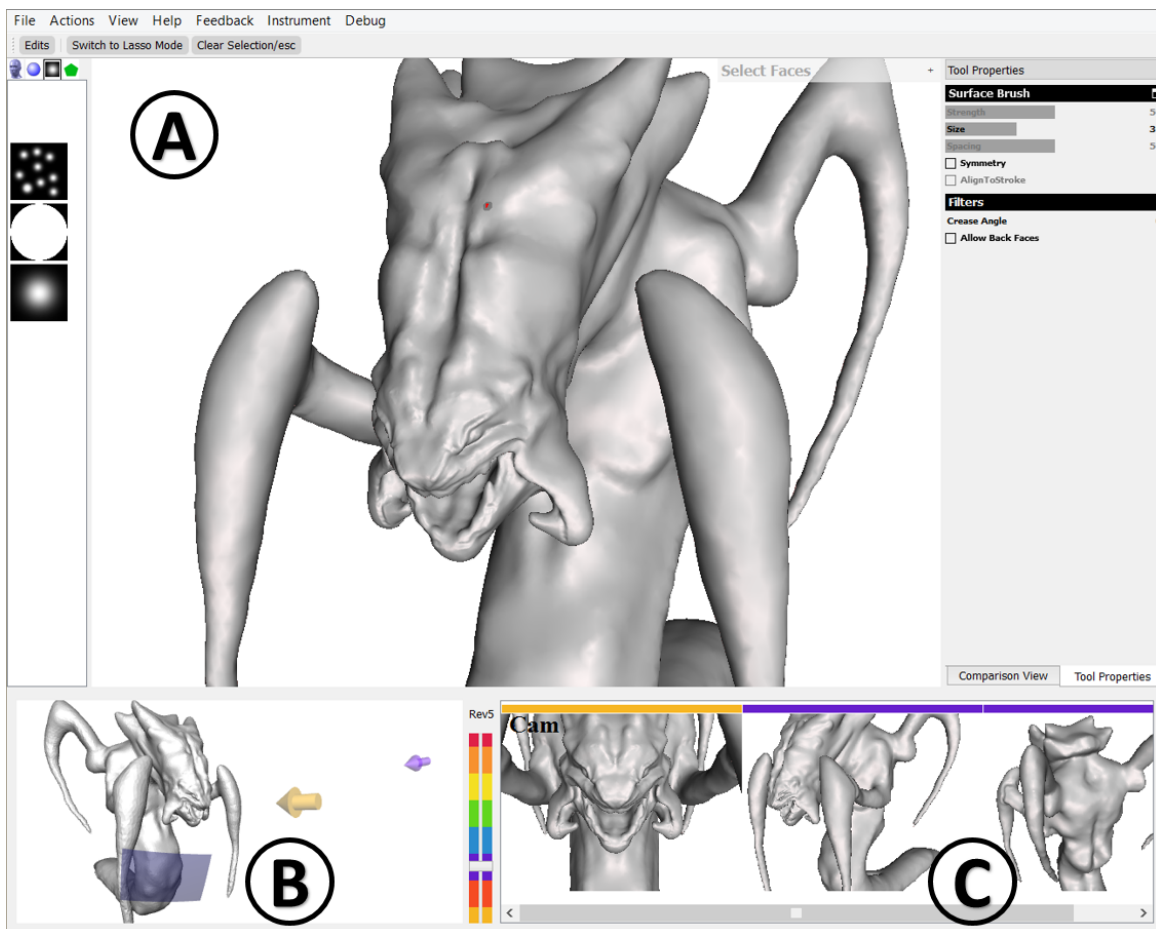


Figure 1: (A) the main MeshMixer modeling window, (B) the authoring overview panel, and (C) the navigation panel

In this project, we contribute a history assisted view authoring and navigation system, which is integrated with the MeshMixer sculpting tool, for 3D modelers (Figure 1). Our system stores the editing and viewpoint history along with discrete versions of the underlying model. By analyzing both the history and the model, our system can automatically generate informative views that may consist of viewpoints, view paths, and view surfaces, from which the original authors can further customize. The system also provides a region-specific view authoring function, where modelers can directly specify regions of interest on the object surface, allowing our system to automatically suggest the corresponding views.

Our system produces an interactive summary of the 3D model, allowing viewers to easily navigate between authored views and along view paths, and compare different versions from the model’s revision history. The authored views can also be exported as an interactive viewpoint collage for brochure/catalogue-like display.

2 Algorithm and Implementation

2.1 Editing History Instrumentation

The editing history recorded by our instrumented MeshMixer comprises of a complete log of the two most significant interactions in a 3D sculpting tool: interactive sculpting brush strokes, and 3D camera manipulations. For each sculpting brush operation we store: a unique operation ID, elapsed time of the stroke, a list of the affected vertices, and the current camera viewpoint. For the camera history we store all intermediate cameras (location, look-at, and up vector) during each camera transformation operation, as well as the elapsed time of the entire camera manipulation.

2.2 Region-Specific View Suggestion

Our system interactively suggests candidate views based on the region specified by the modeler. The following paragraphs describe the algorithms that generate candidate viewpoints, view paths and view surfaces.

Candidate Viewpoint and View Path Construction Our approach is motivated by the camera oscillations discussed in our formative observations. The captured sculpting views and inspecting views tend to form dense, disjoint spatial clusters whose weighted centers we interpret as candidate viewpoints for the surface region specified by the modeler.

Given the modeler-specified region, we identify sculpting views as those where the modeler applied a sculpting brush to that region. We then apply spatial clustering to

these views to pick a specific camera. Our algorithm takes a set C of sculpting-view cameras, and builds an octree spatial decomposition based on the camera positions. We then compute the accumulated camera time in each cell, by summing the camera times in its sub-tree, and finding the dominant cell whose accumulated time consists of 70% of the total time. Finally, we calculate the weighted average center (based on time) of the viewpoints in the dominating cell, and select the camera in C that is closest to the calculated center. We denote this camera viewpoint c_{sculpt} , the representative sculpting-view for the region of interest.

Note that in our prototype system, we use the octree spatial decomposition among many other possible clustering techniques (e.g. k-mean), mainly because it is how MeshMixer stores the geometric primitives and many query functions are optimized for the octree data structure. To identify inspection views, viewpoints used by the author to assess the model, we search for camera manipulations which occur between two sculpting operations applied to the same segment. We treat these as inspection operations. For each such camera manipulation, the sequential camera positions create a 3D piecewise-linear curve L . We compute the opening angle at each vertex (camera position) of L , and choose as a turning point, c_{turn} , the vertex with largest opening angle (Figure 2). For a given region, we take the 3D cameras associated with all of the turning points and then apply the same clustering technique described above to extract the inspection-view, c_{inspect} , for a segment. Iterating the procedure described above, we can extract multiple sculpting views and inspection views. We then treat these extracting views as candidate viewpoints and the oscillating camera traces these views belong to as candidate view path of the specified region.

Candidate View Surface Construction We hypothesize that a view surface that covers a cluster of sculpting views would be a good candidate for inspecting the selected region. Given a surface region of a 3D model, we calculate the average center v_{center} of the surface vertex and obtain its dominating octree cell of sculpting views and representative sculpting-view c_{sculpt} with the algorithm described above. Next, we construct a spherical patch with the center at v_{center} , radius as $\|c_{\text{sculpt}} - v_{\text{center}}\|$, and spanning angles that cover the octree cell.

2.3 Global Viewpoint Suggestions

In addition to the interactive region-specific view suggestions, our system can also automatically segment the 3D model into meaningful surface regions and suggest good viewpoints for each of them. This allows our system to recommend a set of global viewpoints that summarize the entire model. Our observations suggest that modelers sculpt sequentially and locally. This leads to a feature-centric approach—we explicitly segment the surface into (potentially overlapping) regions based on

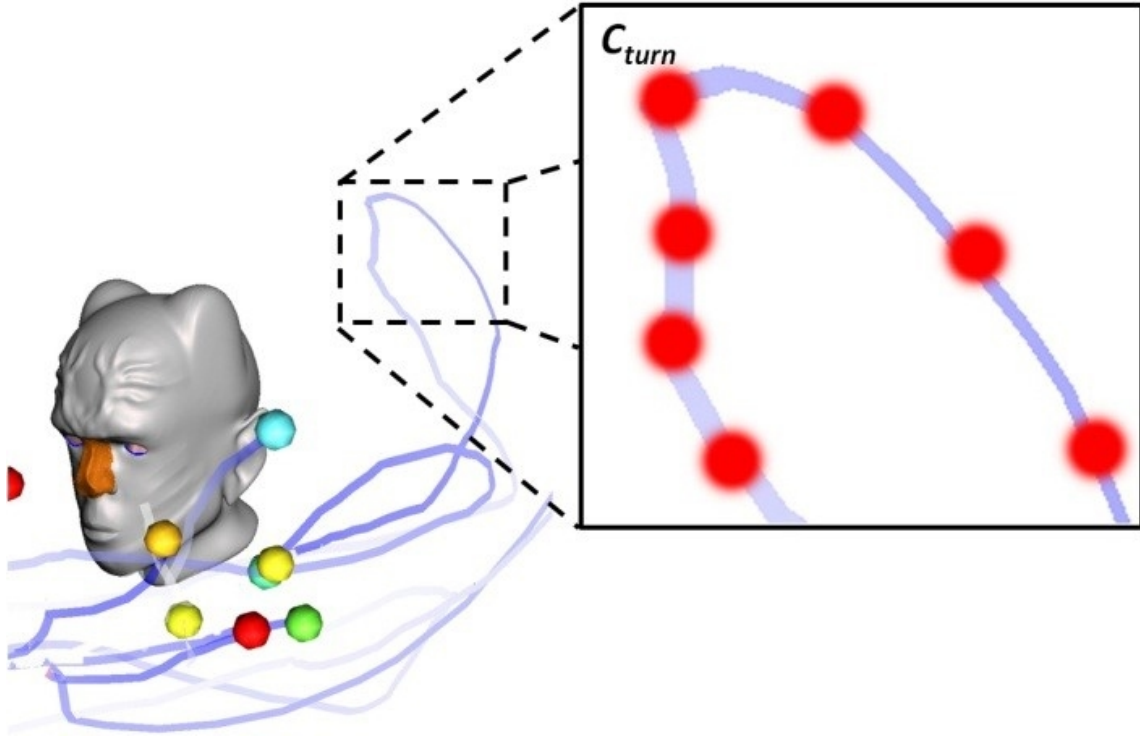


Figure 2: C_{turn} is the turning point of the oscillation camera movement pattern

the editing history and then rank them by an importance factor. We first define a segment as a region in which each vertex has been given a roughly similar amount of attention (i.e. accumulated sculpting time) by the modeler. We then extract segments via standard greedy region-growing [18]. To create a segment S , we first choose as the segment seed the vertex v_s with the largest accumulated sculpting time $T(v_s)$. We then incrementally include vertices v adjacent to S which have a similar $T(v)$ to the average accumulated sculpting time for S , denoted $T(S)$. Specifically, we add v to S if $\|T(S) - T(v)\| < 0.3 * T(S)$. The set S is then removed from the candidate set and we repeat the above process until either the required number of segments is found, or all vertices have been consumed.

Per our observations, most individual brush strokes do not span multiple semantic features of a model (e.g. modifying the nose and ear in a single stroke). As our greedy segmentation does not enforce this, we apply a refinement post-process. For each modeling operation that affects any of the vertices contained in segment S , we grow S to include all vertices of that operation. We note that this refinement step has the added benefit of cleanly handling cases where sculpting brushes have automatically been applied symmetrically, as is very frequently in sculpting workflows (Figure 3).

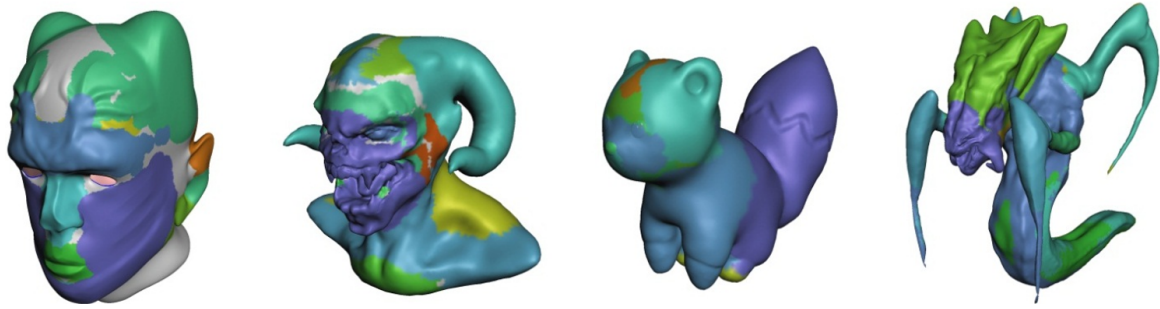


Figure 3: Segments generated by our algorithm

3 Related Work

3D Viewpoint Selection At the core of our system is the viewpoint selection algorithm. Previous work [7] infers the importance of viewpoints based on the visible geometric attributes of a given 3D surface. Secord et al. [8] summarize existing techniques, and explore viewer preferences via crowd sourcing. They then generate weightings of the various criteria explored in the literature such that the fitted model can predict people’s preferred views. By additionally considering the editing history of a 3D model, our approach could potentially create viewpoint selection results that better convey the original author’s intentions.

Capturing and Utilizing Workflow Histories Numerous systems have been proposed to automatically record users’ workflows within a software application. Captured interaction history and workflows have been shown to be useful for creating step-by-step tutorials [3] or allowing users to explore and replay editing history of a document [2]. Our work on viewpoint summarization demonstrates a previously unexplored way to utilize captured workflows: to select views and create model summarizations.

4 Conclusion

We propose a history assisted view authoring system for 3D models. Our system provides modelers the ability to traverse and author the camera views, paths, and surfaces across different model versions. We also introduce the region-specific view authoring technique where candidate views are automatically calculated based on the surface region specified by the 3D modeler. Internally, we introduce a unified algorithm framework for 3D model segmentation and automatic view selection based on the 3D model and editing history. Our user studies show that the quality of

the viewpoints generated by our algorithm is comparable to ones manually selected by the modeler, and the history-assisted authoring system elicited a high level of interest from potential end-users.

References

- [1] N. Burtnyk, A. Khan, G. Fitzmaurice, R. Balakrishnan, and G. Kurtenbach. “StyleCam: Interactive Stylized 3D Navigation Using Integrated Spatial & Temporal Controls”. In: *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*. UIST ’02. Paris, France: ACM, 2002, pages 101–110. DOI: 10.1145/571985.572000.
- [2] H.-T. Chen, L.-Y. Wei, and C.-F. Chang. “Nonlinear Revision Control for Images”. In: *SIGGRAPH ’11*. 2011, 105:1–10.
- [3] P.-Y. Chi, S. Ahn, A. Ren, M. Dontcheva, W. Li, and B. Hartmann. “MixT: Automatic Generation of Step-by-step Mixed Media Tutorials”. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. UIST ’12. Cambridge, Massachusetts, USA: ACM, 2012, pages 93–102. DOI: 10.1145/2380116.2380130.
- [4] J. D. Denning, W. B. Kerr, and F. Pellacini. “MeshFlow: Interactive Visualization of Mesh Construction Sequences”. In: *SIGGRAPH ’11*. 2011, 66:1–66:8.
- [5] M. Feixas, M. Sbert, and F. González. “A Unified Information-theoretic Framework for Viewpoint Selection and Mesh Saliency”. In: *ACM Trans. Appl. Percept.* 6.1 (Feb. 2009), 1:1–1:23. DOI: 10.1145/1462055.1462056.
- [6] V. G. Kim, W. Li, N. J. Mitra, S. DiVerdi, and T. Funkhouser. “Exploring Collections of 3D Models Using Fuzzy Correspondences”. In: *ACM Trans. Graph.* 31.4 (July 2012), 54:1–54:11. DOI: 10.1145/2185520.2185550.
- [7] C. H. Lee, A. Varshney, and D. W. Jacobs. “Mesh saliency”. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH ’05. Los Angeles, California: ACM, 2005, pages 659–666. DOI: <http://doi.acm.org/10.1145/1186822.1073244>.
- [8] A. Secord, J. Lu, A. Finkelstein, M. Singh, and A. Nealen. “Perceptual Models of Viewpoint Preference”. In: *ACM Trans. Graph.* 30.5 (Oct. 2011), 109:1–109:12. DOI: 10.1145/2019627.2019628.
- [9] P. Shilane and T. Funkhouser. “Distinctive Regions of 3D Surfaces”. In: *ACM Trans. Graph.* 26.2 (June 2007). DOI: 10.1145/1243980.1243981.

Service-Oriented Integration and Processing of Massive 3D Point Clouds

Sören Discher

Computer Graphics Systems Group
Hasso-Plattner-Institut
soeren.discher@hpi.uni-potsdam.de

Advances in remote and in-situ sensing technology allow for the creation of dense, spatial overlapping, and multitemporal 3D point clouds that are updated continuously. However, workflows of traditional geoinformation systems commonly limit the use of 3D point clouds to the generation of mesh-based 3D models which in turn serve as the basis for subsequent processing and analysis tasks. Since the generation of accurate mesh-based 3D models from 3D point clouds is time-consuming, the overall performance can be improved notably by conducting these tasks directly on the point data. In this report, concepts for the service-oriented integration, processing, and provision of massive 3D point clouds are discussed. A prototypical implementation of a corresponding service-oriented infrastructure is presented and its performance is evaluated based on two different real-world scenarios.

1 Introduction

Traditionally, geoinformation systems (GIS) use *3D point clouds*, i.e., point-based digital representations of real-world surfaces that can be created efficiently and automatically by means of in-situ and remote sensing technology, to derive 3D meshes, such as building or terrain models [1, 6]. These, in turn, can be applied in different areas such as urban planning and development, environmental monitoring, disaster and risk management, or homeland security [3]. Deriving mesh-based 3D models from 3D point clouds is however a time consuming task as it often requires manual post processing. Consequently, that approach is inefficient for large data sets, especially when parts of the data have to be updated continuously.

These limitations have become more apparent in recent years due to advances in 3D laser scanning technology leading to an ever-increasing density (e.g., 400 points per m^2), availability, and capturing frequency of point-based 3D models. As an example, an increasing number of german federal states captures their territory on a regular basis (e.g., once a year) by means of aerial laser scanning or photogrammetric approaches to facilitate the process of updating existing 3D city models or cadastre data. Terrestrial laser scanning technology is becoming more prevalent in every

day life, e.g., as part of navigation and driver assistant systems, that use various sensors to constantly capture the environment of a car and to identify relevant objects (e.g., cars or pedestrians) within. Furthermore, that data can also be used to derive georeferenced models of the environment [7]. Simultaneously, several consumer applications (e.g., *Autodesk 123D* or *Microsoft Photosynth*) have been developed in recent years that simplify the creation of digital models from real-world objects by means of photogrammetric approaches [14], thus making that technology accessible to a broad audience.

By combining these different data sources, point-based models of cities and landscapes can be created that are not only *dense* and highly detailed, but also *spatial overlapping*, inherently featuring different levels of detail (LoD) since the given area has been captured from different point of views (i.e., airborne, terrestrial, or focused on single objects). In addition, by combining information from multiple data sources, the model can be updated more frequently, thus, creating a more realistic representation of the real world. Keeping track of these changes, i.e., storing not only positional but also temporal information, serves as a basis for various processing tasks (e.g., detecting changes in an area over a certain amount of time) and speeds up subsequent processing tasks in general as it allows for the efficient selection of areas that have been subjected to changes in a given period of time.

However, to use dense, spatial overlapping, and multitemporal 3D point clouds that are updated continuously in an efficient way, the workflows of traditional GIS are not sufficient. Instead, we propose to conduct - whenever possible - processing and analysis tasks directly on the point data. In this report, concepts for a service-oriented infrastructure for 3D point clouds are discussed allowing for (1) the integration of heterogeneous 3D point clouds into a homogeneous spatial data model, (2) the efficient conduction of common analysis and processing tasks (e.g., calculating distances for multitemporal point data), and (3) the efficient provision of the point data for subsequent processes (e.g., to integrate processing results into process chains of existing GIS). To display the practicability of the presented concepts, a prototypical implementation of such an infrastructure is presented and evaluated for two different use cases.

2 Service-Oriented Infrastructure for 3D Point Clouds

In workflows of traditional GIS the use of 3D point clouds is mostly limited to the generation of mesh-based 3D models which in turn are used subsequently as the *geographic base data* for various processing and analysis tasks. However, conducting these tasks directly on the point data, thus, avoiding the time-consuming generation of 3D meshes, can speed up the performance of these tasks notably [8]. To establish

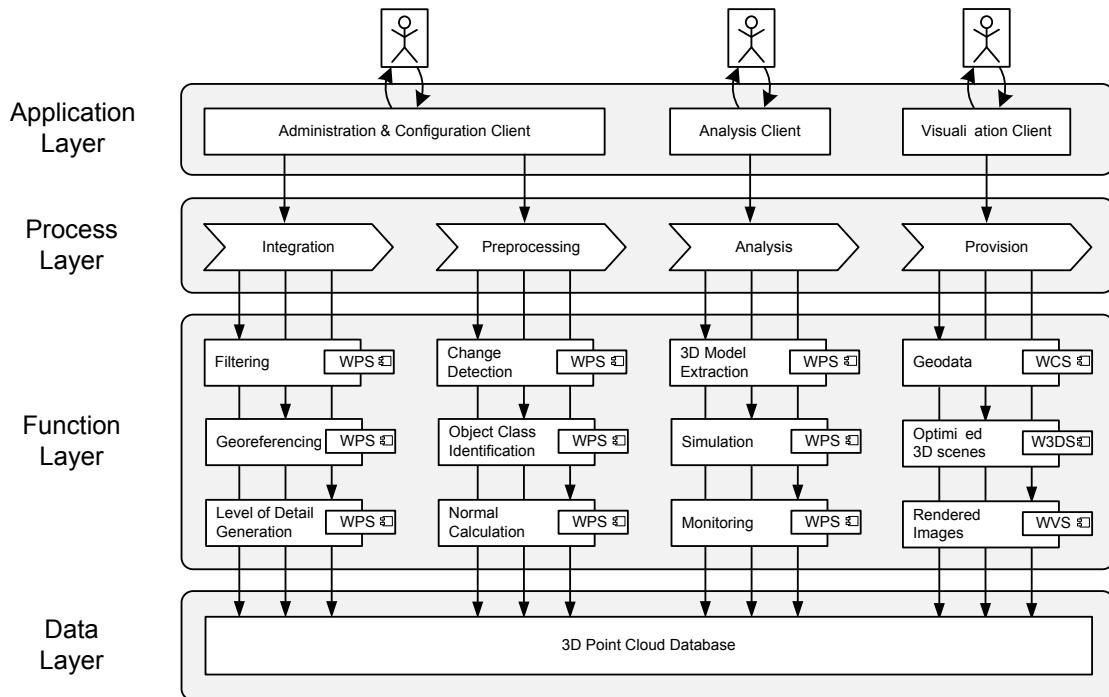


Figure 1: Service-oriented infrastructure allowing for the management of dense, spatial overlapping, and multitemporal 3D point clouds

3D point clouds as a universal type of geographic base data, we propose the use of a specialized software and geodata infrastructure that *manages* 3D point clouds. That means, it facilitates the integration of heterogeneous point data from various data sources, it updates, processes, and analyses the managed point data and it allows for the provision of that data based on different attributes (e.g., spatial, temporal, or semantic information).

To ease the integration of such an infrastructure into existing workflows and process chains, its functionality should be made available by standardized interfaces. To ensure high performance rates both for the processing and the provision of the point data, it should be organized in spatial data structures, that allow for an efficient selection of points based on object classes as well as spatial or temporal attributes. In addition, point-based processing tasks typically benefit from massive parallelization; therefore distributed computing concepts—combined with a distributed storage of the data—should be utilized. In summary, the following requirements need to be matched:

- Integration of heterogeneous 3D point clouds (e.g., airborne, mobile, or terrestrial) that may be specified in different formats or georeference systems into a homogenous spatial data model.

- Distributed storage of the managed point data.
- Distributed, scalable, adaptive and selective updating, processing and analysis of existing point data (e.g., computation of point specific object classes such as vegetation, building, terrain).
- Efficient provision (e.g., interactive visualization) of point data based on semantic (e.g., certain object classes), temporal (e.g., certain acquisition periods) or spatial attributes.
- Integration of the infrastructure into workflows of existing geoinformation systems and applications by means of standardized interfaces.

An infrastructure that matches these requirements is the one depicted in Figure 1. That *service-oriented* infrastructure provides several functions to integrate, analyze and provide point-based geodata. All functionality can be accessed via standardized web services such as *Web Processing Services (WPS)* [12], *Web Coverage Services (WCS)*, *Web 3D Services (W3DS)*, or *Web View Services (WVS)*; therefore it can be easily combined with existing workflows and processing chains. Simultaneously, web services may also be integrated *by* the given infrastructure to support certain processing tasks (e.g., hybrid computation of object classes based on the point data and additional geodata such as infrastructure maps).

The different services provided by the infrastructure can be assigned to the following categories:

- **Data integration.** Services allowing for the continuous updating of the managed data. This includes the cleaning newly acquired 3D point clouds as well as the transformation of point coordinates into a unified georeference system.
- **Data preprocessing.** Services computing basic attributes stored a per-point basis that are used by several analyses or visualization approaches (e.g., color, object class or topologic information) and therefore should only be computed once.
- **Data analysis.** Complex analyses such as noise and flooding simulations or the updating of 3D city models based on 3D point clouds. Typically, these tasks require additional information (i.e., apart from a point's spatial position) that are either computed during the preprocessing or extracted from additional geodata.
- **Data provision.** Services allowing for the export of the managed data or analysis results respectively, either directly in a standardized geodata format (i.e., via a WCS) or in a format optimized for the visualization of the data (i.e., via a W3DS or WVS).

In the following, these categories are discussed in more detail.

2.1 Data integration

The integration of heterogeneous 3D point clouds into a homogenous data model is essential both for the overall performance of the infrastructure as well as its compatibility with existing workflows. Compatibility can be ensured by supporting the import and export of standardized geodata formats (e.g., LAS, GML, PLY), especially of those following OGC standards or the INSPIRE directive. To improve the overall performance, a data model should be chosen that allows for the efficient selection of points based on different attributes, especially on a point's spatial position. Therefore, *spatial data structures*, such as quadtrees, octrees, or kd-trees, should be used to organize the point data [9]. For many analyses and visualization approaches, only a representative subset of the data for a given area is needed. Thus, the overall performance may be improved further by using so-called level-of-detail data structures, i.e., specialized spatial data structures storing different subsets of the data for each area, thus allowing for the efficient provision of variously detailed representations of the data [5].

Apart from the selection of points based on different attributes, the spatial data structure also needs to allow for an efficient integration of newly acquired and updating of existing point data. Kd-trees subdivide the given space into multiple parts of varying size based on the given data. This proves to be disadvantageous when the data basis is constantly updated since the integration of additional data typically initiates a rebuilding of the whole structure. Octrees and quadtrees on the other hand, conduct a three- or two-dimensional subdivision of the space into *equally* large parts. The three-dimensional subdivision applied by octrees is beneficial for *terrestrial* laser scans as the points in that case typically are spread uniformly across the space. If, by contrast, a 3D point cloud has been acquired by means of *aerial* laser scanning, large parts of the space typically remain empty, resulting in unbalanced octrees that are inefficient to traverse. Thus, the use of quadtrees would be favorable in general for airborne laser scans. To support the efficient integration of *both*, terrestrial and airborne laser scans, a combination of octrees and quadtrees should be considered.

2.2 Data preprocessing & Data analysis

Complex point-based analysis tasks typically require additional information about the respective area aside from each point's spatial position. Some information (e.g., two-dimensional land use information) can be extracted directly from additional geodata, whereas other information has to be calculated on a per-point basis.

Object classes, for example, can be determined by computing and weighting different features (e.g., normal distribution or surface variation) describing the topology

of a point's proximity [15]. By comparing newly acquired 3D point clouds with the point data already managed by the infrastructure, changes that occurred in-between these different acquisitions (e.g., newly constructed buildings) can be identified [10]. That *change detection* is an integral part of the described infrastructure as it allows for the selective updating of those parts of the data that have been subjected to changes lately.

While calculations like the change detection or the identification of object classes are fundamental for a broad range of complex analyses, they are usually time consuming. Therefore, calculating these kinds of information only once for parts of the data that have been newly integrated or updated and reusing it subsequently can speed up the performance of those analyses notably.

Whereas preprocessing tasks always generate per-point attributes that are integrated into the data basis, subsequent analysis tasks may also generate different types of geodata such as mesh-based 3D models or cadastre data. In conclusion, the described infrastructure speeds up the conduction of common analysis tasks, because (1) required subsets of the data can be selected efficiently due to the underlying spatial data model and (2) crucial information (e.g., object classes) is precalculated in a preprocessing phase.

2.3 Data provision

To integrate the infrastructure into existing workflows, the managed point data as well as analysis results have to be accessible via various web services. If a component requires full access to the requested data, e.g., to conduct subsequent analyses on it, a *Web Coverage Service (WCS)* [2] should be implemented, that allows for the selection of points based on various attributes (e.g., spatial position, temporal or object class information). Since all the details for a given area are rarely needed, the WCS should support the selection of different levels of detail as well. In addition, standardized protocols (e.g., JPIP) and data formats (e.g., GMLJP2) optimized for the efficient transfer of data in networks [13] should be used to ensure interoperability with existing web coverage services and to speed up transfer rates for point data.

If the point data is only requested for visualization purposes, the usage of a WCS is not always the best option since a lot of application logic is required on the client that is responsible for both the selection and the configuration (e.g., color, size, and primitive type of each point) of the visualized data. As opposed to that so-called *thick client* approach, a *Web 3D Service (W3DS)* [11] selects and configures the requested data on the server before transferring it to the client in a format optimized for rendering purposes. However, such a *medium client* approach still requires the client to render the transferred point data itself, thus limiting its applicability to clients featuring a certain minimal hardware requirements (e.g., a stand-alone GPU).

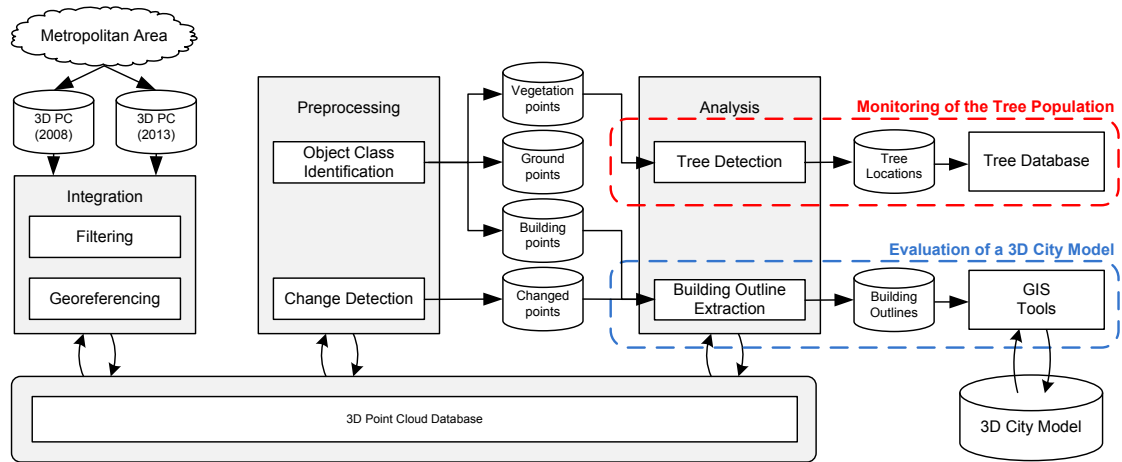


Figure 2: Prototypical implementation of the service-oriented infrastructure, configured for two real-world scenarios

To allow for the visualization of arbitrarily large 3D point clouds even on mobile devices, a *Web View Service (WVS)* should be used instead, that renders the data directly on the server and only transfers the rendered images to the client. This *thin client* approach notably reduces the minimal hardware requirements on client side. Since only the rendered images but not the visualized data itself have to be transferred, the data traffic is independent from the amount of data that is visualized. Therefore, when visualizing massive amounts of data, the data traffic of a WVS is typically lower than the one of a W₃DS [4].

3 Evaluation

A prototypical implementation of the proposed infrastructure was evaluated based on two real-world scenarios. The first scenario was the evaluation of a 3D city model based on two aerial laser scans of a metropolitan area. As a second scenario, the same data basis was used to monitor changes in the tree population in the respective area.

The data basis consisted of two aerial laser scans captured in 2008 and 2013, respectively. The former one featured 7-10 points/m², 5 billion points overall, and 75 GB of data; the latter one featured 100 points/m², 80 billion points overall, and 1200 GB of data.

As depicted in Figure 2, the scenarios were based on a change detection as well as an object class identification which were conducted as part of the preprocessing phase. For the first scenario, all building points with a certain degree of change (e. g.,

more than two meter distance between the old and the new laser scan) were selected and used to derive building outlines (i.e. shape files) describing all newly constructed or modified buildings in the given area. Subsequently, the exported building outlines were used in combination with external tools to manually update the modified parts of the 3D city model. Since the modified parts were already identified by our infrastructure, only a small subset (i.e., approximately two percent) of the 3D city model had to be reevaluated. For the second scenario, all points representing vegetation were analyzed to derive the location of individual trees as well as several attributes (e.g., height, volume) characterizing these trees in more detail. The results of that analysis were exported and—in conjunction with external tools—used to evaluate an existing tree cadastre.

The processing tasks were conducted in parallel on a total of *six* computers—each of which featuring an Intel Xeon CPU with 2.66 GHz, 12 GB main memory, and a NVIDIA GeForce GTX 660 with 2 GB device memory. In Table 1, the data throughput for the most relevant (pre-)processing tasks is specified. Data throughput for the integration and the provision of the data is defined by the overall network and memory bandwidth and was at around 80 MB/second.

Table 1: Data throughput for the most relevant processing tasks for the given scenarios in billion points/hour.

Object Classification	Change Detection	Building Extraction	Tree Detection
0.33 B pts/hour	5.00 B pts/hour	0.42 B pts/hour	6.51 B pts/hour

The use of the presented service-oriented architecture clearly proved to be beneficial for the evaluated scenarios. For once, the distributed storage and processing of the point data allowed for high data throughput. Secondly, the spatial data structure used to organize the point data allowed for the efficient selection of those points relevant to the respective processing task. Finally, a WVS allowed for the interactive exploration of the managed point data as well as analysis results on a broad range of hardware devices (see Figure 3).

4 Conclusion and Outlook

In this report concepts for a service-oriented infrastructure were discussed, that allows for (1) the continuous integration of heterogeneous 3D point clouds into a homogeneous spatial data model, (2) the conduction of various complex analyses



Figure 3: Aerial laser scan of a metropolitan area. Points are colored based on color information extracted from aerial images (left) or based on their respective object classes

directly on the point data, and (3) the efficient provision of that data either in standardized geodata formats or in a format optimized for the visualization of the data. Since additional per-point attributes that are required by multiple analyses are pre-calculated for newly integrated or recently modified data, the performance of these analyses could be sped up notably. In addition, the data structure used to organize the point data allows not only for the efficient selection of points based on semantic, temporal, or spatial attributes but also supports the selection of different levels of detail.

The presented infrastructure is especially suited for the management of 3D point clouds that are dense, spatial overlapping, and multitemporal. Thus, it demonstrates an alternative approach to the workflow of traditional GIS, since it uses 3D point clouds not as a mere basis to create mesh-based 3D models, but facilitates the conduction of various common analysis tasks directly on the point data itself. To ease the integration of the infrastructure into existing workflows and process chains, established web services are used for the external communication.

The practicability of the discussed concepts was evaluated based on a first, prototypical implementation and two real-world scenarios. Future steps include the integration of 3D point clouds from different data sources (e.g., aerial, terrestrial, and mobile) into a homogeneous data model, since the infrastructure so far has only

been tested for aerial laser scans. In addition, novel, more efficient concepts should be evaluated for the compression and transfer of point data in networks.

5 Acknowledgements

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

References

- [1] M. Arikan, M. Schwärzler, S. Flöry, M. Wimmer, and S. Maierhofer. "O-snap: Optimization-based Snapping for Modeling Architecture". In: *ACM Trans. Graph.* 32.1 (2013), 6:1–6:15.
- [2] P. Baumann. *Web Coverage Service (WCS) Interface Standard, Version 2.0.1*. Open Open Geospatial Consortium Inc. 2012.
- [3] J. Coutinho-Rodrigues, A. Simão, and C. H. Antunes. "A GIS-based multicriteria spatial decision support system for planning urban infrastructures". In: *Decision Support Systems* 51.3 (2011), pages 720–726.
- [4] J. Döllner, B. Hagedorn, and J. Klimke. "Server-based Rendering of Large 3D Scenes for Mobile Devices Using G-buffer Cube Maps". In: *17th International Conference on 3D Web Technology*. 2012, pages 97–100.
- [5] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. "An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees". In: *The Visual Computer* 29.1 (2013), pages 69–83.
- [6] F. Lafarge and C. Mallet. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *International journal of computer vision* 99.1 (2012), pages 69–85.
- [7] H. Lin, J. Gao, Y. Zhou, G. Lu, M. Ye, C. Zhang, L. Liu, and R. Yang. "Semantic decomposition and reconstruction of residential scenes from LiDAR data." In: *ACM Trans. Graph.* 32.4 (2013), page 66.
- [8] S. Nebiker, S. Bleisch, and M. Christen. "Rich point clouds in virtual globes—A new paradigm in city modeling?" In: *Computers, Environment and Urban Systems* 34.6 (2010), pages 508–517.

- [9] R. Richter and J. Döllner. “Concepts and techniques for integration, analysis and visualization of massive 3D point clouds”. In: *Computers, Environment and Urban Systems* 45 (2013), pages 114–124.
- [10] R. Richter, J. E. Kyprianidis, and J. Döllner. “Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds”. In: *Transactions in GIS* 17.5 (2012), pages 724–741.
- [11] A. Schilling and T. H. Kolbe. *Draft for Candidate OpenGIS Web 3D Service Interface Standard, Version 0.4.0*. Open Open Geospatial Consortium Inc. 2010.
- [12] P. Schut. *OpenGIS Web Processing Service, Version 1.0.0*. Open Open Geospatial Consortium Inc. 2007.
- [13] Y. Shao, D. Liping, L. Kang, and W. Han. “Using Open Standards to Integrate LIDAR and Geoprocessing”. In: *LiDAR Magazine* 10/2013 3.5 (2013), pages 66–69.
- [14] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. “Finding paths through the worlds photos”. In: *ACM Transactions on Graphics (TOG)*. Volume 27. 3. 2008, pages 11–21.
- [15] Q.-Y. Zhou and U. Neumann. “Fast and extensible building modeling from airborne LiDAR data”. In: *16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2008, pages 1–8.

Consciousness in Artificial Agents – A Theory

Fahad Khalid

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

The phenomenon of consciousness has consistently proven a bottleneck in simulating human-like behavior in artificial agents. On the one side, consciousness has been studied in neuroscience, with the intention of developing an understanding of, and the scientific basis for the phenomenon. On the other hand, research in artificial intelligence has focused on simulating the phenomenon in artificial agents, by developing algorithms from first principles. In this report, a theory is developed, which reveals insights into the potentially beneficial interplay of neuroscience and artificial intelligence; a novel approach to studying consciousness.

1 Introduction

The theory of artificial consciousness presented here is founded on the following premise: If we consider consciousness as a set of functions—such as awareness, emotions, etc.—then an artificial agent can be endowed with the same set of functions as a human being. Such an agent would appear conscious to us in the same way other humans do. The *functions of consciousness* [10] form the basis of comparison between the theories of consciousness developed in neuroscience, and the theory of artificial consciousness developed in this paper (see Figure 1 for an overview).

In addition to the discussion of functions as mechanisms, we describe the architectural details of an artificial implementation of these functions. In doing so, we elaborate on the hierarchical and modular nature of such an architecture. Moreover, we comment on the interconnectedness of the different components/modules involved in the various computations.

It is also our hypothesis that for a system to be able to generate conscious behavior, it must be able to interact with its environment. We consider conscious agents, both natural and artificial, as *reactive systems*. The mechanisms to perceive stimuli and respond to them are essential for any reactive system. Therefore, we argue that any conscious artificial agent capable of mimicking human consciousness must be able to interact with the environment just as humans do. However, this assumption raises an issue: the algorithms available today for stimulus-response mechanisms are neither accurate enough, nor computationally feasible for mimicking human abilities [12].

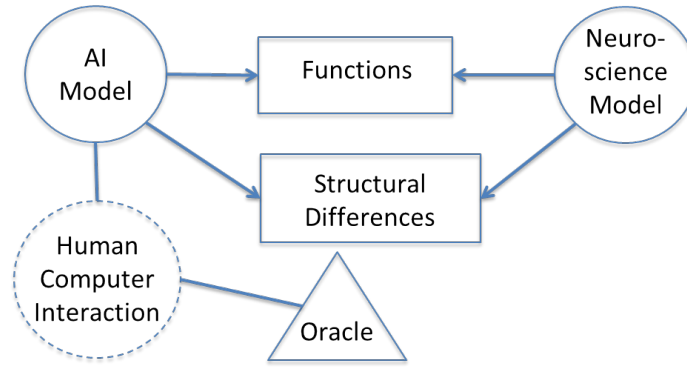


Figure 1: The figure highlights *functions* and *structural elements* as points of connection between theories of consciousness in AI and neuroscience. Moreover, *human computer interaction* is presented as an *Oracle*, to which the AI theory refers.

As the focus of this paper is on functions of consciousness rather than stimulus-response mechanisms, and since we do not aim to address the origin of consciousness, we assume that the entire agent-environment interaction functionality is an Oracle. Consequently, we can proceed with suggestions on how to implement consciousness in an artificial agent without having to delve into the details of interaction mechanisms.

It is also important to note that given today’s processor architectures, any reference to software based artificial agents implicitly pertains to software designed for the Von Neumann machine architecture [11]. This is fundamentally different from the neural network architecture utilized by the human brain. It is well known [2] that both Von Neumann machines and neural networks are capable of implementing the basic logic operations required for universal computation. However, biological neural networks appear to be superior in performance for certain specific tasks, e.g., vision. Such tasks fall under the category of *interaction functions* comprising the Oracle. Therefore, we ignore these. For the functions of consciousness though, there is no empirical evidence suggesting that the neural network architecture is superior. The one aspect that may benefit from a neural network machinery is *memory*. Our hope is that with the progress in memory and microprocessor technology, this will become less of a barrier to implementing human-like memory function.

In the following Section, we illustrate the mechanisms and processes involved in artificial consciousness. We use *awareness* and *emotions* as illustrative functions of consciousness.

Artificial Agents—Introducing John and Doe

In order to illustrate the mechanisms of artificial consciousness, we present examples of two hypothetical artificial agents named *John* and *Doe*. These are both mobile

robots that look and behave exactly like humans, i.e., a human cannot identify John and Doe as robots based solely on physical appearance and behavior. Therefore, our descriptions of these agents are highly anthropomorphized.

Note: This report is an excerpt of this author's contribution to the "*How Artificial Intelligence Can Inform Neuroscience: A Recipe for Conscious Machines?*"¹ project report, which resulted from an interdisciplinary collaboration between researchers from the 2014 Santa Fe Institute Complex Systems Summer School. The figures included in this report were prepared by Claire Lagesse².

2 Awareness in Artificial Agents

A high level software-based model of awareness involves the following components:

- Sensor input and fusion
- Sensor input prioritization
- Attaching the awareness property to the highest priority stimulus

We explain these processes in the following Subsections using hypothetical scenarios centered on the artificial agents introduced earlier.

2.1 The Process by way of Illustration

Let us consider a scenario where John is standing at the platform of a train station, waiting for the train. John is facing the track, looking at the captivating scenery in the background. At one point, he hears the train coming from the right. He instantly turns his head to the right to see the train. By looking at the oncoming train, he has visual confirmation of his interpretation of the auditory stimulus.

In the above mentioned scenario, as John sees the train, he also hears the sound that gets louder as the train comes closer. John's brain is concurrently receiving two stimuli; visual and auditory. The artificial brain fuses the two stimuli to refine the perception of the object. This is because in many cases, one stimulus is not enough to assess the situation completely. E.g. hearing the sound of the train does not guarantee that it is the train for which John is waiting, and not a different train passing by the station. Similarly, if the train is coming in over a bend, John might not be able to see it up until a certain point, but might still be able to hear the sound. Therefore, sensor

¹http://santafe.edu/media/cms_page_media/598/CSSS_2014_Consciousness.pdf, accessed December 16, 2014.

²Claire Lagesse is one of the co-authors of the SFI CSSS 2014 project report.

fusion is an important process to increase the amount of information available about a certain situation.

Before John hears the sound of the train, he can hear many other sounds as well, e.g., the birds chirping, some other passengers talking to each other, etc. As soon he hears the sound of the train, it captures his attention, and he suddenly looks right. Most of the other sounds, even though being perceived, are being assigned low priorities. The sound of the train, however, is assigned a high priority, which enables John’s artificial brain to filter out the rest and focus on that one sound. Prioritization of sensor input plays an important part in the process of attention.

The attention focuses the visual system on acquiring visual information about the train. This results in an increase in the amount of information, and therefore an increase in certainty, about the belief that the train is coming. The sound tells John that a train is coming; the visual perception further tells him that his train is coming. At this point John is *aware* of the fact that his train is coming. Awareness in this case is an additional property that is assigned to an object once enough information has been acquired [5]. The amount of information is increased or acquired using the process of attention.

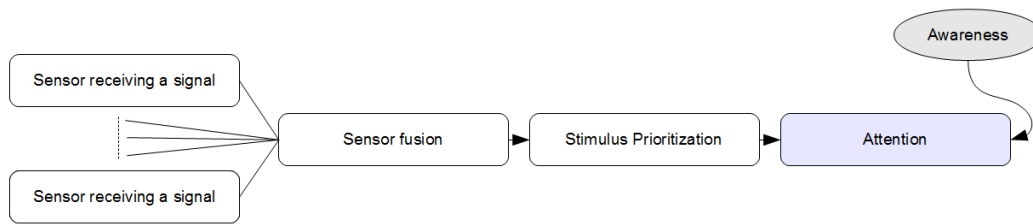


Figure 2: The process of generation of awareness is depicted. From left, a stimulus is received by one or more sensors and subsequently *fused* and *prioritized*. *Attention* is given to the highest priority stimulus, to which *awareness* is then attached.

2.2 Sensor Fusion

Sensor fusion is a widely known and commonly used method in robotics. There are several algorithms available for implementing sensor fusion, one of which is based on the Kalman Filter [8]. The important point here is that the algorithms for sensor fusion are available, and we assume that the software module responsible for performing sensor fusion in our agents can be implemented using existing technology.

2.3 Sensor Input Prioritization

The details of an algorithm for the sensor input prioritization would be too complicated to include in this paper. Nevertheless, the basic concept is presented here. Let us assume that the auditory system (a software module) receives several different inputs. These are:

- Sound of birds chirping
- Sound of music playing
- Sound of people talking amongst a group (of which the observer is not a member)
- Sound of an oncoming train

Each sound is prioritized based on two major properties.

- Loudness
- Type

We assume that the agent is already familiar with all the different sounds. Therefore, when a sound enters the artificial brain, it is compared against a pattern that is stored in the memory, and immediately recognized as belonging to a certain category³, such as, the sound of birds chirping.

Given a certain state of the artificial mind, e.g., anticipation of the arrival of the train, a certain ‘sound pattern’ has a high priority. This is based on the *type* of sound. Moreover, the *loudness* can always result in a high priority for any sound if the loudness is above a certain threshold value. E.g., even while John is anticipating the arrival of the train, Doe shouting at him would capture his attention.

This concept can be distilled down to a very simple algorithm.

```

1 Loop: forever
2     Loop: For each auditory stimulus
3         Compute Loudness
4         if Loudness > threshold
5             focus attention on the stimulus
6             break Loop
7         Compute Type
8         if Type in Types of Interest
9             focus attention on the stimulus

```

³The machine learning savvy reader will notice that the process is analogous to *pattern recognition* or *classification learning*.

```
10         break Loop
11     End Loop
12 End Loop
```

Please note that even though the computation of loudness is more or less similar in most situations, the computation of the type of interest can vary significantly depending on the given situation, i.e., the state of the artificial mind. E.g., when one is anxiously waiting for a phone call, the sound of phone ringing has a very high priority. However, when one is in a meeting, the sound of phone ringing has a lower general priority, which is further dependent on the loudness.

Note: Whether an algorithm is formulated in terms of if-then-else clauses, or edge weighted directed graphs (such as neural networks), is merely a matter of representation. Both can generate equivalent results [2].

2.4 Attaching the Awareness Property to a Stimulus

Based on idea of the relationship between awareness and attention presented in the theory of consciousness by Graziano and Kastner [5], once the prioritization module assigns the highest priority to a stimulus, the *awareness module* enables the *Is Aware* property for that stimulus. Therefore, the awareness module can be thought of as an independent module that keeps track of attention and enables the awareness property for the stimulus to which the mind is paying attention.

The awareness module can assign awareness object by object in a multi-node chain of objects. This is useful for situations such as: Doe is aware of the fact that John is aware of the oncoming train, which represents a 2–deep awareness chain. Moreover, the same awareness module/function can be used to generate self-awareness. In this case, the awareness property is attached the object *self*. This is similar to Graziano and Kastner’s idea that self-awareness stems from the ability of the brain to reuse the perceptual social machinery for awareness.

Note: For the sake of simplicity, we have ignored the process of object recognition. Algorithms for object recognition exist [1, 3, 7, 14], although these do not perform nearly as well as the human brain.

3 Emotions and Internal Loops

3.1 Emotions as Functions

Let us consider a scenario where John is taking a walk, and he arrives at a Koi pond. He looks at the fish, which triggers a slight change (towards a calming feeling) in

his state of mind. What is the process that takes place between *seeing the fish* and *feeling calmer*?

In the previous Section, we discussed the process that takes place between *sensing the stimulus* and *generating attention and awareness*. In the above mentioned scenario, we are adding another stage to the process. Once John's attention is on the Koi, this information is forwarded to the *emotion module*. This module implements the *emotional logic*, which is the algorithm used to determine the emotion triggered by a given stimulus. Once a specific emotion (or a set of emotions) has been determined, in addition to possible physiological changes such as a change in facial expression, a change is triggered in the state of mind. This new state of mind (in John's case) is perceived as *feeling calmer*.

Each emotion can be viewed as a *function*, which is called when a certain set of properties/stimuli is evaluated to be true. The function itself implements an algorithm that constitutes the set of actions to be taken when the function is called. One such action would be the change in facial expression, such as smiling, when an agent perceives happiness. Please note that algorithms circumscribed by these functions change over time (these are governed by an adaptive/learning algorithm), depending on a given state of mind, and also on a long-term or permanent change (possibly caused by damage to the artificial brain) in how a certain stimulus affects the agent.

3.2 Emotional Logic

The question we will try to answer in this Section is, "*How does a stimulus trigger an emotion in an artificial agent?*"

Once a stimulus has been assigned a high enough priority, information about the stimulus (the associated data structure; the *stimulus object*) is forwarded to the emotion module. This module reads certain properties of the stimulus object, e.g., in the Koi fish example, the *beauty associated with the scene*. The *property scan* function, i.e., the function that reads the properties, is followed by a call to the emotional logic algorithm. This algorithm analyzes list of properties received from the property scan function. In our example, beauty is considered a positive property. In the absence of any negative properties, the algorithm determines the positive emotion that should be triggered by this property. In this simple case, the *feeling calm* emotion is triggered. A highly simplified version of the algorithm is depicted in Figure 3.

In order to represent a more realistic multi-emotional state, a more elaborate decision process would have to be taken into account. The binary branching between positive and negative perception of a property would not be enough. This too is not an issue, since we know from machine learning that *decision trees* [4] can be used to represent very complex decision processes.

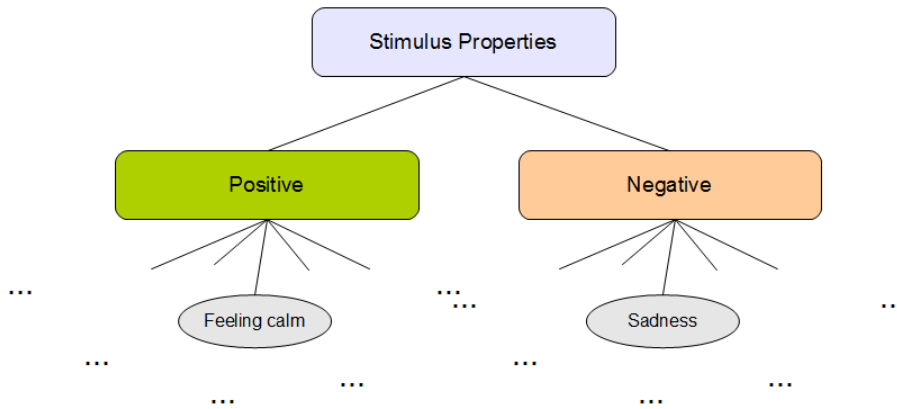


Figure 3: Simple *emotional logic*: Based on its properties, a stimulus is classified as either positive or negative. Then, the particular emotion to be triggered is determined.

Furthermore, the action associated with an emotion function can also be quite elaborate; in addition to a change in the state of mind, multiple actuators can be activated. Also, the intensity or significance of the stimulus can affect the intensity and/or nature of the response. Please note that all this boils down to a complex set of rules, which can be represented as a set of if-then-else clauses. Other representations such as trees, networks, etc. can be used as well [17].

3.3 Internal Loops

The above discussion raises the question, “*How can an artificial agent experience emotions in a dream? How does an artificial agent feel emotion in the absence of external stimuli?*”

The experiences of day to day life are stored in the agent’s memory in the form of patterns. Once the agent is in a dream state, the sensory system can be temporarily replaced by memory, i.e., the stimuli are loaded from memory rather than being received from the sensory system. Once a stimulus is present, it can be processed using the algorithms described earlier, regardless of whether the stimulus was received from the environment, or loaded from memory. A similar mechanism is applicable to how actuators can be perceived as interacting with the environment during dreams while the body is completely at rest. Similarly, an agent’s own thoughts can trigger emotions without any external stimuli.

The above stated explanations imply that there are internal connections between the memory and the other functional modules mentioned earlier. It is these connec-

tions that make it possible for the mind to bypass the sensor-actuator modules in the dream states. The architecture is depicted in Figure 4.

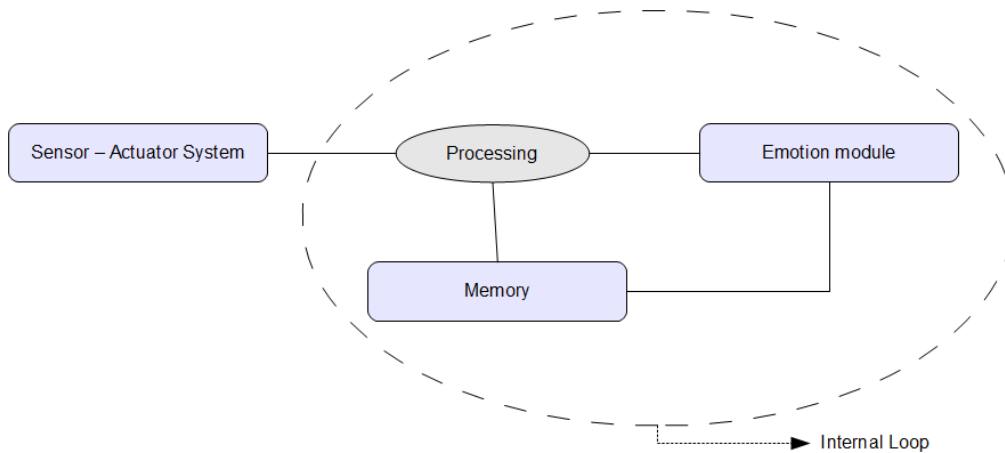


Figure 4: *Internal loop:* The loop constitutes loading a *stimulus pattern* from memory and processing it. During processing, the emotion module might be used, which might in turn load further stimuli from memory.

Nevertheless, it is important to note that the sensor-actuator system is a necessity for constructing experiences in the first place, which are later used by the mind to create dreams. If the agent never had the sensor-actuator system, it would not have experienced sensory input and actuator responses, and therefore would not have the necessary information in memory to construct dreams.

Please note that the same mechanism is used in situations when the agent is wide awake, but thinking. E.g., consider a scenario where John is sitting at his desk, developing a simulation in NetLogo [16]. Ideas are generated in his mind as he thinks about how to solve certain problems. In this case, the thinking process does not solely rely on the stimuli currently being perceived; much of the problem solving is done using the information already stored in memory (information retrieval followed by processing to arrive at ideas).

4 Hierarchical Architecture of an Artificial Brain

The concept of a hierarchical architecture is intrinsically related to the concept of *modularity*. Modular system design is an established practice in software engineering.

This makes it possible to independently evolve a module over time, without the propagation of side-effects to other modules. A common method of determining how to divide the code into modules is to look at the coupling relations between different functions. Tightly-coupled, highly interdependent, and functionally similar functions are often included into a single module. On the other hand, loosely-coupled, independent, and functionally dissimilar functions are kept in different modules. There are other more complex decisions that come into play when making these distinctions, but for brevity, only these simple cases are considered here.

There is evidence from neuroscience that certain functionally similar modules in the brain are located in close spatial proximity [5]. This indicates that modularity is a common structural property of complex decision making systems, be they artificial or natural.

The architectural model we present of an artificial brain is strongly influenced by the modularity principle employed in software engineering. In order to illustrate how this hierarchical and modular structure functions, we present the following illustration.

John is running late for a meeting with Doe this morning. He enters the train station, and is pacing toward the platform, so that he can catch the train in time. It is morning rush hour and the station is full of people walking to different platforms, purchasing tickets, etc. As John is rushing toward the platform, he suddenly hears the sound of a child talking; emanating in close proximity on his right hand side.

4.1 Sensor-Actuator Layer

The sound is a stimulus that is received at the bottom layer of the artificial brain hierarchy, called the *sensor-actuator layer*. This layer is responsible for interaction with the environment, and consists of *sensors* and *actuators*.

4.2 Attention and Awareness Layer

Once the stimulus (i.e., sound) is received by the sensor (John's ears), it is forwarded to the *Attention and Awareness layer*. This triggers the prioritization algorithm that immediately gives the stimulus a high priority. This results in John's attention being focused on the sound of the child talking.

4.3 Reactive Layer

Once John's attention is focused on the sound, it immediately results in him turning his head to the right to see the child. This happens because any stimulus object to which awareness is attached is sent onward to the *reactive layer*. The reactive layer is

adaptive in nature, which means that over time it can learn to react to stimuli that are not pre-programmed. *Turning the head toward the sound* is such a learned reaction.

As soon as the stimulus object is received by the reactive layer, the corresponding response is searched for in a map (similar to a hash map data structure), where the key is a stimulus pattern and the value is the corresponding action. The action is a function that can itself consist of a complex algorithm. In the case being discussed here, the action function sends a request to the sensor-actuator layer with an instruction to turn the head right. It also activates the vision system and sets its status to *actively looking for the child object*.

At this point, John sees the child. The vision system, which is a part of the sensor-actuator layer, receives the stimulus of the child, which is fused with the auditory stimulus, resulting in the strengthening of the belief of the agent that there is a child in close proximity. In addition, sensor fusion results in the computation of estimation of distance between John and the child. In this case, the distance is small enough that another function in the reactive layer is triggered. This function sends a *strafe left* message to the sensor-actuator layer. As a result, John suddenly strafes left.

The reactive system can trigger a function not just for the agent's own safety, but also for the safety of the other agents in the environment. The above mentioned reaction where John strafes left was learned through the social perceptual machinery of the artificial brain. The object of strafing left was to make sure that the child is not harmed.

Note: There can be reactions analogous to the knee-jerk response of the human nervous system. These are pre-programmed reactions or *instinctive reactions* [10] as opposed to *learned reactions*.

4.4 Decision Making Layer

The significance of causing a child harm is high enough that as the reactive system triggers the *strafe left* action, it also forwards the stimulus object to the emotion module, which is located in the *decision making layer*. The emotion module evaluates the stimulus object and decides to call the *fear* function. The fear function activates the *fear center* (analogous to *Amygdala* [9] in the human brain). The fear center broadcasts a special fear message to the sensory-motor layer, which increases the perceptiveness of the sensory system. This ensures that for a certain short period of time in the future, the agent is hyperalert.

The decision making layer is primarily responsible for: 1) evaluating stimuli (external or internal) that are not processed by the reactive layer, and 2) processing ideas generated inside the artificial brain. In addition to the emotion module, this layer comprises complex functions that implement algorithms for deliberating and assessing situations. A simple example is an agent walking through the station and

trying to find its way to the correct platform (assuming it is not familiar with the layout of the station building). At certain points, it is lost, in which case it decides whether to ask someone, or to turn and move in a different direction, etc. The agent may need to make several deliberative decisions in such a situation.

4.5 Reflective Layer

John managed to reach the platform in time, and is now seated inside the train. As the agent is in a resting position now, and there are no stimuli that require specific attention, the agent's brain turns attention to the *self object*. There is an *idle time* based algorithm; when the idle time, i.e., the amount of time without any significant stimulus, crosses a certain threshold, awareness is attached to the self object. This is when the control is passed on to the *reflective layer*.

As John settles down in the seat, he starts to reflect on the incident with the child. He thinks about the possible outcomes had he not reflexively strafed left in time. This negative feedback results in the thinking process looking back one step in the history of the decision process. John starts to realize that the situation could have been completely avoided had he not been walking too fast. This leads him to think that the origin of this Markov chain is in *not waking up early enough*.

The reflective layer constitutes functions that evaluate decisions made earlier. This is similar to the function of *self-reflection* in humans. The purpose of the functions in this layer is to adapt the functions in the other layers according to previous outcomes. This is similar to the feedback loop used in machine learning algorithms [6] and adaptive filters [13]. In certain cases, the feedback results in an update in the functions of the decision layer. In other cases, if the reflective layer determines that the situation was severe enough that the response should be immediate for a similar situation in the future, the corresponding functions of the reactive layer are updated. This results in the creation of a learned reaction.

Please note that the reflective layer can directly influence the function of most of the decision making layer, however, it has only limited access to the emotion module. It is not possible for the reflective layer to directly alter the emotion functions⁴. If this were not the case, unlike humans, the agent would be able to directly manipulate the influence of stimuli on its emotional state. Since we intend for the artificial agent to mimic human consciousness, we have proposed an implementation that corresponds to the working of emotions in humans.

⁴This can be implemented by having only indirect connectivity between the emotion module and the reflective layer via other modules and layers.

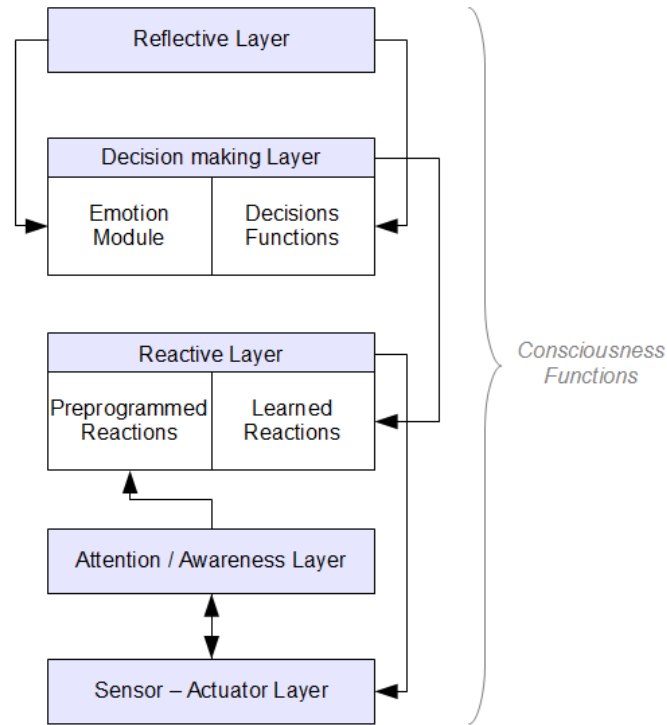


Figure 5: Hierarchical architecture of the artificial brain. In order to generate conscious behavior, functions in the various layers must be integrated.

5 The Artificial Mind as a Highly Interconnected State Machine

Whenever we use the term *state of mind*, we (perhaps unintentionally) refer to the mind as a state machine. Given the myriad processes concurrently being performed by the brain at a certain point in time, it is not feasible to enumerate all possible states. Moreover, different states of the mind can be combinations of other different states, which renders the state space practically unexplorable. *How then can such a system be implemented in an artificial agent?*

In this Section we provide a speculative theory of the interconnection architecture that would be required to create an artificial agent with a complex brain, i.e., one with states of mind similar to that of a human brain.

The solution lies in a highly interconnected network of functions, implemented in a hierarchical structure. High interconnectedness is required so that many different functions can call each other without going through an intermediary. Also, depending on the current state and any given set of stimuli, it should be possible to represent

the next state as a graph of interacting functions. This can be represented by an edge-weighted graph (possibly a hypergraph) with an edge set of high cardinality, and edge weights representing connection strength. For such an architecture, a *state* would constitute the set of active paths through the graph. The *experience* of the state could then be the particular set of edge weights – assuming that edge weights can change in response to learning.

The above mentioned description of interconnectedness highlights a possible point of comparison with the Integrated Information Theory of consciousness [15]. Perhaps it is the case that when we try to implement consciousness functions, the nature of computation of these functions naturally requires a highly interconnected structure. That is, even if we approach artificial consciousness from a functional rather than structural point of view, a highly interconnected structure emerges as a result of the necessary interactions between functions and layers.

6 Limitations and Future Work

In this paper, we have presented mechanisms and architectures that serve as a road map for implementing consciousness in artificial agents, and inform the theories of consciousness in neuroscience. Our approach, nevertheless, has been to keep our discussion at a high level of abstraction.

We realize that in order to uncover major obstacles in the implementation of conscious agents, we need to design and analyze algorithms that uncover low level details, and extend the architecture to various systems involved in conscious behavior that have not been covered in this paper. Therefore, one direction for future research would be to delve deeper into the design process, and move towards implementation of software agents to provide proof of concept.

There are various possible points of connection between the Integrated Information Theory (IIT) [15] and our approach to artificial consciousness. These also lead to open questions that are worth pursuing. A high priority task for us would be to use artificial neural networks as toy models for doing an in depth empirical analysis of the structures introduced in IIT, and their relationship to information integration. It would be interesting to see how this compares to the concept of modularity, which is a cornerstone of software systems engineering.

References

- [1] M. Bennamoun and G. J. Mamic. *Object Recognition: Fundamentals and Case Studies*. New York, NY, USA: Springer-Verlag New York, Inc., 2002.

- [2] L. Fausett. *Fundamentals of Neural Networks-Architectures, Algorithms, and Applications*. Prentice Hall, 1994.
- [3] S. Gong, S. J. McKenna, and A. Psarrou. *Dynamic Vision: From images to face recognition*. London: Imperial College Press, 2000.
- [4] K. Grabczewski. *Meta-Learning in Decision Tree Induction (Studies in Computational Intelligence)*. Springer, 2014.
- [5] M. S. A. Graziano and S. Kastner. "Human consciousness and its relationship to social neuroscience: a novel hypothesis". In: *Cognitive neuroscience 2.2* (2011), pages 98–113.
- [6] S. Haykin. *Neural networks and learning machines*. Volume 3. Pearson Education Upper Saddle River, 2009.
- [7] M.-K. Hu. "Visual Pattern Recognition by Moment Invariants". In: *IRE Transactions on Information Theory IT-8* (1962), pages 179–187.
- [8] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME-Journal of Basic Engineering 82*.Series D (1960), pages 35–45.
- [9] E. Kandel, J. Schwartz, and T. Jessell. *Principles of neural science*. Volume 4. McGraw-Hill New York, 2000.
- [10] M. Minsky. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. SIMON & SCHUSTER, 2007.
- [11] J. von Neumann. *First Draft of a Report on the EDVAC*. Technical report. University of Pennsylvania, 1945.
- [12] R. W. Proctor and K.-P. L. Vu. *Stimulus-Response Compatibility Principles: Data, Theory, and Application*. CRC Press, 2006.
- [13] A. H. Sayed. *Adaptive filters*. John Wiley & Sons, 2008.
- [14] B. Schiele and J. Crowley. "Probabilistic object recognition using multidimensional receptive field histograms". In: *Proceedings of the 13th International Conference on Pattern Recognition (ICPR 96)*. Volume B. 1996, pages 50–54.
- [15] G. Tononi. "Consciousness as integrated information: a provisional manifesto". In: *The Biological Bulletin 215.3* (2008), pages 216–242.
- [16] U. Wilensky. "NetLogo. <http://ccl.northwestern.edu/netlogo/>". In: *Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.* (1999).
- [17] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

Implementing an Object-Constraint Extension Without VM Support

Tim Felgentreff

Software Architecture Group
Hasso-Plattner-Institut

Tim.Felgentreff@hpi.uni-potsdam.de

Constraints provide a useful technique for ensuring that desired properties hold in an application. As a result, they have been used in a wide range of applications, including graphical layout, simulation, scheduling, and problem-solving. We have previously proposed a design, Babelsberg, for a family of object-constraint languages that cleanly integrates constraints with the underlying language in a way that respects encapsulation and standard object-oriented programming techniques. However, our work on Babelsberg has relied on modifying the underlying virtual machine, but that is not an option for web-based applications, which have become increasingly prominent.

We thus present an approach to implementing Babelsberg without virtual machine support, along with an implementation as a JavaScript extension. We demonstrate the resulting language, Babelsberg/JS, on a number of applications and provide performance measurements. Programs without constraints in Babelsberg/JS run at the same speed as pure JavaScript versions, while programs that do have constraints can still be run efficiently. Our design and implementation also incorporate incremental re-solving to support interaction, as well as a novel cooperating solvers architecture that allows multiple solvers to work together to solve more difficult problems.

1 Introduction

Constraints are relations among objects that should hold. This could be that all parts in an electrical circuit simulation obey the laws of physics, that the rows in a Sudoku include each digit from 0 to 9, or that a streamed video plays smoothly in the presence of changing CPU and network load. We also want to support interactive use of constraints, for example, continuously re-satisfying a set of layout constraints on screen widgets as they are moved with the mouse. In addition, it is useful to extend the constraint formalism to allow soft constraints as well as required ones, where the system should try to satisfy the soft constraints if possible, but it is not an error if they cannot be satisfied. For example, we might have a soft constraint for video quality that we are willing to relax if necessary, given the current network

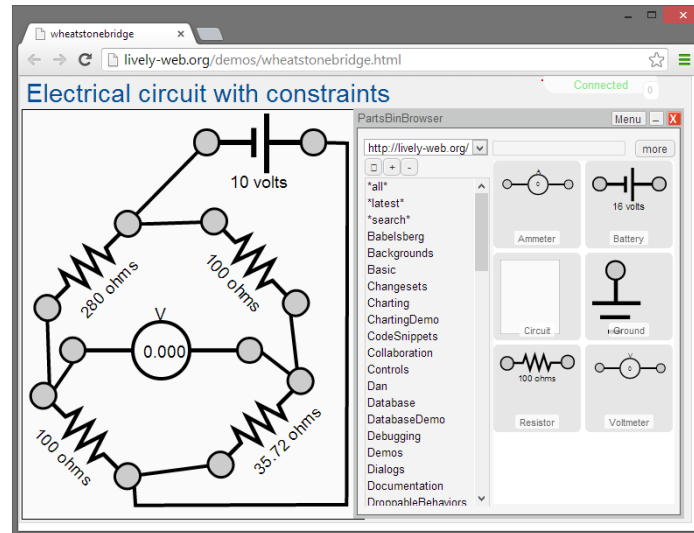


Figure 1: Constructing a Constraint-based Wheatstone Bridge Simulation

load, or a desired ideal spacing between two widgets that again can be relaxed if need be. In the work reported here, we want to support constraints in a clean way in an object-oriented language running in a lightweight, web-based programming environment.

Figure 1 shows a screenshot from our prototype system implemented in the Lively Kernel environment [12] that illustrates the kinds of capabilities we want. It shows a constraint-based simulation of Wheatstone Bridge being constructed. (A Wheatstone Bridge is used to measure an unknown electrical resistance by balancing two pairs of resistors so that the electrical potential between them is 0.) Parts representing batteries, resistors, and meters are copied from the Lively Kernel parts bin [12] on the right, dropped into the circuit on the left, and wired together. These parts carry constraints representing Ohm’s Law, Kirchhoff’s Current Law, and so forth. The system automatically solves the constraints when the parts are first connected, and re-solves them if the battery’s supply voltage or a resistance is edited, updating the voltage displayed by the meter.

The capability to graphically construct constraint-based simulations dates back to Sketchpad [18] and ThingLab [2]. Babelsberg [6] enables a closer integration of constraints with the host object-oriented programming language in the spirit of constraint-imperative programming in Kaleidoscope [13] and Turtle [10].

We now want to support this design in the existing web-based environment. Our work is related to other approaches to constraint satisfaction in object-oriented programs, which use libraries [4, 19], domain-specific languages [5, 17], or (more recently) functional-reactive programming [14, 16] to specify and solve constraints. These approaches do not need special runtime support, but require the programmer

to call specific application programming interfaces (APIS) or follow certain rules to not accidentally circumvent the constraints. With Babelsberg, we provide a syntax and semantics for unified declaration and execution of constraints and object-oriented program entities.

The first implementation of Babelsberg in Ruby, called Babelsberg/R [6], extended the Ruby Virtual Machine (VM). However, applications written in e.g. JavaScript typically have to work on a variety of client VMs included in different Web browsers. This makes it infeasible to implement Babelsberg in a JavaScript VM. JavaScript is currently of considerable interest in the industry and research communities. Thus, an implementation as an extension written entirely in JavaScript enables us to apply constraint programming to a variety of existing problems, and to compare it directly with alternative solutions on a variety of platforms.

A useful Object Constraint Programming language requires sufficiently powerful constraint solving capabilities. In prior work [6], we identified an important requirement, namely support for a new design for *cooperating constraint solvers* [1]. The motivation is that it is often infeasible to provide a single constraint solver that works well for all aspects of a problem; instead, different solvers may be more appropriate than others for some aspects, and which need to work together to solve the problem. Our design and implementation in Babelsberg/JS provides this capability, in a way that supports incremental re-solving of constraints without requiring access to the VM [7].

2 Object Constraint Programming Without VM Support

In industry, JavaScript has become the de-facto standard for Web programming, and a huge amount of code exists in the language. This fact, along with JavaScript's unique design and its execution environment in a Web browser, also make it of great interest to the research community, motivating work on revising and adapting useful features of other languages to include in it [11, 20].

To provide practical support for Object Constraint Programming (OCP) in JavaScript, we adapt the Babelsberg design to not require support from the underlying VM. This enables us to run Babelsberg/JS in modern browsers and use it in a variety of practical Web applications.

For Babelsberg/JS, since we do not have access to the VM, we cannot redefine the operation of `LOAD` and `STORE` instructions to handle variables with constraints on them. Instead, the unmodified JavaScript VM is used only for imperative evaluation mode. To intercept accesses and assignments to constrained variables, we wrap properties with property accessors that interact correctly with the constraint solver.

To get the value of a constrained variable, the accessor gets the value for that variable from its solver. For a store, the setter in general calls the appropriate constraint solver to solve an equality constraint between the variable and its new value for a store.

For constraint construction mode, we use a custom JavaScript interpreter, itself written in JavaScript. This custom interpreter is about three orders of magnitude slower than the underlying one. However, since evaluating code in constraint construction mode is a much less common activity, and one that doesn't occur in inner loops, the performance penalty is not a significant issue.

Generalizing our approach, we have thus identified the following requirements for implementing the Babelsberg scheme without VM support:

- The host language must support a means to intercept variable lookup, so names can refer to different objects.
- The VM-based implementation ignores encapsulation and modify VM data structures directly. In contrast, here the extension must enable calling the appropriate API functions.
- The host language must provide a means to modify interpretation of a block of code to implement the constraint construction mode.

The first requirement is only partially supported in JavaScript, namely for object fields using property accessors. We therefore limit ourselves to constraining field storage in Babelsberg/JS, but not storage into local variables. (Some compiled OO languages, for example C#, also support property accessors; and other dynamic OO languages, such as Python and Smalltalk, support *method wrappers* to enable intercepting accessors, again within the limitation of only constraining field access.) As with the original Babelsberg/R design, it does not matter whether the fields are constrained directly or whether they are used in the execution of a method that was constrained to produce a certain result. A property that is accessed in the execution of a constraint expression is wrapped with property accessor that intercepts lookup and storage.

Property Accessors for Constrained Objects

When an object has been used in a constraint, its constrained properties have been replaced with property accessors. The *property getter* is a simple wrapper that reads from the solver variable in the most upstream region in which the field is referenced (cf. 2.1). Instead of returning the field value of the object, it returns the value of that variable in the solver data structure. The *property setter* distinguishes two cases. If the variable is writable from a solver, an equality constraint for that solver is created and the updated constraint system is solved, potentially triggering other solvers. On the other hand, if the variable is not writable (either because it is of a type that

no available solver supports or because it has been marked as read-only by the programmer), its new value is stored, and all dependent constraints are recalculated. These dependent constraints have treated the variable as a constant (because they cannot modify it). To recalculate them, the constraints are deactivated in the solvers, and the expressions that created them are re-evaluated in constraint construction mode to create new constraints based on the new value. (The implementation of edit constraints (subsection 2.2) handles the situation of repeated changes much more efficiently.)

Creating Constraints

As an example of defining constraints, consider an interactive temperature converter, which maintains the relation between sliders representing values on the Fahrenheit, Celsius, Rankine, and Kelvin scales.

```

1 var converter = {},
2   cassowary = new CLSimplexSolver();
3 always: { solver: cassowary
4   converter.C * 1.8 == converter.F - 32 &&
5   converter.C + 273.15 == converter.K &&
6   converter.F + 459.67 == converter.R
7 }

```

In Babelsberg/JS, a source-to-source transformation creates a call to a global function — `always` — from an `always:` expression of this form (this transformation just provides syntactic sugar — the function can also be called directly with function object.) Once this function has executed, a change to any one of the temperature values in the `converter` object will trigger changes to the other three values to keep the constraint satisfied through property accessors described above.

The `always` function passes the predicate expressing the constraint and information about the context into a custom JavaScript interpreter. This interpreter is used to evaluate expressions in constraint construction mode, which is provided as part of the Babelsberg/JS library. The custom interpreter creates property accessors (getters and setters) for the `C`, `F`, `K`, and `R` fields of the `converter` object. The appropriate accessor is then called whenever some other part of the program uses one of those fields. However, within the constraint expression, accesses to these fields do not use these accessors, but instead return *ConstrainedVariable* objects. Messages are then sent to these objects, and instead of calculating values, build up networks of primitive constraints that can then be satisfied by a solver. The `always` function returns a *Constraint* object that provides meta-level access to the asserted relations, using the protocol described for Babelsberg/R.

In this example, the constraints are on the fields of the object. However, constraints in Babelsberg/JS (as with any instance of the Babelsberg scheme) can also invoke methods that perform computations. For example, imagine the `converter` uses the `getCelsius` method to return a cached temperature value that is updated in regular intervals from a Web service:

```
1 var converter = {},
2   cassowary = new CLSimplexSolver();
3
4 converter.getCelsius = function() {
5   if (!converter.updater) {
6     updateCelsius(converter); // updateCelsius omitted for brevity
7     converter.updater = setInterval(5000, function() {
8       updateCelsius(converter);
9     });
10  }
11  return converter.C;
12 }
13
14 always: { solver: cassowary
15   converter.getCelsius() * 1.8 == converter.F - 32 &&
16   converter.getCelsius() + 273.15 == converter.K &&
17   converter.F + 459.67 == converter.R
18 }
```

By placing the constraint on the result of sending messages rather than on fields, Babelsberg respects object encapsulation. The value returned from the message send in this example is simply a float, but return values can also be arbitrary objects and computed values. For example, we could constrain the maximum pressure of a volume of dry air with a fixed density and gas constant, which would effectively limit the maximum temperature to around 36° Celsius.

```
1 converter.pressure = function () {
2   var gasConstantDryAir = 287.058, // J/(kg * K)
3     density = 1.293; // kg/m^3
4   return density * gasConstantDryAir * converter.K / 1000;
5 }
6
7 always: { solver: cassowary
8   converter.pressure() <= 115 // kPa
9 }
```

2.1 Cooperating Constraint Solvers

The temperature converter described above has no graphical representation. Cassowary only works on reals, yet in order to display the temperature scales, we need to convert the values into strings and update the Web browser's Document Object Model (DOM) using the appropriate API. This is best done with a local propagation solver, which can invoke arbitrary methods to satisfy the constraints, in this case by calling the API. (The constraints that define the temperature converter are simple enough that we could have used a local propagation solver for all of them, but this is unsatisfactory for many problems, such as the Wheatstone bridge example in Figure 1, since local propagation cannot handle such situations as simultaneous equations or inequalities.)

There is currently no single solver that can efficiently handle all constraints that arise in a typical application (and it seems unlikely that one can be created). To address this, we extend the work presented in [6] to include an architecture for cooperating constraint solvers, allowing a problem to be partitioned among multiple solvers. For this example, we use two solvers: one for linear arithmetic on the reals, and one for local propagation constraints.

Our architecture for cooperating solvers partitions constraints into regions that are connected via read-only variables, implementing the design proposed in [1]. The result is a very loose coupling among the cooperating solvers. This approach is in contrast to the more commonly-used Satisfiability Modulo Theory (SMT) technique for supporting cooperating constraint solvers [15], which uses inferred equality constraints as the means for the cooperating solvers to communicate (including the case when neither of the equated variables has a specific value). Our experience so far indicates that our approach is more suited to integration with imperative constructs, in which variables do always have specific values, and lends itself well to support edit constraints for incremental re-solving. (While we have not yet done so in our implementation, the architecture described in [1] in fact allows hierarchies of cooperating solvers, so that within a single region, there could be multiple solvers that cooperate by sharing inferred equality constraints.)

In the cooperating solvers architecture, each constraint belongs to exactly one solver. All constraints that belong to the same solver are in the same region. While constraints belong to exactly one region, variables may be shared across regions. This happens if variables occur in multiple constraints that belong to different regions. These variables must be read-only in all but one of the regions. Read-only variables are represented in a solver-specific manner, either using stay constraints for solvers that support them, or through required equality constraints. To support this, solver libraries should provide a method that makes a variable read-only for them.

In this architecture, the regions must form an acyclic graph, so that solving can simply proceed from the upstream to the downstream regions, propagating variable values. Figure 2 shows an example configuration. Solving proceeds from the left and each solver propagates values for its variables to downstream solvers that need them. The downstream solvers can only read, not write to those variables. This architecture prohibits loops and a system that oscillates without finding a solution. To create this graph, the system determines an order for the solvers based on the dependencies between the constraints. The programmer can explicitly control the position of a solver in this graph, or the libraries can provide information so the system can create the order without the programmer's support. Applications can use multiple instances of the same solver type that are used one after the other (for example, for a problem that first uses Cassowary to solve simultaneous linear constraints, then DeltaBlue for local propagation constraints, then Cassowary again).

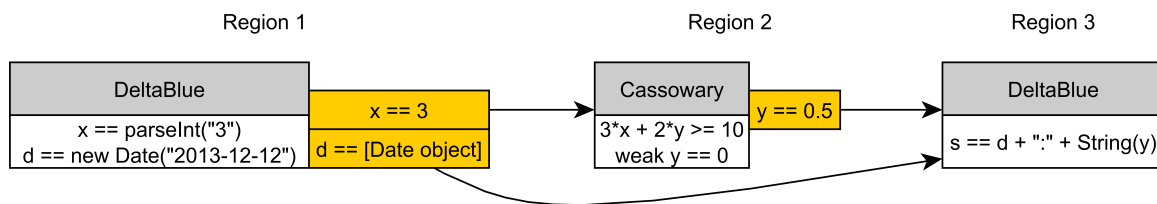


Figure 2: Regions propagating variable values downstream

Once the solver regions are sorted, solving proceeds from the furthest upstream region. Each region will determine values for the variables it can write to, and the downstream regions will adjust to accommodate the new values propagated to their read-only variables from higher level regions. Soft constraints are solved for just within each region — in keeping with the theory of hard and soft constraints in the presence of read-only variables [3], if a soft constraint in an upstream region restricts a variable to a certain value, then a downstream region must use that value and can in fact not distinguish if this value was determined by a required or a soft constraint. If constraints in a downstream region cannot be satisfied due to an upstream soft constraint, we do not backtrack.

Given these additional capabilities, we can now add a graphical representation to our temperature converter. We want the color of a `div` element to change when the temperature is above 30° C.

```

1 var el = jQuery("#tooHotWarning");
2
3 always: { solver: deltablue
4   el.color.formula([converter.getCelsius()], function(celsius) {
5     var color = celsius > 30 ? "red" : "blue";
6     el.setAttribute("class", color);
7     return color;
8   });
9 }

```

Note that for the DeltaBlue local propagation solver, we do not provide a predicate (although we could — in that case it would be run to test whether re-solving is necessary). Instead, local propagation solvers need formulas for all writable variables that state their dependencies and how to update the variable. In this case, we want the Celsius value to be used as input for the color, but not vice versa, so we only provide one formula. The only dependency here is on the return value of `converter.getCelsius()`, passed explicitly in line 4. (Note that this could be omitted Babelsberg/R, because its version of DeltaBlue supports deducing the dependencies from the formula function — a feature we have not yet implemented here.) The dependencies are passed as arguments to the formula function, so we can use them directly to update the DOM using the browser's `setAttribute` API and return the new value. These functions, just like the predicates for Cassowary, are

evaluated in constraint construction mode which wraps variables with property accessors — the function `formula` is simply a function defined by DeltaBlue.

2.2 Incremental Re-Solving for Cooperating Constraint Solvers

Some applications involve repeatedly re-satisfying the same set of constraints with differing input values. A common such case is an interactive graphical application with a constrained figure, in which we move some part of the figure with the mouse. For such applications, it is important to re-solve the constraints efficiently, and a number of constraint solvers, including DeltaBlue and Cassowary, support this using edit constraints that allow a new value for a variable to be repeatedly input to the solver.

The original Babelsberg design did not include support for incremental re-solving at the language level — it was up to the solver library to provide access to such functionality. However, to integrate with our cooperating solvers architecture, Babelsberg/JS does include support for incremental re-solving through a solver-independent `edit` function that takes the variables to be edited and returns a callback function. The process that produces new values can use this callback to input new values into the solvers for the variables to be edited.

The `edit` function gathers all the constraints in which the passed variables participate. Only variables that occur solely in solver regions that support edit constraints can be edited; otherwise an exception is raised. The read-only annotations for variables in the solvers for downstream regions are converted to edit constraints, reflecting the fact that the upstream regions will be providing new values for these variables. Finally, the `edit` function creates a callback function and returns it. This callback can then be used to feed new values into the solvers.

As an example, suppose we wanted to connect the Celsius value of our temperature converter to a graphical slider. We wrap the original `onDrag` (which updates the slider's `value`) to input the new value into the edit callback as well.

```

1 var callback = edit(converter, ['C']);
2 slider.onDrag = slider.onDrag.wrap(function (originalOnDrag, evt) {
3   originalOnDrag(evt);
4   callback([ slider.value ]);
5 });

```

Two restrictions apply to the use of incremental re-solving with cooperating solvers: first, all variables that are edited must be only in regions of solvers that support edit constraints; and second, while the edit callback is used, no new constraints can be created. (Edit constraints are just a technique for optimizing the sequence of repeatedly replacing a constraint that a variable equal a constant with a new constraint with a new constant. Thus, if the restrictions aren't met, it is still possible to express and solve the desired constraints, just not as efficiently.)

3 Future Work and Conclusion

We have presented a design for implementing an Object Constraint Programming language without VM support, which is realized as a JavaScript extension called Babelsberg/JS. We have also implemented a number of features from the original OCP design, including unified language constructs for constraint definition and object-oriented code, automatic maintenance of constraints, integration with the existing syntax and semantics, an interface to add new solvers and constraint solver constructs such as read-only variables and incremental re-solving; and also extended the design to support cooperating constraint solvers. There are a number of directions for future work.

Usability of Babelsberg/JS An important area for future work is the evaluation of the usability of our approach in general applications. We are interested in the comprehensibility of Babelsberg/JS code, especially to the target group for this language, i.e. imperative programmers with little prior experience with constraint programming. This will also provide opportunity to compare performance on more practical examples.

Debugging, Explanation, and Solver Selection It is currently difficult to tell why a solver may not be able to satisfy a given constraint, why it produced an unexpected result, or why finding a solution is slow. Our ConstraintInterpreter should include support for reasoning about the constraint system it builds. Prolog (or just a direct backtracking algorithm) may be useful as a “meta-solver” to automatically find a solver (or set of solvers) for a particular configuration of constraints. We are experimenting with a debugging tool for constraint construction in a Squeak/Smalltalk implementation of Babelsberg [9].

A Complete Formal Semantics The Babelsberg design has evolved alongside with its implementation. However, both because the implementations were driven in part by practical considerations about the expectations of programmers familiar with the underlying host language, as well as because of the complexities of integrating objects, state, and constraints, this has led to a number of semantic choices that are muddled with implementation specifics. For example, there have been a number of confusing issues with respect to constraints and object identity, how to represent assignment, what are the appropriate restrictions on expressions that define constraints, and what should happen if the solver can't find a solution to a set of constraints. In an effort to understand these better and to provide a complete design

that instances of Babelsberg can implement, we are in the process of completing a formal operational semantics for Babelsberg [8].

*

Babelsberg/JS, compared to the earlier Babelsberg/R implementation, can be applied more directly to existing problems. It runs unmodified in different Web browsers, and integrates with the existing imperative language and libraries. The work reported here is quite recent, and we expect to continue to evolve both the language and its implementation.

References

- [1] A. Borning. *Architectures for Cooperating Constraint Solvers*. Technical report VPRI Memo M-2012-003. Glendale, California: Viewpoints Research Institute, May 2012.
- [2] A. Borning. "The Programming Language Aspects of ThingLab, A Constraint-Oriented Simulation Laboratory". In: *PLS* 3.4 (Oct. 1981), pages 353–387.
- [3] A. Borning, B. Freeman-Benson, and M. Wilson. "Constraint Hierarchies". In: *LISP and Symbolic Computation* 5.3 (1992), pages 223–270.
- [4] L. De Moura and N. Bjørner. "Z3: An efficient SMT solver". In: *TACAS*. Springer. Mar. 2008, pages 337–340. DOI: 10.1007/978-3-540-78800-3_24.
- [5] Enthought Inc. *Enaml 0.6.3 Documentation*. Feb. 6, 2014.
- [6] T. Felgentreff, A. Borning, and R. Hirschfeld. "Specifying and Solving Constraints on Object Behavior". In: *Journal of Object Technology* 13.4 (Sept. 2014), 1:1–38. DOI: 10.5381/jot.2014.13.4.a1.
- [7] T. Felgentreff, A. Borning, R. Hirschfeld, J. Lincke, Y. Ohshima, B. Freudenberg, and R. Krahn. "Babelsberg/JS". English. In: *ECOOP*. Springer, 2014, pages 411–436. DOI: 10.1007/978-3-662-44202-9_17.
- [8] T. Felgentreff, T. Millstein, and A. Borning. *Developing a Formal Semantics for Babelsberg: A Step-by-Step Approach*. Technical report 2014-002. Viewpoints Research Institute, Sept. 2014.
- [9] M. Graber, T. Felgentreff, R. Hirschfeld, and A. Borning. "Solving Interactive Logic Puzzles With Object-Constraints - An Experience Report Using Babelsberg/S for Squeak/Smalltalk". In: *REBLS*. To appear. ACM, 2014, 1:1–1:5.
- [10] M. Grabmüller and P. Hofstedt. "Turtle: A constraint imperative programming language". In: *RDIS*. Springer, 2004, pages 185–198. DOI: 10.1007/978-0-85729-412-8_14.

- [11] S. Kang and S. Ryu. “Formal Specification of a JavaScript Module System”. In: *OOPSLA*. ACM. 2012, pages 621–638.
- [12] J. Lincke, R. Krahn, D. Ingalls, M. Roder, and R. Hirschfeld. “The Lively PartsBin—A Cloud-Based Repository for Collaborative Development of Active Web Content”. In: *HICSS*. IEEE. Jan. 2012, pages 693–701. DOI: 10.1109/HICSS.2012.42.
- [13] G. Lopez, B. Freeman-Benson, and A. Borning. “Kaleidoscope: A Constraint Imperative Programming Language”. In: *Constraint Programming*. Volume 131. NATO Advanced Science Institute Series, Series F: Computer and System Sciences. Springer-Verlag, 1994, pages 313–329. DOI: 10.1007/978-3-642-85983-0_12.
- [14] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. “Flapjax: A Programming Language for Ajax Applications”. In: *SIGPLAN* 44.10 (2009), pages 1–20.
- [15] G. Nelson and D. Oppen. “Simplification by Cooperating Decision Procedures”. In: *PLS* 1 (1979), pages 245–257.
- [16] Y. Ohshima, A. Lunzer, B. Freudenberg, and T. Kaehler. “KScript and KSWorld: A Time-aware and Mostly Declarative Language and Interactive GUI Framework”. In: *Onward!* Indianapolis, Indiana, USA: ACM, 2013, pages 117–134. DOI: 10.1145/2509578.2509590.
- [17] E. Sadun. *iOS Auto Layout Demystified*. Addison-Wesley, Oct. 2013.
- [18] I. Sutherland. “Sketchpad: A Man-Machine Graphical Communication System”. In: *Proceedings of the Spring Joint Computer Conference*. IFIPS. 1963, pages 329–346.
- [19] E. Torlak and D. Jackson. “Kodkod: A relational model finder”. In: *TACAS*. Volume 4424. Springer, Apr. 2007, pages 632–647. DOI: 10.1007/978-3-540-71209-1_49.
- [20] T. Van Cutsem and M. S. Miller. “Proxies: Design Principles for Robust Object-Oriented Intercession APIs”. In: *SIGPLAN* 45.12 (2010), pages 59–72.

Comparing the Layout Stability of Treemap Algorithms

Sebastian Hahn

Computer Graphics Systems Group
Hasso-Plattner-Institut
sebastian.hahn@hpi.uni-potsdam.de

Different approaches were presented to evaluate this property of a layout algorithm. Since most of these approaches focus on the position change of the rendering artifacts there is minimal work that connects the effects that different changes in the data causes on the geometric properties of the depicted items and the perception of the users and his mental map to support the comparison of different layout algorithms of hierarchical visualization techniques (e.g., different treemap layout algorithms). In this report I present the first two stages of a 3-stage framework for the evaluation of layout stability.

1 Introduction

The visualization of hierarchical datasets is a well investigated research field with a various range of use cases and data sets, e.g. the visualization of software system's package structure at a single state of the development process in so called "*software maps*" to get an overview of the system [3]. The main idea of such "*software maps*" is to give a shape to the shapeless information pieces of software systems by using different visual variables, like geometrical shape, height and color [2], use the metaphor of a landscape or a map to arrange theses items - by a *layout algorithm* - and support a user to create a mental map of the software system that allows for fast recognition of certain points of interests.

Since software systems are evolving over time the layout algorithms that creates those depictions need to fulfil a special requirement: Small changes in the datasets should lead to similar depictions (see Figure 1). This requirement is called layout stability and should make it possible for a long-time user of a software map to preserve his before created mental map. A lot of publications claim that different novel approaches are more stable than older ones and present different metrics to evaluate this property of a layout algorithm [8, 9]. Although, each of the presented approaches focus certain aspects of this ability, they are not entirely useful for a full evaluation with respect to all stages of the visualization pipeline (see Figure 2). Therefore, we present an approach that focusses not just on the position change of

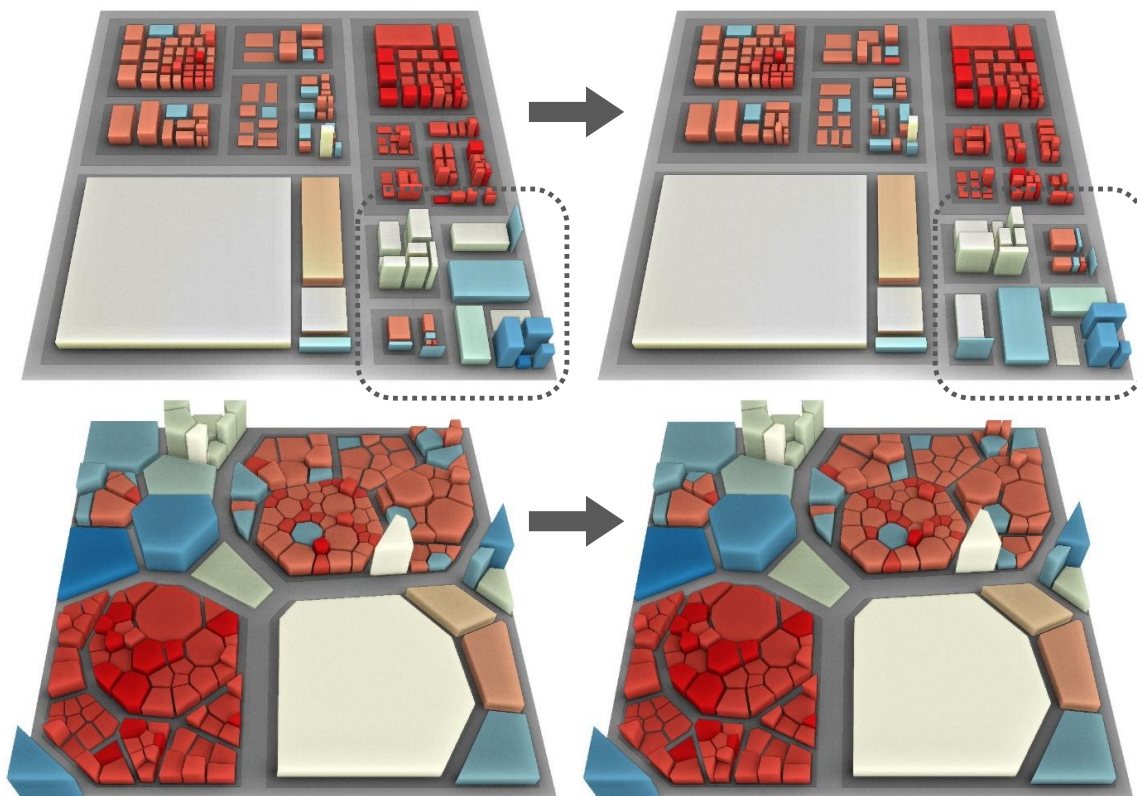


Figure 1: Comparison of two snapshots of a software system depicted with a squarified treemap algorithm (top) [5] against depictions using voronoi treemaps with stable initial distributions (bottom) [6]. Note the switching artifacts in the bottom right corner of the rectangular-based approach and the minimal change in the polygonal approach.

the rendering artifacts, but also connects the effects that different changes in the data causes on the geometric properties of the depicted items and the perception of the users and his mental map to support the comparison of different layout algorithms of hierarchical visualization techniques (e.g., different treemap algorithms [1, 5, 6, 8]). In this report the *data stage* and *mapping stage* are presented.

2 Data Stage

First, to compare two snapshots of a varying hierarchical dataset metrics for the change in the input data itself is needed. Boorman and Oliver differ the comparison

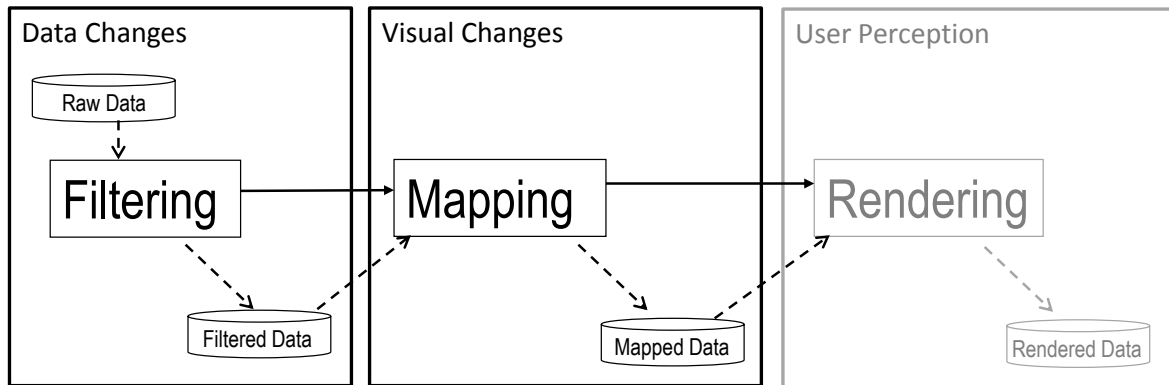


Figure 2: This report presents metrics to evaluate layout stability of treemap algorithms on the data (left) and mapping stage (middle)

of tree structures in two different ways: a.) the definition of atomic functions (e.g., `addChild`, `removeChild`) that can be used to describe the changes from one tree to another b.) find a metric that represents the changes of two tree structures [4]. In addition to the structural changes of the hierarchies we also have to handle changes in the attributes of the tree items. Our primary goal is to get an overview over several changes of the dataset properties (e.g., the change in the hierarchical structure itself, the changes of a certain attribute of tree items). For it, we declare metrics with a range of 0 to 1 for different aspects of the changes that are connected to the layout stability, while 1 means totally unchanged and 0 means that the compared datasets are completely disjunct.

2.1 Hierarchical Change

The change in the hierarchical data is measured by a depth-weighted approach of the Jaccard index applied on each subtree of the dataset in both directions. This allows for comparing the hierarchical dataset with respect to their structure and includes changes induced by *add*-, *remove*- or *move*-operations, claimed as the basic operations applied on hierarchies of software system structure [4].

$$\begin{aligned}
 T &= \{\text{uid}, d, C\} & (1) \\
 C &= \{N_0, \dots, N_n\} \mid n \in \mathbb{N}, N \in T & (2) \\
 \text{depth}(T) &= T_d & (3) \\
 \text{maxDepth}(T) &= \begin{cases} 0, & \text{if } T_C = \emptyset \\ 1 + \max(\text{maxDepth}(T_{N_0}), \dots, \text{maxDepth}(T_{N_i})), & \text{otherwise.} \end{cases} & (4) \\
 J(A, B) &= \frac{|A \cup B|}{|A \cap B|} & (5) \\
 J(T, T') &= \begin{cases} 0, & \text{if } T_C = \emptyset \\ (\text{maxDepth}(T_{\text{Root}}) - \text{depth}(T)) \cdot J(T_C, T'_C) + \sum_{i=0}^n J(T_{N_i}, T'_{N_i}), & \text{otherwise.} \end{cases} & (6) \\
 W(T, T') &= \begin{cases} 0, & \text{if } T_C = \emptyset \\ (\text{maxDepth}(T_{\text{Root}}) - \text{depth}(T)) + \sum_{i=0}^n W(T_{N_i}, T'_{N_i}), & \text{otherwise.} \end{cases} & (7) \\
 \text{HierChange}(T, T') &= \frac{J(T, T') + J(T', T)}{W(T, T') + W(T', T)} & (8)
 \end{aligned}$$

Figure 3: Data model for a hierarchy and calculation of metric for hierarchical changes in two hierarchies T and T'

2.2 Attribute Changes

In addition to the hierarchical structure of the aforementioned datasets, the information contained in each node, or the nodes *attributes* used for the mapping stage, change a lot. Therefore, a metric that covers the changes of a certain attribute of the hierarchical nodes seems useful. Steinbrückner proposes the use of the *Relative Weight Change* metric (see Figure 4). Since this metric proceeds the assumption that all items appear in both compared states it is not sufficient for most of our datasets in general, but for a metric just regarding the nodes attributes.

$$\text{RWC}_w(\text{DAHG}, t) := \frac{\sum_{\{v \in G \mid f_e(v, t) = \text{True}\}} |a_w(v, t) - a_w(v, t-1)|}{\sum_{\{v \in G \mid f_e(v, t) = \text{True}\}} a_w(v, t-1)}$$

Figure 4: Relative weight change proposed by Steinbrückner [10]. The normalized sum of each nodes attribute (a_w) difference in two snapshots of a tree t and $t-1$ is taken to calculate a similarity metric.

3 Mapping Stage

The mapping stage uses the mapped data from the visualization pipeline as an input for the layout stability metrics calculation, that means, e.g., the items' shape, position, and size. Basically, we have two groups of mapping stage metrics: a.) the *intra-node* and b.) the *inter-node* metrics. Intra-node metrics only takes the nodes appearance itself for each node into account. Bederson et al. present such a intra-node metric for layout stability with the so called *distance change* (see Equ. 9) [1]. For each leaf-node of a tree the Euclidean distance of its change in position (x, y) and size (w, h) is calculated and added up. The metric itself worked fine for the comparison of rectangular treemaps without border sizes, but is not sufficient to compare rectangular treemaps against those that are based on polygons. Additionally, neither, the adjacency between nodes nor the (parent-)node appearances are used to define the stability.

$$\text{distChange} = \sqrt{d_x^2 + d_y^2 + d_w^2 + d_h^2} \quad (9)$$

The aforementioned example (Figure 1) shows one of the main problems with the distance change. If a high number of small changes in the layout appear (as in the polygon-based approach) the distance change can be equal or even higher compared to rectangular approaches with a few high changes. Nevertheless, the layout itself appears more stable to the user because of the small changes in the overall structure, coming from the adjacency informations of the nodes. Therefore, the additional use of an inter-node metric would be sufficient.

The computation of the adjacency similarity is done in the following steps:

1. For each snapshot of a hierarchy, a diagonal adjacency matrix M_t is set up for each subtree.
2. The matrix values are populated by computing the directional vectors from each node to its neighbours (see Figure 5).
3. The difference of the adjacency matrix between all M_{t_i} is computed added up and normalized.

By using this adjacency similarity metric the we can make sure to compare rectangular as well as polygon-based treemap algorithms with respect to their inner structure and have an additional metric that also is aware of rotations to a certain degree.

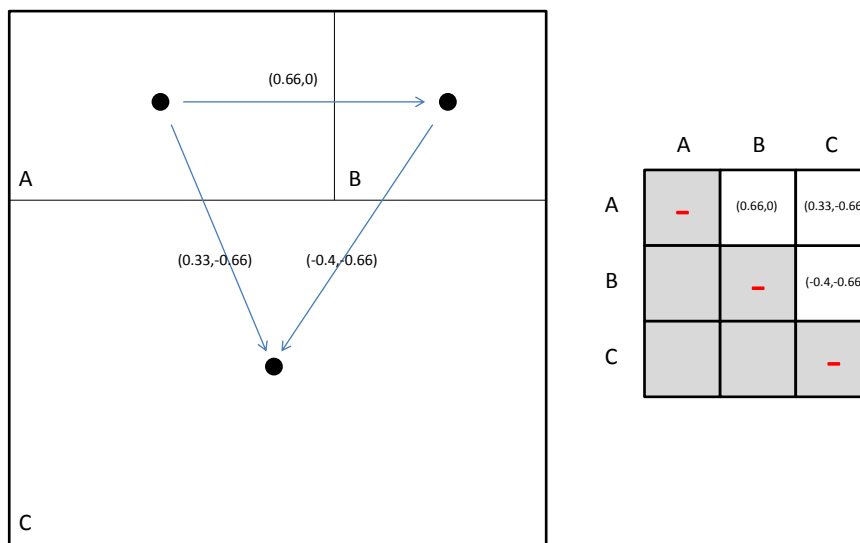


Figure 5: The adjacency matrix is populated by the computed directional vectors for each item

4 Next Steps

In the next weeks I'm planning to finally run the evaluation of the layout stability of the most common treemap algorithms (strip treemap, squarified treemaps, slice'n dice treemaps) against those that are claimed to have a higher layout stability (voronoi treemaps with stable initial site distribution, Hilbert-Moore treemaps [11] and template-based treemaps [7]). After that, I want to investigate in the dynamic visualization of changes between two states by using animated transitions, to make the appearing changes more reasonable. In addition to that, the static visualization of changes between two states of a software map has high potential to increase the usability of software maps.

5 Further Activities

Besides to my research in the field of layout stability for software maps, I supervised a master thesis focussing on the visualization of multi-threaded software behaviour using a level-of-detail-based software-city approach, called *Thread city* (see Figure 6). Moreover, I worked on an interactive visualization of a dependency graph for analysis purposes as a part of a software visualization project with the SAP Innovation center (see Figure 7). Furthermore, I participated as a tutor for the 3D computer

graphics lecture at the HPI and contributed to our open source computer graphics middle ware libraries (github.com/hpicgs).

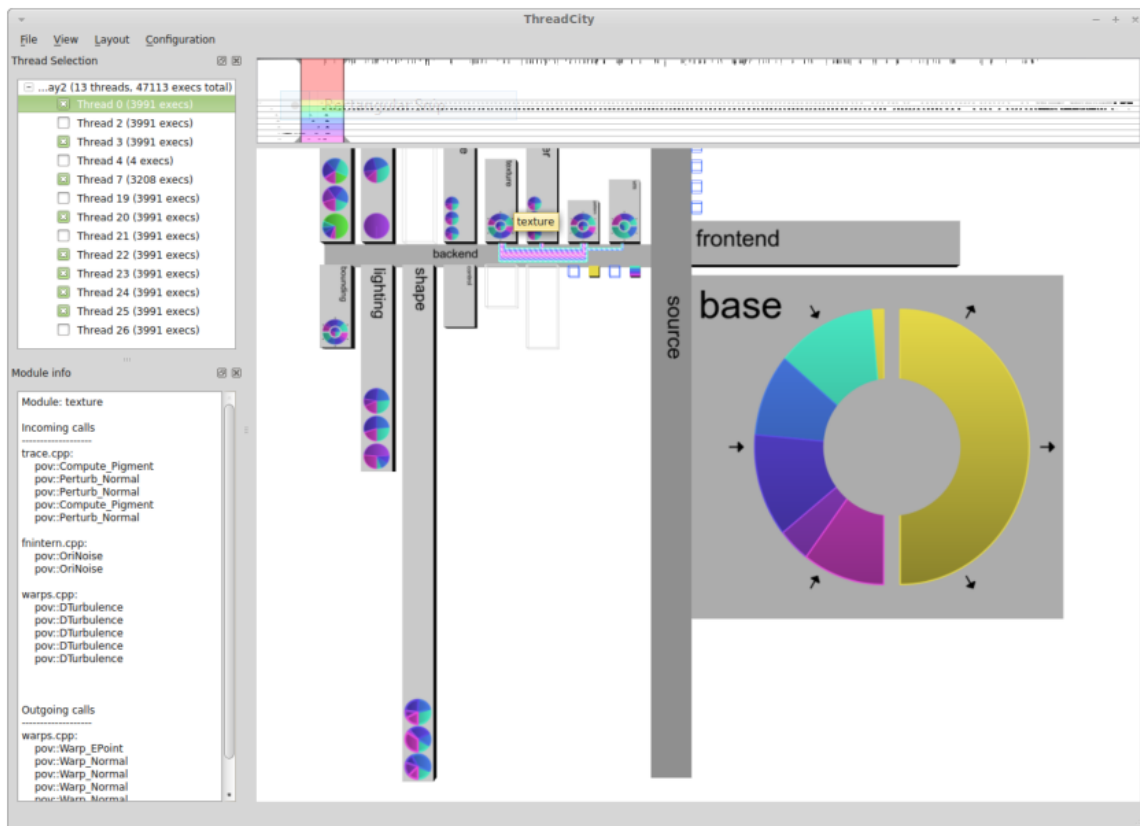


Figure 6: ThreadCity tool

References

- [1] B. B. Bederson, B. Shneiderman, and M. Wattenberg. “Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies”. In: *AcM Transactions on Graphics (TOG)* 21.4 (2002), pages 833–854.
- [2] J. Bertin and M. Barbut. *Sémiologie graphique: les diagrammes, les réseaux, les cartes*. Mouton Paris, 1967.
- [3] J. Bohnet and J. Döllner. “Monitoring Code Quality and Development Activity by Software Maps”. In: *Proceedings of the 2Nd Workshop on Managing Technical*

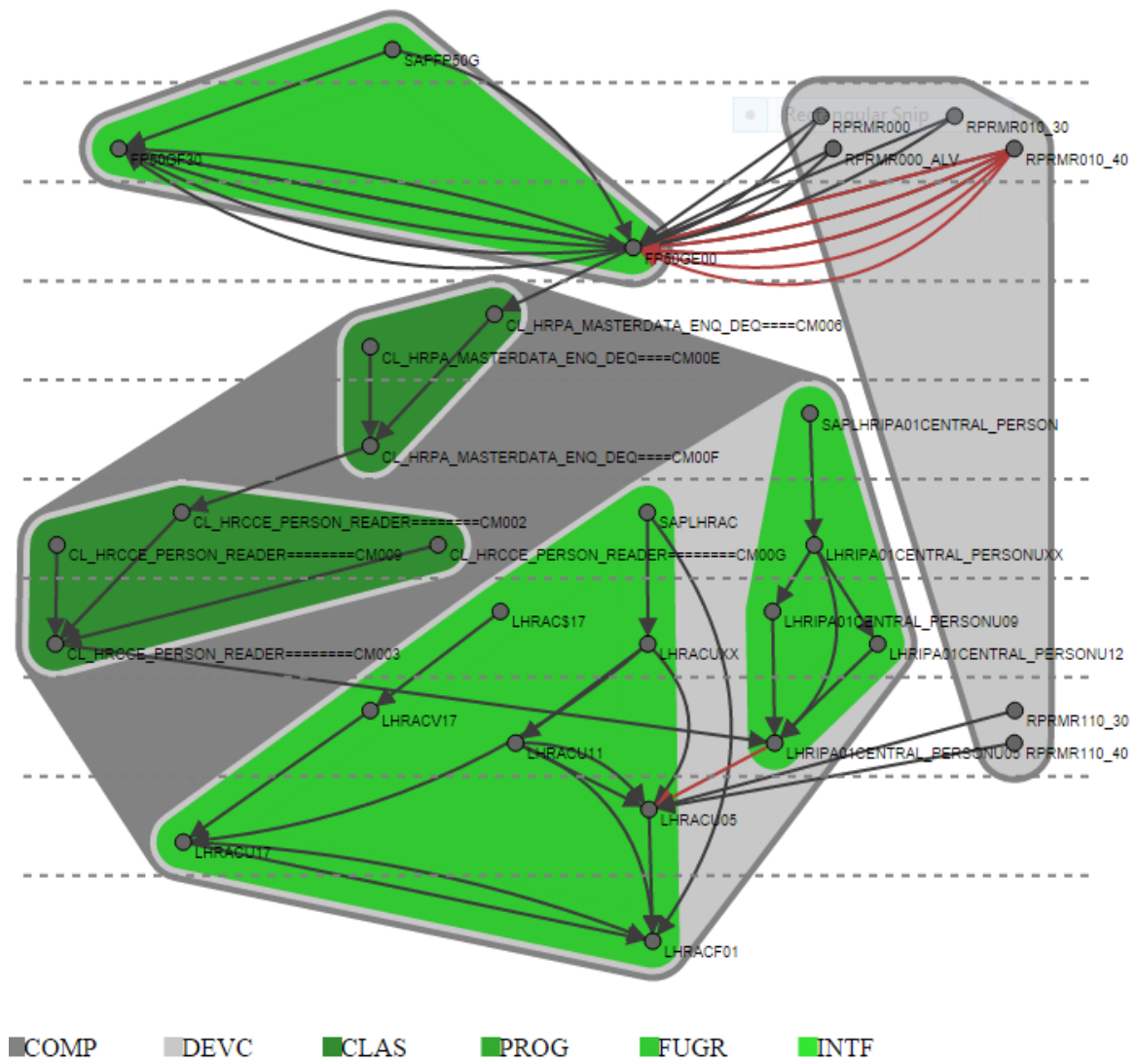


Figure 7: SAPIC dependency analyzer

- Debt*. MTD '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pages 9–16. doi: 10.1145/1985362.1985365.
- [4] S. A. Boorman and D. C. Olivier. “Metrics on spaces of finite trees”. In: *Journal of Mathematical Psychology* 10.1 (1973), pages 26–59.
- [5] M. Bruls, K. Huizing, and J. J. Van Wijk. “Squarified treemaps”. In: *Data Visualization 2000*. Springer, 2000, pages 33–42.
- [6] S. Hahn, J. Trümper, D. Moritz, and J. Döllner. “Visualization of Varying Hierarchies by Stable Layout of Voronoi Treemaps”. In: *Proceedings of the 5th International Conference on Information Visualization Theory and Applications (IVAPP 2014)*. SCITEPRESS – Science and Technology Publications, 2014, pages 50–58.
- [7] N. Kokash, B. de Bono, and J. Kok. “Template-based Treemaps to Preserve Spatial Constraints.” In: *IVAPP*. 2014, pages 39–49.
- [8] B. Shneiderman and M. Wattenberg. “Ordered treemap layouts”. In: *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*. IEEE. 2001, pages 73–78.
- [9] F. Steinbrückner. “Consistent Software Cities: supporting comprehension of evolving software systems”. PhD thesis. Universitätsbibliothek, 2012.
- [10] F. Steinbrückner and C. Lewerentz. “Representing development history in software cities”. In: *Proceedings of the 5th international symposium on Software visualization*. ACM. 2010, pages 193–202.
- [11] S. Tak and A. Cockburn. “Enhanced Spatial Stability with Hilbert and Moore Treemaps”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.1 (2013), pages 141–148. doi: <http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.108>.

Generating Dynamic Dependability Models from Call Traces

Lena Herscheid

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

In order to make complex software dependable, its failure behaviour needs to be well understood. Dependability modeling languages, such as fault trees, provide a means for quantitatively and qualitatively assessing and comparing the reliability of different systems. Reliability, i. e., the probability of providing a service without interruption for a period of time, is an important dependability attribute also for software systems, where any crashes and downtime lead to economic loss and bad customer experience.

To assess software reliability, runtime information from realistic program executions needs to be included when modeling software systems. A prototype tool, PyFT, which generates dynamic fault trees from Python programs, is introduced in this paper. PyFT traces the program execution and constructs a dependability model, including information about exceptions, by harnessing the introspection features of the Python language. Two ways of obtaining unreliability numbers are suggested—from developer annotations, and from code complexity.

This article is a critical discussion of the approach of deriving quantitative dependability models based on source code, as exemplified by PyFT. The lack of a useful software fault model, which incorporates runtime aspects, remains an unsolved problem.

1 Introduction

A major threat to dependability of modern software systems is their complexity, which can lead to unforeseen error propagation chains and various different failing paths through the program. Complexity arises from the composition of heterogeneous software modules to form systems of increasing scale and feature richness. While single modules may be thoroughly tested and well understood by their engineers, the emerging complexity of large, composed software systems is much harder to reason about.

Heisenbugs—bugs that are activated only under certain environment conditions—are becoming increasingly common in large scale software systems [9] [6]. Such seemingly non-deterministic heisenbugs stem from the ordering of concurrent oper-

ations, the availability of shared resources, or the unpredictable behaviour of third party dependencies.

Formal methods for assuring dependability fall short in large scale software systems due to combinatorial and variable range explosion. Naturally, this state space explosion problem also affects humans: Manually creating and maintaining models for analyzing software dependability has become tedious and error-prone. The dynamic behaviour of software systems is rarely thoroughly understood by its developers.

To assess and compare the dependability of complex software systems, automated ways of obtaining dependability models are therefore needed. Such dependability models should include *runtime information* in order to reflect the dynamic behaviour of software realistically and also cover heisenbugs.

This paper introduces a prototype tool, *PyFT*, which generates fault trees from the callgraphs of Python programs. By harnessing Python 3 features, *PyFT* observes single executions of the program, generating a basic event for each function call which may fail. Approximate reliability metrics for these basic events are obtained in two different ways: first, by evaluating potential developer annotations and second, by computing code complexity metrics for important encountered function calls.

2 Related Work

Software Fault Trees Leveson et al. [11] propose a static mapping of programming language constructs to fault trees without static elements. This mapping takes place at the granularity of source code lines, and is thus applicable to any imperative programming language. This transformation has been discussed, extended and compared with static analysis methods such as weakest precondition analysis by Clarke et al. [5]. To our knowledge, dynamic fault trees containing sequence-dependent gates, such as SPARE gates, have not yet been generated from source code in related literature.

Runtime Dependability Models Exception propagation behaviour, as an error recovery mechanism, has been formalized in a variety of related literature, e.g. [7]. At a much higher level of abstraction, the generation of models from runtime information, especially in the context of self-adaptive and distributed systems has been studied [1]. In this context, the research area of *runtime verification* explores how desirable correctness properties can be monitored and verified dynamically [2]. Verified properties are usually expressed in temporal logic formulae. A distributed monitoring mechanism for these properties can then be synthesized, which in itself needs to be fault tolerant. However, the transition to such methods is hindered by

human difficulties of using temporal logic correctly [8]. Since the advent of accessible, increased parallel computing powers, the application of other formal methods such as model checking during runtime has also been proposed [3].

3 PyFT - a Prototype Dependability Tracer for Python

PyFT generates dependability models at the granularity level of function calls, serving two purposes: First, aspects of the program's runtime behaviour relevant for fault tolerance, are visualized in an intuitive fashion. Second, the resulting models should be analyzable by existing tools, so programs can be assessed using established methods from the reliability engineering domain.

To achieve the first objective, we use program traces, ideally obtained from test suite executions, as a basis. In order to secondly re-use existing reliability modelling tools, and to provide an intuitive graphical model of the execution dynamics, we generate a *dynamic fault tree* from the traced call graph.

Implementation Due to its dynamic nature and in-built reflection and debugging features, the Python programming language is well suited for tracing experiments. Based on the open source `pycallgraph` package¹, *PyFT* uses the `with` statement to trace the execution of an arbitrary code block using a `Callgraph` object. The `Callgraph` object's `__enter__` and `__exit__` functions, called at the entry and exit of the `with` block, take care of setting up and tearing down a custom tracer. This tracer uses inbuilt the `sys.settrace()` facility. Custom code, keeping track of which functions were invoked for how long, can be executed in the tracer. Python's introspection features allow for a straightforward implementation of the tracer—method and variable names, memory usage and execution time can be figured out easily. In case the processing takes a long time, for instance when additional code metrics are computed, the tracer can also asynchronously work on a queue of incoming trace events, containing all necessary frame information. Listing 1 shows an invocation of the main function with the tracer.

¹<http://pycallgraph.slowchop.com/en/master/>, accessed December 16, 2014.

Listing 1: Example code, with traced execution. Function calls in the Eratosthenes class are annotated by the developer. In `primesUpTo`, exception handling is used to recover from wrongly typed inputs.

```
1
2 #!/usr/bin/env python
3 from output import FaultTreeOutput
4 from callgraph import CallGraph
5
6 class Eratosthenes:
7     def __init__(self) -> 'extremely_robust':
8         self.primes = []
9
10    def clear(self):
11        self.primes.clear()
12
13    def primesUpTo(self, limit) -> 'robust':
14        self.clear()
15        try:
16            n = limit+1
17        except TypeError:
18            n = int(limit)+1
19
20        multiples = set()
21        for i in range(2, n):
22            if i not in multiples:
23                self.primes.append(i)
24                multiples.update(range(i*i, n, i))
25
26    def __str__(self) -> 'extremely_robust':
27        return("Here are some primes: " + str(self.primes))
28
29 def main():
30     output = FaultTreeOutput()
31     with CallGraph(output=output):
32         e = Eratosthenes()
33         e.primesUpTo('100')
34         print(e)
35
36 if __name__ == '__main__':
37     main()
```


3.1 Fault Trees from Call Graphs

While call graphs are easy to obtain from traces, how to interpret them as dependability models is not obvious. *Fault trees* are a deductive dependability modelling language in failure space [15]. In our case, the undesired top event is a failure of the program execution to meet its specification. In this Section, we discuss first how to derive a static fault tree from traced function calls. This fault tree is then enriched by fault tolerance semantics using the dynamic *SPARE gate*.

Fault and Error Model Finding a realistic fault model for software systems at the source code level is challenging. In theory, a fault is any syntactic deviation from a correct program [14]. Moreover, error propagation and error handling mechanisms can take on various forms and are therefore hard to detect automatically. Even with runtime introspection, the program itself is devoid of information about the developers' original intent. In the absence of a complete, formal specification, which is not provided for most modern software, a simplified fault and error propagation model must be postulated. In this work, the following assumptions are made:

1. **Binary Failure Model:** Any function call can either fail or succeed. No other nuances of the notion of "failed" software exist. When the traced execution represents a test case, the assumption is that the test suite provides a *specification* for the system. If the test case fails, the top event occurs, otherwise, the system is assumed fully functional.
2. **Immediate, Deterministic Error Propagation:** Unless explicitly handled (by catching exceptions), an error resulting from the failure of one function call immediately propagates to the calling function.
3. **Fault Tolerance By Exception Handling** A common way of handling and mitigating errors are *exceptions*, as supported by the programming language. We assume that exceptions are only used for fault tolerance mechanisms, and not for regular control flow. Speaking in terms of Hamner's terminology for fault tolerant software [10], we assume that the raising of the exception represents a *detected error*, its handling represents *error mitigation* or *error recovery*. This is a simplified stance which disregards the potential use of exceptions as a syntactic tool for non-erroneous control flow manipulation.

Figure 1 depicts the dynamic fault tree for the code in Listing 1. A function call fails, if the function itself fails to do what it is intended, or if a callee of the function fails. Therefore, the callgraph is simply transformed to a hierarchy of OR gates.

According to Cristian [7],

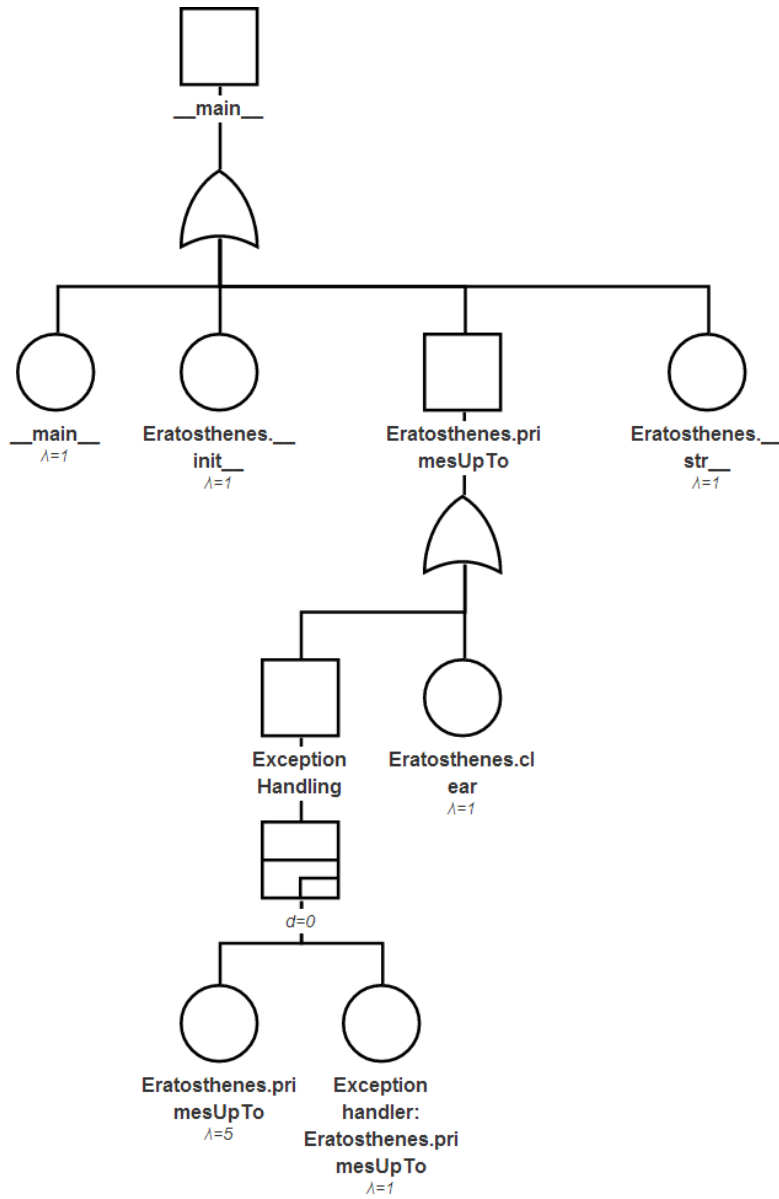


Figure 1: Dynamic fault tree, generated from the example code in Listing 1. In the model, each traced function call is represented by a basic event, with a rate parameter representing the complexity and developer annotation regarding function robustness. Exception handling is assumed to be a fault tolerance mechanism, and therefore represented by a SPARE gate.

Exceptions can therefore be viewed as being a software structuring concept that helps bridge the conceptual gap which exists between behaviors as opposite as “correct standard service provided” at one extreme, and “program failure” at the other extreme.

Consider the `primesUpTo` function in our example. If the input variable type is incorrect, exception handling is used to mitigate or recover from this error. Exceptions—assumed fault tolerance mechanisms—are therefore transformed to SPARE gates in the dependability model. Here, the function in which the exception was raised becomes the “primary spare”. The function in which the exception is handled becomes the “secondary spare”. We use cold spare semantics, because the secondary spare is not “used”, or executed, unless the exception really occurs.

3.2 Obtaining Software Reliability Metrics

Fault tree analysis can be either qualitative or quantitative. Qualitative fault tree analyses, such as the determination of minimal cut sets, is meaningful only if the fault tolerance mechanisms are also reflected in the model. As discussed in the previous Section, we hoped to achieve that by mapping exception propagation to SPARE gates.

On the other hand, quantitative analyses such as computing top event probabilities from the model, can be used to compare any two fault trees. In this case, failure rates for the basic events—function calls in our case—are needed.

We present two approaches to obtaining such quantitative metrics:

1. **Developer Annotations** Developers are usually able to provide an estimation of the reliability of the code they implemented, taking into consideration its complexity and their own experience. Since the dependability model should incorporate as much information as possible, PyFT parses optional Python 3 annotations, provided by the developers, to obtain a vague reliability value. For instance, in Listing 1, functions of the `Erastothernes` class have been annotated with estimations of their robustness by developers.
2. **Complexity Metrics** Various code complexity metrics, as well as their relation to software reliability, have been studied in the literature. McCabe’s cyclomatic complexity [12] has been found to correlate with fault-proneness of software modules [13]. Therefore, PyFT uses the `radon` package² to estimate the failure proneness of a function from its cyclomatic complexity.

Both developer annotations and complexity metrics can be mapped to basic event failure rates. This mapping is somewhat arbitrary, hence, the resulting unreliability

²<https://pypi.python.org/pypi/radon>, accessed December 16, 2014.

numbers are meaningful only in relation with other, similarly obtained, numbers. PyFT does *not* claim that the absolute value, without comparison to other such values, can be used as a realistic estimate of the system unreliability³.

Since both approaches are highly imprecise and speculative, we propose that the resulting values should be regarded as “risk indicators” rather than probabilities in the strict mathematical sense: the risk of different models (representing implementations of different code quality) can be compared and partially ordered, but the numbers should not be used to compute scalar values such as times to failure.

To better reflect the imprecision inherent to the automated PyFT approach, we propose to use triangular fuzzy numbers, as supported by the *FuzzEd* fault tree analysis tool [16].

4 Towards Runtime Dependability Models

Figure 2 shows the bigger picture around the PyFT approach. The vision is a comprehensive framework which uses various sources to automatically run suitable fault injection experiments on a software system. From the observed runtime behaviour under faultload, more realistic dependability models can be derived automatically.

5 Conclusion and Future Work

PyFT needs to be applied to more real world examples, in order to validate its usefulness in non-trivial software systems. To yield realistic dependability models at the source code level, some further impediments still have to be overcome:

Tracing The approach of PyFT relies on tracing as a means for getting detailed insights into the program’s runtime behaviour. Tracing obviously comes at a cost: First, the execution of the program is slowed down significantly. Therefore, profiled information such as the average time spent in a function can be very inaccurate. Second, non-intrusive tracing is much harder to implement in compiled languages such as C++, than it is in Python. Since many safety-critical software systems are still written in such languages, a useful implementation of the tool presented here in a compiled language might become an engineering challenge.

³Hardware fault trees frequently use quantitative analysis to compute the probability of failure before mission end time. We do not believe that software fault trees can be applied analogously at the source code level of granularity.

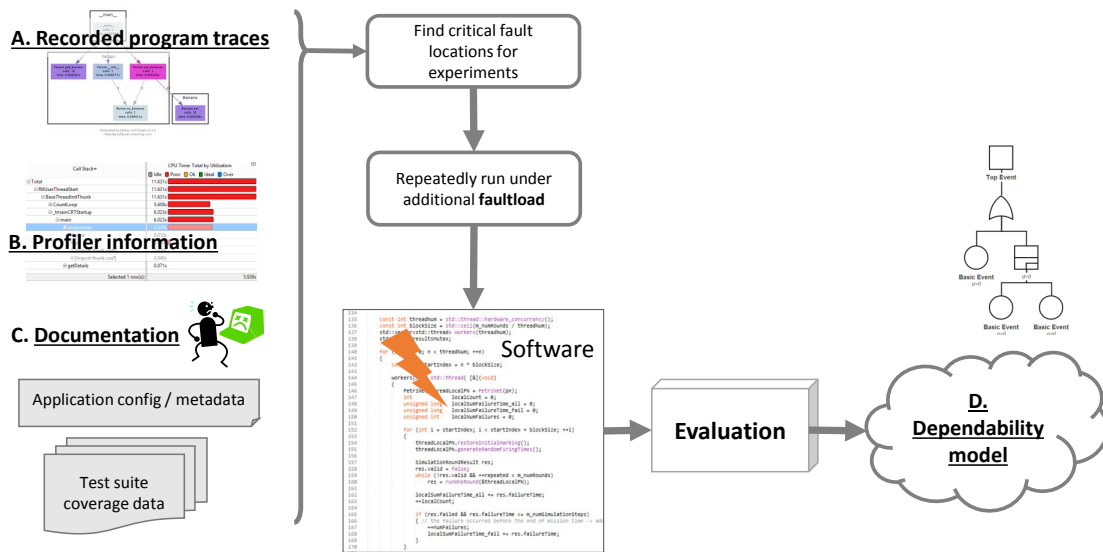


Figure 2: PyFT in context. The highlighted (underlined) steps have been tackled partially by PyFT. **A:** Program executions are traced using Python language support such as `settrace`. **B:** Time and memory spent inside a function are also recorded by the framework. Future work will investigate the usage of such measured values for computing reliability metrics. **C:** Python 3 annotations can be used by developers to document the confidence they have in their code. These annotations are used as a basis for quantitative per-function reliability metrics (basic event probabilities). **D:** PyFT outputs dynamic fault trees as a preliminary, quantitative reliability model.

Fault Models Most important, the approach lacks a unified, language-independent software fault model, which takes into consideration runtime phenomena. There is a vast amount of related efforts: literature such as the *orthogonal defect classification* (ODC) [4], as well as online resources. A recent trend is mining data from bug trackers and using collaborative defect databases such as the *common weakness enumeration* (CWE)⁴. Pretschner et al. [14] propose a fault model tailored for automatic testing.

Incorporating a realistic software fault model into the architecture in Figure 2 is part of our future plans. To this end, we are currently doing a systematic literature study of software fault models. We believe that *fault activation patterns* are of special interest for a broad class of software failures. Understanding such environment-

⁴<http://cwe.mitre.org/index.html>, accessed December 16, 2014.

dependent fault activation patterns would pave the way for more efficient mitigation approaches relying on the prevention of fault/bug activation instead of fault removal in the source code.

Fault Tolerance Patterns Assessing the program structure with regards to dependability makes sense only when fault tolerance mechanisms are included. This raises the question: Is it possible at all to extract developer intent from source code in a fully automated fashion? We found this hard to achieve, and based PyFT on the assumption that structured exceptions are the sole means for fault tolerance.

Future efforts of automatically detecting fault tolerance patterns in software are necessary for more realistic models. Promising in this respect may be the *Erlang* language, which, designed with fault tolerance in mind, offers dedicated syntactic constructs and handling mechanisms for all kinds of errors.

As shown in Figure 2, PyFT is nevertheless a step towards the automated generation of dependability models from runtime behaviour. PyFT can be used to derive a preliminary runtime dependability model, used to further guide analysis and experiments. For instance, one might envision using the *minimal cut sets*, i.e., the minimal paths to system failure modelled in the fault tree, for finding promising locations for fault injection locations in the source code.

References

- [1] A. Bauer, M. Leucker, and C. Schallhart. “Model-based runtime analysis of distributed reactive systems”. In: *Software Engineering Conference, 2006. Australian*. Apr. 2006, DOI: 10.1109/ASWEC.2006.36.
- [2] A. Bauer, M. Leucker, and C. Schallhart. “Runtime verification for LTL and TLTL”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20.4 (2011), page 14.
- [3] R. Calinescu and S. Kikuchi. “Formal Methods @ Runtime”. In: *Proceedings of the 16th Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems*. FOCS’10. Redmond, WA: Springer-Verlag, 2011, pages 122–135.
- [4] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong. “Orthogonal defect classification-a concept for in-process measurements”. In: *Software Engineering, IEEE Transactions on* 18.11 (1992), pages 943–956.

- [5] S. Clarke and J. McDermid. "Software fault trees and weakest preconditions: a comparison and analysis". In: *Software Engineering Journal* 8.4 (July 1993), pages 225–236.
- [6] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. Trivedi. "Fault triggers in open-source software: An experience report". In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. Nov. 2013, pages 178–187. DOI: 10.1109/ISSRE.2013.6698917.
- [7] F. Cristian. "Exception Handling". In: *Dependability of Resilient Computers*. 1989, pages 68–97.
- [8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. "Patterns in Property Specifications for Finite-state Verification". In: *Proceedings of the 21st International Conference on Software Engineering. ICSE '99*. Los Angeles, California, USA: ACM, 1999, pages 411–420. DOI: 10.1145/302405.302672.
- [9] M. Grottke and K. S. Trivedi. "A classification of software faults". In: *Journal of Reliability Engineering Association of Japan* 27.7 (2005), pages 425–438.
- [10] R. Hanmer. *Patterns for fault tolerant software*. John Wiley & Sons, 2013.
- [11] N. G. Leveson and P. R. Harvey. "Software fault tree analysis". In: *Journal of Systems and Software* 3.2 (June 1983), pages 173–181. DOI: 10.1016/0164-1212(83)90030-4.
- [12] T. J. McCabe. "A complexity measure". In: *Software Engineering, IEEE Transactions on* 4 (1976), pages 308–320.
- [13] N. Nagappan, T. Ball, and A. Zeller. "Mining metrics to predict component failures". In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pages 452–461.
- [14] A. Pretschner, D. Holling, R. Eschbach, and M. Gemmar. "A generic fault model for quality assurance". In: *Model-Driven Engineering Languages and Systems*. Springer, 2013, pages 87–103.
- [15] M. Stamatelatos and J. Caraballo. *Fault tree handbook with aerospace applications*. Office of safety and mission assurance NASA headquarters, 2002.
- [16] P. Tröger, F. Becker, and F. Salfner. "FuzzTrees-Failure Analysis with Uncertainties". In: *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*. IEEE. 2013, pages 263–272.

Physical Motion Displays

A Wearable Device for Producing a Strong Tactile Stimulus

Alexandra Ion

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

We propose communicating simple two-dimensional shapes/messages to users by dragging a physical tactor across their skin. The main benefit of this approach is that it produces not only a sense of touch, but also stretches the user's skin, thereby reaching more skin receptors than the currently prevalent modality, i.e., vibrotactile. We present a prototype that implements the concept as a wearable device that users wear on their wrist. We built it in a very compact form factor so that it fits under a watch.

1 Introduction

Wrist-worn devices are in continuous physical contact with the wearer's skin. This allows these devices to send simple messages to the user, e.g., by pulsing a message using a single vibrotactile actuator [2, 5, 8].

In order to allow sending more "mnemonic" messages, e.g., individual characters or simple icons, researchers extended the concept to two-dimensional arrays of vibrotactile actuators [5, 11]. Fading from one vibration motor to the next, these devices produce the illusion of a dot moving across the skin, which allows them to draw simple shapes [3].

This approach is popular because it is easy to implement and miniaturizes well to mobile and even wearable size. Unfortunately, vibrotactile units excite only a subset of tactile receptors—so called fast-acting receptors. We argue that this limits their effectiveness in communicating tactile messages.

Li et al. [6], in contrast, have explored a different type of tactile display. Their device "rubs" the user's skin using a tactor that moves back and forth. While their objective was to extend the expressiveness of tactile devices to better represent human-to-human touch, we believe that this approach reaches more skin receptors than vibrotactile. Because of its lateral motion, it also stretches the skin, thereby reaching so-called skin stretch receptors.

We appropriate this concept of moving a physical moving tactor across the user's skin as a means for delivering tactile messages to the user. In order to do so, we

extend the concept to 2D actuation. We call the resulting tactile displays physical motion displays. We show that this concept can be implemented in a compact form factor suitable for wearable use.



Figure 1: Physical motion displays drag a tacton over the wearer's skin in order to communicate a spatial message. Our self-contained miniaturized prototype is the size of a watch.

2 Related work

2.1 Vibrotactile feedback

Due to their compact size, vibrotactile actuators have not only been integrated in large objects like chairs [3, 13], but also in mobile devices [2, 14] and wearable devices [4, 12].

Vibrotactile devices allow sending non-visual messages to the user using patterns of varying amplitude. Tactons [2] are a general concept of tactile icons, which can be combined to send more complex messages. Lee et al. designed a wrist-worn device featuring three vibration motors that produced 24 distinguishable patterns [4]. Pasquero et al. [8] provided tactile feedback with a piezoelectric actuator that was built into a wrist-watch.

To communicate spatial messages to the user, researchers proposed 2D arrays of vibrotactile actuators. Tan and Pentland [11] proposed a 3×3 array of vibration motors to create a directional tactile display using apparent motion. Lee et al. [5] investigated sending directional and letter-like shapes (e.g. "L") to the user using a wrist-worn 4×4 array of vibrotactile units. More recently, Israr et al. [3] provided gamers with directional strokes produced by a 3×3 array of vibrotactile actuators mounted on the back of a chair.

2.2 Techniques that create tangential forces on the skin

Researchers have investigated technologies that provide directional tactile cues by exciting skin stretch receptors. Bark et al. [1] compared skin stretch feedback with vibrotactile feedback in a cursor positioning task and found a significant performance benefit for skin stretch feedback. These results confirm findings from physiology [7] that suggest the skin’s directional sensitivity is higher (8 mm on the forearm) than its location acuity (30 mm–40 mm on the forearm).

Gesture output [9] ports the concept to displays that address the user’s proprioceptive sense. They send eyes-free messages to users by dragging their thumb across a mobile device. Li et al. [6] proposed a device that “rubs” and “taps” the user’s skin. Stanley et al. [10] wrapped the concept in wrist-worn form factor. We build on this work, extend it to 2D, and use it to encode tactile messages.

3 Implementation

To demonstrate the concept of physical motion displays, we have implemented a watch-style prototype (shown in Figure 2). The device features a round actuation area of 32 mm diameter. It creates a tactile message by dragging the shown factor across the wearer’s skin.

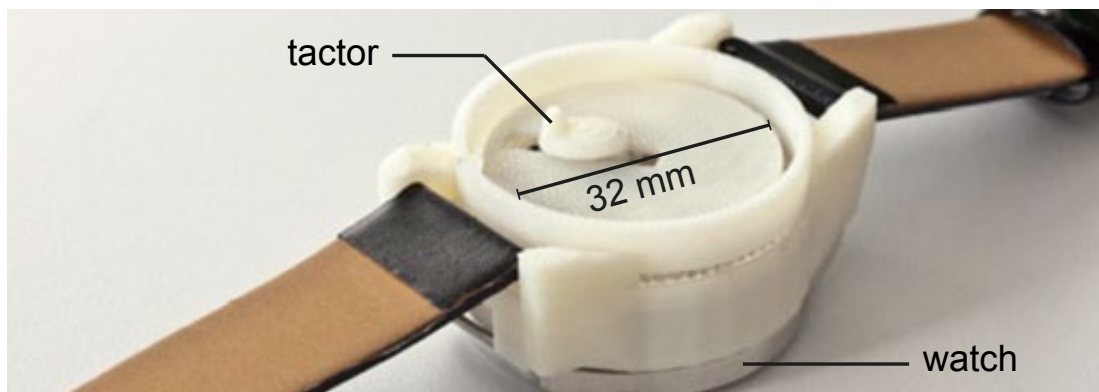


Figure 2: Our wrist-worn device actuates a 32 mm area in a 41 mm casing.

As discussed earlier, the main benefit of this design is that it produces a tangential force when dragging a factor across the user’s skin, because the motion of the factor excites the slow-acting as well as fast-acting mechanoreceptors [1].

3.1 Mechanical design

As shown in Figure 3, our prototype is based on a rotating mechanism, which implements a polar coordinate design i.e., it actuates the 2-dimensional factor motion in the form of azimuth and radius. We chose this design because it enables us to place all mechanical and electronic components “within the actuation area” yielding a self-contained device. Unlike the more traditional XY-table design, this allows us to fill the actuation area with components.

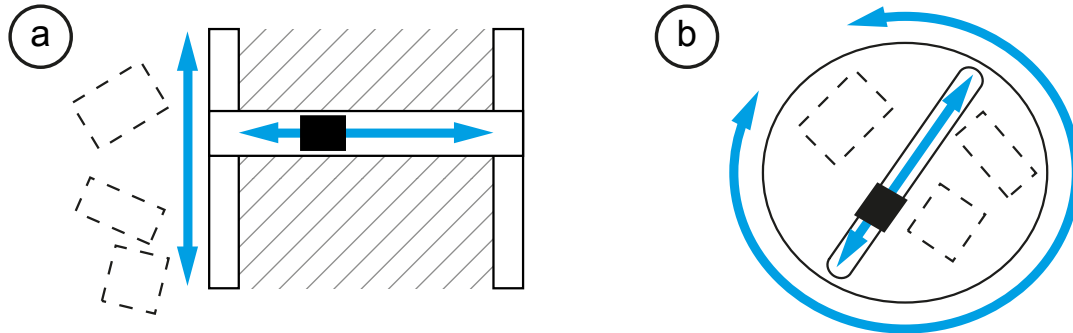


Figure 3: (a) Traditional XY table design. (b) Our polar coordinate design allows components to be placed on the actuation area.

3.2 Watch-size prototype providing tactile feedback

Our watch-size prototype is based on the polar coordinate design. The key component behind this design is a capsule that rotates inside of the casing. As shown in Figure 4, the casing features an inside gear along its inner circumference. A pinion rides along the inside gear. It is driven by a 6 mm motor via a worm drive. A linear micro servo (Spektrum AS2000L) drives the radius.

3.3 Electronics & Software

Figure 5 shows the design in semi-assembled state. The “bottom” layer of the device holds all mechanical parts. It is connected to the top layer, which holds electronics. The electronic part holds the battery, a motor controller and a microcontroller.

The microcontroller (ATtiny85) controls the motors. A li-ion battery powers the device. The device runs self-contained. Currently, the shapes to be played back are stored on the microcontroller and played back in fixed order as soon the device is

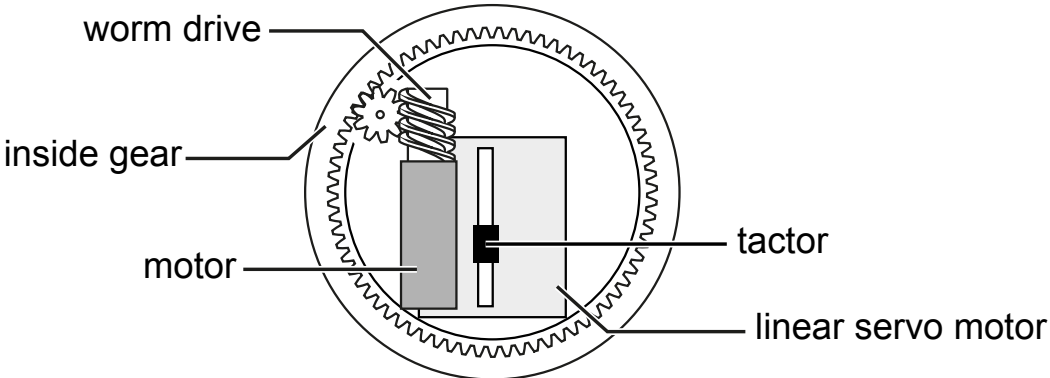


Figure 4: We use a worm drive with an inside gear for rotation and a linear servo for actuating the radius.

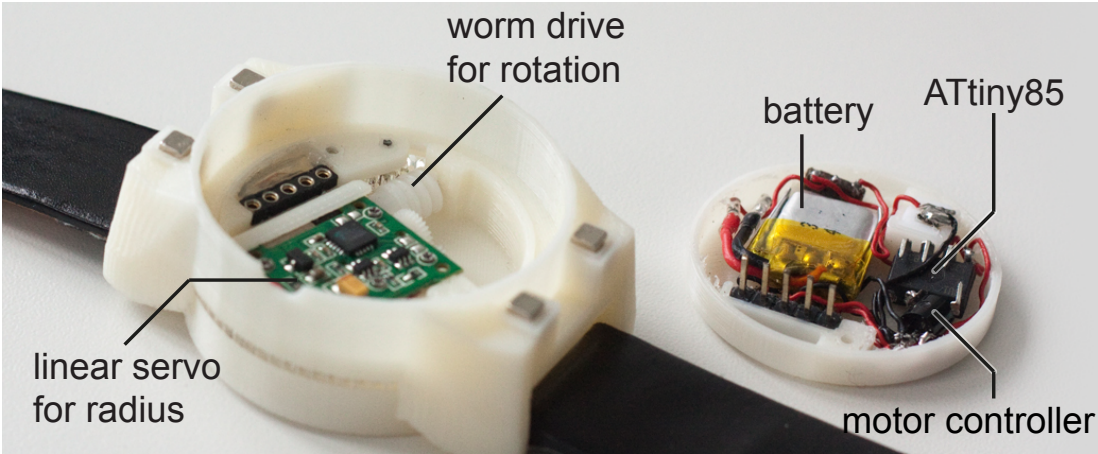


Figure 5: Electronics are placed on the actuation area and rotate with it.

switched on. We plan to send data from a notebook computer wirelessly (e.g. via the a Bluetooth module).

4 Conclusion

We presented the concept of physical motion displays that are able to display simple spatial shapes and a prototype that implements that concept. Our main contribution is that we demonstrate more effective ways of communicating tactile messages than the today's prevalent modality, i.e., vibrotactile. We also show that the approach is mechanically feasible and that we can fit it in a small package, opening up the potential for use in actual wearable devices.

In the future, we want to study the differences between sending vibrotactile stimuli and sending stimuli by producing physical motion on the user's skin. We want to learn how well participants recognize shapes drawn on their skin with physical motion compared to vibrotactile stimuli.

References

- [1] K. Bark, J. Wheeler, S. Premakumar, and M. Cutkosky. "Comparison of Skin Stretch and Vibrotactile Stimulation for Feedback of Proprioceptive Information". In: *Haptic interfaces for virtual environment and teleoperator systems, 2008. haptics 2008. symposium on*. Mar. 2008, pages 71–78. DOI: 10.1109/HAPTICS.2008.4479916.
- [2] S. Brewster and L. M. Brown. "Tactons: Structured Tactile Messages for Non-visual Information Display". In: *Proceedings of the Fifth Conference on Australasian User Interface - Volume 28. AUIC '04*. Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pages 15–23.
- [3] A. Israr and I. Poupyrev. "Tactile Brush: Drawing on Skin with a Tactile Grid Display". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '11*. Vancouver, BC, Canada: ACM, 2011, pages 2019–2028. DOI: 10.1145/1978942.1979235.
- [4] S. " Lee and T. Starner. "BuzzWear: Alert Perception in Wearable Tactile Displays on the Wrist". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '10*. Atlanta, Georgia, USA: ACM, 2010, pages 433–442. DOI: 10.1145/1753326.1753392.

- [5] S. C. Lee and T. Starner. "Mobile Gesture Interaction Using Wearable Tactile Displays". In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '09. Boston, MA, USA: ACM, 2009, pages 3437–3442. doi: 10.1145/1520340.1520499.
- [6] K. A. Li, P. Baudisch, W. G. Griswold, and J. D. Hollan. "Tapping and Rubbing: Exploring New Dimensions of Tactile Feedback with Voice Coil Motors". In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. UIST '08. Monterey, CA, USA: ACM, 2008, pages 181–190. doi: 10.1145/1449715.1449744.
- [7] U. Norrsell and H. Olausson. "Spatial cues serving the tactile directional sensibility of the human forearm". English. In: *The Journal of physiology* 478.3 (1994), pages 533–540.
- [8] J. Pasquero, S. J. Stobbe, and N. Stonehouse. "A Haptic Wristwatch for Eyes-free Interactions". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pages 3257–3266. doi: 10.1145/1978942.1979425.
- [9] A. Roudaut, A. Rau, C. Sterz, M. Plauth, P. Lopes, and P. Baudisch. "Gesture Output: Eyes-free Output Using a Force Feedback Touch Surface". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, 2013, pages 2547–2556. doi: 10.1145/2470654.2481352.
- [10] A. Stanley and K. Kuchenbecker. "Design of body-grounded tactile actuators for playback of human physical contact". In: *World Haptics Conference (WHC), 2011 IEEE*. June 2011, pages 563–568. doi: 10.1109/WHC.2011.5945547.
- [11] H. Tan and A. Pentland. "Tactual displays for wearable computing". English. In: *Personal Technologies* 1.4 (1997), pages 225–230. doi: 10.1007/BF01682025.
- [12] K. Tsukada and M. Yasumura. "ActiveBelt: Belt-Type Wearable Tactile Display for Directional Navigation". English. In: *UbiComp 2004: Ubiquitous Computing*. Edited by N. Davies, E. Mynatt, and I. Siio. Volume 3205. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pages 384–399. doi: 10.1007/978-3-540-30119-6_23.
- [13] Y. Yanagida, M. Kakita, R. Lindeman, Y. Kume, and N. Tetsutani. "Vibrotactile letter reading using a low-resolution tactor array". In: *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS '04. Proceedings. 12th International Symposium on*. Mar. 2004, pages 400–406. doi: 10.1109/HAPTIC.2004.1287227.

- [14] K. Yatani and K. N. Truong. “SemFeel: A User Interface with Semantic Tactile Feedback for Mobile Touch-screen Devices”. In: *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*. UIST '09. Victoria, BC, Canada: ACM, 2009, pages 111–120. DOI: 10.1145/1622176.1622198.

Profiling the Web of Data

Anja Jentzsch

Information Systems Group

Hasso-Plattner-Institut

anja.jentzsch@hpi.uni-potsdam.de

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

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

1 Problem Statement

Over the past years, an increasingly large number of data sources has been published as part of the Web of Data¹. At the time of writing the Web of Data comprised already roughly 1,000 datasets totaling more than 82 billion triples², including prominent examples, such as DBpedia, YAGO, and DBLP. Furthermore, more than 17 billion triples are available as RDFa, Microdata and Microformats in HTML pages³. This trend, together with the inherent heterogeneity of Linked Datasets and their schemata, makes it increasingly time-consuming to find and understand datasets that are relevant for integration. Metadata gives consumers of the data clarity about the content and variety of a dataset and the terms under which it can be reused, thus encouraging its reuse.

A Linked Dataset is represented in the Resource Description Framework (RDF). In comparison to other data models, e.g., the relational model, RDF lacks explicit schema information that precisely defines the types of entities and their attributes. Therefore, many datasets provide ontologies that categorize entities and define data types and semantics of properties. However, ontology information is not always available or may be incomplete. Furthermore, Linked Datasets are often inconsistent

¹The Linked Open Data Cloud nicely visualizes this trend: <http://lod-cloud.net>, accessed December 16, 2014.

²<http://datahub.io/dataset?tags=lod>, accessed December 16, 2014.

³<http://webdatacommons.org>, accessed December 16, 2014.

and lack even basic metadata. Algorithms and tools are needed that profile the dataset to retrieve relevant and interesting metadata analyzing the entire dataset.

Data profiling is an umbrella term for methods that compute metadata for describing datasets. Traditional data profiling tools for relational databases have a wide range of features ranging from the computation of cardinalities, such as the number of values in a column, to the calculation of inclusion dependencies; they determine value patterns, gather information on used data types, determine unique column combinations, and find keys.

Use cases for data profiling can be found in various areas concerned with data processing and data management [12]:

Query optimization is concerned with finding optimal execution plans for database queries. Cardinalities and value histograms can help to estimate the costs of such execution plans. Such metadata can also be used in the area of Linked Data, e.g. for optimizing SPARQL queries.

Data cleansing can benefit from discovered value patterns. Violations of detected patterns can reveal data errors, and respective statistics help measure and monitor the quality of a dataset. For Linked Data, data profiling techniques help validate datasets against vocabularies and schema properties.

Data integration is often hindered by the lack of information on new datasets. Data profiling metrics reveal information on, e.g. size, schema, semantics, and dependencies of unknown datasets. This is a highly relevant use case for Linked Data, because for many openly available datasets only little information is available.

Schema induction: Raw data, e.g. data gathered during scientific experiments, often does not have a known schema at first; data profiling techniques need to determine adequate schemata, which are required before data can be inserted into a traditional DBMS. For the field of Linked Data, this applies when working with datasets that have no dereferencable vocabulary. Data profiling can help induce a schema from the data, which then can be used to find a matching vocabulary or create a new one.

Data Mining: Finally, data profiling is an essential preprocessing step to almost any statistical analysis or data mining task. While data profiling focuses on gathering structural metadata about a dataset, data mining is usually more concerned with gaining new insights about data.

2 Relevancy

There are many commercial tools, such as IBM's Information Analyzer, Microsoft's SQL Server Integration Services (SSIS), or others for profiling relational datasets.

However these tool were designed to profile relational data. Linked Data has a very different nature and calls for specific profiling and mining techniques.

Finding information about Linked Datasets is an open issue on the constantly growing Web of Data due to the use cases mentioned above. While most of the Linked Datasets are listed in registries as for instance at the Data Hub (datahub.io), these registries usually are manually curated, and thus incomplete or outdated. Furthermore, existing means and standards for describing datasets are often limited in their depth of information. VoiDand Semantic Sitemapscover basic details of a dataset, but do not cover detailed information on the dataset's content, such as their main classes or number of entities. More detailed descriptions, e.g. information on a dataset's RDF graph structure, topics etc., is usually not available. Data profiling techniques can help to fulfil the need for information about, e.g. classes and property types, value distributions, or entity interlinking.

3 Related Work

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

One relevant use case is schema induction, because the lack of a fixed and well-defined schema is a common problem with Linked Datasets. One example for this field of research is the ExpLOD tool [9]. ExpLOD creates summaries for RDF graphs based on class and property usage as well as statistics on the interlinking between datasets based on `owl:sameAs` links.

Li describes a tool that can induce the actual schema of an RDF dataset [11]. It gathers schema-relevant statistics like cardinalities for class and property usage, and presents the induced schema in a UML-based visualization. Its implementation is based on the execution of SPARQL queries against a local database. Like ExpLOD, the approach is not parallelized. Both solutions still take approximately 10h to process a 10 million triples dataset with 13 classes and 90 properties. These results illustrate that performance is a common problem with large Linked Datasets.

An example for the query optimization use-case is presented in [10]. The authors present RDFStats, which uses Jena's SPARQL processor to collect statistics on Linked Datasets. These statistics include histograms for subjects (URIs, blank nodes) and histograms for properties and associated ranges.

Others have worked more generally on generating statistics that describe datasets on the Web of Data and thereby help understanding them. LODStats computes

statistical information for datasets from the Data Hub [2]. It calculates 32 simple statistical criteria, e.g. cardinalities for different schema elements and types of literal values (e.g. languages, value data types).

In [4] the authors automatically create VoID descriptions for large datasets using MapReduce. They manage to profile the BTC2010 dataset in about an hour on Amazon's EC2 cloud, showing that parallelization can be an effective approach to improve runtime when profiling large amounts of data.

Finally, the ProLOD++ tool allows to navigate an RDF dataset via an automatically computed hierarchical clustering [5] and along its ontology class tree [1]. Data profiling tasks are performed on each cluster or class dynamically and independently to improve efficiency.

4 Challenges

This section describes selected challenges that I identified as specific to profiling Linked Data and web data, as opposed to profiling relational tables.

Profiling along hierarchies

Vocabularies define classes and their relationships. Ontology classes usually are arranged in a taxonomic (subclass–superclass) hierarchy. While the Web of Data spans a global distributed data graph, its ontology classes build a tree with `owl:Thing` as its root. Analyzing datasets along the vocabulary-defined taxonomic hierarchies yield further insights, such as the data distribution at different hierarchy levels, or possible mappings between vocabularies or datasets.

Keys are clearly of vital importance to many applications in order to uniquely identify individuals of a given class by values of (a set of) key properties. In OWL 2 a collection of properties can be assigned as a key to a class using the `owl:hasKey` statement [8].

Nevertheless it has not yet fully arrived on the Web of Data: only one Linked Dataset uses `owl:hasKey` [7]. Thus, actually analyzing and profiling Linked Datasets requires manual, time consuming inspection or the help of tools.

Many languages have a so-called “unique names” assumption. On the web, such an assumption is not possible as real-world entities can be referred to with different URI references.

Heterogeneity

A common practice in the Linked Data community is to reuse terms from widely deployed vocabularies whenever possible, in order to increase homogeneity of descriptions and, consequently, easing the understanding of these descriptions. There

are at least 416 different vocabularies to be found on the Web of Data⁴. Some datasets, however, also exist without any defined or dereferencable vocabularies. And even if common vocabularies are used, there is no guarantee that the specifications and constraints are followed correctly.

Nearly all datasets on the Web of Data use terms from the W3C base vocabularies RDF, RDF Schema, and OWL. In addition, 191 (64.75 %) of the 295 datasets in the Linked Open Data Cloud Catalogue use terms from other widely deployed vocabularies [3].

As Linked Datasets cover a wide variety of topics, widely deployed vocabularies that cover all aspects of these topics may not exist yet. Thus, data providers often define proprietary terms that are used in addition to terms from widely deployed vocabularies in order to cover the more specific aspects and to publish the complete content of a dataset on the Web. Currently 190 (64.41 %) out of the 295 datasets use proprietary vocabulary terms with 83.68 % making the term URIs dereferenceable.

Topical profiling

The Web of Data covers not only a wide range of topics, it also contains a number of topically overlapping data sources. Since it provides for data-coexistence, everyone can publish data to it, express their view on things, and use the vocabularies of their choice. Integrating topically relevant datasets requires knowledge on the datasets' content and structure.

The State of the LOD Cloud document [3] gives an overview of the Linked Datasets for each topical domain but there is no fine-grained topical clustering for Linked Datasets. With 504 million inter-dataset links the Web of Data is highly interlinked; 1.6 % of all triples are links stating the relationship between the real-world entities in different datasets. Thus a huge topical overlap amongst the datasets is given.

Large scale profiling

With more than 82 billion triples distributed among roughly 1,000 Linked Datasets and more than 17 billion triples available as RDFa, Microdata and Microformats, the need for efficient profiling methods and tools is apparent.

The runtime of profiling tasks as presented in Sec. 7 takes up to hours, e.g., for determining property co-occurrences [6]. Profiling tasks often have the same preprocessing steps, e.g., filtering or grouping the dataset. Thus there is a large incentive and potential to optimize the execution of multiple scripts.

5 Research Questions

The main question in my doctoral research is:

⁴<http://lov.okfn.org/>, accessed December 16, 2014.

What are the challenges that are specific to profiling Linked Data and web data, as opposed to profiling relational tables?

After identifying four selected challenges, the following questions arise:

Profiling along hierarchies Does analyzing Linked Datasets along the vocabulary-defined taxonomic hierarchies, such as the data distribution at different hierarchy levels, yield further insights?

Heterogeneity How does profiling help analyzing the heterogeneity on the Web of Data?

Topical profiling How can topical clusterings for unknown datasets on the constantly growing Web of Data be derived efficiently?

Large scale profiling How can these huge amounts of Linked Data be profiled efficiently?

6 Approach

My approach to address the research questions is to tackle each of the identified challenges. The main goal is to reuse existing profiling techniques and adapt them to the Linked Data world.

This section presents possible and if available developed solutions by me to the presented challenges.

Profiling along hierarchies

One example of profiling tasks along the class hierarchy is determining the *uniqueness* of properties as well as the unique property combinations, which can bring insights into the property distribution inside the dataset. It allows for finding relevant (key-candidate) properties for each level in the class hierarchy and see if the relevance is increasing or decreasing along hierarchy.

As I have found, due to the sparsity on the Web of Data, usually neither full key-candidates of properties nor unique property combinations can be retrieved using traditional techniques. Thus I defined the concept of *keyness* as the Harmonic Mean of uniqueness and density of a property⁵, allowing to find potential key candidates.

Heterogeneity

Data profiling can be used to provide metadata describing the characteristics of a dataset, for instance its topic and more detailed statistics, like the main classes and properties. Furthermore, data profiling can not only determine the usage of vocabularies but also help understanding and reusing existing vocabularies. Additionally, it can assist when mapping vocabulary terms.

⁵We define the *uniqueness* of a property as the number of unique values per number of total values for a given property; and the *density* of a property as the ratio of non-NULL values to the number of entities.

Topical profiling

The first profiling task is, of course, to discover (and possibly label) these topical clusters. The discovery of which topics an unknown dataset is even about, is already a very helpful insight. Next, any profiling task can be executed on data of a particular topic and compared against the metadata of other topics.

Large scale profiling

The runtime of the profiling tasks takes up to hours already on 1 million triples, e.g., for determining property co-occurrences [6]. A number of different approaches can be chosen when trying to optimize the execution time of algorithms dealing with RDF data in general and data profiling tasks in particular.

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

Parallelization: When dealing with large datasets, a good approach for improving performance is to perform calculations in parallel when possible [12]. This can be done on different levels: dataset, profiling run, profiling task and triples. Cluster-based parallelization based on MapReduce is a reasonable choice when working with Linked Data.

Multi-Query Optimization: A data profiling run usually consists of a number of different tasks, which all have to be computed on the same dataset. Depending on the set of data profiling tasks, different tasks may require the same preprocessing steps, or perform similar computation steps. Overall execution time can be reduced by avoiding duplicate computations. Similar computation steps may be interweaved to reduce runtime and I/O costs. If different tasks require similar intermediate results, these can be stored in materialized views.

7 Preliminary Results

Initially, I have defined a set of 56 useful data profiling tasks along various groupings to profile Linked Datasets. They have been implemented as Apache Pig scripts and are available online⁶.

Furthermore, I illustrated the Web of Data's diversity with the results for four different Linked Datasets [6].

⁶<http://github.com/bforchhammer/lodop/>, accessed December 16, 2014.

Profiling along hierarchies

When analyzing the uniqueness in the class hierarchy for DBpedia, I found that there are properties that become more specific by class level, thus their uniqueness gets higher for subclasses. For instance, `dbpedia:team` becomes more unique for athletes than it is for all persons. I also found properties that are generic, their uniqueness stays constant throughout the class hierarchy. For instance, `dbpedia:birthDate` is not specific to persons or their subclasses.

Furthermore, I have defined the concept of *keyness* of the property to gap the sparsity on the Web of Data and thus the possibility to find potential key candidates where traditional approaches fail.

Large scale profiling

We have addressed the different approaches to improve Linked Data profiling performance and not only developed LODOP, a system for executing, benchmarking and optimizing Linked Data profiling scripts on Hadoop but also developed and evaluated 3 multi-query optimization rules [6]. We experimentally demonstrated that they achieve their respective goals of optimizing the amount of MapReduce jobs or the amount of data materialized between jobs, thus reducing the profiling tasks runtimes by 70 %.

8 Evaluation Plan

For the evaluation, there are three main lines of interest.

Metadata The main goal is to provide comprehensive dataset metadata that helps analyzing the datasets. The metadata can be evaluated on quantity and quality wrt existing metadata on the Data Hub, VoiD and Semantic Sitemaps.

Usability Tools and techniques should have a high usability in terms of results being presented in both human and machine readable ways to achieve better decision making when working with datasets.

Performance evaluation Various aspects of the developed tools should be tested for performance, especially the for huge amounts of data as it is present on the Web of Data.

9 Reflections and Conclusion

The main difference in my approach with existing work on Linked Data profiling is to address the shortcomings mentioned in section 3, in particular gathering comprehensive metadata in an efficient way. Within my research I am building on existing

profiling techniques for relational data and adapting them according to the different nature of Linked Datasets.

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

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

References

- [1] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. "Mining and Profiling RDF Data with ProLOD++". In: *Proceedings of the International Conference on Data Engineering (ICDE)*. Demo. 2014.
- [2] S. Auer, J. Demter, M. Martin, and J. Lehmann. "LODStats – an extensible framework for high-performance dataset analytics". In: *Proceedings of the Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*. 2012.
- [3] C. Bizer, A. Jentzsch, and R. Cyganiak. *State of the LOD Cloud*. 2011. URL: <http://lod-cloud.net/state/>.
- [4] C. Böhm, J. Lorey, and F. Naumann. "Creating VoiD Descriptions for Web-scale Data". In: *Journal of Web Semantics* 9.3 (2011), pages 339–345.
- [5] C. Böhm, F. Naumann, Z. Abedjan, D. Fenz, T. Grütze, D. Hefenbrock, M. Pohl, and D. Sonnabend. "Profiling Linked Open Data with ProLOD". In: *Proceedings of the International Workshop on New Trends in Information Integration (NTII)*. 2010.
- [6] B. Forchhammer, A. Jentzsch, and F. Naumann. "LODOP - Multi-Query Optimization for Linked Data Profiling Queries". In: *ESWC Workshop on Profiling & Federated Search for Linked Data (PROFILES)*. 2014.
- [7] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres. "OWL: Yet to arrive on the Web of Data?" In: *WWW Workshop on Linked Data on the Web (LDOW)*. 2012.
- [8] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 2009.
- [9] S. Khatchadourian and M. P. Consens. "ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud". In: *Proceedings of the Extended Semantic Web Conference (ESWC)*. Heraklion, Greece, 2010.

- [10] A. Langegger and W. Wöß. “RDFStats – An Extensible RDF Statistics Generator and Library”. In: *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*. Los Alamitos, CA, USA, 2009, pages 79–83.
- [11] H. Li. “Data Profiling for Semantic Web Data”. In: *Proceedings of the International Conference on Web Information Systems and Mining (WISM)*. 2012.
- [12] F. Naumann. “Data Profiling Revisited”. In: *SIGMOD Record* 42.4 (2013).

Enterprise Simulations based on Value Driver Trees

Stefan Klauck

Enterprise Platform and Integration Concepts
Hasso-Plattner-Institute
stefan.klauck@hpi.de

Value driver trees are a well-known method to model dependencies such as the definition of key performance indicators. While the models have well-known semantics, they lack the right tool support for business simulations, because a flexible implementation that supports multidimensional, hierarchical value driver trees and data bindings is very challenging. My research tackles this problem by developing an approach to build enterprise simulations which base on value driver trees. This new approach consists of two parts: the definition of a simulation meta model and the concept of a simulation tool. The simulation meta model describes how simulation model instances look like, meaning the structure of the simulation model graph, the data binding to a database and the parameterization of the model to simulate data changes. The simulation tool is used to create and edit simulation model instances and run simulations. Besides the formal description of the approach, this report presents a prototypical implementation of the simulation tool.

1 Introduction

Companies measure their success based on key performance indicators (KPIs) as the operating profit or return of investment. Thereby, they are not only interested in the current state of these KPIs, but also in forecasted values. Companies invest a significant amount of time in their yearly budgeting process and the resulting quarterly or monthly forecasts. Value driver trees such as the DuPont model [1] are a well-known technology to model KPIs with linear equations and dependencies among another, as introduced by Zwicker [8]. Figure 1 shows a driver tree for the operating profit. While their semantics are well-known, there is a lack of tool support to enable simulations based on value driver trees.

For many years, the biggest challenge was the speed to access the data the simulations base on. Value drivers comprise aggregated values for multiple dimensions, e.g. time and location. To allow flexible simulations on all dimension levels for the operating profit, sales documents with its line items have to be scanned. The corresponding tables comprise billions of records, specifying the for the simulation rele-

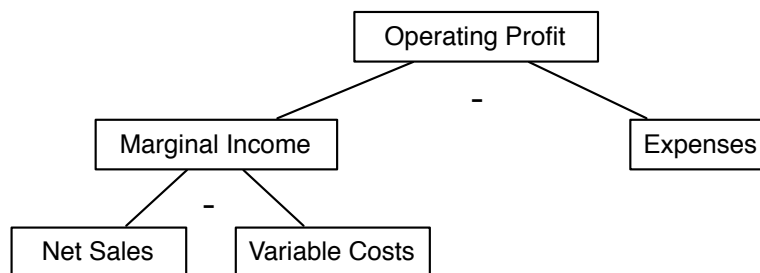


Figure 1: Value driver tree for the operation profit

vant attributes date, product, sales volume, price and customer, but also hundreds of other attributes. The advent of columnar in-memory databases has increased the performance of queries accessing few attributes of large datasets, which enables the development of new enterprise applications on top of it, e.g. enterprise simulations [4, 5]. A flexible approach can replace existing forecast tools like Microsoft (MS) Excel, which have several drawbacks:

- Lack of flexibility: Existing simulation tools are targeted for specific processes and are difficult to modify or extend to capture new use cases.
- Complexity: Value drivers can be defined to comprise multiple dimensions as time, customer, location, product. Many of them can be hierarchically structured. Simulation tools have to support the drilldown into hierarchies and filter for specific dimension values.
- Inconsistencies: Simulations with MS Excel enable flexibility at a very high level, but they are missing efficient data binding techniques which can cause inconsistencies in the simulation with respect to the underlying data.
- Missing collaboration: Simulations and what-if analyses are often done in many iterations, with many users in potentially different roles. Collaboration is a cumbersome and error-prone process with respect to inconsistencies.

My research targets these deficiencies and tries to find a solution to flexibly create, configure and run simulations interactively. This comprises two things: First, a way to define multidimensional dependency graphs, the specification of data bindings between nodes and data of database tables, and the supported simulations. Second, a concept to define and edit the graph as well as to specify the parametrization for the simulation.

This report describes the current state of that research. After this short introduction into the problem domain, Section 2 presents challenges for dependency graphs and

derives requirements for the simulation model and the tool. Section 3 describes the solution approach in a formal way, before an implementation is shown in Section 4. Related work is presented in Section 5. Section 6 closes the report by presenting the conclusions and offering an outlook for future work.

2 Problem Description

The operating profit is an important KPI of companies to measure their success. This section uses it as example to describe the challenges and derives requirements for the model definition and the simulation tool.

2.1 Model Definition

Figure 2 shows a more comprehensive calculation model for the operating profit. It extends the model shown in Figure 1 by specifying the calculation for net sales and variable costs. Both nodes are influenced by the sales volume. In general, nodes can influence multiple other nodes. That is why, simulation models should be expressed as dependency *graphs* instead of *trees*.

Operations connect two or more nodes. Thereby, the operation specifies not only the calculation for a single node, but the dependency between the nodes. The operation between marginal income, net sales and variable costs in Figure 2 means that given two values of these three, the third one can be calculated.

The values of nodes can be queried from data sources, e.g. database tables, or calculated with values of connected nodes. The data source specification is independent of the dependency graph and is called data binding. When specifying, the dependency graph and the data binding, one has to ensure that the values of all nodes can be calculated unambiguously, meaning that the data has to be sufficient and consistent.

Besides the values, the data binding specifies the dimensions and finest hierarchy level for each node. A single node with a data source contains values for all attribute combinations of the dimensions. Dimensions and hierarchies allow to filter the data. Instead of querying the sales volume of the whole company, one could request it for a specific product or a selected month. The available dimensions and hierarchy levels for nodes with calculated values depend on the connecting operation and dimensions of the connected nodes (cf. Figure 2).

The following points summarize the requirements to model multidimensional dependency graphs.

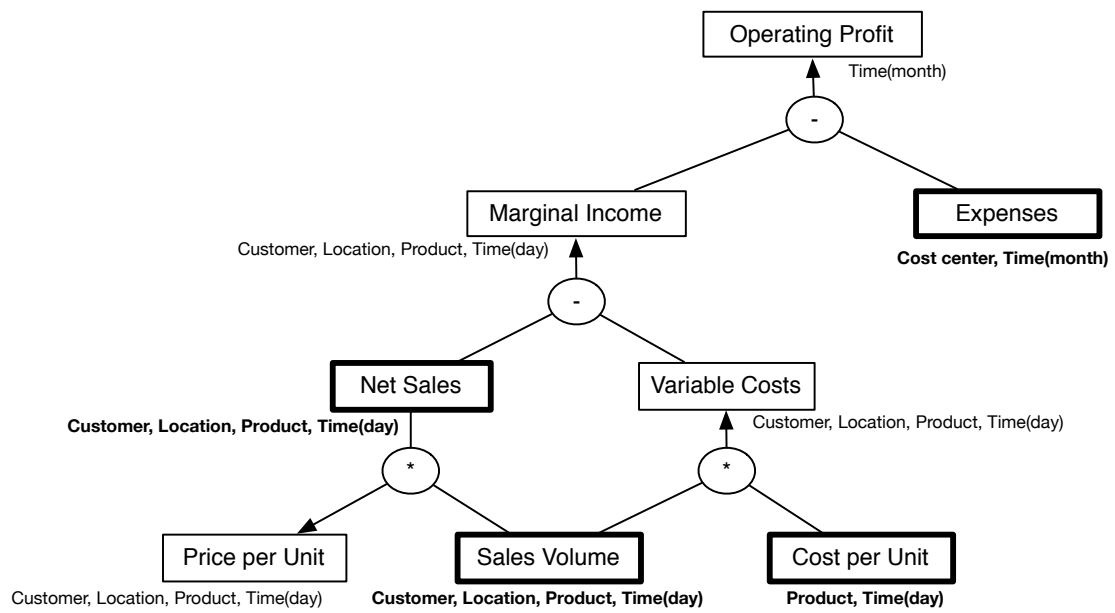


Figure 2: Dependency graph for the operation profit: Bold nodes represent data-sources with its dimensions. The values of nodes which are marked by arrows can be calculated recursively by following the operations from which the arrows come.

1. **General driver model:** The driver model describes the nodes and its relationships via operations. It has to support multiple dimensions, which can be hierarchically structured.
2. **Data binding:** Data bindings connect nodes with data from database tables. They define the available dimensions and the finest level of hierarchy.

2.2 Simulation Tool

The structure of the dependency graph can change. New KPIs can be defined, which can be the basis for new simulation scenarios. Data bindings should be editable. Finally, the simulation tool should allow to parameterize the node values to simulate changes. The changes of node values can propagate via connected nodes through the whole model. The simulation model has to define how changes affect connected nodes. Using the connection between net sales, price per unit and sales volume as example: Increased net sales can result in an increased price per unit or sales volume. Besides it could be desirable to ignore the dependency between these nodes, because one is interested in the effects for the other dependencies of net sales. Following requirements have been identified for the simulation tool.

1. Model editor: The model editor should enable users to create and change calculation models as described in Section 2.1.
2. Simulation mode: Simulations specify changes of the queried or calculated values. They can be defined on all nodes and additionally specify filter criteria to limit the affected data. The simulation model has to define which nodes are allowed to influence the values of which connected nodes.

3 Approach

This section describes the approach to develop generic simulation models. It is divided into three parts: the definition of simulation models, the declaration of data bindings and the introduction of the simulation tool concept. The following subsections present the single parts in detail.

3.1 Simulation Model

Simulation models are hypergraphs. Each node has a name, available dimensions and can be connected with other nodes by *operations*, which are hyperedges. The simulation model specifies the available dimensions with its hierarchy levels. Time can for example be hierarchically structured into years, months, days. The dimensions of a specific node are determined by the data binding or the operation and dimension specification of connected nodes in case the node has no data source.

Operations define the dependencies between nodes. Each operation has to define the way it calculates the values for the result nodes. Thereby, the approach can define the calculations of common operations as addition, subtraction, multiplication and division, which work in a similar way as for one-dimensional data. Combining the values of nodes with different levels of hierarchies may require a preaggregation phase. In other cases, the operation cannot be executed at all. The signature and algorithm of each operation has to be defined. The addition operation connects two nodes with the same dimensions and creates a node with values for these specific dimensions. The hierarchy level of the resulting node is the minimum of both input nodes.

3.2 Data Binding

Nodes of the value driver tree can obtain their values in two ways. On the one hand, they can query their data directly from database tables. On the other hand, they can calculate their values by solving the equation defined by the operation between

connected nodes and itself. For the first case, data bindings are required. This report focuses on relational databases as data source, although it is also possible to use other sources. Data bindings define the database connection and SQL queries for all supported dimension filter criteria and hierarchy levels. A generic way to implement it without specifying each single possible query is to define the SQL query to request the data for all dimensions at the lowest hierarchy levels. Based on this data, filter criteria and aggregations can be applied to support selections and views of the data at higher hierarchy levels.

3.3 Concept of Simulation Tool

The simulation tool supports creating, changing and running simulation models. In general, different user groups are responsible for model editing and specifying simulation scenarios. The choice of KPIs and its adaption to better fit the company's structure is executed at management level. As the management level does not have a detailed knowledge of the data schema, it will delegate the task of defining data bindings to more technical staff. The individual simulations can then be done in collaboration with controllers. Hence, the tool has two modes to separate these activities: one for editing simulation model instances, the other for simulating data changes. The editor mode comprises an interface to create dependency models as described in Section 3.1 and allows to specify the data binding. A graphical modeling tool can be used as user-friendly interface, as well as a text editor to describe the model with structured text, JSON or XML.

Simulation model instances could then be validated before switching to the simulation mode. The simulation mode allows drilldowns, i.e. filtering the node values by dimensions and hierarchy levels. The number of dimensions specifies the number of drilldown possibilities. The depth of the hierarchy specifies how deep the drilldown goes. If a node does not support the current drilldown level, it is excluded from the simulation. For each drilldown level, simulation parameters can be specified. Thereby, the model has to define directions for edges to specify which nodes are affected by the parameterization of connected nodes.

Figure 3 shows an example of a simulation. Based on the dependency graph as given in Figure 2 with data sources for net sales, sales volume, cost per unit and expenses, the other nodes can be calculated forming the graph as presented in Figure 3a. The values are the basis of the simulation. The double arrows specify in which directions changes are propagated. Each value, independent whether it was directly queried from a database table or was calculated, can be overwritten by simulation parameters. Figure 3b shows the simulation of an increased price per unit (\$ 30 instead of \$ 15) and how it propagates via the endpoints with double

arrows. When parameterizing nodes with incoming double arrows, e.g. setting net sales to \$ 200, the connected nodes are ignored (cf. Figure 3c).

4 Simulation Tool Implementation

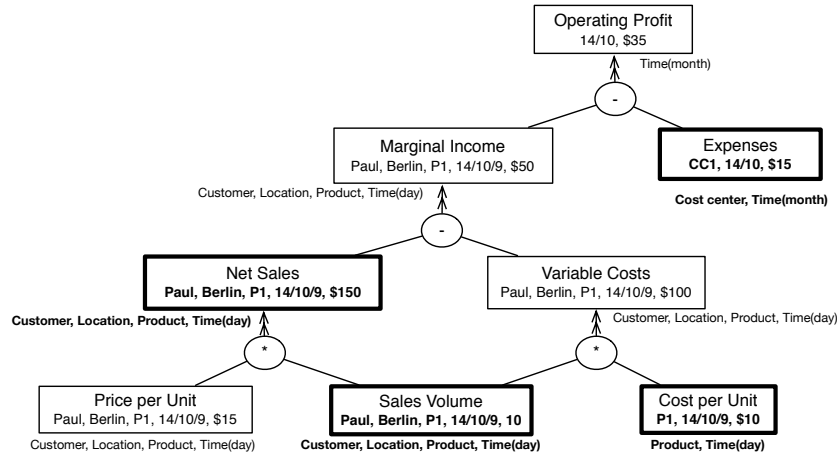
The simulation tool concepts has been implemented as a prototype. A graphical editor, implemented with *jsPlumb*¹, can be used to create and edit calculation models. JSON is used to persist the calculation model, which specifies the dimensions, nodes and operations as objects. Nodes with data sources have an *sql* property. The prototype uses a single database so that connection details are not stored within the model. The dimensions and level names map directly to column names so that mappings are given implicitly. The available dimensions for nodes with no datasource are calculated on model load or change. Calculations are described as strings and are parsed to calculate the values for corresponding nodes. The *influenced* property states that changes for the specified node do not result in changes for the connected nodes. Instead they are ignored.

5 Related work

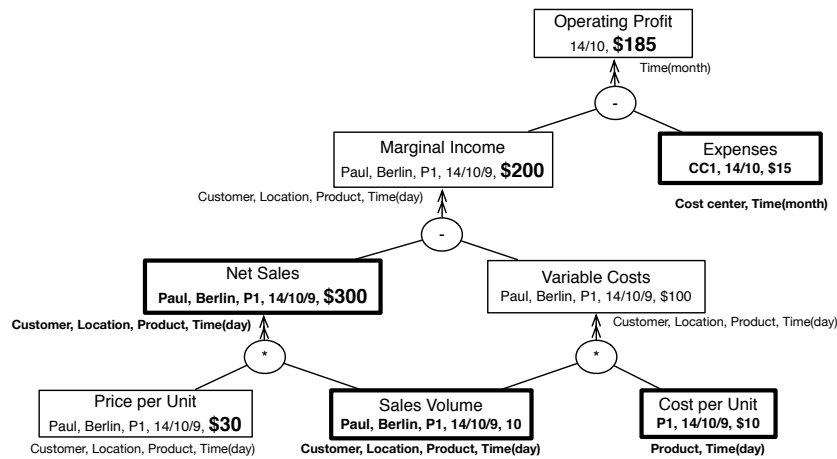
Besides sophisticated controlling software, spreadsheets, MS Excel in particular, are the main simulation tool in enterprises. The RS-Controlling-System² offers Excel templates for target-performance comparisons of balance sheets, profit and loss statements and capital flows. Spreadsheets have an easy to understand interface, but do not build on consolidated enterprise data, which is stored in a RDBMS. On the other side, SQL lacks the support for array-like calculations as Witkowski et al. claim in [6]. Their idea is to combine both and offer a spreadsheet-like computation in RDBMS through SQL extensions. In [7], they continue that research and introduce a way to translate MS Excel computations in SQL. The simulation model is thereby expressed in MS Excel. This approach comes with two drawbacks. First, it does not encapsulate the definition of the simulation model so that it is not tangible, but only expressed by multiple formulas spread over many table cells. Second, MS Excel is bound to the two-dimensional representation and cannot visualize graphical dependencies very well.

¹<https://jsplumbtoolkit.com/>, accessed December 16, 2014.

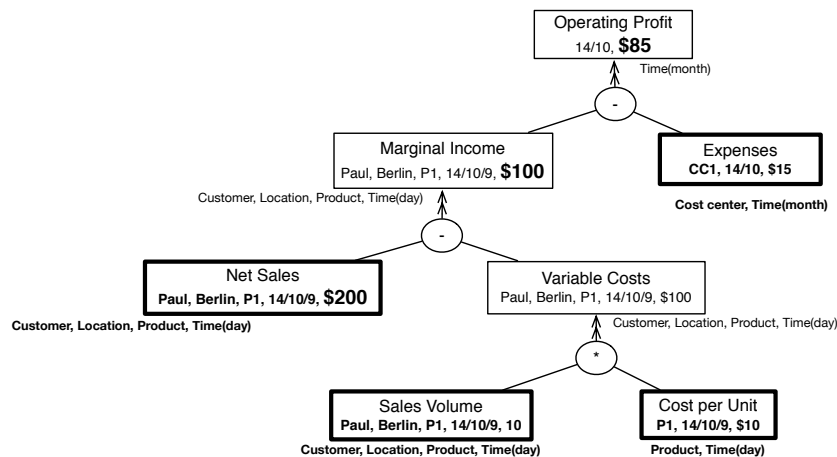
²<http://www.controllingportal.de/Shop/Excel-Tools/RS-Controlling-System.html>, accessed December 16, 2014.



(a) Initial state with data from the database table



(b) Simulation with an absolute value for price per unit



(c) Simulation with an absolute value for net sales

Figure 3: Examples of Simulations

```
1 {
2   "dimensions": {
3     "time": ["year", "month", "day"],
4     "customer": ["customer"],
5     "location": ["country"],
6     "product": ["product"],
7   },
8   "node1": {
9     "name": "Operating Profit",
10  },
11  "node2": {
12    "name": "Marginal Income",
13  },
14  "node3": {
15    "name": "Expenses",
16    "sql": "SELECT cost center, month, sum(amount)
17           FROM expenses"
18    "dimensions": {
19      "Cost center": "cost center", "time": "month"
20    },
21  },
22  "operation1": {
23    "calculation": "node1 = node2 + node3",
24    "influenced": "node1"
25  }
```

To describe simulation models, Golfarelli et al. introduce a methodology to formally express and build what-if analyses in [3]. They divide the process to design simulations into seven phases: goal analysis, business modeling, data source analysis, multidimensional modeling, simulation modeling, data design and implementation, and validation. To describe the actual simulation model they propose an extension of UML 2 activity diagrams [2]. However, they do not focus on how to develop generic simulation applications that are based on the formal descriptions.

6 Conclusion

This paper proposes a new approach to build enterprise simulations. It consists of a multidimensional generic model to describe the dependencies of value drivers as basis of simulations and the concept of a simulation tool to create, edit and parameterize model instances to simulate scenarios. The simulations base, thereby, on enterprise data, stored in columnar databases.

The main benefits of the solution can be summarized as follows: The generic enterprise simulation approach enhances the focus on key factors that drive the business. Further, it reduces the planning effort and drives cross-functional collaboration and alignment. By directly using transactional data as a basis for the simulation, it minimizes consistency issues. Most importantly, the simulation solution provides leadership with visibility into different alternatives to achieve desired outcomes.

The calculation model and database binding are described in a semi-formal way in prose text and JSON. My future work will formalize the model and the multidimensional operations. Of special interest are the possibilities and specifications of simulation changes. Beside, an investigation of optimization strategies for the required database queries, which base on groupings with different granularities, will be done.

References

- [1] A. Chandler and S. Salsbury. *Pierre S. Du Pont and the Making of the Modern Corporation*. BeardBooks, 2000.
- [2] M. Golfarelli and S. Rizzi. "UML-Based Modeling for What-If Analysis". In: *DaWak*. 2008, pages 1–12. DOI: 10.1007/978-3-540-85836-2_1.
- [3] M. Golfarelli, S. Rizzi, and A. Proli. "Designing What-if Analysis: Towards a Methodology". In: *DOLAP*. 2006, pages 51–58. DOI: 10.1145/1183512.1183523.

- [4] H. Plattner. "A common database approach for OLTP and OLAP using an in-memory column database". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*. Edited by U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul. ACM, 2009, pages 1–2. doi: 10.1145/1559845.1559846.
- [5] H. Plattner. "The Impact of Columnar In-Memory Databases on Enterprise Systems". In: *PVLDB* 7.13 (2014), pages 1722–1729.
- [6] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Shen, and S. Subramanian. "Spreadsheets in RDBMS for OLAP". In: *ACM SIGMOD*. 2003, pages 52–63. doi: 10.1145/872757.872767.
- [7] A. Witkowski, S. Bellamkonda, T. Bozkaya, A. Naimat, L. Sheng, S. Subramanian, and A. Waingold. "Query by Excel". In: *VLDB*. 2005, pages 1204–1215.
- [8] E. Zwicker. *Prozeßkostenrechnung und ihr Einsatz im System der integrierten Zielverpflichtungsplanung*. Techn. Univ. Berlin, 2003.

Proprioceptive Interaction

Pedro Lopes

Human Computer Interaction - Prof. Patrick Baudisch

Hasso-Plattner-Institut

pedro.lopes@hpi.uni-potsdam.de

Wearable and mobile devices allow users to access computing on-the-go. However, these devices interfere with the users' primary task, such as having a conversation, because they heavily rely on the visual sense. We propose a new way of eyes-free interaction for wearable devices. It is based on the user's proprioceptive sense, i.e., rather than seeing, hearing, or feeling an outside stimulus, users feel the pose of their own body. We have implemented a proof-of-concept device called Pose-IO that offers input and output based on proprioception. Users wear Pose-IO on their forearms. They communicate with Pose-IO through the pose of their wrists. Users enter information by performing an input gesture by flexing their wrist, which the device senses using a 3-axis accelerometer. Users receive output from Pose-IO by finding their wrist posed in an output gesture, which Pose-IO actuates using electrical muscle stimulation. This mechanism allows users to interact with Pose-IO without visual or auditory senses, but through the proprioceptive sense alone.

1 Introduction: why wearable devices?

As computing devices decrease in form-factor, from desktops to mobile phones, the tendency for mobility emerges. This is a manifestation of our desire to access computing power on-the-go, from gaming to cloud-based productivity applications, which happens through small devices that fit in our pockets or through devices that become an extension of our clothing, such as smartwatches and glasses.

Unfortunately, most wearable I/O devices use modalities that require visual or auditory attention and interfere with common primary tasks, such as having a conversation. To enable devices to be interference-free from daily tasks, researchers focus on developing eyes-free interactions, i.e., that do not require the visual channel (which is typically busy with the primary task). The predominant eyes-free non-auditory type of output is vibrotactile output. Unfortunately, it is subject to a range of limitations. Karuei et al. found that vibrotactile output degrades in mobile situations, such as while walking [4]. Furthermore, vibrotactile output was found to offer only limited bandwidth [8] and is hard to learn because it lacks mnemonic properties [6]. To cope with these limitations of vibrotactile feedback, Roudaut et al.

proposed an eyes-free symmetric input and output technique [9]. Their prototype, Gesture Output, unifies input and output vocabularies in the form of 2D finger gestures on a mobile phone with an actuated touchscreen. Here, users feel the system's reply in the form of a gesture, which is sensed, not visually, but through their sense of proprioception.

2 Background: what is proprioception?

The human proprioceptive sense allows sensing the position, orientation, and movement of limbs, joints, and muscles [2]. Some debate has taken place whether proprioceptive and kinesthetic are separate senses. However, the medical community points that the term proprioception encompasses kinesthesia within [2]. More importantly, research suggests that proprioception is independent from the visual and auditory channels [2, 3]. Apart from research in cognitive and medical domain, proprioception has been often evoked in HCI with regards to haptic feedback. Bark et al. showed how the skin stretch sensation (part of proprioception), which is elicited once a muscle is moved, is superior to vibrotactile feedback for input tasks in which the user controls a cursor [1].

3 Related work

The work proposed in this paper builds on proprioceptive feedback and wearable interfaces for eyes-free interaction. Furthermore it is inspired by different approaches to muscle-based input or output. Given that we already presented wearable devices and proprioception, we now turn into muscle input and output.

3.1 Reading Muscle Activity

More recently, researchers started measuring muscle activity as a means of achieving small wearable input devices that are controlled using finger flexion, such as the forearm electromyography input device by Saponas et al. [10]. We gathered inspiration from these findings and designed Pose-IO to be a wearable bracelet mounted onto the user's arm, which can read input signals from hand motion.

3.2 Actuating Users: Mechanically and using EMS

Actuating the user's body using mechanical actuators is common in haptics. The most common mechanical approaches actuate the body using pulley mechanisms. These mechanical actuators displace the user's body by pulling one limb using motors; a notable example is the SPIDAR [12]. The aforementioned Gesture Output [9] is yet another example of such a system. Likewise, exoskeletons such as FlexTorque [12] are a sub-class of pulley systems but attach the complete infrastructure to the user. While mechanical actuators are great for their accuracy and output power, they require attaching exoskeletons, motors and large batteries to the user.

As such, Electrical Muscle Stimulation (EMS) has been proposed as a means of mobile actuation [7]. EMS has been applied in a range of interactive applications, ranging from gaming [5] to assistive learning (e.g., Possessed Hand [11]). Lopes et al. demonstrated EMS as a means to implement force-feedback in mobile devices [7]. Since these systems do not use a control loops, actuation works through pre-calibration.

4 Proprioceptive Interaction

Figure 1 illustrates the concept of proprioceptive interaction. Proprioceptive interaction allows for input and output by posing one of the user's limbs, here the user's wrist. Users enter information by performing an input gesture, here flexing their wrists inwards. The device senses this using its accelerometer. Users can perform such a gesture eyes-free, as their proprioceptive sense informs them about the position of their wrist. Users receive output from a proprioceptive interface by finding their body posed in an output gesture, here again the wrist. Also this they perceive eyes-free by means of their proprioceptive sense.

The interaction shown in Figure 1 is of the "purest" form in that input and output occur through the same limb, here the wrist. This "symmetric" interaction results in a particularly intuitive interaction [9]. However, proprioceptive input and output may also occur through different limbs, e.g., when the application requires more input than output or more output than input. We call this asymmetric proprioceptive interaction, which is depicted in Figure 2.

4.1 Prototypes

As shown in Figure 3, Pose-IO's hardware design has the form of a bracelet with attached electrodes. The bracelet is comprised of 3D-printed sections, each hosting parts of the embedded electronics.

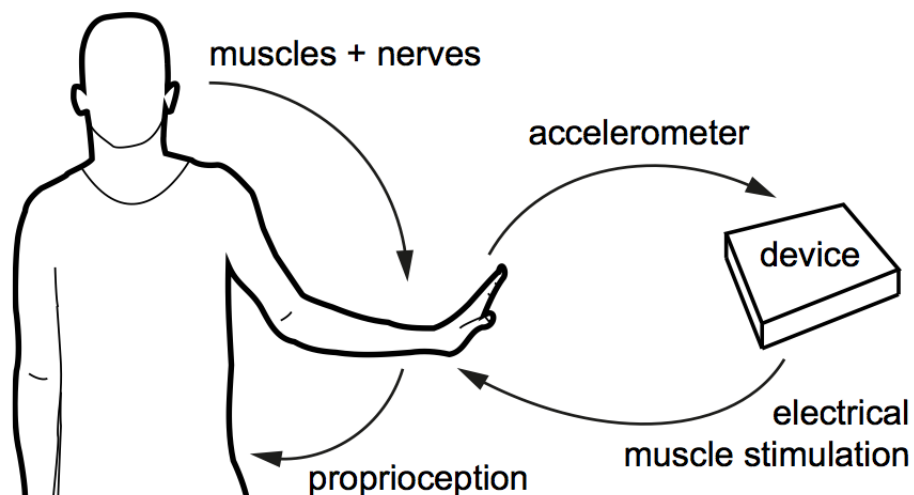


Figure 1: Symmetric proprioceptive interaction revolves around the pose of one of the user's limbs, here the wrist.

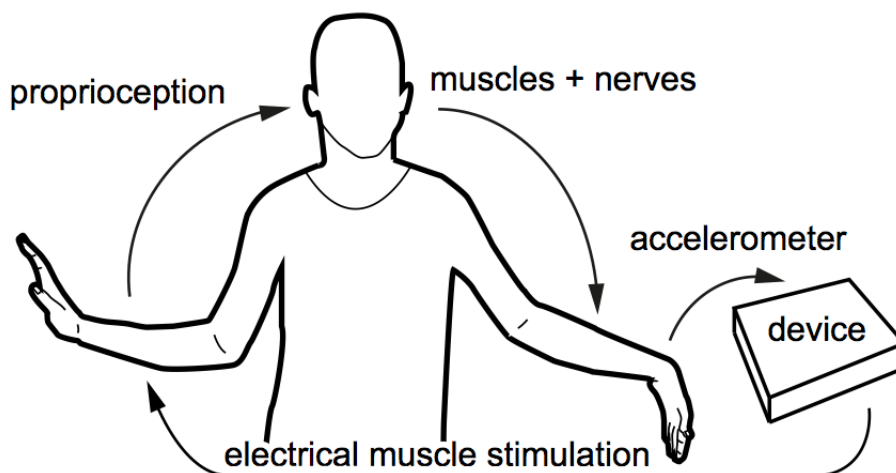


Figure 2: In an asymmetric proprioceptive interaction, input and output occur through different limbs.

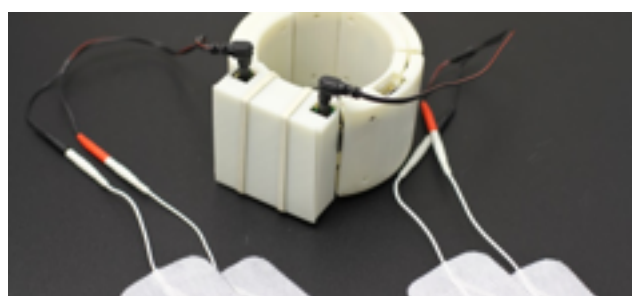


Figure 3: The 3D printed Pose-IO bracelet

Figure 4 shows the opened-up bracelet. Pose-IO writes output to the muscles using a medically-compliant electrical muscle stimulation unit (TruTens V3) connected to four pre-gelled electrodes (50×50 mm). The amplification for the Electrical Muscle Stimulation is regulated by two CMOS digital potentiometers with non-volatile memory (X9C103, 10 k Ω) controlled by an Arduino Nano using a three-wire serial interface.

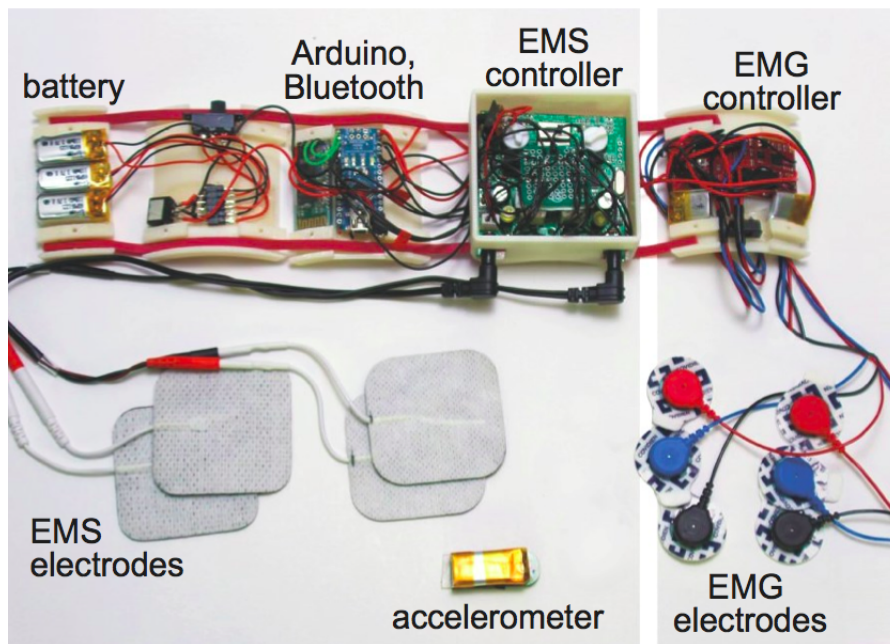


Figure 4: A Pose-IO bracelet unrolled and with the top enclosure removed. On the left side, the additional EMG unit.

Electrical Muscle Stimulation (EMS)

Pose-IO creates its EMS signal as a biphasic waveform. Its signal pulsates at 120 Hz with a pulse-width of 150 μ s. The current is limited to 100 mA, allowing for safe operation. Given that the lowest power settings of the EMS unit did not achieve muscle actuation with the users we tested with, we stepped up the control curve by adding a 10 k Ω resistor in parallel to the EMS unit's variable output and input pins. This allows Pose-IO to achieve a smoother output current curve. Upon first use, users calibrate Pose-IO's EMS device by specifying the lowest stimulus that still leads to a recognizable sensation (there is no visible hand motion at this level) as well as the maximum signal that the user perceives as comfortable. At all times, Pose-IO use is pain-free. Figure 5 shows the placement of the EMS electrodes on the

extensor digitorum (wrist extension) and on the flexor digitorum superficialis (wrist flexion).

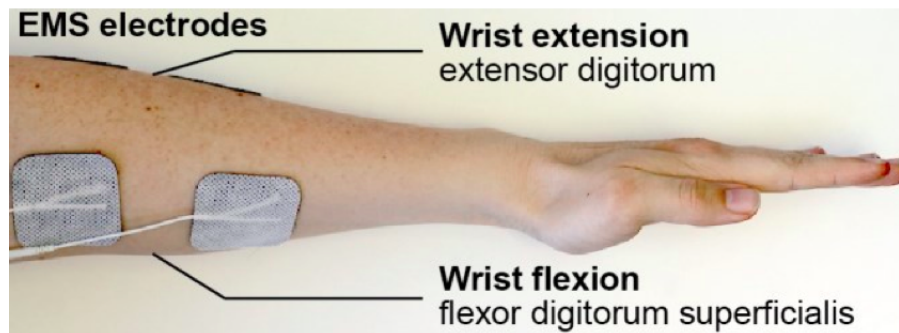


Figure 5: Placement for Pose-IO's EMS electrodes for wrist flexion and extension

Accelerometer

Pose-IO uses the accelerometer's Y-axis to determine the user's wrist position as the tilt of the user's hand against the horizontal, sampled at 50 Hz. We also made an extended version of the device that determines the wrist pose with respect to a second accelerometer worn on the forearm. This allows users to operate Pose-IO in any body posture. Users invoke and dismiss Pose-IO by holding the hand horizontally and then shaking it, which Pose-IO senses by looking for acceleration along any of the other accelerometer axes. Pose-IO sends commands to its applications, such as aforementioned games, which it runs on the Arduino microcontroller. Pose-IO talks to software running on other computers, such as the video player in the presenter tool scenario, via Bluetooth. Depending on version, Pose-IO either uses an Axivity WAX3 or WAX9 3-axis wireless accelerometer that user wear on their ring finger. After removing them from their casing, both measure 34.5 mm × 16 mm × 15 mm and offer 4 mg resolution. The WAX-3 sends data over IEEE802.15.4 radio, which we convert using a laptop computer with a radio dongle; this issue we resolved with the newer WAX9, which sends data directly to Pose-IO's Arduino via Bluetooth. Pose-IO is powered by a 9 V power supply comprised of three 3.7V lithium ion rechargeable batteries connected in series and regulated using an L7809CV. Under continuous use, Pose-IO offers around 4 hours of battery-life on the bracelet and 8 hours on the accelerometer.

Control Loop for Symmetric Input and Output

When tracking an external signal, such as the video play position, Pose-IO actuates the user's muscles using a PID control loop. We obtain oscillation-free behavior

using the gain factors: $K_p = 1.2$, $K_v = 1.1$, $K_i = 0.5$ (Figure 6). Pose-IO calculates the error derivative on a moving average over the 10 last measured velocities.

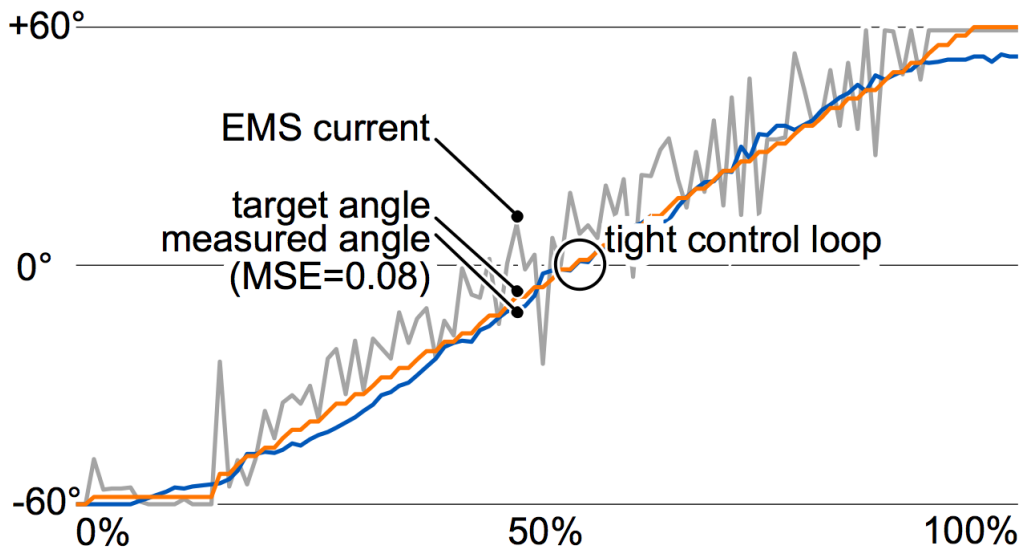


Figure 6: Behavior of Pose-IO's PID control

If the accelerometer value deviates from the system state by 10° , Pose-IO assumes that the user intends to override the system state—in the video scrubbing scenario this allows users to override the video play head position.

4.2 Study 1: symmetric proprioceptive input and output

The purpose of our first study was to verify our basic proprioception interaction concept, i.e., interaction by means of posing the wrist. The study focuses on symmetric input/output. For each trial, Pose-IO posed the participant's wrist at a target angle and held it stable for one second, played a sound, and dismissed the pose, causing the hand to drop. Participants' task now was to recreate the previous pose. When satisfied, participants pressed a keyboard button, which caused the system to record the trial. For each trial, we recorded Pose-IO's accelerometer reading during the target pose and during the recreated pose.

Procedure

There were 7 target angles in 20° steps between 60° (flexion) and 60° (extension) reflecting the biomechanical constraints of the human wrist. Each participant performed $(7 \text{ target positions} \times 4 \text{ repetitions}) = 28$ trials. Target angles were presented

in random order. Prior to the first trial, we calibrated the device for the respective participant. However, participants received no training on the task, as our expectation was that the unified correspondence between input and output would allow participants to complete the task based on their natural sense of proprioception alone.

Participants

We recruited 10 participants (3 female) from our local organization. All were right-handed.

Results

Participants recreated poses with an average error of 5.8° ($SD = 5.1^\circ$) across all trials. A linear regression found the overall model fit to be $R^2 = 0.954$. Furthermore, tilting the hand up further resulted in slightly larger errors (mean = 6.49 , $SD = 5.93$) (mean = 5.19 , $SD = 4.18$). However, we found no statistical significant difference between any of the angles. For simplicity we present the results for the pairs with larger deviation, e.g., 40° vs. -60° ($Z=-1.886$; $p=0.059$) and 60° vs. -60° ($Z=-1.274$; $p=0.203$).

Discussion

Our results show that the interaction did actually use and benefit from proprioception. As observed participants re-posing was accurate and included reposing of fingers. The results suggest that proprioceptive interaction's ability to allow for symmetric input and output offers benefits when it comes to recreating poses, as users can perceive and re-pose their wrist using their proprioceptive sense. On a practical level, participants acquired the target poses with comparably high accuracy, i.e., 5.8° error on average. With respect to the motion range of 60° to $60^\circ = 120^\circ$ this represents an error of under 5%. If we map these values back to the motivating example of video scrubbing, this means that Pose IO would allow users to jump to a spot in a 1-minute video clip with ± 3 seconds accuracy, which we would argue to be a useful level of performance. Obviously, these values were obtained under idealized conditions, i.e., a sitting user and immediate recreation of a pose, so should be interpreted as a lower bound for error.

4.3 Study 2: asymmetric proprioceptive input and output

While our discussion so far was focused on the technical aspects and abilities of proprioceptive interaction, one of its aspects that we are personally intrigued about is its way of representing the computer as part of the user's body. Very unlike traditional human computer interaction systems, where computers are outside and

different from the user, proprioceptive interfaces make the user's limbs themselves serve as the interface, which we argue is the very essence of proprioception—after all the work stems from Latin “*proprius*”, meaning “one's own”. Our video-scrubber application makes the hand partially “owned” by the user and partially owned by the machine. The two games we showed earlier push this approach even further by questioning that notion of “ownership” in that they cause one hand to be fully owned by the machine.

We were wondering how users perceive this aspect of proprioceptive interaction and what emotional response it might produce. This is what we investigated in this second, exploratory study. We had participants play the red hands game described earlier. We recorded participants' response to the interface on video and had them fill in a question-naire.

Setup

We recruited 12 participants (3 female) from our local organization, which did not partake in the previous study, and asked them to play the red hand game for about 5 minutes. As discussed earlier, participants' objective was to evade getting slapped by the computer-controlled hands. All participants received candy as a small incentive to participate and we promised additional candy to whoever would score highest. The game offered levels of increasing difficulty. As mentioned earlier, these were implemented in the form of barely noticeable actuation of the slapping hand before the actual slap (750 ms to 50 ms, depending on level). We did not tell participants about this “warning” mechanism, making it part of the game to either consciously or unconsciously figure it out. For this study we used an earlier Pose-IO prototype, i.e., same hardware but not wearable yet. To make sure participants only responded to the pose interaction, we canceled out any auditory signals by making them wear noise-cancelling headphones that played music. Each study session started by participants calibrating Pose IO, familiarizing themselves with the game mechanics, and playing one training level with 10 slaps. They then played 5 levels with about 10 slaps each. If they got hit less than 3 times per level, they proceeded to the next level and their score continued to go up. If not, their score stopped increasing, but we still let them finish the remaining levels to give them a chance to experience the complete game.

Findings

We had designed the game to be challenging and participants took the game quite seriously. All except one participant completed level 3; three level 4, and two made it to the fifth and final level. Participants rated the first two levels as easy (mean = 2.0 of 5; 1 = very easy, 5 = very hard) and the last levels as hard (mean = 3.92 of 5). All participants rated the game as fun (mean = 4.6 of 5). When asked why, seven participants pointed to the fact that they felt as if they were “playing against themselves”.

Several participants went back and forth between referring to the hand wearing the device as “me” and as “computer”, such as “it is weird that you lose against yourself” and “sometimes the computer hand was faster than I”.

In the last two levels, ten of twelve participants stopped looking at their hands while awaiting the slaps. When asked about it, only two stated to have played always looking at their hands, while five participants stated to play eyes-free and seven stated to have rarely looked. This suggests that participants operated our application by means of proprio-ception alone.

Most participants agreed that the muscle output had contributed to their excitement (mean = 4.91 of 5). One of the two participants who had continued to look at their hands explained: “it was so remarkable to see my hand moving without my intention that I could not look away”.

In summary, watching the twelve participants confirmed that Pose IO affords being operated eyes-free and that players perceived the “blurred ownership” of the user’s body as compelling. We discuss this further in the conclusions.

5 Conclusion and outlook

In this paper, we proposed the concept of proprioceptive interaction, which we instantiate through a wearable prototype. Proprioceptive interaction leverages the user’s proprioceptive sense for both input and output, i.e., the use of pose as a bidirectional communication channel between human and computer. We demonstrated how proprioceptive interactions allows for (1) eyes-free and ears-free use; (2) invoke the device any time and return to their primary task immediately when necessary; and, provides (3) two modalities: symmetric, in which input and output occur in the same limb, and asymmetric (i.e., one limb for input, another for output). A unique aspect of proprioceptive interaction is that it represents the computer using a part of the user’s body, which our second study suggests that participants found this experience compelling.

5.1 Future Work

As next steps for my research I plan to investigate how these type of interfaces (i.e., poses as input and output) provide support in interacting with unfamiliar objects. Thus the next step is a deeper understanding of the usability implications of my research.

5.2 Projects under submission

- Proptioceptive Interaction (*working title*), Pedro Lopes, and Patrick Baudisch
- Affordance++ (*working title*), Pedro Lopes, Patrik Jonell, and Patrick Baudisch

5.3 Pedagogical Activities and Other

- Certificate of International Teaching Professional, University of Potsdam.
- Editor of ACM XRDS Magazine.

References

- [1] K. Bark, J. W. Wheeler, S. Premakumar, and M. R. Cutkosky. "Comparison of Skin Stretch and Vibrotactile Stimulation for Feedback of Proprioceptive Information". In: *Proceedings of the 2008 Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. HAPTICS '08. IEEE Computer Society, pages 71–78. DOI: 10.1109/HAPTICS.2008.4479916.
- [2] S. Daniel. *Handbook of Phenomenology and Cognitive Science*. Edited by S. Gallagher. Springer, 2010.
- [3] E. Gardner and J. Martin. *Coding of Sensory Information, Principles of Neural Science*. McGraw-Hill.
- [4] I. Karuei, K. E. MacLean, Z. Foley-Fisher, R. MacKenzie, S. Koch, and M. El-Zohairy. "Detecting Vibrations Across the Body in Mobile Contexts". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, pages 3267–3276. DOI: 10.1145/1978942.1979426.
- [5] E. Kruijff, D. Schmalstieg, and S. Beckhaus. "Using Neuromuscular Electrical Stimulation for Pseudo-haptic Feedback". In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. VRST '06. Limassol, Cyprus: ACM, pages 316–319. DOI: 10.1145/1180495.1180558.
- [6] K. Li. "Designing easily learnable eyes-free interaction". PhD thesis. SanDiego.
- [7] P. Lopes and P. Baudisch. "Muscle-propelled Force Feedback: Bringing Force Feedback to Mobile Devices". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, pages 2577–2580. DOI: 10.1145/2470654.2481355.

- [8] T. Ni and P. Baudisch. “Disappearing Mobile Devices”. In: *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*. UIST '09. Victoria, BC, Canada: ACM, pages 101–110. doi: 10.1145/1622176.1622197.
- [9] A. Roudaut, A. Rau, C. Sterz, M. Plauth, P. Lopes, and P. Baudisch. “Gesture Output: Eyes-free Output Using a Force Feedback Touch Surface”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, pages 2547–2556. doi: 10.1145/2470654.2481352.
- [10] T. S. Saponas, D. S. Tan, D. Morris, R. Balakrishnan, J. Turner, and J. A. Landay. “Enabling Always-available Input with Muscle-computer Interfaces”. In: *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*. UIST '09. Victoria, BC, Canada: ACM, pages 167–176. doi: 10.1145/1622176.1622208.
- [11] E. Tamaki, T. Miyaki, and J. Rekimoto. “PossessedHand: Techniques for Controlling Human Hands Using Electrical Muscles Stimuli”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, pages 543–552. doi: 10.1145/1978942.1979018.
- [12] D. Tsetserukou, K. Sato, and S. Tachi. “ExoInterfaces: Novel Exoskeleton Haptic Interfaces for Virtual Reality, Augmented Sport and Rehabilitation”. In: *Proceedings of the Augmented Human*. AH '10. Megève, France: ACM, pages 1–6. doi: 10.1145/1785455.1785456.

Question Answering for Biomedicine

Mariana Neves

Enterprise Platform and Integration Concepts

Hasso-Plattner-Institut

Mariana.Neves@hpi.uni-potsdam.de

Question answering systems can support biologists and physicians when searching for answers in the scientific literature or the Web. It allows users to pose questions in natural language instead of keywords and to receive short answers in return instead of documents which might be relevant to the query. Further, multilingual question answering systems provide more possibilities and flexibility to users, by allowing them to write questions and get answers in their native language, and the exploration of resources in other languages by means of machine translation. In this report I present my recent work on biomedical question answering and its evaluation during participation on the BioASQ challenge and on a multilingual dataset for English, German and Spanish.

1 Introduction

Question answering (QA) is the task of posing questions to search engines and receiving an exact answer in return [4]. For instance, biologists frequently look for answers in the scientific literature or in the Web to confirm results obtained in the laboratory, e.g., whether a certain disease could be associated to a particular genetic mutation. However, answers to such questions can be scattered over different scientific publications (abstracts and full text), biological databases and Web pages.

QA differs to information retrieval in two main aspects: (a) queries are presented as natural language questions (long input) instead of a set of keywords (short input); and (b) an exact and short answer (yes/no, a fact or a paragraph) is returned instead of a list of potential documents which might contain a answer. A variety of QA systems have been developed for the so-called open domain (e.g., START¹) and the field has received increasing attention from the scientific community since the IBM Watson system beat human participant in the Jeopardy TV show [3].

Multilingual question answering systems offer a variety of new possibilities to the user, also for the biological domain. Although most of the relevant scientific publications are available in the English language, there is a myriad of other resources in other languages that could be explored, such as non-English biomedical

¹<http://start.csail.mit.edu/index.php>, accessed December 16, 2014.

scientific literature (e.g., Scielo² for Spanish and Portuguese), as well as the whole Web. Further, a multilingual QA system allows non-native English speakers to pose questions and receive answers in their native language while the system queries English (or any other language) documents by means of machine translation.

In this report I describe the architecture of my question answering system and present its evaluation on the BioASQ challenge [8] and on a multilingual collection of questions which includes the German and Spanish languages [9]. The QA system was developed on the top of an in-memory database which include built-in text analysis functionalities for eleven languages. Additionally, I have created and made available two parallel collections of 50 questions each for English/German and English/Spanish.

2 Related work

Despite the importance of QA systems for Biology, not much previous work has been carried out for this domain, when compared to the state-of-art solutions in the medical domain [1]. Currently, there seems to be only three QA systems available for Biomedicine [2]: EAGLi³, AskHermes⁴ and HONQA⁵. However, both AskHermes and HONQA have a focus on Medicine and might not be suitable for biologists. Nevertheless, this scenario has been changing in the last couple of years thanks to new initiatives such as the QA4MRE [7] and BioASQ [11] challenges which support improvements in biomedical question answering.

Previous research is scarce in the biomedical multilingual question answering field and in biomedical natural language processing (NLP) in general. From the QA systems cited above, only HONQA allows questions to be posed in other languages than English, namely Italian and French, whose performance has been evaluated in [10]. The biomedical research seems to be always one (or more) step behind state-of-art in NLP for other domains due to its complexity and the lack of suitable resources for system development and evaluation. For instance, QA systems need to rely in high performing tools for the many pre-processing steps, such as tokenization, part-of-speech tagging and parsing. However, previous research have proved that sometimes specific models might be necessary for some domains, such as Biomedicine [6]. One important step towards improvements in biomedical NLP is the recent EU-funded Multilingual Annotation of Named Entities and Terminology Resources Acquisition (Mantra) project [13] which has supported improvements in

²<http://www.scielo.org/>, accessed December 16, 2014.

³<http://eagl.unige.ch/EAGLi/>, accessed December 16, 2014.

⁴<http://www.askhermes.org/>, accessed December 16, 2014.

⁵<http://www.hon.ch/QA/>, accessed December 16, 2014.

named-entity recognition of biomedical terms during the CLEF-ER challenge last year. Finally, improvements in machine translation for Biomedicine have also been recently published [5].

3 Methods and Material

I have developed a system which relies on the in-memory database technology [12] and built-in text analysis provided by the SAP HANA database (hereafter called “HANA”). In this section I start by presenting the datasets which have been used and/or developed for the evaluation and describe the system architecture.

3.1 BioASQ dataset

The BioASQ challenge⁶ is an EU-funded project which aims to foster research and solutions on biomedical question answering. A first challenge was run in 2013 as part of CLEF 2013 [11] and a new edition has been held in 2014 together with other QA and machine reading-related tasks⁷. The BioASQ challenge consists of two main tasks: (2a) Large-Scale Online Biomedical Semantic Indexing and (2b) Biomedical Semantic QA, the one in which I participated. The later is sub-divided in two phases: Phase A and Phase B. In Phase A of task 2b, questions and their respective type (yes/no, factoid, list or summary) were released and participants were requested to provide the following information:

1. a list of relevant concepts belonging to five predefined ontologies and terminologies (GO, DO, MeSH, Jochem and Uniprot);
2. a list of relevant articles from PubMed⁸;
3. a list of relevant snippets, including the PubMed document of origin, the start and end sections and offsets in the documents, and the text of the snippet;
4. a list of relevant RDF triples.

In Phase B of task 2b, participants were provided with the questions and respective types released for Phase A as well as gold-standard information for Phase A, i.e, manually curated relevant concepts, documents, snippets and RDF triples. This time participants were requested to submit the following answers:

⁶<http://bioasq.org/>, accessed December 16, 2014.

⁷<http://nlp.uned.es/clef-qa/>, accessed December 16, 2014.

⁸<http://www.ncbi.nlm.nih.gov/pubmed>, accessed December 16, 2014.

1. an exact answer for the question: “Yes” or “No” for yes/no questions, and a single or a list of short answers for factoid and list questions, respectively.
2. an ideal answer, which consists of a short paragraph for the summary questions as well as an extended answer to the yes/no, factoid and list questions.

A training dataset which includes of 310 questions and manually curated information for both Phases A and B was released for the participants to allow training and/or evaluation during system development.

3.2 Multilingual question corpus

A collection of biomedical question in English, German and Spanish [9] has been created together with a visiting Master student at the EPIC group. The construction of the collection of parallel questions was based on the Medline documents in Spanish and German of the CLEF-ER resource (cf. below) which contain a corresponding document in English. We chose this approach to allow a comparison between results obtained with German and Spanish with those in English on the same documents.

Batches of 50 documents (titles) were randomly retrieved from the above datasets and titles were manually chosen according to their relevance to the biomedical domain. During manual screening of the titles, we avoided documents related only to the medical domain, an area where state-of-art in question answering is more advanced in comparison to the biological one [1]. In particular for the Spanish questions, we selected titles related to tropical neglected diseases, which is a frequent topic on publications in the Medline collection for this language.

Questions were manually written in a way that at least one answer could be found in the corresponding document, i.e., the document’s title. However, not all of the information cited in the text was always used and the questions sometimes are more general than the respective text. We have generated only factoid questions, i.e., questions which requires one or more specific short answer in return, such as a chemical compound, an organism or a disease. While writing the questions, we tried to rephrase the text, used synonyms for both the named entities and remaining words (whenever possible), changed the word’s lexical class (e.g., from verb to noun), and converted passive voice to active voice, or the other way round. Synonyms for the lexical terms were supported by making queries to a variety of on-line language-specific dictionaries.

The semantic concepts referred in the text were also, whenever possible, changed to a equivalent synonyms. This task was supported by a variety of on-line resources, such as Wikipedia (for the three languages), NCBI Taxonomy and other web sites which were returned by the Google search engine. Therefore, we did not make use of the thesaurus made available by the CLEF-ER challenge (cf. below), instead, we

have tried to use the resources that the users (biologists) might use while posing questions to a QA system.

Questions were initially written in English and were reviewed by an expert in molecular biotechnology. In a second step, the questions were translated into Spanish and German by the authors, who are either native or have advanced knowledge on the languages. We have also sought to use synonyms for the words and concepts in this step. Finally, we tried to make the questions with similar difficulty level in both languages. Some examples of questions are presented in Table 1 and the list of parallel questions is available for download⁹.

Table 1: Examples of parallel questions from the English/German and the English/Spanish datasets

English/German (document d6357751)
Which methods can be used to determine the living cell count of cariogenic microorganisms? Welche Methoden bieten sich zur Bestimmung der Lebendzellzahl von kariogenen Mikroorganismen an?
English/Spanish (document d16888692)
What are possible drug targets for eye related infections? Cuáles son los posibles objetivos farmacológicos en infecciones relacionadas con el ojo?

CLEF-ER resources

The CLEF-ER challenge took place in 2013 as part of the Mantra project¹⁰ which aims to provide multilingual documents and terminologies for the biomedical domain. For the scope of this challenge, Medline and patent documents have been released in five languages: English, German, French, Spanish and Dutch. Mapping to English documents were provided for all documents in each of the languages other than English but no mapping seems to exist between documents from two other languages, e.g., between Spanish and German.

For creating the collection of questions described above, we have utilized the Medline documents collections, which in fact consists only of the title of the documents.

⁹<https://sites.google.com/site/marianalaraneves/resources/>, accessed December 16, 2014.

¹⁰<https://sites.google.com/site/mantraeu/home>, accessed December 16, 2014.

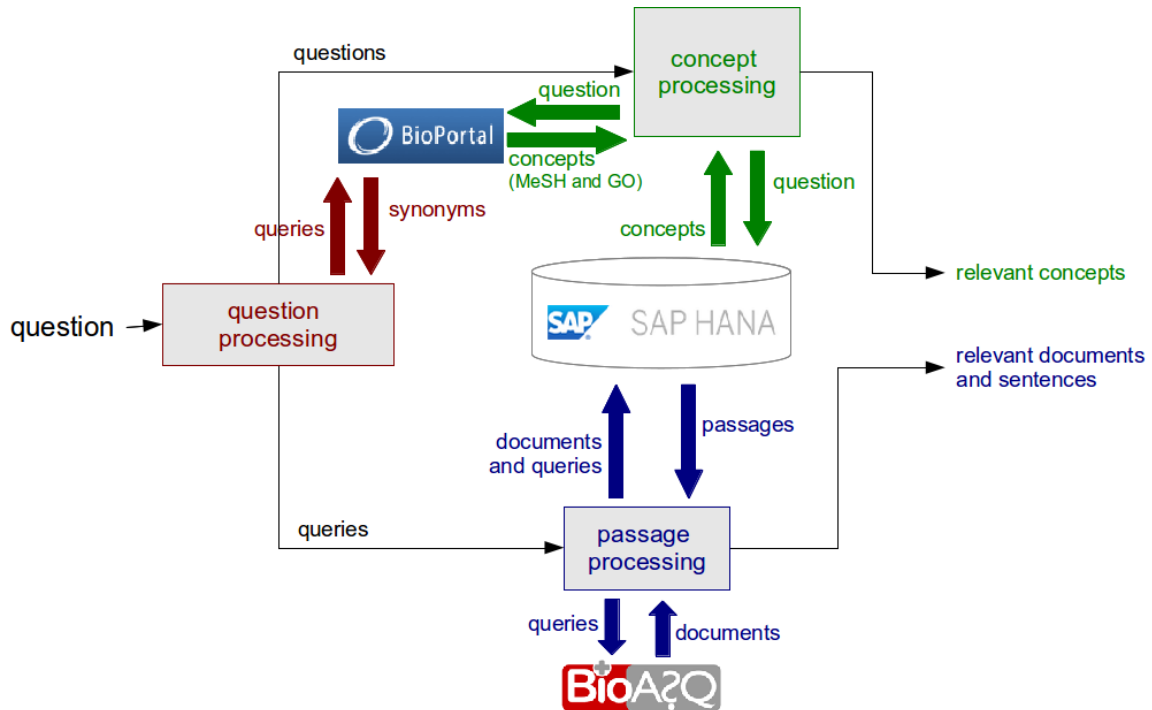


Figure 1: Architecture of the system. The data work-flow for the three components are identified by distinct colors: “red” for question processing, “green” for concept recognition and “blue” for document and passage retrieval.

We have restricted our work to Spanish and German, given the support of the SAP HANA database for these languages and the knowledge of the authors on them.

Organizers have also released a terminology containing synonyms for terms in the above languages, which has been compiled from three resources: Medical Subject Headings (MeSH), Systematized Nomenclature of Human and Veterinary Medicine (SNOMED-CT) and Medical Dictionary for Regulatory Activities (MedDRA).

3.3 System architecture

Question answering systems usually include three main components [1]: question processing, passage retrieval and answer processing. My system currently includes only the two first steps. A schema of the system is shown in Figure 1 and each of its components are described in details below.

3.4 Question processing

The question processing component includes sentence splitting, tokenization, part-of-speech tagging and chunking using the Stanford CoreNLP package¹¹. For the multilingual prototype, I use the OpenNLP Maximum Entropy-based tokenizer and part-of-speech tagger and the corresponding available models for English and German¹². Given that no models are available for the Spanish language, I have used the one available for Portuguese for the tokenization step, given the similarity between these languages and that tokenization is even more challenging in the later due to the composed words, which are not common in Spanish. For the part-of-speech tagging in Spanish, I have used the Maxent model made available by Juan Manuel Caicedo Carvajal¹³. Queries was generated from the original question based on both tokens and chunks.

When evaluated on the BioASQ dataset, I performed query expansion based on the extracted tokens and by calling services from BioPortal¹⁴. I required exact match of the term to avoid potential wrong synonyms. For evaluation over the multilingual corpus of questions, query expansion was performed using the CLEF-ER terminologies resources (cf. above). The CLEF-ER thesaurus has been loaded into the HANA database and a fuzzy search is carried out for each token in the query against the synonyms available for the corresponding language.

Weights are assigned for the terms of the query and the corresponding synonyms and are calculated based on the popularity of the term in the BioPortal (or CLEF-ER) resource. The higher the number of synonyms which match to a term, the lower the weight of the later. Tokens which do not match any synonyms are assigned weight 0.5, i.e., an average weight. Otherwise, weights are calculated based on the number of terms which matched to this particular token ($\#MatchesToken$) and the total number of terms matched to all tokens of the query ($\#MatchesTotal$), following the expression below:

$$\text{weight} = 1 - \frac{\#MatchesToken}{\#MatchesTotal} \quad (1)$$

3.5 Concepts retrieval

For each question in the BioASQ challenge, participants were required to return relevant concepts in five ontologies or terminologies: MeSH, GO, SwissProt, Jochem

¹¹<http://nlp.stanford.edu/software/corenlp.shtml>, accessed December 16, 2014.

¹²<http://opennlp.sourceforge.net/models-1.5/>, accessed December 16, 2014.

¹³<http://cavorite.com/labs/nlp/opennlp-models-es/>, accessed December 16, 2014.

¹⁴<http://data.bioontology.org/documentation0>, accessed December 16, 2014.

	ID	TA_TOKEN	TA_TYPE	TA_NORMALIZED	TA_OFFSET
1	5118dd1305c10fae75000001	Rheumatoid Arthritis	DO	DOID:7148	3
2	5118dd1305c10fae75000001	Rheumatoid Arthritis	MESH	D001172	3
3	5118dd1305c10fae75000001	men	MESH	D008571	39
4	5118dd1305c10fae75000001	women	MESH	D014930	46

Figure 2: Concepts retrieved by the HANA database for the sentence “Is Rheumatoid Arthritis more common in men or women?” (5118dd1305c10fae75000001) from the training data

and DO. Concepts were retrieved using two approaches: by matching previously compiled dictionaries to the text of the question using HANA and by making queries to the BioPortal web services.

The original ontologies and terminologies were retrieved from the respective web sites¹⁵ and one dictionary was compiled for each of them. The dictionaries were converted to HANA database XML format, compiled and used for matching the terms in the text of the questions, which were previously loaded into a table in the database. I generated an index which includes the identifier of the document (question), the text span which was matched, the terminology/ontology, the respective identifier and the start offset with respect to the original text of the question, as shown in Figure 2. The terms recognized by HANA were retrieved from the database and I skipped those matches whose text length was less than 3 and which coincided with stopwords or Greek letters.

The second approach retrieved concepts of the MeSH and GO ontologies by making queries to the BioPortal Recommender¹⁶. Queries were created based on the full text of the question¹⁷ and all returned concepts were considered.

¹⁵<http://www.ncbi.nlm.nih.gov/mesh>, <http://www.geneontology.org/>, <http://www.uniprot.org/>, <http://www.biosemantics.org/>, <http://disease-ontology.org/>, all accessed December 16, 2014.

¹⁶<http://data.bioontology.org/documentation>, accessed December 16, 2014.

¹⁷e.g., “http://data.bioontology.org/recommender?text=Is+Rheumatoid+Arthritis+more+common+in+men+or+women%3F&ontologies=GO%2CMESH&include_classes=true&apikey=7795d203-29ce-4f89-85aa-02c3555b21dd”

3.6 Passages and documents retrieval

For each question from the BioASQ dataset, I perform four queries to the BioASQ PubMed service¹⁸ according to whether considering query expansion or not and whether using “OR” or “AND” operators between the tokens. I retrieve up to the 500 top ranked documents and I only consider titles and abstracts. However, an analysis of the BioASQ training dataset shows that these constitute about 90% of the relevant passages (5240 out of 5781 snippets).

The text of the titles and abstracts were inserted into the HANA database and a full text indexing was performed on them which include sentence splitting and tokenization. Queries were posed to the HANA database based on the terms of the query (including synonyms extracted during query expansion) and an approximate matching was performed by requiring at least 90% similarity. Sentences were ranked according to a score, which was calculated based on the similarity of the tokens, their weights in the query (cf. query processing above) and the total number of tokens which were matched.

For evaluation on the multilingual dataset, Medline documents available for the English, Spanish and German languages from the CLEF-ER (cf. above) were loaded into HANA and a full text index was created for each collection. I have experimented with three search strategies provided by HANA: exact, fuzzy matching (at least 90% of similarity) and linguistic.

4 Results and evaluation

In this section I present the preliminary results obtained for the BioASQ and the multilingual datasets.

4.1 BioASQ dataset

I performed experiments with the training/development dataset which contains 310 questions and their respective relevant concepts, documents and snippets. I calculated metrics of precision, recall and F-score for the concepts, documents and passages retrieval, but I did not considered their ranks in their respective lists. Concepts were evaluated based on their identifiers and documents based on the PubMed identifiers. Passages were evaluated based on the particular document and section they come from and I consider a true positive if there is an overlap of any length

¹⁸<http://gopubmed.org/web/gopubmedbeta/bioasq/pubmedmedline>, accessed December 16, 2014.

Table 2: Results in terms of precision, recall and F-score for the training dataset

	Evaluation	Precision	Recall	F-Score
Concepts	HANA	0.28	0.18	0.22
	BioPortal	0.25	0.15	0.18
	HANA+BioPortal	0.26	0.21	0.23
Documents	w/o query expansion	0.022	0.11	0.037
	with query expansion	0.022	0.12	0.037
Passages	w/o query expansion	0.015	0.077	0.025
	with query expansion	0.015	0.078	0.025

between the text of the gold standard and the one returned by our system. Results for the training dataset are presented in Table 2.

The evaluation phase of Phase A of task 2b consisted of five batches of questions which were released every 2/3 weeks. Participants had 24 hours to process the dataset, obtain the outputs for the corresponding information, build the JSON output file and submit it to the BioASQ web site.

My system was under development while the BioASQ challenge was running and my submissions to the various batches of test questions varied accordingly. I did not submit runs for batches 1 and 5 and the only major change between the system used for batch 2 (HPI-S₁) and batches 3 and 4 (HPI-S₂) was that the first did not include synonyms for terms when performing queries to the BioASQ services for retrieving relevant documents. Predictions for concepts were only provided for batches 3 and 4. Table 3 presents the results for the three batches (as June 24th 2014) based on the metrics of mean precision, recall, f-measure and MAP which are described in details in the BioASQ guidelines¹⁹.

4.2 Multilingual dataset

We randomly split the two parallel collections of 50 questions in English/Spanish and English/German in two sets of 25 questions each, i.e., two sets of 25 questions for development and two sets of 25 questions for testing purposes. The development datasets were used for choosing the best strategies, adding of extra stopwords and error analysis, but have not been used to train any of components of the system.

Evaluation of the development and test datasets consisted in running the system for each of them, retrieving the 10 best ranked passages (sentences) and checking

¹⁹http://bioasq.lip6.fr/Tasks/b/eval_meas/, accessed December 16, 2014.

Table 3: Results in terms of mean precision (P), recall (R), f-measure (FM) and MAP (as June 24th 2014), along with our position (Rank) in this batch with respect to the total number of runs. * indicates whether our position was ranked higher than the Top 100 and Top 50 baselines provided by the organizers. [§] indicates that no system outperformed any of the two baselines.

Documents	P	R	FM	MAP	Rank
Batch 2	0.0235	0.1341	0.0376	0.0733	10/18
Batch 3	0.0216	0.1773	0.0343	0.1016	11/19
Batch 4	0.0159	0.1399	0.0271	0.0558	10/18
Snippets	P	R	FM	MAP	Rank
Batch 2	0.0117	0.0746	0.0191	0.0521	1/10*
Batch 3	0.0126	0.0857	0.0195	0.0538	5/10*
Batch 4	0.0084	0.0882	0.0146	0.0339	6/12 [§]
Concepts	P	R	FM	MAP	Rank
Batch 3	0.1134	0.1318	0.1034	0.0567	8/10 [§]
Batch 4	0.1042	0.1080	0.0959	0.0522	8/8 [§]

whether the original document from which the question had been derived was present in this list. With such an experiment, we sought to evaluate the precision of our QA prototype for finding relevant passages to the questions as well as the difficulty level of the questions. Text passages were retrieved only for documents in the same language of the question, e. g., Spanish questions were queried only against the Spanish documents, thus, no machine translation was used.

We have evaluated the following settings of our QA system:

1. HANA exact search;
2. HANA fuzzy matching (at least 90 % similarity);
3. HANA fuzzy matching (above), plus query expansion of the question words using the CLEF-ER terminology;
4. HANA fuzzy matching (above), query expansion (above), plus HANA linguistic matching for those words in the question which did not match to any synonym.

For the evaluation, we calculate the R-Precision, which is the precision on the r-th position where a first match with the original document was found, or zero, if not

found. We then calculate the mean of the R-precision over the collection of questions, i.e., over 25 questions for each dataset. Table 4 shows the results for each language in the English/German and English/Spanish parallel collections of questions.

Table 4: R-Precision and number of sentences found (in parenthesis) for each dataset and for various setting of the question answering system. Results for the training dataset are shown for the many setting options while the ones for the test dataset were obtained when using query expansion and fuzzy matching (i.e., “+ synonyms”). Codes for the languages are the following: “EN”: English, “DE”: German, “ES”: Spanish.

Settings	English-German		English-Spanish	
	EN	DE	EN	ES
exact	0.04 (1)	0.04 (1)	0.01 (1)	0.09 (4)
fuzzy	0.05 (3)	0.02 (1)	0.10 (5)	0.10 (5)
+ synonyms	0.09 (3)	0.06 (2)	0.11 (4)	0.09 (6)
+ linguistic	0.10 (4)	0.04 (2)	0.03 (2)	0.08 (6)
Test	0.05 (5)	0.05 (4)	0.05 (3)	0.06 (4)

5 Conclusion and outlook

In this report I have shown the architecture of my question answering system developed based on the in-memory database technology and its evaluation on two datasets: during its participation in the BioASQ challenge and on a multilingual dataset for German, Spanish and English.

Comparison between the training and test batches of BioASQ shows that results for the later have been only slightly lower than what have been obtained in the training dataset, excepts for the concept retrieval whose results were far below. In general, except for the concept retrieval step, recall is much higher than the precision because I provided the top 100 snippets (and respective documents) for each question without specifying a minimum threshold score for that. Given that the MAP results were always higher than the precision for all document and snippets batches, I believe that precision (and consequently the F-Measure) can be improved by experimentally defining a minimum threshold in the passage retrieval step.

Results for document retrieval was tightly dependent on the query processing step, i.e., the conversion of the question to a appropriate query, to the performance of the BioASQ services, which was used for retrieving the documents, and to our passage retrieval step, from which the list of document was compiled. From a total of 3847 false positives in the training data, 2782 (72 %) referred to documents which were not contained in our database and 368 (9.5 %) to documents which were in the database but that were not directed linked to the question, i.e., the documents were retrieved by BioASQ for another question. As future work on document and passage retrieval, I plan to query PubMed directly, instead of the BioASQ services, I also experimenting with various queries.

Regarding the multilingual dataset, preliminary results show that passage retrieval is not a straightforward task for none of the three languages, even when using a small collection of Medline document titles. Although the best results were obtained for Spanish, this was probably due to the smaller set of documents available for this language than to the simplicity of the task or the language. On the other hand, results for German were particularly low in comparison to the other languages because of the presence of many compound words, as will be discussed below. Despite the overall low performance of our results, these experiments provide baseline results for the proposed collection and gives an estimation on the quality of our dataset.

In comparison to the development dataset, performance for the test dataset was higher for the English/German dataset and a little bit lower for the English/Spanish questions. Curiously, this is the best performing result for the German questions. Discrepancies between development and test data are expected given the small number of questions and the distinct topics they refer. The development dataset was used only for comparing the different setting of the features, updating the stopwords list with additional terms and performing an analysis of the results.

Results vary significantly for the many configurations of our system when evaluated for the three languages (cf. Table 4). Despite the less complexity of the English language, more exact matches have been obtained for the Spanish (4) than for English (1 for each dataset) or German (1). The higher number of matches for Spanish have occurred mainly due the impossibility in getting better synonyms in the Spanish language when writing the question, together with the fact that less documents are available for this language. For instance, the question “*Qué métodos histoquímicos permiten la observación y caracterización de glicoconjugados?*” got the right match (document d9279022) in the third rank, while no correct match was found for the corresponding English question (“Which histochemical methods allow observation and characterisation of glycoconjugates?”).

References

- [1] S. J. Athenikos and H. Han. “Biomedical question answering: A survey”. In: *Computer Methods and Programs in Biomedicine* 99.1 (2010), pages 1–24. DOI: 10.1016/j.cmpb.2009.10.003.
- [2] M. Bauer and D. Berleant. “Usability survey of biomedical question answering systems”. English. In: *Human Genomics* 6.1 (2012), pages 1–4. DOI: 10.1186/1479-7364-6-17.
- [3] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. “Building Watson: An Overview of the DeepQA Project.” In: *AI Magazine* 31.3 (2010), pages 59–79.
- [4] L. Hirschman and R. Gaizauskas. “Natural Language Question Answering: The View from Here”. In: *Nat. Lang. Eng.* 7.4 (Dec. 2001), pages 275–300.
- [5] A. Jimeno Yepes, E. Prieur-Gaston, and A. Neveol. “Combining MEDLINE and publisher data to create parallel corpora for the automatic translation of biomedical text”. In: *BMC Bioinformatics* 14.1 (2013), page 146. DOI: 10.1186/1471-2105-14-146.
- [6] D. McClosky, E. Charniak, and M. Johnson. “Automatic Domain Adaptation for Parsing”. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT ’10. Los Angeles, California: Association for Computational Linguistics, 2010, pages 28–36.
- [7] R. Morante, M. Krallinger, A. Valencia, and W. Daelemans. “Machine Reading of Biomedical Texts about Alzheimer’s Disease”. In: *CLEF (Online Working Notes/Labs/Workshop)*. 2012.
- [8] M. Neves. “HPI in-memory-based database system in Task 2b of BioASQ”. In: *Working Notes for the CLEF BioASQ Challenge*. 2014.
- [9] M. Neves, K. Herbst, M. Uflacker, and H. Plattner. “Preliminary evaluation of passage retrieval in biomedical multilingual question answering”. In: *Proceedings of the Fourth Workshop on Building and Evaluation Resources for Biomedical Text Mining (BioTxtM 2014) at Language Resources and Evaluation (LREC) 2014*. 2014.
- [10] M.-D. Olvera-Lobo and J. Gutiérrez-Artacho. “Multilingual Question-Answering System in Biomedical Domain on the Web: An Evaluation.” In: *CLEF*. Edited by P. Forner, J. Gonzalo, J. Kekäläinen, M. Lalmas, and M. de Rijke. Volume 6941. Lecture Notes in Computer Science. Springer, 2011, pages 83–88.

- [11] I. Partalas, E. Gaussier, and A.-C. N. Ngomo. “Results of the First BioASQ Workshop”. In: *1st Workshop on Bio-Medical Semantic Indexing and Question Answering, a Post-Conference Workshop of Conference and Labs of the Evaluation Forum 2013 (CLEF 2013)*. Nov. 2013.
- [12] H. Plattner. *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases*. 1st. Springer, 2013.
- [13] D. Rebholz-Schuhmann, S. Clemenide, F. Rinaldi, E. V. Mulligen, I. Lewin, D. Milward, A. J. Yepes, U. Hahn, J. Kors, C. Bui, J. Hellrich, and M. Poprat. “Multilingual semantic resources and parallel corpora in the biomedical domain: the CLEF-ER challenge”. In: *CLEF Lab*. Sept. 2013.

Adaptive Just-in-time Value Class Optimization

Efficient Compaction of Immutable Data Structures in Virtual Machines

Tobias Pape

Software Architecture Group

Hasso-Plattner-Institut

tobias.pape@hpi.uni-potsdam.de

The performance of value classes is highly dependent on how they are represented in the virtual machine. Since value classes are supposed to be very lightweight and fast it is important to optimize them well. In this paper we present a technique to detect and compress commonly occurring patterns of value class usage to improve memory usage and performance. The set of micro-benchmarks we use in a small prototype language show two to ten-fold speedup over value classes in other object oriented languages.

1 Introduction

Objects are at the heart of object-oriented languages and the choice of how to represent them in memory is crucial for the performance of a language implementation [2, 13]. Additionally, on the language level, typical best practice techniques of object-oriented modelling and design, such as delegation or the composite design pattern, have an inherent influence on performance. Every additional indirection between delegators and delegates or composites and their parts has the overhead of a new object. This includes memory consumption, but also execution time to navigate the referenced objects. A simple example is traversing linked lists or trees. For collection structures, this problem has been tackled with arrays and their optimization [15], e.g. with Z-rays [14]. However, for *composite* structures like trees or composite objects, optimization is not as straightforward.

We want to provide performance similar to that of such flat structures while retaining desirable properties of composite structures, such as to be recursive or composable. Considering performance, the optimization of composite structures is similar for both general objects and *value class* [1] objects. The latter is a special form of objects that is immutable, may reference only other value class objects or primitive data, and has no object identity. Value classes are available in, e.g. Java, Scala, and .Net.

Composite structures involving value classes have certain usage patterns. This is obvious in linked lists (a single list element probably references another list element and so forth) or trees (a tree node references a number of other nodes or a value). Data structures with such patterns can be transformed into lower-level structures more fitting to the machine model. While simple variants of such patterns can be statically inferred, many become apparent only at runtime. Especially recursive structures exhibit patterns that are opaque to static inference, e.g. while a tree apparently may have sub-nodes, it is statically unknown whether trees are used as deep trees or rather flat ones. However, each case could be optimized differently. Hence our optimization approach works at run-time in conjunction with a JIT compiler.

2 Tracing Just-In-Time Compilers

One approach for writing JIT compilers is using a *tracing* JIT compiler. A tracing JIT compiler records the steps an interpreter takes to obtain an instruction sequence, called *trace* and then uses this trace instead of the interpreter to execute the same part of a program [12]. This produces specialized instruction sequences, e.g. only one path in if-then-else constructs; missed branches still use the interpreter. Tracing JIT compilers have been used for optimizing native code [3] and also for efficiently executing object-oriented programs [11].

Meta-tracing goes a step further and traces not the program execution but rather the interpreter execution. Hence, a resulting trace is not specific to the program executed but the interpreter [6]. It is not necessary for a language implementer to program a language-specific JIT compiler but rather to provide a language-specific interpreter in RPython, a type-inferreable subset of Python. *Hints* to the meta-tracing JIT allow fine-tuning of the resulting JIT compiler. Meta-tracing been successfully applied to Python with PyPy [8].

3 Optimization Approach

Our optimization detects common patterns of how instances of value classes (*value objects* for short) reference each other. It then introduces short forms for these patterns, which we call *shapes*, that make it possible to represent these patterns more efficiently in memory.

A straightforward value object representation would be a chunk of memory that stores a pointer to the class of the value object, followed by its contents that typically consist of pointers to all the fields of that value object. We call the contents the value object's *storage*. The class describes the storage, e.g. how many fields there are and

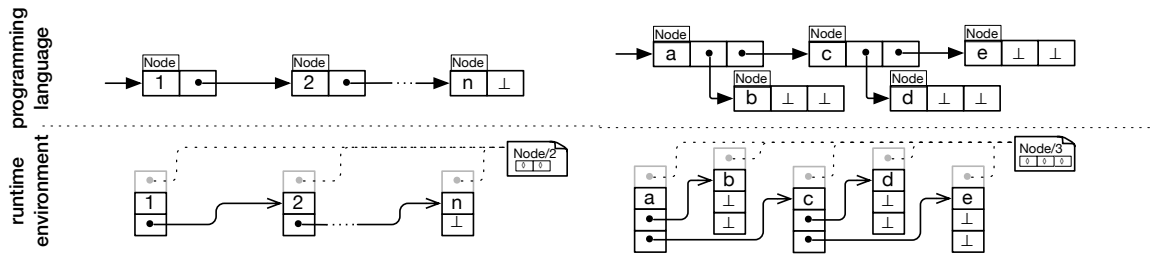


Figure 1: Value class representation for a linked list and a tree. Top: the language view; bottom: runtime view with *storage* and *shape*

how they are to be interpreted. This representation corresponds very closely to the programmers' view on value classes.

In our approach, the storage area remains, but the pointer to the class is replaced by a pointer to the shape. Like in the regular representation, the shape determines the class of the value object and hence the meaning of its contents. Two examples for this separation can be found in Figure 1. For every value class there is a *default shape* that has no additional information compared to directly storing the class. If the default shape was always used, the representation would be completely equivalent to the straightforward one.

The difference to the straightforward representation is that a shape does not necessarily describe only the class of the value object. Rather, a shape can additionally describe the shape of referenced value objects, recursively. If a referenced value object's shape is not specified in the referencing object's shape, it is stored as a reference in the storage. If the shape is specified, that value object's content is inlined into the referencing value object's storage. This process can be applied recursively.

To actually save memory, a shape has to be shared by as many value objects as possible. Indeed, if every shape was used by only one object, the memory use is not improved. Therefore, a new shape must only be introduced after runtime profiling makes sure that it occurs often enough.

3.1 Shapes and their recognition

A *shape* describes the abstract, structural representation of composite value objects and is shared between all identically structured value objects of the same value class, denoted by its name.

Shapes are recursive; they consist of sub-shapes for each field in a value object's storage. A special type of shapes denotes unaltered access to object content and termination of shape recursion. Value objects with these shapes are treated as black boxes, e.g. scalar data or unoptimized objects, they are stored directly in the storage. This is depicted in the bottom part of Figure 1; all three nodes in the list share the

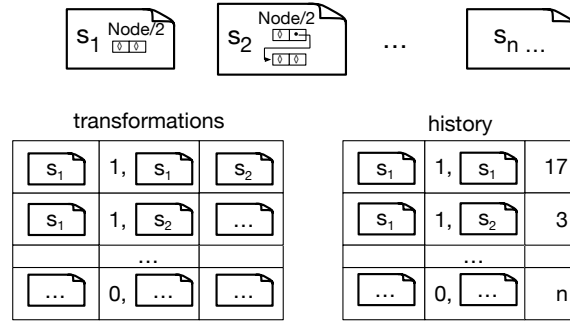


Figure 2: Shapes and supplementary data structures: transformations rules and history of encountered sub-shapes

same shape, which denotes that each node consists of two references with *direct access* shapes. The same holds for the nodes of the tree in that figure, but with three references.

Storing the shape of value objects may seem redundant given that the shape matches what it tries to describe. This only holds as long as no optimization has taken place. In this case, a value object’s shape is the *default shape* of its value class and solely consist of *direct access* sub-shapes. The shapes in Figure 1 are the default shapes for their value classes.

Further, a mapping of replacement options for inlining (the *transformation rules*), and profiling data built up during object creation to aid the creation of new transformation rules (the *history*) are supplementary structures that we use to aid the inlining process.

History

The immutability of value objects demands that all to-be-referenced value objects that will constitute its content have already been constructed beforehand. Hence, their shapes will be available at construction time and we can count occurrences of *sub-shapes* at specific positions in the value object. That way, we obtain a histogram of all possible shapes a referenced value object can have. In Figure 2, e.g. for shape s_1 at position 1, the shape s_1 itself has been encountered 17 times as sub-shape, while shape s_2 has been encountered 3 times as sub-shape in that position.

When a certain threshold of encounters has been reached, we generate a new transformation rule.

Transformation rules and recognition

The transformation rules are mappings $Shape \times Position \times Shape \rightarrow Shape$ that drive the inlining process. When constructing a new value object, they are consulted by

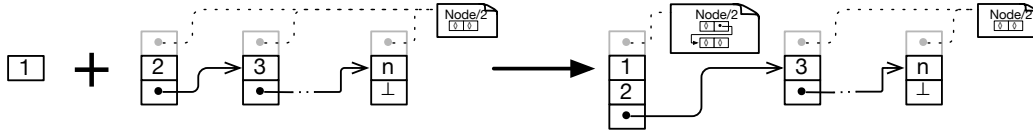


Figure 3: When creating a new node value object that should contain “1” and the list as shown, a new value object that merges the “1” with the “2” object and a different shape is created instead.

the inlining algorithm. These mappings can be specified prior to program execution or inferred dynamically based on shape history.

Upon value object creation, just after updating the shape history, we check whether the sub-shape counters hit a certain threshold, and if so, proceed to create a new shape that combines the value object’s current shape with the sub-shape that hit the threshold. In this new shape, we replace the *direct access* sub-shape at the position of the threshold hit with the sub-shape found in the history entry. The position of the hit, the sub-shape at that position, and the newly created shape are then recorded as new rule in the transformations table. Considering Figure 2 as example, shape s_2 would be the result of turning the history entry $(s_1, 1, s_1, 17)$ into the transformation rule $(s_1, 1, s_1) \mapsto s_2$. The structure of shape s_2 is the structure of shape s_1 but with another s_1 structure in the final position.

We call the process of recording the shape history and inferring transformation rules *shape recognition*.

3.2 Compaction though inlining

Since value objects are immutable, compaction does only need to happen when creating new ones. With this premise, our optimization technique works by inlining the to-be-referenced value objects into the to-be-created value object upon its creation.

When a new value object is created, we handle the default shape s for the type and the value object’s new content c as specified in the algorithm in algorithm 1. We iterate over the given new content and for each to-be-referenced value object o_i at position i and its sub-shape s_{c_i} , we look up a replacement shape in the transformations table. If the table contains a mapping, the replacement shape s' is assumed as shape for the to-be-created value object. At that point, the *storage* of the current value object c_i is *spliced* into the current content c instead of the current value object c_i itself; the value object c_i is now *inlined*. After a successful inlining, the new shape s' becomes the to-be-created value object’s shape s and the current content is reiterated from start to allow for possible other transformations due to the shape change. That way, transition chains are possible that may quickly lead to shapes of deeply nested

Algorithm 1: Merging composite objects based on shape

```

Input:  $s$  : Shape,  $c$  : [Object]
 $i \leftarrow 0$ 
while  $i < |c|$  do
   $o \leftarrow c_i$ 
   $s_o \leftarrow O\{shape\}$ 
   $s' \leftarrow transformations_{s,i,s_o}$  or  $s$ 
  if  $s' \neq s$  then
     $c' \leftarrow [c_0, \dots, i-1, O\{storage\}, c_{i+1}, \dots, |c|]$ 
     $s \leftarrow s'$ 
    // rewind over new storage:
     $i \leftarrow 0, c \leftarrow c'$ 
  else
     $i \leftarrow i + 1$ 
  end
end
return  $s, o$ 

```

structures. Once no further transitions are found, the value object's shape s and the current content c are returned as the shape and storage of the new value object.

The effect of this process is shown simplified with the example in Figure 3: creating a new node consisting of "1" and a rest list as in the figure. We start with a list of "1" and the rest list as initial content for the new value object and shape s_1 as the initial default shape. We iterate over the list and encounter "1" at position 0. For this example, we assume that the transformation table does not contain a mapping for "1" at position 0, thus s' will be s and we continue with the next position. At position 1, we find the rest list with the sub-shape s_1 . In the transformation table, the entry for $(s_1, 1, s_1)$ holds a replacement shape, s_2 . Thus, we inline the current value object's storage into the current content as c' , which now has three elements. Note that it is not the shape of the rest list s_c that is changed but rather the shape s of the to-be-created value object. The content, now c , is reiterated but no further transformations are found. The resulting value object is that to the right in Figure 3.

The shape of thusly optimized value objects are themselves subject to the shape-recognition process and eventually, transition rules to more optimized shapes can be created in the default shapes for the value classes. Thus, more specific shapes are directly available for the inlining process. Value objects can be more directly transitioned into the most optimized shape compared to working off a long transition chain.

This inlining technique has two main advantages. First and foremost, inlined value objects take up less space than individual, interreferenced value objects. But even more, the shape of a value object provides structural information in a manner the

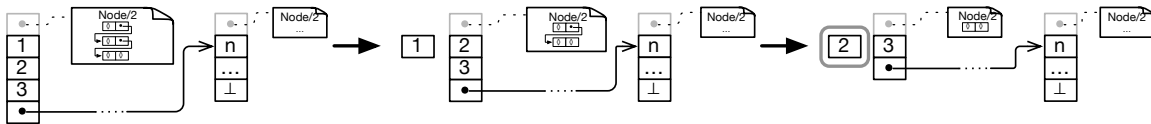


Figure 4: Referenced value object reification. Accessing the second item 2 of the list $l \leftarrow \text{Node}/2[1, \text{Node}/2[2, \text{Node}/2[3, \dots]]]$ by two operations $\text{head}(\text{tail}(l))$ results in two differently reified rest lists to be created. Note how the shapes of the rest lists differ.

meta-tracing JIT compiler can speculate on. This is crucial for optimizing the access to references of a value object.

3.3 Transparent field access

While optimization of data structures takes place during construction, we have to apply the reverse during deconstruction, i.e. when accessing a value object referenced by another. This is no longer trivial, as several (formerly referenced) value object may have been inlined into their referencing value objects. Therefore, we construct new value objects whenever a reference is navigated, essentially reifying it. We use the information a value object’s shape provides to identify which parts of the value object’s storage comprise the value object to be reified. The structural information allows a direct mapping from the language view of the data structure to the actually stored elements. In Figure 4, the structural information in the shape of the leftmost list allow the reasoning that the first element of the storage is equivalent to the head of the language level node value object and the remaining three storage elements are equivalent to the tail of that value object, as recored in the shape. Hence the middle view in that figure; both the element “1” and the rest list have been reified. The same goes for the rightmost view.

Note that this reification is completely transparent to the programming language view. Taking, e.g. the tail of a node value object or accessing the third element of a ternary tree repeatedly, the operations remain the same on the language level, no matter the inlining status of the value objects at hand on the implementation level.

4 Implementation in RPython with a tracing JIT compiler

We implemented our optimization approach in a simple execution model. It provides a λ -calculus with pattern matching as sole control structure and is implemented


as a direct application of the CEK-machine [10]. The only structured data types currently available are value classes. We used the RPython tool chain to incorporate its meta-tracing JIT compiler [4].

To improve performance, the JIT compiler needs to reduce the overhead of these operations. The first step is to treat the transformation tables as constant when a function is compiled. This allows the JIT compiler to compile value object creation down to a series of type checks for the types of the referenced value objects. While the transformation tables are not constant *per se*, we instruct the JIT compiler to treat them as such for all practical purposes.

Second, we have to avoid the otherwise necessary reification of referenced value objects when it is being read from a value object it has been inlined into. For that, the observation that most of these intermediate value objects are actually short-lived is crucial; most value objects are created just to be either immediately discarded or consumed in another, typically larger data structure. As a concrete example, typical linked list operations deconstruct the list they are working on. Hence, if the tail is read off a linked list node which has the tail inlined (as the transition from left to middle in Figure 4) and needs to be reified, that tail is usually soon deconstructed itself into its head and tail components (as the transition from middle to right in the same figure). This allows the tracing JIT compiler to optimize the reading of fields that need reification. Since the value objects allocated when reifying a field are short-lived, the built-in escape analysis [5] will fully remove their allocation and thus remove the overhead of reification.

5 Results

We report the performance of five micro-benchmarks, i.e. their execution time and their maximal memory consumption (resident set size). The benchmarks chosen are *append*, *filter*, *map*, and *reverse* on very long linked lists and the creation and complete prefix traversal of a binary *tree*.

In the left part of Figure 5, the execution time of all benchmarks is reported. Our implementation, labeled *prototype* , is significantly faster—from two to ten times faster—for all but the tree benchmark, where our implementation is second to just the ahead-of-time (AOT) compiled OCaml version. However, the other two RPython implementations are significantly slower than expected; the Pycket interpreter uses the same CEK execution model as our implementation. It is possible that not the value class implementation but the interpreter style is responsible for most of the execution time. Nevertheless, our implementation is still significantly faster than both RPython implementations.

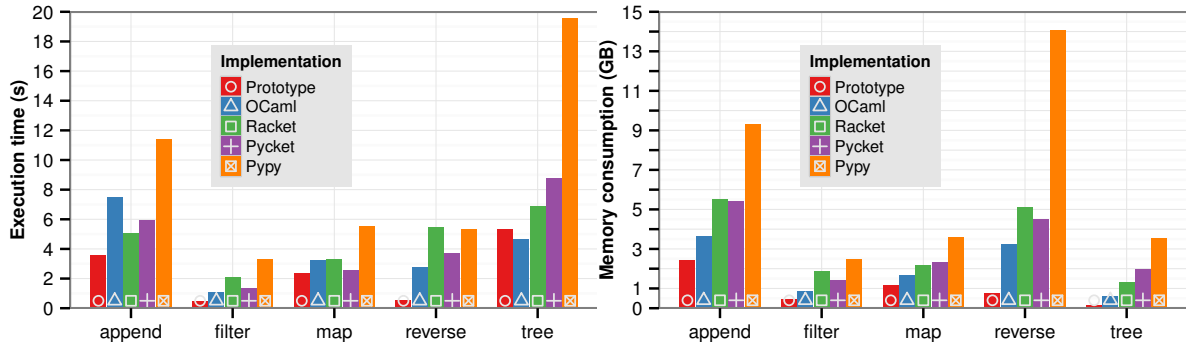






Figure 5: Benchmarking results. Each bar shows the arithmetic mean of ten runs for execution time (left) and memory consumption (right). Lower is better

For memory consumption, shown in the right part of Figure 5, our implementation always uses significantly less memory than the other implementations.

The results suggest that our approach is viable and warrants application to other programming languages.

Compared Implementations

We included OCaml , Racket , Pycket , and Pypy  in the comparison. OCaml provides a concept similar to value classes with its algebraic data types and its execution model similarities to our implementation. OCaml produces native binaries. Racket’s *cons* cells, *structs* and classes can act as value classes. Racket acts as virtual machine with hand-written JIT compiler. Pycket [7] is a RPython implementation of Racket and provides a meta-tracing JIT compiler. Pypy is the RPython implementation of Python and has a meta-tracing JIT compiler. While Python has no actual concept of value classes, we used regular classes in a value class manner, and then Pypy is able to handle them as value classes. We intended to also include the standard Python (CPython) but it was too slow and would have rendered the comparison meaningless.

6 Related Work

From a data structure optimization point of view, value classes are similar to the notion of *algebraic data types* as found in languages in the ML family. Hence, optimizations done to this category of data structures are relevant to value classes.

Wimmer has proposed object inlining [18] as a general data structure optimization for structured objects in Java. Superficially, this approach is similar to this work, yet

object inlining is restricted to statically typed object oriented languages like Java, as the approach needs full knowledge of all class layouts.

The idea to improve data structures to gain execution speed was proposed especially to improve operations on linked lists in functional languages, e.g. by unrolling [16]. Typically, those optimizations are restricted to linked lists.

One of the key effects in our optimization is avoiding to allocate intermediate data structures. In that respect, *hash consing* [9], as used in functional languages for a long time, is related to this work. However, hash consing typically works at the language level using libraries, coding conventions, or source-to-source transformations. It is not adaptable at runtime.

Deforestation [17] has the aim to eliminate intermediate data structures and is in this respect related to our approach. However, deforestation deliberately works through program transformation and does not incorporate dynamic usage information.

7 Conclusion and Future Work

Our approach to just-in-time optimization of value classes provides very good first results both for execution time and memory consumption for a prototype implementation on selected micro-benchmarks. They are promising and allow us to investigate the matter further. Immediate next steps include the integration of our approach into existing programming language implementations. Here, languages that already have an implementation with a meta-tracing JIT compiler would be obvious candidates. Then, larger and more real-world benchmarks can be tackled.

Our aim is then to broaden the scope of our approach beyond value classes. We want to support objects that have identity as well as mutable objects. Yet, in the context of our optimization, these need more in-depth investigation.

References

- [1] D. F. Bacon. “Kava: a Java dialect with a uniform object model for lightweight classes”. In: *Concurrency and Computation: Practice and Experience* 15.3-5 (Feb. 12, 2003), pages 185–206. doi: 10.1002/cpe.653.
- [2] D. Bacon, S. Fink, and D. Grove. “Space- and Time-Efficient Implementation of the Java Object Model”. In: *ECOOP 2002 — Object-Oriented Programming*. Edited by B. Magnusson. Volume 2374. Lecture Notes in Computer Science.

- Springer Berlin / Heidelberg, May 29, 2002, pages 13–27. DOI: 10.1007/3-540-47993-7_5.
- [3] V. Bala, E. Duesterwald, and S. Banerjia. “Dynamo: A Transparent Dynamic Optimization System”. In: *ACM SIGPLAN Notices* 35.5 (2000), pages 1–12.
 - [4] C. F. Bolz. “Meta-tracing just-in-time compilation for RPython”. PhD thesis. Mathematisch-Naturwissenschaftliche Fakultät, Heinrich Heine Universität Düsseldorf, 2012.
 - [5] C. F. Bolz, A. Cuni, M. Fijałkowski, M. Leuschel, S. Pedroni, and A. Rigo. “Allocation Removal by Partial Evaluation in a Tracing JIT”. In: *Proceedings of the 20th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. PEPM ’11. Austin, Texas, USA: ACM, 2011, pages 43–52. DOI: 10.1145/1929501.1929508.
 - [6] C. F. Bolz, A. Cuni, M. Fijałkowski, and A. Rigo. “Tracing the Meta-level: PyPy’s Tracing JIT Compiler”. In: *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*. ICPOOLPS ’09. Genova, Italy: ACM, 2009, pages 18–25. DOI: 10.1145/1565824.1565827.
 - [7] C. F. Bolz, T. Pape, J. Siek, and S. Tobin-Hochstadt. “Meta-tracing makes a fast Racket”. In: *Dyla’14*. Edinburgh, United Kingdom, June 2014.
 - [8] C. F. Bolz and L. Tratt. “The impact of meta-tracing on VM design and implementation”. In: *Science of Computer Programming* (2013). DOI: 10.1016/j.scico.2013.02.001.
 - [9] A. P. Ershov. “On Programming of Arithmetic Operations”. In: *Communications of the ACM* 1.8 (Aug. 1958), pages 3–6. DOI: 10.1145/368892.368907.
 - [10] M. Felleisen and D. P. Friedman. “Control operators, the SECD-machine and the λ -calculus”. In: *Proceedings of the 2nd Working Conference on Formal Description of Programming Concepts - III*. Edited by M. Wirsing. Elsevier, 1987, pages 193–217.
 - [11] A. Gal, C. W. Probst, and M. Franz. “HotpathVM: An Effective JIT Compiler for Resource-Constrained Devices”. In: *Proceedings of the 2nd International Conference on Virtual Execution Environments*. VEE ’06. Ottawa, Ontario, Canada: ACM, June 14, 2006, pages 144–153. DOI: 10.1145/1134760.1134780.
 - [12] J. G. Mitchell. “The Design and Construction of Flexible and Efficient Interactive Programming Systems”. PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University, 1970.
 - [13] M. E. Noth. “Exploding Java Objects for Performance”. PhD thesis. University of Washington, 2003.

- [14] J. B. Sartor, S. M. Blackburn, D. Frampton, M. Hirzel, and K. S. McKinley. “Z-rays: Divide Arrays and Conquer Speed and Flexibility”. In: *SIGPLAN Notices* 45.6 (June 2010), pages 471–482. doi: 10.1145/1809028.1806649.
- [15] J. B. Sartor, M. Hirzel, and K. S. McKinley. “No Bit Left Behind: The Limits of Heap Data Compression”. In: *Proceedings of the 7th International Symposium on Memory Management*. ISMM '08. Tucson, AZ, USA: ACM, 2008, pages 111–120. doi: 10.1145/1375634.1375651.
- [16] Z. Shao, J. H. Reppy, and A. W. Appel. “Unrolling lists”. In: *SIGPLAN Lisp Pointers* VII.3 (July 1994), pages 185–195. doi: 10.1145/182590.182453.
- [17] P. Wadler. “Deforestation: transforming programs to eliminate trees”. In: *Theoretical Computer Science* 73.2 (1990), pages 231–248. doi: 10.1016/0304-3975(90)90147-A.
- [18] C. Wimmer. “Automatic object inlining in a Java virtual machine”. PhD thesis. Linz, Austria: Johannes Kepler Universität, 2008.

Processing Over Encrypted Data: Between Theory and Practice

Eyad Saleh

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

In this report, we summarize our efforts in the past six months into two parts. First, we present our solution towards the question of how to minimize the risk of co-resident tenants in the context of SaaS. Second, we discuss the state-of-the-art in the area of processing over encrypted data.

1 Secure Tenant Placement: SecPlace

Researchers study the tenant placement problem from a workload and resource utilization perspective to optimize the performance and decrease the cost [16, 20, 38, 48]. Although cost and performance are crucial, security is critical as well. We investigate the possibility of developing a security-aware placement algorithm that minimizes the threats of placing several competing tenants on the same database instance.

Consider “Butterfield Bank” and “Baloise Group”, two financial organizations working in the banking field. Both organizations use an on-demand “Performance and Goals” system provided by SuccessFactors. “Performance and Goals” is an on-demand application used to develop and enhance employees’ performance. Both companies compete in the market, and therefore, the data of their personnel, data of employees and managers, and financial records are of much importance. The data for both companies is at risk of being exposed to third parties, either accidentally or maliciously. Accidentally, when for example a software bug or malfunction resulted in sharing one tenant’s data with others, such as if the database developer forget to filter a query that retrieve data by the appropriate filter, this would result in returning the data of all tenants. Or maliciously, when an attacker exploits a weakness in the software stack and gain illegal access to the data, such as SQL-injection or Cross Site Request Forgery (CSRF) attacks. Therefore, our proposed approach is capable of minimizing the risk of such issues by preventing any competing tenants from being hosted on the same database instance. Thus, if one database instance is compromised, data for existing tenants will be at risk while other competitors will not be affected.

1.1 Overview of SecPlace

We introduce *SecPlace*. A mechanism that minimizes the risk of co-resident tenants by implementing the concept that no two competing tenants should share the same infrastructure with each other. *SecPlace* enables the SaaS provider to control the resource (i.e., database instance) allocation process while taking into account the security of tenants as a requirement. *SecPlace* is based on the idea of hosting no more than one tenant T of any business type BT on a any database instance DB . This approach is useful to a certain extent. Yet, due to the fact that several tenants of similar business types will be hosted by the SaaS provider, it is not possible to have distinct tenants on the database server, i.e., at some point in time, we will not be able to avoid hosting more than one tenant with similar business type on the same database instance. Thus, we consider the second factor in our algorithm which is the tenant size S . According to Salesfore and SuccessFactors, tenant's size are classified into three categories: Small, Medium, and Enterprise. When we can not find any database instance to host the new tenant (because all instance are having other tenants of the same business type BT), we look for an instance DB where a tenant of the same BT exist but with different S . For instance, assume that we have five new database instances DB , each one can host up to 20 tenants T . When a new T_1 comes in, we host it onto DB_1 . Now if T_2 is of a different BT than T_1 , it can be hosted on DB_1 , otherwise, it will be hosted on DB_2 . Now assume that all DB_1 - DB_5 contains for example a T of *High-Tech* as its BT , when a new *High-Tech* tenant comes in, there is no room for it because all DB_1 - DB_5 contains the same BT . Thus, we search for a DB that has the same BT but with different S . This is based on our assumption that tenants of same BT are more likely to compete with each other if they have similar S . This is due to the perspective that tenants of different S , for instance *Enterprise* and *small* are hard to find as competing with each other. Take an example of a bank (Enterprise) and a small credit institution (Small), the operations for both of them are completely different, client-base is different, target-market is different, and so on.

The main contribution of our approach is the design and implementation of a mechanism that minimizes the security risk of co-resident tenants without breaking the functionality of the SaaS application or changes to the back-end database. We leverage the fact that competing tenants should not be hosted on the same DB instance, because compromising the instance would result in data leakage, and if competitors are hosted on the same infrastructure, the loss for the tenants will be much worse. Imagine the competitors obtain the data for each other. Furthermore, we implement our approach using Java and the source code is available to download from github [14]. Our system is designed as an independent tool to allow any SaaS provider to download and start using it directly, transparent of the architecture of their applications or data center operations.

The goal of *SecPlace* is to improve the security isolation between the tenants of any SaaS applications hosted in the cloud. We propose the usage of subscription data such as business type BT and tenant size S , so our algorithm can recognize the competing tenants and ensures that no two competing tenants of the same BT and similar S are hosted on the same database instance DB . We assume that a compromised DB that holds data for competing tenants is worse than other DB instances. This is due to the fact that the attacker would be able to blackmails both competing parties because he hold their information. While if no data for conflicting parties is present, the risk remains, but it would be smaller. *SecPlace* can be used to redistribute existing tenants to minimize the security risk of co-location, and can be also used to place the new tenants according to the security requirements managed by our algorithm. When the SaaS provider wants to board a new tenant, our algorithm start looking for the most-secure DB (i.e., DB that does not have similar BT and S of this tenant) to host this tenant.

In practice, tenant-placement is accomplished in two ways. First, place it randomly. Second, the provider uses a formula to select the best-fit DB instances to maximize the resources utilization. Our approach does not conflict with such practice, rather we build on them. In the random placement, our algorithm minimizes the risk as we show later in this work. When the formula is used, our approach is applied to the subset that is resulting from the formula, so a smaller subset that achieve both resource-utilization and our security requirements will be selected. *SecPlace* works in transparent of the provider parameters, such as database schema, number of database instances, number of tenants, etc.

1.2 The Placement Algorithms

We have two placement algorithms. Both are close to each other with a slight difference. We describe them in details in the next section.

Placement Algorithm I

Our algorithm is designed to find the best DB instance to place the tenant into. Best instance means the instance that minimize the risk of being hosted with competing companies. In *Algorithm I*, we do this by checking some attributes of the subscription data, namely the business type (BT). In *Algorithm II*, we include also the tenant size (S).

The algorithm starts by looping on all DB instances. First, it checks if the instance can accept additional tenants, if yes it proceeds. Otherwise, it moves to the next instance. When the database can accept the new tenant, it start checking the list of all tenants in this tenants and compare their BT with the BT of the new tenant, if no match is found, it place it in the instance, increase the number of tenants by one,

Algorithm 2: Tenant Placement Algorithm

input : NewTenant, DBsList, DBsList.TenantsList, and MAX
output: DBsList.NewTenantsList (A new distribution of tenants in all DBs)
initialization foundSameBizType \leftarrow false; foundSameSize \leftarrow false;
isPlaced \leftarrow false;

foreach (db element in DBsList) **do**
 / Check if the DB instance is not full */*
 if db.TenantsCount \neq MAX **then**
 foreach (t element in db.TenantsList) **do**
 / Check if the DB instance has a tenant of same business type */*
 if db.t.BizType = NewTenant.BizType **then**
 foundSameBizType \leftarrow true;
 exitfor;
 end
 end
 if foundSameBizType = false **then**
 / This DB instance does not have a tenant with similar business type */*
 db.TenantsList.Add(NewTenant);
 db.TenantsCount ++;
 isPlaced \leftarrow true;
 exitAlgorithm;
 else
 / We cannot place the tenant here and therefore move ahead to the next DB instance to check */*
 foundSameBizType \leftarrow false;
 end
 end
end

if isPlaced = false **then**
 / We cannot place the NewTenant because every DB instance has at least one tenant with similar business type. Therefore, we run algorithm 2 which includes a check of the second factor - tenant size */*
 Call Algorithm2;
end

and exit the algorithm. If a tenant with the same *BT* is found, we break the loop and move to the next instance. After checking all the *DB* instances, if the value of the parameter *isPlaced* is *false*, this means that all *DB* instances have tenants with similar *BT*. Therefore, we call *Algorithm II*.

Placement Algorithm II

Algorithm II is similar to *Algorithm I*, but with a slight change. *Algorithm I* works only on a single parameter which is the *BT* of the tenants. However, in certain cases, we fail to find a *DB* instance that do not have tenants with similar *BT*, thus, *Algorithm II* is used.

According to *Algorithm I*, we compare the *BT* in line 7, and since the result of *Algorithm I* indicates that all *DB* instance have tenants with similar *BT*, then we replace this by another statement that checks if there is a tenant with s size (*S*) that is similar to the *S* of the new tenant. If not similar *S* is found, then we place the tenant. Otherwise, we break the loop and move to the next *DB* instance. We repeat this until we find a *DB* instance that do not have similar *S*. If *Algorithm II* fails to find such an instance, then it means all instances have tenants with similar *BT* and *S*. And therefore, all instances become identical under our model, then we place the tenant randomly.

1.3 Experiments and Discussion

In our experiments, we assume that we have a cluster with 5 homogeneous *DB* instances. We further assume that every *DB* instance can accommodate up to 20 tenants.

The experiments can be classified into two main categories. In the first category, we assume that the cluster is full and cannot accept new tenants. Our task is to re-distribute the tenants to minimize the risk of co-locating competing tenants. In the second category, we assume that the cluster can accept new tenants. Further, we took the result of the first category of experiments (i.e., the cluster after re-distribution) and assume that this is the baseline for the second category.

In all experiments, we normalize the values based on the minimum of the average of variations in the *DB* instance. Variations means the percent of the existence of any business type in the *DB*. For instance, if an instance has only three *BT*, such as *High Tech*, *Travel*, and *Financial*. Then the variation for this instance would be close to 33 %. Due to space limitations, we only describe experiment 1 and 2 that belong to the first category.

In the Figures 1 and 2, the x-axis represents the if of the *DB* instance, and the y-axis represents the degree of variation between tenants. The maximum the degree of

variation, the better it would be because it means that the co-locating of competing tenants is as minimum as possible.

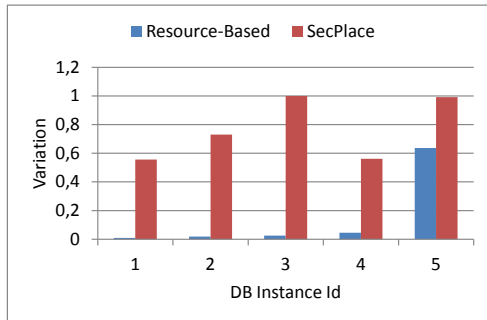


Figure 1: Worst-Case Placement

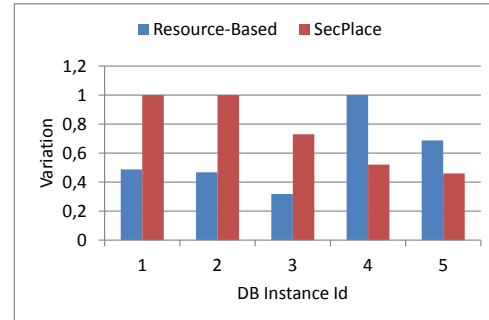


Figure 2: Uniform Distribution

Experiment I: Worst-Case Placement. We assume in the this experiment that the current status of distribution is at the worst case. This means that all tenants of same business type BT and same size S are co-located at the same DB instance. Figure 1 shows the result after applying our approach. It is obvious that the variation of tenants in reference to our factors (i.e., BT and S) has been increased very much in all DB instances.

Experiment II: Uniform Distribution. We assume in the this experiment that the distribution is uniformly random. We can see from Figure 2 that we enhance the level of security on 3 instances, yet instance 4 and 5 still have issue. We justify this by two reasons. First, If we stop this experiment in the middle of execution, we can tell that all DB instances would have the same level of variation, but as we continue with the algorithm, the enhancement is done sequentially, so the better variation will go to the DB instances that our algorithm hits first. Second, we assume in all the experiments that every tenant could be migrated only once. Thus, we cannot balance the variation on instances 4 or 5 by re-distribute the tenants again.

2 Processing Over Encrypted Data

Encryption was previously used to encrypt data during transmission to prevent eavesdroppers from intercepting the communication and receive the data. In addition, it prevents unauthorized disclosure of confidential data in storage. However, in the modern era, and due to the advances in the computer hardware that yielded to fast processing units, and motivated by the increasing adoption of the Cloud model, the need and possibility of processing over encrypted data is no longer infeasible.

The idea of processing over encrypted data was first introduced by Rivest et. al. [46] in 1978. The main hypothesis was that useful privacy homomorphisms (i.e., encryption schemes) may exist to support processing data while being encrypted. They discussed some examples of basic operations that could be applicable, such as addition on integers. It was shown later by Brickell and Yacobi [12] that security flaws exist in Rivest et. al. approach. In particular, known plaintext and known ciphertext attacks. In 1985, Blakley and Meadows [6] followed Rivest approach and propose an encryption scheme that supports some statistical operations such as sum and average. Despite the previous initiation efforts, Feigenbaum [22] in 1986 and Abadi et. al. [1] in 1987 can be considered as the first proposals to discuss the concept of processing over encrypted data in its general form, and the first to use formal definitions and strict security requirements.

Although several approaches and efforts were discussed by cryptographers as explained above, the hype of processing over encrypted data did not receive a considerable attention by the database community until 2002, when Hacigümüs et. al. [31] discussed the idea in the context of database applications. A restricted version that focuses only on search over encrypted documents has been previously published by Song et. al. [49] in 2000. Since then, a rapidly growing literature evolved, and yielded to several approaches and solutions that have been discussed in the community, such as *Fully Homomorphic Encryption (FHE)* that has been introduced by Gentry [25], *CryptDB* [44], *CloudProtect* [17], and *Silverline* [45].

This introduction is followed by a discussion of homomorphic encryption schemes, both fully and partially. We then present the recent advances of processing on encrypted data. Afterwards, we discuss the current industry offerings. Finally, we outline the limitations and open issues in processing over encrypted data.

2.1 Homomorphic Cryptosystems

Existing encryption schemes can be classified into two main categories in terms of homomorphic properties. Namely, *Fully Homomorphic Encryption (FHE)* and *Partially Homomorphic Encryption (PHE)*. Homomorphic is an adjective that describes a special property of an encryption scheme. That property, at an abstract level, can be defined as the ability to perform computations on the ciphertext without decrypting it or even knowing the keys.

Fully Homomorphic Encryption (FHE)

In the cryptography community, FHE was thought to be impossible to achieve until 2009, when Gentry announced his new approach [26, 27]. It is considered one of the recent breakthrough of cryptography. FHE supports arbitrary computation over encrypted data and remains secure (achieve semantic security) as well. In his

PhD thesis [25], he discussed how his schemes can be constructed. Before Gentry's achievement, all encryption schemes that preserve a homomorphic property were able to support only a single operation over encrypted data. The main contribution of Gentry's work is the supporting of two homomorphic operations at the same time. Namely multiplication and addition. Correspond to AND (\wedge) and XOR (\oplus) in boolean algebra. The remarkable value of supporting these two boolean functions is that any computation can be converted into a function that contains only (\wedge) and (\oplus) as we explained below. Finally, an open-source implementation of FHE is available [34, 35].

In algebraic terms, any computation can be expressed as a boolean circuit. Usually, several techniques can be used to convert a function (i.e., computation) into a more simple or efficient one. However, they can also be used to transform a function to use specific boolean operations. For instance, $\neg A$ can be expressed as $A \oplus 1$, another example would be $A \vee B$, this can be transformed into $(\neg A) \wedge (\neg B)$ which is equivalent to $(A \oplus 1) \wedge (B \oplus 1)$. By utilizing such techniques, all functions can be converted into a series of (\wedge) and (\oplus) operations. This is the basis behind the remarkable achievement of Gentry's work.

Clearly, converting even a simple application into a series of boolean circuits requires enormous number of operations. Moreover, both the complexity of encryption and decryption and the size of the ciphertext hugely grow. Despite that Gentry is trying with the support of his colleagues at IBM to optimize the first version of his algorithm [10, 19, 28], his approach remains very expensive and hence impractical.

Partially Homomorphic Encryption (PHE)

Several PHE systems have been discussed in the Literature. As we explained earlier, Rivest et. al. [46] in 1978 was the first to introduce the concept of privacy homomorphism. Then, several researchers follow such as ElGamal and Paillier. Below is a brief discussion of the most well-known partially homomorphic cryptosystems.

Un-Padded RSA Cryptosystem: Conceptually introduced by Diffie and Hellman [18] in 1976 (as the first proposal of public-key cryptography). Motivated by Diffie and Hellman article, Rivest et. al. [47] followed in 1978 and invented the first practical public-key encryption scheme that is widely known as RSA. The RSA scheme (without padding) is homomorphic for the multiplication operation with the modulo n . Given two plaintexts m_1 and m_2 with their corresponding ciphertexts c_1 and c_2 , where

$$c_1 = \mathcal{E}(m_1) \bmod n, \quad c_2 = \mathcal{E}(m_2) \bmod n$$

$$\begin{aligned} c_1 \times c_2 &= \mathcal{E}(m_1) \times \mathcal{E}(m_2) \bmod n \\ &= \mathcal{E}(m_1 \times m_2) \bmod n \end{aligned}$$

is an encryption of $m_1 \times m_2$. In short, the multiplication of the ciphertexts modulo n is equivalent to the multiplication of the plaintexts.

ElGamal Cryptosystem: In contrast to RSA scheme, that is based on the difficulty of factoring $n = p \cdot q$, where p and q are two large prime numbers. ElGamal [24] in 1984 proposed what is known as ElGamal Cryptosystem. His scheme is based on problem of solving discrete logarithms. The homomorphic operation that ElGamal supports is the multiplication over encrypted messages. Given two ciphertext c_1 and c_2 that are encryption of m_1 and m_2 , using random values k_1 and k_2 respectively, then

$$\begin{aligned}(c_1, c_2) &= (\alpha^{k_1} \alpha^{k_2}, (m_1 \cdot y^{k_1})(m_2 \cdot y^{k_2})) \\ &= (\alpha^{k_1+k_2}, m_1 m_2 \cdot y^{k_1+k_2})\end{aligned}$$

is a valid encryption of $m_1 m_2$. One notable drawback of ElGamal scheme is that the size of ciphertext is double the size of the plaintext message. Interestingly, several variants of ElGamal have been proposed, such as Cramer et. al. [15] that is homomorphic on the additive operation.

Paillier Cryptosystem: As discussed in [42], the Paillier scheme is based on the problem of *Composite Residuosity Class*. i.e., given a composite n and an integer z , it is hard to decide whether there exists y such that $z \equiv y^n \pmod{n^2}$. The difference of Paillier from RSA is the usage of square number as modulus, where $n^2 = pq$ is the product of two large primes. As for homomorphic property, the scheme supports two main operations, one is addition. Let $c_1 = g^{m_1} r_1^n \pmod{n^2}$ and $c_2 = g^{m_2} r_2^n \pmod{n^2}$, then

$$\begin{aligned}c_1 c_2 \pmod{n^2} &= g^{m_1} r_1^n g^{m_2} r_2^n \pmod{n^2} \\ &= g^{m_1+m_2} r_1 r_2^n \pmod{n^2}\end{aligned}$$

is a valid encryption of $m_1 + m_2$.

Based on the above discussion, it is very clear that homomorphic encryption schemes are useful. However, they lack general computation support since they can perform limited types of operations, and hence the question of designing full functional systems that process encrypted data using only homomorphic schemes is still an open challenge.

2.2 State of the Art

Existing literature of processing over encrypted data can be classified into three main categories: (i) Systems that utilize homomorphic encryption schemes, (ii) Client-Server Splitting Approaches, and (iii) Trusted-Hardware Systems. Below we discuss systems that fall under these categories.

Systems Based on Homomorphic Schemes

CryptDB [44] can be considered as one of the “partially” practical systems that utilized several homomorphic schemes to support database functionality. Their approach is basically built on two main ideas. First, Use SQL-aware encryption schemes to efficiently execute queries. And second, use onions of encryption and adjust them dynamically at the run-time based on the required functionality. The idea of SQL-aware encryption schemes is a kind of mapping between the operation required and the homomorphic scheme that can supports it. It is an interesting idea and will be taken into consideration in future research on data confidentiality in the Cloud. However, onions of encryption cause extra overhead. One major draw back of *CryptDB* is the lack of support for *Stored Procedures* (where the SQL code is integrated into the DBMS itself). This is crucial because the best-practice in developing enterprise applications is to use *Stored Procedures*. Moreover, *CryptDB* is targeting only transactional workloads, applications with analytical workloads or very large and complex databases are infeasible to run over *CryptDB*.

Client-Server Splitting Approaches

Several approaches that utilize the concept of client-server query split have been discussed by the community [31–33, 36, 45, 51]. Below we discuss examples of such systems.

Silverline [45] keeps the data at the server-side confidential by encryption in away that is transparent to the application and being able to have some functionality on it as well. *Silverline* proposed to dynamically analyse the application to determine which parts of the data can be functionally encryptable; it assumes that any data that is never interpreted or manipulated by the application is encryptable. For instance, a SELECT query in an HRM applications that searches for all records match the employeeID ‘Jan’ is not required to interpret the actual string ‘Jan’ and hence can execute the query if it would be encrypted. As for key-management, it divides the users into groups, and assigns a single encryption key to this group, facilitates encryption and information sharing at the same time. While *Silverline* seems to be a practical to some extent, the main drawback is that it requires analysis of the application and the data to determine which parts can be encrypted, which we believe to be an expensive task, also a repetition of this process will be required whenever a change to the application or upgrade is taking place. Also, major part of the data will still be stored in plain-text, thus privacy and data compromise issues still open.

In contrast to *Silverline*, Hacigümüs et. al. [31] proposed to store the entire data in an encrypted form on the provider’s side, and introduced an algebraic framework for query rewriting. The framework divides every query into two parts, execute the first part on the encrypted version (i.e., stored on the server’s side), and then perform client-side post-processing on the result come from the server. The efficiency of

this approach relies on how data partitioning and query splitting and rewriting is accomplished.

Finally, *Monomi* [51] utilizes both techniques, PHE and split client-server query execution. In contrast to CryptDB [44] that focuses on transactional workloads, *Monomi* is mainly targeting analytical workloads. Since queries are not known ahead of time, and to maximize efficiency, *Monomi* introduces an optimization designer that choose an appropriate database design (on the server) according to the target workload. Further, it provides a planner that selects the query execution path for every query. Additionally, it provides some techniques such as per-row pre-computation and pre-filtering. From practical point of view, *Monomi* is far from being practical for several reasons. First, performance cost is very expensive. Queries over large (plain) datasets often have the problem of I/O Bottleneck, imagine adding the cost of being encrypted. Second, choose a physical design at the runtime, pre-filtering and pre-computation are complex tasks and depend mainly on the targeted workload. Thus, the task need to be repeated for every workload or application.

Trusted-Hardware Systems

To perform a computation on encrypted data, the keys need to be present at the server to decrypt the data, compute, and then encrypt again. The drawback of this model is the vulnerability of compromising cryptographic keys. Therefore, several techniques and approaches have been discussed by researcher and industry to overcome such vulnerabilities. These approaches use a secure, tamper-proof hardware components attached to the server to store cryptographic keys and perform computation over encrypted data [4, 5]. Examples of industrial solutions that are in use include secure co-processors, Hardware Security Modules (HSM), and FPGAs.

A more detailed discussion about processing on encrypted data using secure hardware is presented in [21, 40].

2.3 Current Industry Offerings

Oracle introduced *Transparent Data Encryption (TDE)* [41] that provides data-at-rest encryption. The data will be stored on the file systems as encrypted. Yet, and upon request, it transparently decrypt the data for the application to process. TDE supports both column-level and table-level encryption. However, a single key is used for the entire table regardless of how many columns are encrypted. By default, TDE utilizes AES with 192-bit length key as a standard encryption algorithm. However, 128 and 256 bits are also supported in addition to 3DES as an alternative encryption algorithm. To prevent unauthorized disclosure, the keys for all tables are encrypted with a database-server master key and then stored in a dictionary table in the database. Afterwards, the master key is stored in an external secure module outside the

database and is accessible only to the security administrator. As a result, no keys are stored in plain text.

Similar to Oracle, Microsoft offers TDE as well [39]. The main concept of securing data at-rest by utilizing encryption remains the same. However, few differences exist, such as storing the keys for encrypting data in the database boot record in comparison to a dictionary in the case of Oracle. Another major difference is that Microsoft TDE uses three-levels of encryption along with two master keys and one certificate. Namely *Service Master Key (SMK)* and *Database Master Key (DMK)*. First, the SMK is created at the time of SQL Server setup. The *Windows OS-Level Data Protection API (DPAPI)* is used to encrypt the SMK so it remains protected. Second, The DMK is created and then protected by encrypting it using the SMK. Finally, a certificate is generated using the DMK and stored in the master database that is consequently used to encrypt the data encryption key. Worth to mention that the entire user's database is encrypted by the data encryption key.

Navajo Systems (acquired by Salesforce in 2011) [23], CipherCloud [13], and SQL-Cipher [50] all provide techniques to encrypt enterprise data before storing them in the Cloud. For instance, CipherCloud offers, in addition to key management and other things, what they call *Tokenization*. It generates a random values to substitute the original data and store them in the Cloud. The mapping between the random values and the original data is stored at the client's side.

Finally, Google is implementing and testing some partially homomorphic encryptions in a new command-line client-tool that accesses their BigQuery service [7].

Obviously, the above industry offerings are mainly targeted to protect data at-rest and in transit. Supporting functionality over encrypted data, other than basic search or limited queries, remains a challenge and an open issue for both industry and academia.

2.4 Limitations and Open Issues

Encryption schemes that preserve homomorphic property have pros and cons. On one hand, it is a desirable property that allows the user to perform computations on the encrypted data without decrypting it or even knowing the decryption keys. One potential candidate for such need is electronic voting. On the other hand, it is perceived as a drawback or a weakness in the encryption scheme since it cannot satisfy IND-CCA2 requirements, and hence can leak sensitive information. This is drawn from the fact that homomorphic encryption schemes are malleable by design. For instance, a chosen-ciphertext attack by Ahituv et. al. [3] was reported against a homomorphic scheme where the addition operation is supported.

Based on the above discussion, we point out inherited limitations of current schemes and discuss some open problems in the domain of processing over encrypted data.

FHE is impractical

Despite the improvements [10, 11, 19, 30] that follow Gentry's scheme, current proposals of FHE are far from being practical due to the expensive cost to perform operations. For example, An evaluation performed by Gentry et. al. in 2012 [29] for AES-128 circuit showed that it cost about 40 minutes per AES block on a machine with 256 GB of RAM. In addition to the performance cost and high security guarantees it requires, the computation model required by FHE is complex due to the need of converting the application into a boolean circuit that may results in a very large, non-trivial circuit. Therefore, designing an efficient and practical FHE scheme remains an open issue.

PHE schemes are Limited

In contrast to FHE, PHE schemes are more efficient. This is due to the support of only limited functionality. For instance, Paillier takes about 0.005 ms to perform an addition on two ciphertexts. PHE schemes are crucial for systems to process encrypted data because of their practicality. However, they only support partial computations, and hence, cannot be used to build complete functional systems. Yet, and motivated by the previous schemes and advances in cryptography, we believe that more schemes to come that can help in bridging this gap.

Strong Order-Preserving Encryption

Order-Preserving Encryption (OPE) schemes [2, 9] are shown to be insecure and reveal about half of the plaintext [43]. An extension to improve the security of [9] was presented by the same authors in [8]. However, the leakage of nothing except order remains questionable. More recent approaches [37, 43] claim that their schemes achieve ideal security of OPE (i.e., they leaks nothing but order). The applicability of such schemes in a general setting remains a challenge.

3 Conclusion and outlook

In this report, we presented our solution towards the question of how to minimize the risk of co-resident tenants in the context of SaaS. Then, we discussed the state-of-the-art in the area of processing over encrypted data. As for my next step, I am working on a system to answer the third question of my thesis: *How to protect the*

tenant's assets in terms of data confidentiality? by utilizing partially homomorphic encryption schemes mentioned above.

References

- [1] M. Abadi, J. Feigenbaum, and J. Kilian. "On Hiding Information from an Oracle". In: *ACM Symposium on Theory of Computing*. New York, USA, 1987.
- [2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. "Order-preserving encryption for numeric data". In: *ACM SIGMOD Conference*. Paris, France, 2004.
- [3] N. Ahituv, Y. Lapid, and S. Neumann. "Processing encrypted data". In: *Communications of the ACM* 30.9 (1987), pages 777–780.
- [4] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. "Orthogonal Security With Cipherbase". In: *CIDR*. California, USA, 2013.
- [5] S. Bajaj and R. Sion. "TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality". In: *ACM SIGMOD Conference*. California, USA, 2011.
- [6] G. R. Balkley and C. Meadows. "A Database Encryption Scheme Which Allows The Computation of Statistics Using Encrypted Data". In: *IEEE Symposium on Security and Privacy*. Oakland, CA, USA, 1985.
- [7] G. BigQuery. *Encrypted BigQuery Client*. [retrieved: Sep, 2014]. URL: <https://code.google.com/p/encrypted-bigquery-client>.
- [8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. "Order-preserving encryption revisited: improved security analysis and alternative solutions". In: *CRYPTO*. California, USA, 2011, pages 578–595.
- [9] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. "Order-Preserving Symmetric Encryption". In: *EUROCRYPT*. Cologne, Germany, 2009, pages 224–241.
- [10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *Innovations in (Theoretical) Computer Science*. Cambridge, MA, USA, 2012.
- [11] Z. Brakerski and V. Vaikuntanathan. "Fully homomorphic encryption from ring-LWE and security for key dependent messages". In: *CRYPTO*. California, USA, 2011, pages 505–524.
- [12] E. F. Brickell and Y. Yacobi. "On privacy homomorphisms (extended abstract)". In: *EUROCRYPT*. Volume 304. 1987, pages 117–125.

- [13] CipherCloud. *Cloud Data Protection*. [retrieved: Oct, 2014]. URL: <http://pages.ciphercloud.com/Guide-to-Cloud-Data-Protection.html>.
- [14] S. Code. *Implementation of SecPlace*. [retrieved: Sep, 2014]. URL: <https://github.com/johannessianipar/SecPlace>.
- [15] R. Cramer, R. Gennaro, and B. Schoenmakers. "A secure and optimally efficient multiauthority election scheme". In: *EUROCRYPT*. NY, USA, 1997, pages 103–118.
- [16] C. Curino, E. Jones, S. Madden, and H. Balakrishnan. "Workload-aware database monitoring and consolidation". In: *ACM SIGMOD Conference*. Athens, Greece, 2011.
- [17] M. H. Diallo, B. Hore, E. C. Chang, S. Mehrotra, and N. Venkatasubramanian. "CloudProtect: Managing Data Privacy in Cloud Applications". In: *IEEE Cloud*. Hawaii, USA, 2012.
- [18] W. Diffie and M. Hellman. "New Directions of Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pages 644–654.
- [19] M. v. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. "Fully Homomorphic Encryption over the Integers". In: *EUROCRYPT*. Nice, France, 2010.
- [20] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. "Performance prediction for concurrent database workloads". In: *ACM SIGMOD Conference*. Athens, Greece, 2011.
- [21] K. Eguro and R. Venkatesan. "FPGAs for Trusted Cloud Computing". In: *Field-Programmable Logic and Applications*. Oslo, Norway, 2012.
- [22] J. Feigenbaum. "Encrypting Problem Instances, or, ..., Can You Take Advantage of Someone Without Having to Trust Him?". In: *CRYPTO*. Springer-Verlag, 1986.
- [23] Forbes. *Salesforce acquires Navajo Systems*. [retrieved: Oct, 2014]. URL: <http://www.forbes.com/sites/greatspeculations/%202011/08/30/salesforce-com-brings-navajo-into-camp-to-boost-cloud-security/>.
- [24] T. E. Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *CRYPTO*. Santa Barbara, California, USA, 1984, pages 10–18.
- [25] C. Gentry. "A fully homomorphic encryption scheme". PhD thesis. Stanford, 2009.
- [26] C. Gentry. "Computing arbitrary functions of encrypted data". In: *Communications of the ACM* 53(3) (2010), pages 97–105.
- [27] C. Gentry. "Fully homomorphic encryption using ideal lattices". In: *ACM Symposium on the Theory of Computing*. Maryland, USA, 2009, pages 169–178.

- [28] C. Gentry, S. Halevi, and N. P. Smart. “Better Bootstrapping in Fully Homomorphic Encryption”. In: *Public Key Cryptography*. Darmstadt, Germany, 2012.
- [29] C. Gentry, S. Halevi, and N. P. Smart. “Homomorphic Evaluation of the AES Circuit”. In: *CRYPTO*. California, USA, 2012, pages 850–867.
- [30] C. Gentry, A. Sahai, and B. Waters. “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based”. In: *CRYPTO*. California, USA, 2013, pages 75–92.
- [31] H. Hacigümüs, B. Lyer, C. Li, and S. Mehrotra. “Executing SQL over encrypted data in the database-service-provider model”. In: *ACM SIGMOD Conference*. Wisconsin, USA, 2002.
- [32] H. Hacigümüs, B. Lyer, and S. Mehrotra. “Efficient Execution of Aggregation Queries over Encrypted Relational Database”. In: *Database Systems for Advanced Applications*. Jeju Island, Korea, 2004.
- [33] H. Hacigümüs, B. Lyer, and S. Mehrotra. “Query Optimization in Encrypted Database Systems”. In: *Database Systems for Advanced Applications*. Beijing, China, 2005.
- [34] S. Halevi. *HElib: an Implementation of Homomorphic Encryption*. [retrieved: Oct, 2014]. URL: <https://github.com/shaih/HElib>.
- [35] S. Halevi and V. Shoup. “Algorithms in HElib”. In: *CRYPTO*. California, USA, 2014.
- [36] B. Hore, S. Mehrotra, and G. Tsudik. “A Privacy-Preserving Index for Range Queries”. In: *VLDB*. Toronto, Canada, 2004, pages 720–731.
- [37] F. Kerschbaum and A. Schroepfer. “Optimal Average-Complexity Ideal-Security Order-Preserving Encryption”. In: *ACM Conference on Computer and Communications Security*. Arizona, USA, 2014.
- [38] W. Lang, S. Shankar, J. Patel, and A. Kalhan. “Towards Multi-tenant Performance SLOs”. In: *IEEE ICDE*. Washington DC, USA, 2012, pages 702–713.
- [39] Microsoft. *Transparent Data Encryption*. [retrieved: Oct, 2014]. URL: <http://msdn.microsoft.com/en-us/library/bb934049.aspx>.
- [40] R. Müller, J. Teubner, and G. Alonso. “Data Processing on FPGAs”. In: *PVLDB 2.1 (2009)*, pages 910–921.
- [41] Oracle. *Transparent Data Encryption*. [retrieved: Oct, 2014]. URL: <http://www.oracle.com/technetwork/database/%20options/advanced-security/index-099011.html>.
- [42] P. Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *EUROCRYPT*. Prague, Czech Republic, 1999, pages 223–238.

- [43] R. A. Popa, F. H. Li, and N. Zeldovich. “An Ideal-Security Protocol for Order-Preserving Encoding”. In: *IEEE Symposium on Security and Privacy*. Berkeley, California, USA, 2013.
- [44] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. “CryptDB: Protecting Confidentiality with Encrypted Query Processing”. In: *ACM Symposium on Operating Systems Principles*. Cascais, Portugal, 2011.
- [45] K. P. N. Puttaswamy, C. Kruegel, and B. Y. Zhao. “Silverline: toward data confidentiality in storage-intensive cloud applications”. In: *ACM SOCC*. Cascais, Portugal, 2011.
- [46] R. L. Rivest, L. Adleman, and M. L. Dertouzos. “On Data Banks and Privacy Homomorphisms”. In: edited by A. J. R.A DeMillo. D.P. Dobkin and R. Lipton. New York: Academic Press, 1982, pages 169–179.
- [47] R. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public Key Cryptosystems”. In: *Communications of the ACM* 21.2 (Feb. 1978), pages 120–126.
- [48] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. “Predicting in-memory database performance for automating cluster management tasks”. In: *IEEE ICDE*. Hanover, Germany, 2011, pages 1264–1275.
- [49] D. Song, D. Wagner, and A. Perrig. “Practical Techniques for Searches on Encrypted Data”. In: *IEEE Symposium on Security and Privacy*. Berkeley, USA, 2000.
- [50] SqlCipher. *Database Encryption*. [retrieved: Oct, 2014]. URL: <https://www.zetetic.net/sqlcipher/>.
- [51] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. “Processing analytical queries over encrypted data”. In: *VLDB*. Volume 6. 5. Trento, Italy, 2013, pages 289–300.

Migrate Highly-Available Applications to Non-HA-Infrastructure

Daniel Richter

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

High availability can be achieved at different levels. Due to the fact, that implementing high availability at infrastructure or platform level is very expensive, the trend is to focus on application level and improve existing methods and tools and develop new ones. To evaluate the ability to be highly available, an applications has to be examined at three different layers: logic layer, data layer, and execution environment layer. Also, there are tools needed to identify critical code parts that either are used often or are not covered by any test cases. The main goal is to find out—with the help of different testbeds and real world scenarios—which layers affect each other and which characteristics are important to be able to make applications highly available with non-highly-available infrastructures.

1 Introduction

Nowadays, many applications demand (high) availability. More and more services and software components are business critical—simultaneously the number of service users is growing rapidly. Typically, under increasing load systems can be scaled out or scaled up. Scaling up means adding more resources to a machine, whereas scaling out means adding more machines or computing nodes. Which technique one can use heavily depends on an application’s implementation—while scaling up usually is possible without changing any implementation detail, the ability to scaling out is generally limited by the possibility to parallelize a software’s behavior and state.

This fact leads to an interesting dilemma: how can we deal with legacy applications? For many long-time existent applications it is hardly possible to change the code-base. In many cases the effort to find or train programmers, setup necessary tools, or find adequate documentation is very costly for the service providers. To deal with increasing load for these applications and retain its dependability and availability at the same time typically scaling-up is in many cases the only chance.

The migration issue also affects service providers that run applications with special—and in most cases very expensive—highly-available infrastructure. A com-

mon attempt is to abstain from specialized hardware and switch to more cost effective—but usually also more unreliable—infrastructure.

The goal is to find blueprints to migrate applications from highly-available infrastructures to non-highly-available ones.

2 Background

The availability of a system is described by the ratio of the systems uptime to the systems overall runtime. The higher a system's availability is, the lower is its downtime per lifetime. An availability of 99.999 ("five nines") for example means a downtime of 5:16 minutes per year. The availability itself does not indicate at which point in time the downtime (the time an application cannot provide the service its contract promises) occurs. Therefore, there are multiple classifications of availability and therefore high availability. Often, for highly available systems a downtime only is allowed to occur within concrete, pre-defined time slots.

One of the main means to achieve low downtime and high availability is fault tolerance [1, 7]. Fault tolerance consists of two phases: error detection and error processing. Error processing can be performed by either mitigate the impact of an error, or error recovery. Usually, one chooses one of the following options to process errors:

Reconfiguration Change a systems functional structure—add, remove, or replace components—or change the internal processing configuration. The external visible functionality remains the same.

Retry Use redundancy in time (execute the same action again), space (use different service instances), or information (use another data representation),

Repair Remove the fault that causes the error.

A very popular method to implement high availability is spatial redundancy: a service and its components will be distributed among multiple, different machines. In case of one service instance or its hardware infrastructure fails, other instances can take over and compensate a failure. However, this ability to compensate failures depends on the types of errors the chosen fault tolerance patterns can handle (e.g. crash faults, timing faults, omission faults, computation faults), and the understanding of consistency within the application (e.g. strict consistency vs. eventual consistency).

Methods to gain availability can be implemented on different levels:

- The *infrastructure level* consists of the underlying hardware an IT-system is using, such as servers, storage, and network interfaces. Usually, high availability at infrastructure needs special—and mostly expensive—hardware. Those systems generally use redundant power supplies, multiple network cards, or arrays of independent disks.
- The *platform level* consists of the operating system and middleware. Spatial redundancy at this level can be implemented with the use of a hypervisor and virtual machines. Common virtualization solutions provide the ability to replicate a virtual machine images and run them in fail-over configurations such as active-active (all virtual machine instances process the same requests and have to be constantly in a consistent state) or active-passive (one virtual machine instance handles requests and permanently stores its state or snapshots of it; in case of a failure another virtual machine instance is started, restores the state, and takes over the processing).

Important challenges are to detect possible error states (ideally forecast them) and restore respectively migrate virtual machines without having long downtimes [6, 10, 12, 14].

- The *application level* provides a wide range for fault tolerance mechanism implementations—but applicable techniques heavily depend on the specific application. Options to make IT-systems more fault tolerant and available are discussed in section 4
- Some failures only can handled manually at the *user level*. That could be the case when fault tolerance mechanisms at other layers are way to hard or cost intensive to implement, or the error processing step needs human decisions. Because of the duration to process manual tasks, error processing at user layer in most cases cannot guarantee high availability.

3 Existing Testbeds

In order to experiment with various fault tolerance techniques, we designed and implemented some technology demonstrators and testbeds with different execution environments and programming languages, such as:

3.1 The Carrera Racetrack Experiment

The *Carrera Racetrack Experiment* in an embedded system containing a customized slotcar racetrack (extended with several position sensors), and additional custom-

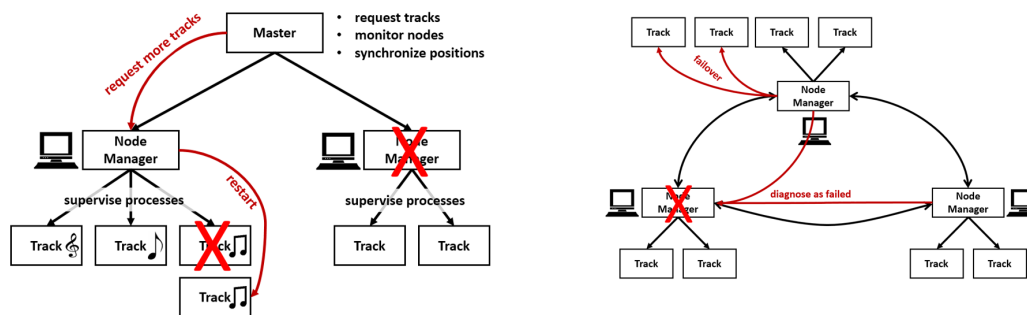


Figure 1: The Unstoppable Orchestra as a distributed fault tolerant system. Left: Crash faults for track playback processes can be tolerated. Right: Fault tolerance for node manager crash faults

made sensor and actuator boards. This testbed is used to investigate fault tolerance mechanisms within an environment with real-time constraints and resource limitations, with embedded components, as well as to handle arbitrary wrong sensor signals. Due to the limited number of sensors, also the most of the time there is only insufficient knowledge about the whole system state. Therefore, control algorithms and fault tolerance mechanisms have to be robust against measurement inaccuracy and incompleteness.

3.2 The Unstoppable Orchestra

The *Unstoppable Orchestra* is a distributed system whose purpose is to evaluate several fault tolerance mechanisms, too. The application's functionality is to play a piece of music, where the music composition itself is divided into separate tracks (usually one track per instrument). A track playback is distributed over multiple computers, so for example a crash-fault of one machine can be compensated through the other running systems. (See figure 1)

This testbed targets the fault tolerant interoperability between multiple execution environments and programming languages (such as C#, C++, or Erlang) and the implementation and evaluation of several fault tolerance strategies. Further investigated aspects are real-time constraints, group communication, client coordination, and leader election.

To evaluate different strategies for varying classes of faults and failures the Unstoppable Orchestra testbed currently covers the following failure types:

Crash faults Crashed track playback processes can be detected and restarted (according to the testbed's parameterization) either by a node manager itself or the orchestra manager. Crashed node managers can be restarted by the or-

chestra manager. A track playback processes can deliver its service without an corresponding node manager.

Omission faults (no result is produced) In case of broken network connections both the node manager (or orchestra manager) and track playback processes detect the inability to communicate with its counterpart. The track playback process will stop its actions autonomously as well as the node manager will start another track playback process.

Timing faults (a result too late or too early) The orchestra master constantly distributes the current position within the tracks to all playback processes. In case of big differences between the orchestra master's send position and the track playback process' own time model this time will be updated (minor time differences can be tolerated due to the characteristics of the human auditory system).

Computation faults (an incorrect value is computed) or corrupted data cannot be simulated because of the lack of computations and missing shared state (with read and write access of all components).

4 Application Level Fault Tolerance and High Availability

While achieving high availability at infrastructure level and platform level, frameworks and middleware providing software fault tolerance at application level still are developing [2, 5, 11].

4.1 Software Fault Tolerance Layers

Approaches for implementing software fault tolerance (and therefore increasing availability) at application level can be views at the following layers: *logic layer*, *data layer*, and *execution environment layer*.

Execution Environment Layer

The execution environment layer is where both the logic layer and the data layer are operated, monitored, and managed. In conjunction with the fact that application level software fault tolerance is desired, the underlying hardware and operating systems are non-highly-available systems. Instead, techniques such as replication and virtualization are used. It is possible to create—based upon template—whole environments with custom settings for networking, computing, and storage

resources. Infrastructure-as-a-service providers offer a wide range of pre-configured VM-images for different purposes.

Within platform-as-a-service environments—such as Microsoft Azure, Amazon WebServices, or Google AppEngine—even the underlying operating system is transparent for developers. The chance to influence things at the execution environment layer is limited or impossible and developers have to use specified tools and frameworks. On the other hand the effort for managing and maintaining hardware and operating systems is minimized. [13, 16]

Logic Layer

The logic layer is where the business logic is executed. The most important property for a logic layer is the ability to parallelize actions, work independently, and handle shared data within a distributed environment. To implement fault tolerance mechanisms, it is also important to know whether standard libraries are available, whether third party components are available, or whether direct resource access (e.g. local file system, threads, or outbound network connections) is possible—which offers possibilities such as synchronization with other instances, delegation of work, or replication and recovery.

Data Layer

The data layer manages data processed by the logic layer. It is responsible for the storage and retrieval of required data as well as caring about data consistency within a distributed cloud environment.

According to the CAP theorem [3, 4], for highly available systems one has to choose between available, partition tolerant, but not (strict) consistent systems; or available, (strict) consistent, but not partition tolerant systems. One consequence is, that one has to analyze the actual needed degree of consistency—is strict consistency really required or is it sufficient to use a storage layer that guarantees eventual consistency so one can use e.g. highly available transactions [2].

Nowadays, there are two popular types of data storage: relational databases with a data schema (e.g. SQL databases) and document databases with more unstructured data (e.g. NoSQL databases, key-value databases, big data stores, message queues, distributed hash tables) [15]. The choice for one of them affects the usable tools, methods, and algorithms.

4.2 Tools for Software Fault Tolerance Evaluation

Failures typically are triggered by environment conditions—such as task scheduling, memory leaks, network outages, or hardware failures. To create reproducible fault

```

1-Verweis
private static TypeAttributes? ClassModifierToTypeAttributes(ClassModifier? modifier)
{
    ClassModifier valueOrDefault = modifier.GetValueOrDefault();
    if (!modifier.HasValue)
    {
        return null;
    }
    switch (valueOrDefault)
    {
        case ClassModifier.Sealed:
            return new TypeAttributes?(TypeAttributes.Sealed);
        case ClassModifier.Abstract:
            return new TypeAttributes?(TypeAttributes.Abstract);
        default:
            throw Error.ArgumentOutOfRange("modifier");
    }
}
}
2-Verweise
private static string AttributeName(System.Type attributeType)
{
    string name = attributeType.Name;
    if (!name.EndsWith("Attribute", StringComparison.Ordinal))
    {
        throw Error.ArgumentOutOfRange("attributeType");
    }
    return name.Substring(0, name.Length - 9);
}
}
1-Verweis
private static bool IsNullableType(System.Type type)
{
    return type != null && type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>);
}
}

```

Figure 2: Example for a code coverage analysis based upon a test suite run. Blue parts a fully covered, yellow ones partial; red code parts were not touched during the test run

tolerance experiments, an environment or framework is needed that can trigger specific failures in a controlled manner. This approach is known as *fault injection* [8, 9].

One way to provoke faults is to stress an application. For example one can increase the number of requests, simulate decreased CPU time or memory scarcity (by adding CPU- or memory-consuming processes), or increase the network latency. Another method to produce faults is to directly manipulate an applications' memory or code and check whether any faults or failures are noticed and can be tolerated. Some of these approaches can be executed at virtual machine level, by modifying the operating system, by patching the language runtime, or just add additional processes.

Fault injection methods can be characterized e.g. by their cost/perturbation (runtime overhead), controllability and triggerability (which faults can be triggered in which granularity) and repeatability [8]. The goal is to create a fault injection tool that has low runtime overhead, can cover and trigger as much failures as possible with reproducible results.

To find suitable points to inject faults, one can use test cases out of a test suite. Using tests has the advantage of reproducible results and—if the tests suite is almost complete—has a high code coverage. With the help of code coverage tools one can find both critical code parts that are used very often—so the fault tolerance and reliability of these code parts could and should be hardened—and code parts that

are used rarely or never—which gives the advantage to complete the test suite and increase its overall code coverage capability (see figure 2).

With the help of code coverage and tracing tools one can prioritize code parts that are worth investigated more precise (usually to check all code parts is very time consuming and expensive). Also, one can find locations that are worth being targeted by software fault injection tools to check whether critical code parts are prone to issues such as data races or concurrency problems, corrupted state, or loop perforation. Ideally, both faults should be detected and also processed by a fault tolerance application.

4.3 Research Questions

The overall goal is to both increase the availability of IT-systems on application level without having high availability guaranteed at infrastructure or platform level. For the main part this can be done by making applications more fault tolerant. Therefore, to first qualify and evaluate the capabilities of different components and even code parts and second make these more fault tolerant and with it available, we want to combine tools for tracing and code coverage, find locations to inject faults and increase the workload within a controlled environment and identify components or code parts, that have to be made more fault tolerant.

The following research questions arise out of the previous considerations:

- What is the impact of load on performance?
- What is the impact of fault tolerance on performance?
- Which influence do the different layers (logic layer, data layer, execution environment layer) have over each other?
- Are there one or more layers more important than others?
- Which layer properties make it easier or harder to migrate existing applications to environments without highly available infrastructure?
- How robust is the software and its components?
- How critical are faults in specific components?
- Which of the points above can be automated?

5 Conclusion and Future Work

Whereas high availability at infrastructure and platform level is a well understood area, the achievement of high availability at application level offers room for improvements. Our existing testbeds—the Unstoppable Orchestra and the Carrera

Racetrack Experiment (see section 3)—provide the potential to try out and evaluate different methods and mechanisms for fault tolerance at application level. However, the available scenarios are mostly artificial, the layers mentioned in section 4.1 are not fully available, and do not provide insights inside real world applications and problems.

This issue is about to be solved in cooperation with the DB Systel, the IT arm of the Deutsche Bahn. They will provide some insights into a real world business critical application running for many years. Based upon documentation, implementation details, experience of developers and end users, and the long time the applications runs, we hope we can get more insights into real world scenarios and can create more complex components with a bigger code base to investigate.

The main goal is to find answers for the questions asked in section 4.3 and to be able to create blueprints for highly-available applications with non-highly-available infrastructures, both for new applications an existing ones.

References

- [1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. “Basic concepts and taxonomy of dependable and secure computing”. In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (2004), pages 11–33.
- [2] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. “Highly Available Transactions: Virtues and Limitations (Extended Version)”. In: *Proceedings of the VLDB Endowment* 7.3 (2013).
- [3] E. Brewer. “CAP Twelve Years Later: How the “Rules” Have Changed”. In: 45.2 (2012), pages 23–29.
- [4] E. A. Brewer. “Towards robust distributed systems”. In: *PODC*. 2000, page 7.
- [5] S. Bykov, A. Geller, G. Kliot, J. Larus, R. Pandya, and J. Thelin. *Orleans: A Framework for Cloud Computing - Microsoft Research*. Technical Report MSR-TR-2010-159. Microsoft Research, Nov. 2010.
- [6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. “Live migration of virtual machines”. In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI’05. Berkeley, CA, USA: USENIX Association, 2005, pages 273–286.
- [7] R. Hanmer. *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.
- [8] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer. “Fault injection techniques and tools”. In: *Computer* 30.4 (1997), pages 75–82.

- [9] R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira. “On fault representativeness of software fault injection”. In: *Software Engineering, IEEE Transactions on* 39.1 (2013), pages 80–96.
- [10] M. Nelson, B.-H. Lim, and G. Hutchins. “Fast Transparent Migration for Virtual Machines”. In: (2005).
- [11] J. M. Paluska, P. Hubert, G. Schiele, C. Becker, and S. Ward. “Vision: a Lightweight Computing Model for Fine-Grained Cloud Computing”. In: (June 2012).
- [12] A. Polze, P. Tröger, and F. Salfner. “Timely Virtual Machine Migration for Pro-active Fault Tolerance”. In: *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, 2011, pages 234–243. DOI: 10.1109/ISORCW.2011.42.
- [13] R. Ranjan and B. Benatallah. “Programming cloud resource orchestration framework: operations and research challenges”. In: (2012). arXiv: 1204.2204 [cs.DS].
- [14] F. Salfner, P. Tröger, and M. Richly. “Dependable Estimation of Downtime for Virtual Machine Live Migration”. In: 5.1 and 2 (June 2012), pages 70–88.
- [15] M. Stonebraker. “SQL databases v. NoSQL databases”. In: 53.4 (Apr. 2010), pages 10–11. DOI: 10.1145/1721654.1721659.
- [16] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Stoesser. “Cloud computing—a classification, business models, and research directions”. In: *Business & Information Systems Engineering* 1.5 (2009), pages 391–399.

Leveraging Programmers' Skills: Interleaving of Modification and Use in Data-driven Tool Development

Marcel Taeumel

Software Architecture Group
Hasso-Plattner-Institut
marcel.taeumel@hpi.uni-potsdam.de

Many programmers use programming tools but hesitate to modify them to accommodate challenging scenarios—taking on the role of a tool builder seems too costly, maybe not even rewarding. We propose a new perspective on graphical tools and provide a framework to build and modify them with low effort, that is, few lines of code and short feedback loops. We applied our framework to context-oriented programming, source code versioning, run-time state exploration, and also its *own evolution*. At the time of writing, we are investigating to which extent programmers do take advantage of our mechanism.

1 Introduction

Programming tools, like other software systems, are created and modified iteratively. Building such tools comes with a subjective trade-off between utility and usability. Thus, programming tools are likely to exhibit deficiencies during actual usage. When tool builders assume an idealized set of prospective tasks and users, there is a chance that they make inadequate assumptions or miss some corner cases. In such a case, the tool user who detects a deficiency can contact the tool builder; bug notices or feature requests can typically be submitted. Then, the user can wait for a resolving response or go on working around the detected deficiency. Now for *programmers* being the tool users, this procedure may be unsatisfactory. If they have access to the tool's sources, they may want to address the problem by themselves to save time.

We propose a new perspective on graphical programming tools and environments such as Eclipse or Visual Studio. We want to express the existing functionality of those tools in *simpler terms* and provide a tool building framework that exposes this expression. Eventually, tool builders and tool users should be roles that any programmer can take on.

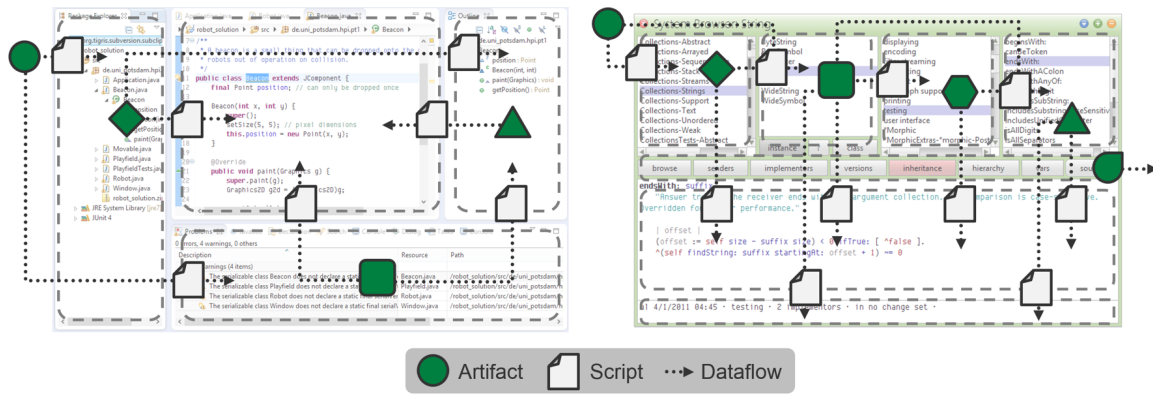


Figure 1: Our data-driven perspective applied to Eclipse (left) and Squeak (right). Software artifacts include projects, files, classes, and methods.

1.1 A Data-driven Perspective

We see graphical tools as data processing pipelines whose intermediate results can be displayed on screen. Software artifacts are repeatedly transformed and prepared for views; programmers interactively explore and modify artifacts through such views. Given this, we want to bring the simplicity and flexibility of creating and composing Unix filter programs from the text-based world into the graphics-based one. Means of *configuration* represent the selection of relevant artifact relationships and the extraction of characteristic information to reveal an appropriate degree of insight. Means of *combination* represent the arrangement of multiple views, each having particular strengths, that cooperate and help programmers see problems from different angles. Means of *abstraction* represent different groups of tools and the notion of tool boundaries like it exists in terms of Eclipse’s perspectives.

Figure 1 applies our data-driven perspective to the programming environments Eclipse and Squeak; each environment consists of rectangular boxes that exchange software artifacts and where scripts prepare them for visualization on screen.

By projecting this data-driven perspective on graphical tools, we support programmers to focus on their domain-specific software artifacts. In contrast to usually not self-explaining interfaces, data-driven tools provide discoverable cues for the whereabouts and happenings of software artifacts. For example, a typical user may reason about the rules of practice in a graphical user interface like this:

If I click on *that file name* on the left-hand side, the environment *somehow* shows that file’s contents in the central, editable area. I have to *remember* that.

In our data-driven perspective, we anticipate thoughts that focus on artifacts and their projections like this:

If I choose *that file representing my module* in the left-hand view, this very artifact will flow to the central, editable view where it is *projected* to its text-based contents. I can *change* that if I want to.

There are already list-based views where programmers can be in control of their software artifacts because each artifact has a distinguishable representation on screen. In our perspective, we can also make the rules of processing such artifacts *explicit and customizable* because we establish *discoverable boundaries* in the graphical user interface.

1.2 VIVIDE: Our Tool Building Framework

We propose a mechanism that supports low-effort construction of graphical tools [16]. It resides between the fields of (1) processing data and (2) presenting data in graphical views on screen. On the one hand, it can provide the necessary glue to decouple both fields and promote their extensibility and reusability. On the other hand, it can reach into one or the other field to provide missing functionality ad-hoc according to specific domains, tasks, or personal preferences. Basically, it is a *script-based, data-driven* approach to *transform* software artifacts and *prepare* them before showing them on screen.

We implemented the framework in the object-oriented programming language Squeak/Smalltalk¹ and use Smalltalk as the scripting language. An exemplary script looks like this; it transforms classes into methods and extracts the selector and some meta information to be displayed in a table widget:

```
[[:class | class methodDict values].
[:method | {
  #text -> method selector.
  #tooltip -> (method author, method timestamp)}]].
```

Our framework has the notion of *panes* as uniform building blocks, which describe where to show information on screen. That is, panes are invisible rectangles with a position and an extent being placeholders for actual content. Encapsulated in each pane, there is set of artifacts, a current script, and a current widget. When new artifacts arrive, the particular pane evaluates its script and updates the intermediate model for its widget.

Panes can talk to each other as illustrated in Figure 2. Such communication allows for modeling the dataflow within one tool and across tools. For each pane, there is an *interaction loop* through the widget, which allows users to influence the dataflow. They can, for example, select which artifacts to process if widgets support such a selection like most list-based ones do.

¹<https://www.github.com/hpi-swa/vivide>

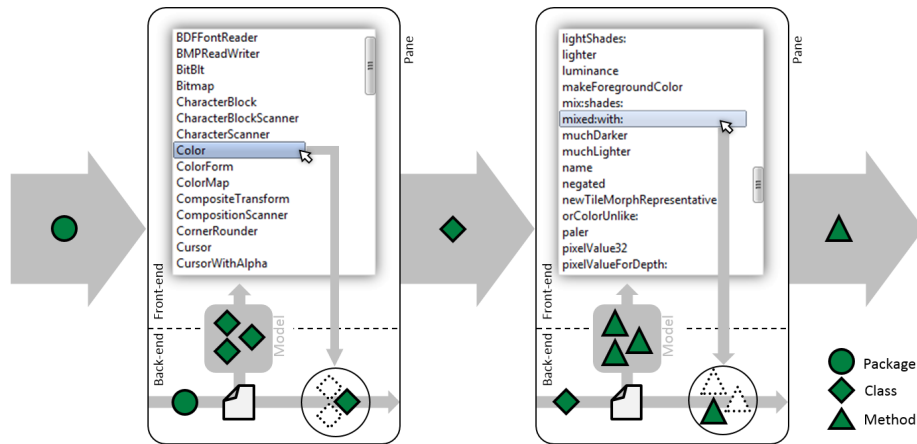


Figure 2: In our framework, graphical tools consist of cooperating panes, which encapsulate interactive widgets and evaluate scripts on incoming artifacts.

1.3 Research Question

We observed that many programmers use their programming tools and environments but often hesitate to adapt those when facing challenging situations in unanticipated domains or tasks. Now this profession of a programmer should encourage not only to create software systems for paying customers but also to constantly improve the own working practice and tool sets. That is why we investigate the following research question:

How can we provide low-effort construction of graphical tools so that programmers can take advantage of informative data sources and insightful visualizations while accommodating unanticipated tasks on their own?

We chose the domain of graphical programming tools because we argue that their interactive views take more advantage of modern display and input technologies than text-based command-line interfaces do. When programmers can conveniently take on both roles—tool builder and tool user—, they may not only create software systems with fewer bugs but also may have more fun in their profession.

2 Applications

We applied our framework to context-oriented programming, source code versioning, run-time state exploration, and also its *own evolution*.

2.1 Improving VIVIDE using VIVIDE

We rebuilt many standard tools of Squeak with scripts to explore the limitations of our framework but also to create a consistent user experience. We argue that a data-driven working practice can only be investigated if programmers do not have to frequently fall-back to traditional tools. The first scripts dealt with listing class and package contents as well as editing source code by means of class definitions and methods. Then, we created scripts for change management and code versioning. At the time of writing, we provide more than 100 scripts that also include debugging and file management. The debugger, for example, can be expressed in our framework like this: the input object for is a suspended *process* object; the first script can extract the *stack* and put it into a list widget; the second script can show *code* for the current selection in the stack; the third script can reveal more context information such as the message *receiver*.

All new kinds of data that VIVIDE introduces can be managed with scripts as well. This includes scripts themselves, which can be treated as software artifacts of their own right and be transformed as well as prepared for some interactive view. We are still in the process of refactoring other framework code and map it to its own concepts to improve maintenance and extensibility.

2.2 Tools for Exploring Run-time Information

When it comes to debugging and exploring run-time information, our framework can save much efforts by adjusting views to concrete, domain-specific data. Typically, default debugging tools fall short on providing appropriate views. They clutter the screen with information that align with the programming language representation (i.e., objects and fields) but not with the particular domain (e.g., persons and telephone numbers).

Scripts are evaluated on concrete objects. As shown in Figure 3, such objects can be part a system's run-time state; scripts can filter, reorder, or extend corresponding information to reveal an appropriate degree of insight. Supporting this, all views can be arranged freely on screen, thus supporting programmers to compare *abstract* code with *concrete* examples side-by-side [17].

2.3 Tools for Context-oriented Programming

There is not only one strategy to design and implement context-oriented programming (COP) for a programming language along with tool support. Indeed, for many languages where COP is available, at least two different implementations exist [1]. Chances are that even basic tools, once created with an effort, cannot be reused

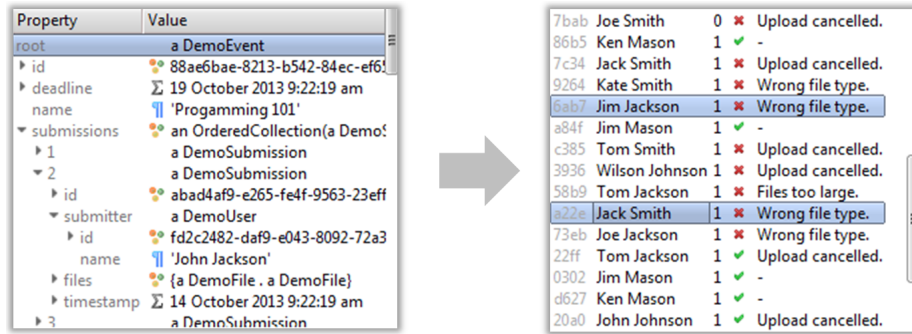


Figure 3: Traditional debugging tools show typically unspecific information (left). Our framework supports tailoring views (right) that fit specific tasks.

across different COP implementations. Even for shared host languages, the extension’s meta-model may differ and hence existing tools have to be adapted. Here, the programmer who implements COP is not necessarily a tool builder, but only a user of similar COP tools. To provide basic tool support for the new implementation, the programmer has to learn the tool framework and become a tool builder, which may involve a steep learning curve.

We applied our framework [15] and created tools that answer essential program comprehension questions related to COP such as “Which layers refine class C?” or “Which layers are currently active in process P?” The resulting tools showed that small scripts are sufficient to provide basic tool support and that extended tasks can be accommodated by configuring or combining those. We assume that they are more or less domain-independent and likely to be reused in any COP projects.

2.4 Tools for Source Code Versioning

During software evolution, representative programming tool support includes means to manage source code versions and log the project’s history. Programmers are advised to follow best practices to maximize the benefits of version control systems (VCS) such as Git or Mercurial. Such practices include *continuous integration* [4], which addresses both committing and testing code on a regular basis, and the advice to only store *small, coherent, complete* change sets with a brief yet descriptive message to improve comprehensibility [18]. In reality, programmers commit frequently about once per day on average [13] but only two-thirds of commits get assigned with an appropriate message [3]. Further research shows [7, 9] that programmers even commit incoherent change sets.

We propose a semi-automatic, interactive approach that supports programmers to (1) identify consecutive changes that indicate continuity of an activity and (2) collect

scattered runs of changes that belong to the same activity. Observing multiple programming sessions, we derived several reusable algorithms that form a multi-stage analysis chain. In each stage, an algorithm assigns changes into *groups* and adds *recommendations* if in doubt. After the analysis, programmers can explore the proposed groups, adjust them manually, or resolve recommendations. We used our framework to provide graphical tool support, that is, visualizing intermediate results concisely and having scripts that reduce the effort for manual adjustments of algorithms.

3 Data-driven Tool Development

In this section, we present our vision of extensible programming environments. Our data-driven tool building approach simplifies not only the creation of throw-away prototypes but also the perfection of well-established tools. It supports the whole tool evolution process.

3.1 Reducing the Costs if the Value is Questionable

If the added value is not clear, the costs for building tools should be low. Programmers should be able to try out any idea, whether it turns out to be beneficial or not. Otherwise, they may miss an opportunity that prevents many errors and saves hours of effortful work. If the added value is obvious and very high, there can be a dedicated tool builder who has the time and means to construct any complex tool. Even in such a case, however, minor adaptation requests of the *users*, which are also programmers, will face the same problem like when building new tools of uncertain value.

Exploring run-time information represents a common example. In debugging sessions, the existing tool support seems to address all tasks to some extent: programmers can navigate the execution stack, browse through object state, or add watchdogs. Typically, this is accompanied with numerous mouse clicks, tool focus changes, and manual note taking. Programmers may notice such inconvenience but hesitate to accommodate the situation because they actually can get their work done the traditional way. The expected improvement might pay off only once. *BUT: Tools could and should automate repetitive tasks, reduce information clutter, reflect the mental model, and hence save time.*

Another example addresses design patterns, which are not meant to be created but to be *discovered*. We argue that this is also true for highly valuable, reusable programming tools—even directly related. For example, existing tools do often support only the underlying programming language concepts such as classes and methods as well as inheritance and polymorphism. When introducing patterns that build

upon these concepts, such as the strategy pattern and the visitor pattern [5], existing tool support falls short. This phenomenon is often called *crosscutting concerns*: modules cannot be expressed concisely at the textual level, code scatters and tangles, and existing tools typically fail to recognize the changed situation appropriately. Language extensions such as COP address this problem by adding concepts while reusing tools for editing text. When programmers would use our approach to create new views on the source code at hand, language extensions could be avoided² and patterns could be discovered along with supportive tools.

3.2 An Evolving Tool Landscape

We see the landscape of tools in a programming environment in a process of constant change and improvement. Fresh prototypes communicate with well-established tools to support programmers in code reading, writing, executing, debugging, versioning, deploying, or documenting. As the programmer may not know whether a prototype should be extended or thrown away, the process of building tools should be transparent to other programming activities.

We envision emergent tool designs. Our framework supports programmers to be both tool user and tool builder. Given that we keep the overhead for switching between roles minimal, there may be no dominant role anymore. When working with concrete artifacts, programmers can create and modify scripts, try out various widgets, and combine them to efficiently use the screen real estate. There is arguably the chance that tools just *happen to be built* while programmers focus on processing their domain- and task-specific data.

Given our framework, we distinguish three stages in the tool building process, which programmers might not notice at all:

Apply (resp. reuse) existing scripts to particular data to explore and modify it in the context of the software system to be built for a customer.

Modify or copy scripts to better fit particular data and widgets. The resulting view can help to get new insights for the problem at hand—or not.

Integrate and persist new scripts to a shared repository of all scripts to use them in different tasks or projects. Once the value of a script can be assessed, it should be possible to promote the resulting tool to distinguish it from other prototypes.

These three stages form a cycle for each script (resp. tool) in the environment. While programmers are focused on their domain-specific tasks and work in a data-driven way, they can mold and improve the whole tool landscape.

²Language extensions need custom tool support anyway.

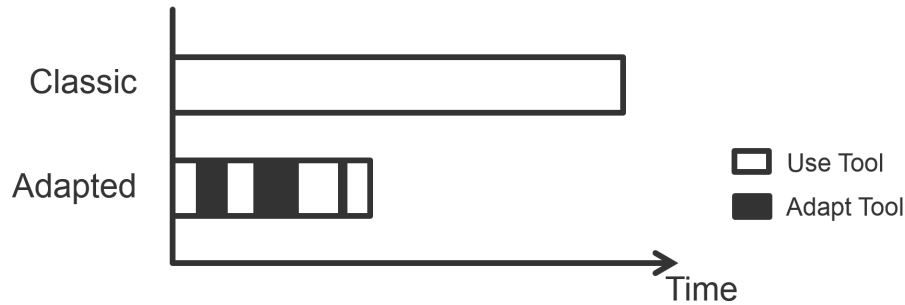


Figure 4: We expect a big effect in exploration tasks when the architectural patterns actually used do not align with the basic language representation.

3.3 Tool Containers

There are many opinions about how to efficiently use the two-dimensional screen space. Layouting flavors include overlapping windows such as in Squeak, tiled areas such as in Eclipse, unbounded spaces [2], and horizontal tapes [6]. They differ in their degree of freedom considering the manual arrangement of widgets.

Our data-driven tool building approach is independent from particular tool containers. Theoretically, panes could be arranged *al gusto*—even on top of each other—because the essence lies in accessing scripts and modifying them *in situ*. Programmers should be able to point with the finger on the screen and say “Why are my artifacts presented this way?” or “I want to see more details about my artifacts in this widget here.”

3.4 Towards a Controlled Experiment

At the time of writing, we are investigating to which extent programmers can benefit from our framework. We are preparing a controlled experiment with a *within subjects* design. The tasks will be about exploring source code and run-time state in the context of fixing bugs or adding features. The control group will not be allowed to access or modify scripts but only use a given set that is small but sufficient and comparable to familiar tools. This includes traditional code browsing, object state exploration, and code execution in a read-eval-print loop.

As for the experimental group, we expect an alternation of tool using and modification activities as shown in Figure 4. To measure both kinds of activities, we want to provide a *modal* interface for reading and writing scripts, which may complicate the use of VIVIDE a little bit compared to its current modeless implementation. In sum, the experimental group should be several times faster than the control group

because they may not have to simulate and remember many things but can try them out and see the results directly.

4 Related Work

The idea of data-driven approaches for building graphical applications manifested itself long time ago in the domain of *visual programming* such as Fabrik [8] and its web-based successor LivelyFabrik [10] do. The programmer can combine scriptable, graphical components and establish dataflow in between. More recent research projects include KScript [12], which employs *functional reactive programming* with declarative, data-driven constructs for building graphical applications.

There are also industry-focused projects such as [14], which combines ActiveX and JavaBeans components as filters into graphical interfaces. As spreadsheet programming is also considered straightforward, the ActiveSheets project [19] explores the possibilities of stream processing and visual output in Excel.

Our approach targets programmers but not necessarily the professional ones. We explicitly appreciate the combination of different language concepts such as object-oriented programming and data-driven scripting. In contrast to the projects mentioned, we consider means of abstraction to facilitate the construction of more complex tools.

5 Conclusion

Live programming systems such as Squeak/Smalltalk provide short feedback loops to promote iterative, low-effort, and high-quality tool construction. Programmers can modify pieces of source code and immediately observe changed behavior in running programs. Graphic frameworks such as Morphic [11] leverage this idea for programs with interactive, visual output. Programmers can directly explore and adapt graphical objects and hence shape the user experience as desired.

With our data-driven perspective, we proposed a novel mechanism to further facilitate the idea of modifying the tools in use and applied it to a range of graphical tools for the programming domain. Given this, programmers can perceive the requirements for being both tool user and tool builder differently. We think that this perspective on graphical tools can inspire the creation of new trade-offs in modularity for both data-providing projects and interactive views.

References

- [1] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid. “A Comparison of Context-oriented Programming Languages”. In: *Proceedings of the 1st International Workshop on Context-Oriented Programming*. ACM. 2009, page 6.
- [2] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola Jr. “Code Bubbles: Rethinking the User Interface Paradigm of Integrated Development Environments”. In: *Proceedings of the 32nd International Conference on Software Engineering*. ACM. 2010, pages 455–464.
- [3] R. P. L. Buse and W. R. Weimer. “Automatically documenting program changes”. In: *Proceedings of the 25th International Conference on Automated Software Engineering (ASE)*. IEEE/ACM. 2010, pages 33–42.
- [4] P. M. Duvall, S. Matyas, and A. Glover. *Continuous integration: Improving software quality and reducing risk*. Pearson Education, 2007.
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: Elements of reusable object-oriented software*. Pearson Education, 1994.
- [6] A. Z. Henley and S. D. Fleming. “The Patchworks Code Editor: Toward Faster Navigation with Less Code Arranging and Fewer Navigation Mistakes”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pages 2511–2520.
- [7] K. Herzig and A. Zeller. “The impact of tangled code changes”. In: *Proceedings of the 10th International Workshop on Mining Software Repositories (MSR)*. IEEE. 2013, pages 121–130.
- [8] D. Ingalls, S. Wallace, Y. Chow, F. Ludolph, and K. Doyle. “Fabrik: A Visual Programming Environment”. In: *ACM SIGPLAN Notices* 23.11 (1988), pages 176–190.
- [9] D. Kawrykow and M. P. Robillard. “Non-essential changes in version histories”. In: *Proceeding of the 33rd International Conference on Software Engineering (ICSE)* (2011), page 351.
- [10] J. Lincke, R. Krahn, D. Ingalls, and R. Hirschfeld. “Lively Fabrik A Web-based End-user Programming Environment”. In: *Proceedings of the 7th International Conference on Creating, Connecting and Collaborating through Computing (C5)*. IEEE. 2009, pages 11–19.
- [11] J. H. Maloney and R. B. Smith. “Directness and Liveness in the Morpnic User Interface Construction Environment”. In: *Proceedings of the 8th Symposium on User Interface and Software Technology*. ACM. 1995, pages 21–28.

- [12] Y. Ohshima, A. Lunzer, B. Freudenberg, and T. Kaehler. “KScript and KSWorld: a time-aware and mostly declarative language and interactive GUI framework”. In: *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*. ACM. 2013, pages 117–134.
- [13] R. Robbes and M. Lanza. “A change-based approach to software evolution”. In: *Electronic Notes in Theoretical Computer Science* 166 (2007), pages 93–109.
- [14] D. Spinellis. “UNIX Tools as Visual Programming Components in a GUI-builder Environment”. In: *Wiley Software: Practice and Experience* 32.1 (2002), pages 57–71.
- [15] M. Taeumel, T. Felgentreff, and R. Hirschfeld. “Applying data-driven tool development to context-oriented languages”. In: *Proceedings of the 6th International Workshop on Context-oriented Programming*. ACM. 2014.
- [16] M. Taeumel, M. Perscheid, B. Steinert, J. Lincke, and R. Hirschfeld. “Interleaving of Modification and Use in Data-driven Tool Development”. In: *Proceedings of the Symposium for New Ideas, New Paradigms, and Reflections on Everything to do with Programming and Software (Onward!) 2014*. ACM. To appear.
- [17] M. Taeumel, B. Steinert, and R. Hirschfeld. “The VIVIDE programming environment: Connecting run-time information with programmers’ system knowledge”. In: *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!) ACM, 2012*, pages 117–126.
- [18] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim. “How do software engineers understand code changes?: An exploratory study in industry”. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM. 2012, page 51.
- [19] M. Vaziri, O. Tardieu, R. Rabbah, P. Suter, and M. Hirzel. “Stream Processing with a Spreadsheet”. In: *Proceedings of the European Conference on Object-oriented Programming (ECOOP)*. Springer, 2014, pages 360–384.

Omniscient Debugging in Database Applications

Arian Treffer

Enterprise Platform and Integration Concepts
Hasso-Plattner-Institut
arian.treffer@hpi.de

Omniscient debuggers can greatly improve developer productivity. Not only do they allow for more efficient navigation in the execution of a program, they can be used as a foundation for dynamic analyses that further help the developer to identify relevant parts of code. Much work has been done on debugging and analyzing object-oriented code.

We present an approach of bringing omniscient debugging and advanced analysis algorithms to stored procedures. Our prototype allows omniscient debugging of SQLScript that handles large amounts of data, while creating only a small overhead through slicing. Furthermore, we show how the trace can be used as a foundation to run dynamic analysis algorithms whilst reducing the amount of data that has to be processed.

1 Introduction

The debugger is one of the most important of a software developer. It allows to observe and inspect a program's execution and is useful for many purposes, such as bug detection and code comprehension. Studies found that developers spend up to 50 % of their time debugging.

The usage of a debugger usually follows the same pattern:

1. The developer forms a hypothesis about the workings of a specific part of the program.
2. She sets a breakpoint inside or before the code of interest. If the control flow through the program is not certain, multiple breakpoints can be used.
3. Once the debugger halts the execution, the program's state can be examined.
4. The execution is continued in small or larger steps, e.g., using step instructions or more breakpoints.
5. If unexpected values or behavior are observed, the hypothesis is adapted.

This process is repeated until the developer's hypothesis is sufficiently confirmed.

Alas, with commonly used debuggers, this approach has several problems. To find good locations for setting breakpoints, extensive knowledge is often necessary. If the hypothesis is changed, other parts of the program may become of interest. If these parts have already been executed, the debug session has to be restarted. This is particularly common in bug hunting, where the infection chain has to be followed from the failure to the code defect, backwards in time. Furthermore, navigation errors such as stepping over a method call instead of into it often make restarting the debug session necessary.

The remainder of the report is structured as follows: The next section gives a brief introduction to omniscient debuggers and presents our previous work on debugging and slicing Java applications. Section 3 shows our ongoing work of adapting these concepts to SQLScript and describes new challenges that emerge when debugging code that handles large amounts of data. A history of omniscient debugging and other related work is presented in section 4, before we conclude in section 5.

2 Omniscient Debugging

A Backwards Debugger is a debugger that allows to step not only forwards, but also backwards in the execution. As an extension, an Omniscient Debugger is a debugger that knows every state of the program, in the past and future of the current point in time.

Working backwards debuggers have been implemented for several programming languages [7, 11–13]. Many of them internally work like omniscient debuggers, but do not reveal this to the user.

2.1 Modeling the execution trace

Debuggers are a special kind of runtime analysis tools. Basically, there are two ways to implement a runtime analysis.

A live analysis evaluates the program as it is executed; as soon as, or even before, the program terminated, the result of the analysis is available. Common debuggers typically fall into this category.

A post-mortem analysis first records aspects of the programs execution and then analyses the recorded data. Sometimes, this approach has the advantage that multiple analyses may be run iteratively, without having to re-execute the program. This disadvantage of this approach is that, depending on the granularity of the recorded data, it requires much more memory.

Backwards debuggers can be implemented with both the live and the post-mortem approach. In the scenario described above, where the debug session begins at the occurrence of a failure, it does not make much of a difference. In other use cases, however, the look-ahead that is possible with the post-mortem approach can make the difference between a backwards and an omniscient debugger.

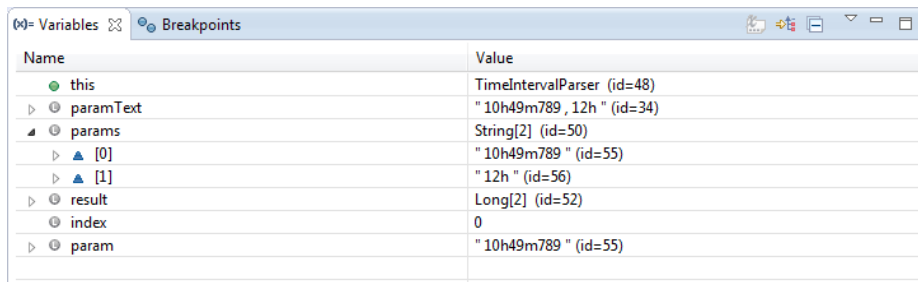
Many strategies have been proposed to reduce the amount of data that has to be captured to allow a replay of the execution. However, since we aim for an omniscient approach, we will record almost everything, including method calls and returns, exceptions, and variable and field accesses.

2.2 Advanced navigation

As described above, a recurring task is to find the source of a value. It seems obvious how a backwards debugger can improve the time required to find the source of an error.

Restarting the debug session becomes virtually unnecessary. Once an error is identified, the developer can step backwards to its source. If a method call is stepped over by accident (in either direction), the operation can be easily reverted.

Nevertheless, this may still require stepping (backwards) through large parts of the application. An omniscient debugger, on the other hand, immediately knows where the value was set.



Name	Value
this	TimeIntervalParser (id=48)
paramText	" 10h49m789 , 12h " (id=34)
params	String[2] (id=50)
[0]	" 10h49m789 " (id=55)
[1]	" 12h " (id=56)
result	Long[2] (id=52)
index	0
param	" 10h49m789 " (id=55)

Figure 1: Variables View in Eclipse

Figure 1 shows the variable view of Eclipse's debugging perspective, which is typically used to spot erroneous values. With the omniscient debugger extension, the developer can directly jump back in time to the assignment of a value simply by double clicking it. This changes the debugging process as follows:

The developer finds the value and double-clicks to jump to its source. She finds that the value is built from three other values, using a formula that seems to be correct. However, she is not sure which of the input values is erroneous. Thus, she

bookmarks the current point-in-time and begins to investigate the first value, again by double-clicking it.

Once she stepped around through the value's creation, she is certain that this value is valid and uses the bookmark to return *back to the future*. Then she begins to investigate the second value. When she realizes that it is invalid, this process is repeated until the fault is reached.

As the example shows, another important task is to determine whether a value is valid by examining how it is produced. Here, the omniscient debugger can assist in multiple ways.

Firstly, instead of showing just the current stack trace, the omniscient debugger can provide a tree of previous and subsequent invocations (cf. Figure 2). Especially after jumping backwards, the developer may have to regain orientation, where this additional context can be helpful.

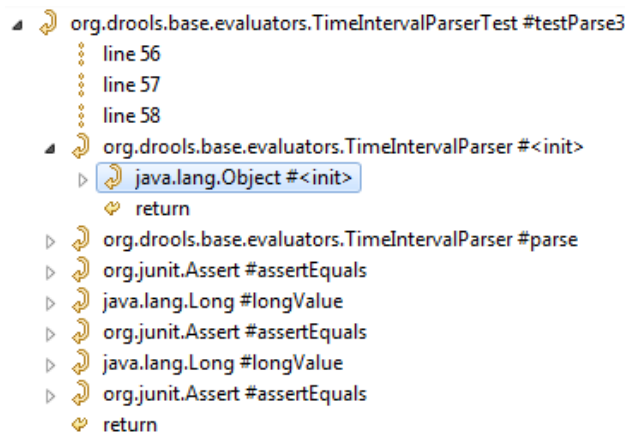


Figure 2: Call Tree

Secondly, the debugger can show the history of a variable, or even an entire object (cf. Figure 3). Mostly, this is helpful when a value is created in a loop or if an object is changed in multiple, different parts of the application over a longer stretch of time.

Finally, the debugger knows whether a given value is used again or at all. By greying out variables and fields that are not accessed again (at least not before their values are changed), the program state that has to be examined by the developer is effectively reduced.

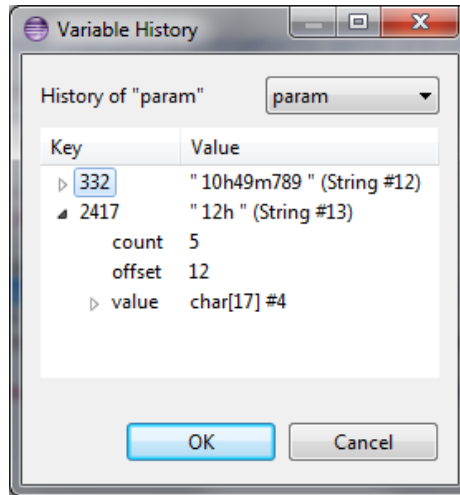


Figure 3: Variable History

2.3 Slicing

According to Weiser [19], a (static) slice S is a subset of the statements of a program P on a slicing criterion C , so that for any input I , S and P produce state trajectories equivalent with respect to C . A typical example for a slicing criterion would be a variable in a given line. Then, all statements that can never impact the value of that variable can be removed from the slice. Dynamic slices are defined similarly, but have to produce the equivalent state trajectories only for specific inputs [10].

It has been shown that slicing represents how programmers naturally think about problems in programming [19] and has many applications, including, but not limited to, debugging [1] and program comprehension [4].

Typically, static and dynamic slicing algorithms focus on finding statements belonging to a slice. However, in many cases statements are executed multiple times in a single program run and not all executions are relevant for the slice. Therefore, our algorithm focuses on state-changing events, i.e., actual executions of statements, instead of the statements themselves.

On the highest level, our algorithm to compute a dynamic slice works as follows: The output of the algorithm will be a sorted set of events. The target event, i.e., the event for which the slice was requested, is added to both the result set and a queue of unprocessed events. Then, until the queue is empty, an event is polled and its dependency events are determined as follows:

Firstly, the event is mapped to a statement in the code. Secondly, the static dependency graph for that method is obtained. Thirdly, the statement is looked-up in the graph and candidate dependency statements are mapped back to events.

Each dependency event that is not yet part of the result set is added to both the result and the queue. Finally, the result set is returned.

Compared to existing Java dynamic slicers, our approach has several advantages. Unlike JSlice, we separated the tracing and the analysis phase. This allows for faster results when computing multiple slices on the same execution. Both JSlice and JavaSlicer allow to visualize the slice by highlighting relevant lines in the source code. Our approach uses the slice in the context of a debug session, which means it not only allows to step through the slice, but also allows the developer to inspect variables and objects at any point in time. Furthermore, the developer can choose to exclude different types of dependencies from a slice to set a focus, for instance, on calculation or reachability questions.

3 Debugging Stored Procedures

Many large and complex applications use a database to persist large amounts of data. However, with the advance of in-memory databases and the decline of RAM cost, the database is no longer seen as a simple data provider [15]. For maximum performance, more and more business logic is moved away from the so-called application layer, which is typically coded in some high-level object-oriented language, into the database, where it has to be rewritten in SQL queries and stored procedures (mostly SQLScript). The increasing complexity of database routines brings an increased need for tool support for debugging.

Regular debuggers for stored procedures already exist. They allow to set breakpoints and to inspect variables and tables, as one would expect. However, often they can not be used as efficiently as debuggers for other languages. It is common for a stored procedure to run several seconds or even minutes, which increases the cost for restarting a debug session. Furthermore, the large amounts of data that can be processed in a single call can make it impossible for the developer to gain a complete understanding of the program state.

Both problems can be solved by testing with minimal example data. However, if the nature of a bug is not yet known, creating such an example may be impossible. In this section, we show how an omniscient debugger for stored procedures can be realized and discuss specific problems such a debugger has to face. A prototypical implementation is currently being developed.

3.1 Tracing and Omniscient Debugging

An omniscient debugger also suffers from the large amounts of data. Our Java debugger traces every field and variable access. Tracing every tuple of a table would create

a dramatic overhead. Using an even-bigger database, just to manage a single debug session, is not feasible. Instead, we take advantage of the same that makes stored procedures so powerful in the first place: the declarative nature of SQL queries.

Unlike in object-oriented programs, where almost every behavior can be changed by virtual method calls, it is not possible to change the behavior of a where-clause. Furthermore, SQL queries are well defined so that it is not necessary to analyze the internal workings to allow an analysis of the overall behavior.

Instead, we only need to trace variable assignments to be able to reproduce the program execution. Queries don't have to be traced at all, although for some purposes it will be helpful to record some meta information, such as the execution time or the number of results. However, we need to be able to reproduce the query results, otherwise the debugger would be quite useless.

3.2 Reproducing Queries

By tracing all variables, the debugger has enough information available to re-execute any query. However, the query will only yield the same results as long as the underlying data has not changed.

In general, one can expect that debugging will take place on a development machine where no other data manipulation occurs. However, in cases where this assumption doesn't hold, the debugger might end up showing wrong or misleading data the developer, which can make the tool outright harmful. Furthermore, the debugged stored procedure itself may change the data, which will cause a query to return different results at different points in time.

We have identified three strategies of dealing with data changes, each with its own advantages and disadvantages.

Temporary Tables Auxiliary tables can be used to track all data changes. Values of deleted tuples and modified attributes have to be recorded, as well as timestamps of the manipulation events. To re-execute a query, it has to be rewritten to include this additional data.

An advantage of this approach is that it allows to efficiently analyze the modifications that were applied by the stored procedure. Disadvantages are that copying data to auxiliary tables greatly increases the tracing overhead, and that rewritten queries are much more complex and thus may take longer to execute.

Transactions If the whole debug session runs in a single database transaction, it is automatically protected from concurrent modifications. Nested transactions can be used to rollback changes done by the debugged code.

This approach requires only little effort by the debugger, as it builds upon existing features of the database. However, to re-execute a single query, the whole stored procedure has to be re-executed up until that point, which can create a significant overhead. Furthermore, depending on how transactions are implemented, the developer might accidentally lock the database for everyone else.

Insert-Only In systems that are relevant to accounting, such as ERP, finance, and CRM systems, data is never deleted. Such behavior often even is a legal requirement. If we require for all tables that data can never be changed or deleted and annotate all tuples with timestamps of when they have been created and invalidated, we can reconstruct the state of the database of any point in time.

This approach combines the advantages of both previous approaches. Especially in in-memory databases, the overhead can be less than expected due to compression, and it may even improve the performance as inserts can be faster than updates or deletes. Finally, adding timestamp filters to select queries does not cause a significant slowdown.

3.3 Slicing

The slicing algorithm that we presented in subsection 2.3 can be easily transferred to stored procedures. Statements depend on each other through the variables they access, and from the variable trace the execution path that is necessary for dynamic slicing can be reconstructed easily.

However, the power of dynamic slicing comes from knowing where fields of objects are read and written. In stored procedures, we would expect that the slicing algorithm can tell us which tuples of a table are actually relevant for the slice. However, without tracing access to tables, this information is not immediately available.

Exact approach One way to find relevant tuples is to reproduce the filter expressions that were used to access a table. By or-chaining all expressions, the result is the union of all individual query results. Additional attributes can be introduced that indicate which of the filter expression matched for a tuple. While this approach might not scale well when nested queries are involved, we are now, at least in principle, able to get a view on the database that reflects our slice.

However, the slicing algorithm does not yet consider this information when computing a slice. It might be the case that a particular filter expression, or a particular variable in a filter expression, does not affect the result set. In this case, the expression or variable should be excluded from the slice.

In principle, it is possible to use this information when constructing a slice. However, actually executing these queries can increase the time required for building a slice until it is no longer desirable.

Approximate approach Another way to build a data slice is to approximate a filter. Bloom filters use hashing to store subsets of large data in comparatively small bitmaps, at the cost of creating some false positives [14]. Inserting bloom filters at selected parts in the stored procedure can reduce the need to analyze larger parts of code and simplify filters that are used to compute slices.

A usable dynamic slicing algorithm could first create a slice without inspecting the tables. In cases where more precision is needed, the developer could then choose to apply one of the approaches outlined above to refine the results.

4 Related work

An omniscient debugger is a debugger that immediately knows about every event in the execution of a program [11]. While reversible execution for debugging purposes has been researched earlier [5], the first omniscient debugger was presented by Lewis [11]. The debugger supported several ways to jump through points-of-interest in the execution, but had no slicing capabilities. Subsequent work in the area focused mostly on memory aspects, for instance by developing a specialized event database [16] or allowing garbage-collection of unreachable past events [13].

The concept of slicing has first been introduced by Weiser, along with a first static slicing algorithm [19]. Korel and Laski, and Agrawal and Horgan later extended the idea to include runtime information to produce more precise slices [2, 10]. Furthermore, Agrawal et al. presented a debugger for C programs with dynamic slicing capabilities [1]. Since then, different slicing algorithms have been proposed and analyzed [3, 8, 20].

For Java, dynamic slicing has been implemented for byte-code traces [17, 18]. JSlice [18] and JavaSlicer [6] are available tools. Ko and Myers used a combination of techniques similar to static and dynamic slicing to automatically answer causality questions [9].

5 Conclusion and Future Work

We have shown how omniscient debuggers can increase developer productivity by allowing backwards navigation and providing fast advanced dynamic analyses.

While these techniques can be applied to all imperative programming languages, a prototype for SQLScript has revealed particular problems that occur when large amounts of data are handled in the analyzed execution.

Future work will consist of developing omniscience-based algorithms for analyzing stored procedure, and evaluating them with regards to required tracing overhead and performance.

References

- [1] H. Agrawal, R. A. Demillo, and E. H. Spafford. “Debugging with dynamic slicing and backtracking”. en. In: *Software: Practice and Experience* 23.6 (1993), pages 589–616. doi: 10.1002/spe.4380230603.
- [2] H. Agrawal and J. R. Horgan. “Dynamic Program Slicing”. In: *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*. PLDI ’90. New York, NY, USA: ACM, 1990, pages 246–256. doi: 10.1145/93542.93576.
- [3] D. Binkley, S. Danicic, T. Gyimóthy, M. Harman, Á. Kiss, and B. Korel. “Theoretical foundations of dynamic program slicing”. In: *Theoretical Computer Science* 360.1–3 (2006), pages 23–41. doi: <http://dx.doi.org/10.1016/j.tcs.2006.01.012>.
- [4] A. De Lucia. “Program slicing: methods and applications”. In: *First IEEE International Workshop on Source Code Analysis and Manipulation, 2001. Proceedings*. 2001, pages 142–149. doi: 10.1109/SCAM.2001.972675.
- [5] S. I. Feldman and C. B. Brown. “IGOR: a system for program debugging via reversible execution”. In: *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging*. PADD ’88. New York, NY, USA: ACM, 1988, pages 112–123. doi: 10.1145/68210.69226.
- [6] C. Hammacher. *Design and Implementation of an Efficient Dynamic Slicer for Java*. Published: Bachelor’s Thesis. Saarland University, Nov. 2008.
- [7] C. Hofer, M. Denker, and S. Ducasse. “Design and implementation of a backward-in-time debugger”. In: *NODE 2006* (2006), pages 17–32.
- [8] T. Hoffner. *Evaluation and comparison of program slicing tools*. Citeseer, 1995.
- [9] A. J. Ko and B. A. Myers. “Debugging reinvented: asking and answering why and why not questions about program behavior”. In: *Proceedings of the 30th international conference on Software engineering*. ICSE ’08. New York, NY, USA: ACM, 2008, pages 301–310. doi: 10.1145/1368088.1368130.

- [10] B. Korel and J. Laski. “Dynamic slicing of computer programs”. In: *Journal of Systems and Software* 13.3 (Nov. 1990), pages 187–195. doi: 10.1016/0164-1212(90)90094-3.
- [11] B. Lewis. “Debugging backwards in time”. In: *Computing Research Repository cs.SE/0310016* (2003).
- [12] H. Lieberman. “Reversible Object-Oriented Interpreters”. English. In: *ECOOP’87 European Conference on Object-Oriented Programming*. Volume 276. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1987, pages 11–19.
- [13] A. Lienhard, T. Gîrba, and O. Nierstrasz. “Practical Object-Oriented Back-in-Time Debugging”. In: *ECOOP 2008 – Object-Oriented Programming*. Edited by J. Vitek. Lecture Notes in Computer Science 5142. Springer Berlin Heidelberg, Jan. 2008, pages 592–615.
- [14] A. Pagh, R. Pagh, and S. S. Rao. “An Optimal Bloom Filter Replacement”. In: *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’05. Vancouver, British Columbia: Society for Industrial and Applied Mathematics, 2005, pages 823–829.
- [15] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.
- [16] G. Pothier, É. Tanter, and J. Piquet. “Scalable omniscient debugging”. In: *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. OOPSLA ’07. New York, NY, USA: ACM, 2007, pages 535–552. doi: 10.1145/1297027.1297067.
- [17] A. Szegedi and T. Gyimothy. “Dynamic slicing of Java bytecode programs”. In: *Fifth IEEE International Workshop on Source Code Analysis and Manipulation*, 2005. Sept. 2005, pages 35–44. doi: 10.1109/SCAM.2005.8.
- [18] T. Wang and A. Roychoudhury. “Dynamic Slicing on Java Bytecode Traces”. In: *ACM Trans. Program. Lang. Syst.* 30.2 (Mar. 2008), 10:1–10:49. doi: 10.1145/1330017.1330021.
- [19] M. Weiser. “Programmers use slices when debugging”. In: *Commun. ACM* 25.7 (July 1982), pages 446–452. doi: 10.1145/358557.358577.
- [20] X. Zhang, R. Gupta, and Y. Zhang. “Precise dynamic slicing algorithms”. In: *25th International Conference on Software Engineering, 2003. Proceedings*. May 2003, pages 319–329. doi: 10.1109/ICSE.2003.1201211.

Learning Deep Semantic Feature for Cross-modal Representation

Cheng Wang

Internet Technologies and Systems
Hasso-Plattner-Institut
Cheng.Wang@hpi.uni-potsdam.de

This report summarizes my research activities in the HPI Research School on Service Oriented Systems Engineer of the past six months on multimodal learning. In this report, deep semantic features are learned for cross modal mapping between visual and textual data. In processing text modality, in order to extract text semantic features, multi-level Online Latent Dirichlet Allocation is proposed and implemented for fitting 3.3 million Wikipedia articles to 336 topics with different topic granularity. For image modality, visual features are learned with deep Convolutional Neural Networks (CNNs) that is pre-trained with 1.2 million images. Those features are used to train a 3-layer neural network for cross representation and thus to bridge the semantic gap across text and image modality.

1 Introduction

The rapid increasing of multimedia data on web brings new challenges for information retrieval. Web applications produce massive multi-modal data every day, such as image, text, audio and video. It means the information we received from various information channels. This trend makes conventional uni-modal based retrieval systems that only considering single modality by using key words or caption more difficult to retrieve user interested information. Besides text-based engines, image [5] and video retrieval systems [4] also been proposed for retrieval multimedia information. But those researches still cannot apply to multi-modal case. Modeling multimodal data is needed as previous research [11] have proved that one modality can be a semantic complementary for another modality, which shown to outperform state-of-the-art information retrieval systems on a uni-modal retrieval task.

Recently, several cross-modal approaches have been proposed to enhance information retrieval performance, in which fusing data modality was widely studied, particularly, image-text modality fusion. Conventional approach is to represent the image component within document to visual words with SIFT [9] descriptor. In processing text modality, Latent Dirichlet Allocation (LDA) [2] is considered, as a probabilistic models, it has proved that this kind of model can effectively exploring the hidden topics in given corpus. One popular approach is to build joint model

for fusing text and image modality and discover the underlying shared “concept” between those modalities. This approach is useful because different data modality actually carry different interesting information. By combining those information can definitely achieve better retrieval results. How to build a joint model is still a difficult problem to be address due to following challenges:

(1) Many previous cross modal retrieval systems tackle retrieval tasks by assigning predefined categories to inquiry text (image), and selecting top-K matched image (text) as retrieval results. One problem of this approach is that existed image labels often ambiguous, which means one image can be classified to different categories.

(2) With respect to feature representation. LDA is a common method that used to infer documents’ major topics for representing text modality. On the other hand, traditional visual representation of use the bag-of-visual-words (BOVW), in which image key points are often extracted by SIFT or SURF [1]. Features of each image utilized to visual topic clustering in image representation. The correlation between word topics and visual topics generally involves Canonical Correlation Analysis (CCA). Thus the features obtained by taking LDA and BOVW vectors are important for cross mapping performance. The problem of those kinds of research is how to appropriately represent image and text in order to improve cross representation performance.

As in [13], “Different modalities typically carry different kinds of information”. Thus it requires both image and text features should be well learned so that those feature can more appropriately represent input data modality. To address those problems, we therefore propose a novel cross modal retrieval architecture which allows cross representation with deep networks. Specifically, image will be represented by text feature and vise-versa in this research. Deep networks [10] has been applied to cross modality feature learning and demonstrated it effectiveness in unsupervised feature learning for single modalities. In this work, similarly, we apply it to learn features from image and text separately. In learning image feature, deep Convolutional Neural Networks (CNN)[8] has achieved considerably outperform previous result in visual representation. In text modality feature, we propose multi-level Online LDA for constructing the bag of topics with different topic granularity, by doing so, text can be semantically represented as topical feature. For mapping between image and text we propose 3-level neural network for training projection layers between image and text modality.

Work that is most relate to our research are [11, 14, 15], which are used as baselines for our research. For compare with those works, in this paper, we will evaluate our model on open benchmark Wikipedia corpus ¹ that used in the baselines for cross model retrieval. Our retrieval tackles two retrieval tasks: (1) retrieving best matched image for a given query text, (2) retrieving best related text for give query image.

¹<http://www.svcl.ucsd.edu/project/crossmodal/>, accessed December 16, 2014.

Different to previous approach in modeling multimodal data, our research concentrate on cross modal representation with deep networks. Consequently, improving cross-modal retrieving performance.

2 Related work

2.1 Modeling Multimodal Data

Modeling and analysis of multimodal data have gained much attention in last several years, much effort have put into discover new joint model for multimodal data. N. Rasiwasia et. al [3] proposed a novel approach to match text and image modality via canonical correlation analysis (CCA). SFIT feature of image and text feature that generated with LDA are considered. Through projecting image feature and text feature into two intermediate spaces and matching different modalities and applied it to cross-modal multimedia retrieval. Similarly, Jin Yu [15] designed a cross-modal retrieval system that considering image-text statistical correlations. The images are represented with SIFT feature and quantized to 100-dimensional vector, correspondingly, 100 topics that generated from LDA use to represent text modality. Motivated by those works, K. Y. Wang [14] proposed an approach which combines common subspace learning and coupled feature selection for cross-modal matching problem. l_{21} -norm was used in this case for selecting features from coupled modalities and coupled linear regression was used to project data to a common space. As deep learning technique received a lot of attention recently, it has been applied to various researches including multimodal data analysis. J. Ngiam [10] applied multimodal deep learning approach in audio-visual speech classification. Greedily training restricted Boltzmann machine (RBM) and deep auto-encoder are used to discover correlation connections across different modalities. In [13] N. Srivastava used a Deep Boltzmann Machine to extract a unified representation from different data modalities, it found out this representation is useful in addressing classification and information retrieval problem.

2.2 Deep Convolutional Neural Networks

Deep Convolutional Neural Networks (CNNs) has already been proved very powerful in image feature extraction and image classification [8, 16]. Conceptually, at a convolution layer, a feature map is formed by convolving the previous layers' feature maps with kernels and activation function. If we denote j -th feature map at given layer l as $x_j^{(l)}$, for given weights $w^{(j)}$ and bias $b^{(j)}$, the feature map $x_j^{(l)}$ combine convolutions with multiple input maps that represented as M_j , and then denoted

as follows:

$$x_j^{(l)} = f\left(\sum_{i \in M_j} x_i^{(l-1)} w_{ij}^{(l)} + b^{(l)}\right),$$

where $f(\cdot)$ is activation function.

At sub-sampling layer, input maps is down sampled to smaller version for reducing the computational complexity, generally, an output map denoted as:

$$x_j^{(l)} = f(\beta_j^{(l)} S(x_i^{(l-1)})) + b^{(l)}),$$

where $f(\cdot)$ and $S(\cdot)$ are activation function and sub-sampling function respectively, and each output map given its own multiplicative bias $\beta_j^{(l)}$ and bias $b^{(j)}$. Inspired by [7] which implemented a fast convolutional architecture for effectively representing sensory inputs. Part of our work is implemented base on Caffe framework.

3 Learning Architecture

To address the problem of cross modal representation we will learn highly representative features for image and text modality separately with different pre-trained visual and textual models. Document contains image-text pair will be represented as feature pairs. For a training dataset S , which contains documents $D = \{D_1, D_2 \dots D_n\}$ and each document is image-text pair that we represented as $D_i = \{I_i, T_i\}$. Modeling multimodal data always need informative method to represent different modalities. To achieve this goal, we designed learning architecture to represent text and image modality separately. First, text and image representation model are pre-trained with deep convolution neuron network and multi-granularity Latent Dirichlet Allocation. As shown in Figure 1, the architecture tackles input document to two different modalities and extracts features. In order to build the connection between visual and textual feature, we designed two project layers, $L^{I \rightarrow T}$ and $L^{T \rightarrow I}$. The function of projection layers is to project visual feature to textual feature and vice versa. The learned features from pre-trained visual and textual model used as training data for learn $L^{I \rightarrow T}$ and $L^{T \rightarrow I}$. In cross modal representation stage, we focus on two problems: (1) represent image with textual feature, (2) vice versa. By doing this, cross modal representation is achieved and further applied to cross modal retrieval field.

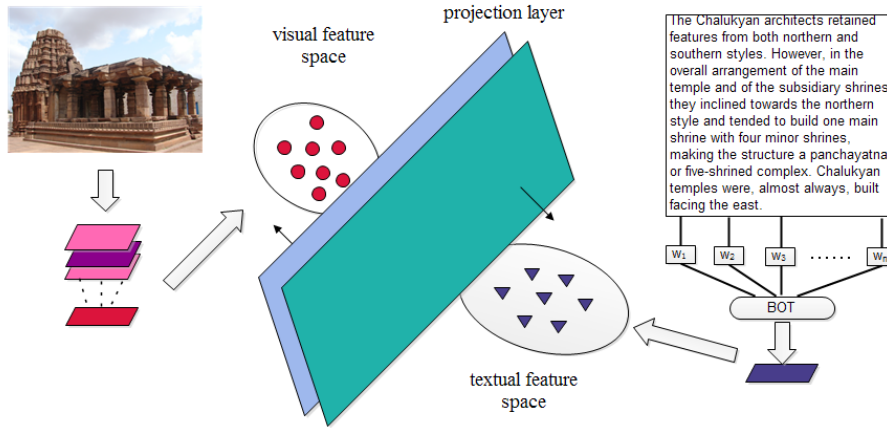


Figure 1: Figure Learning Architecture for text and image modality

4 Modality Representation

4.1 Text Representation

In processing text modality, the problem we need to address is to learn feature representation. To this end, we extend the concept of “bag of words (BOW)” to “bag of topics (BOT)”, which is derived from Latent Dirichlet Allocation (LDA). LDA is a generative probabilistic model for discover latent topics from given corpus. The generative process can be decomposed into doc-topic and topic-word generative process. For a given corpus D , the topic proportion θ follows a Dirichlet distribution with prior probability α . For given θ , the specific topic z_n is draw from a multinomial distribution. Similarly, in topic-word distribution, a word w_n follows multinomial distribution with φ_k that is drawn from a Dirichlet distribution with prior probability β . Thus joint probability distribution can be described as

$$p(\mathbf{w}, \mathbf{z}, | \theta, \beta) = p(\theta | \alpha) \prod_{n=1}^N p(z_n | \theta) p(w_n | z_n, \beta),$$

where w_n means the n -th word in document, z_n means the topic of n -th word. N is the number of words in corpus.

Inspired by recent work in [12] in which a multi-level LDA approach is used for text representation. It regards learned topics from different level as topical feature so that the most discrimination features are extracted. In this work, we redesigned that topic learning architecture to fit our experiment. In order to represent text as topical features, we need to learn a topical feature map from corpus, to address this problem we design our text representation scheme as following In this scheme,

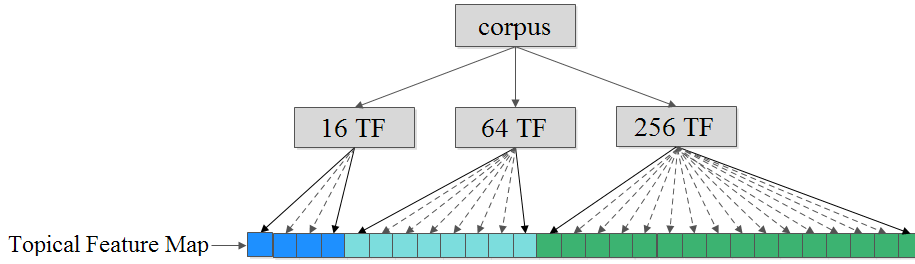


Figure 2: Text representation architecture

3 levels designed to generate 16, 64 and 256 topics from bottom to top respectively, each topic regard as feature which consists of about 50 words. Topical feature map is “bag of topics” which combines of those topics. That is, $16 + 64 + 256 = 336$ topical features. For each input text, pre-trained topical feature map which is pre-trained model used to extract text features. Assume input text with words, we calculate word distribution over topics and generate topical features. It is possible one word belongs to different topics. This assures that different meanings carried by one word can be considered. That’s our text representation mechanism expects to achieve. Here we summarized the algorithm for text representation using topical feature map as in Algorithm: This is a straightforward algorithm for calculating word distribution

Algorithm 3: Text feature extraction with pre-trained topic model

Input: text T , $M^T = M_1^T, M_2^T \dots M_n^T$, $n = 336$

Output: $TF = f_1, f_2 \dots f_n$

For $M_1^T \in M^T$

$W \Leftarrow T \cap M_t^T$

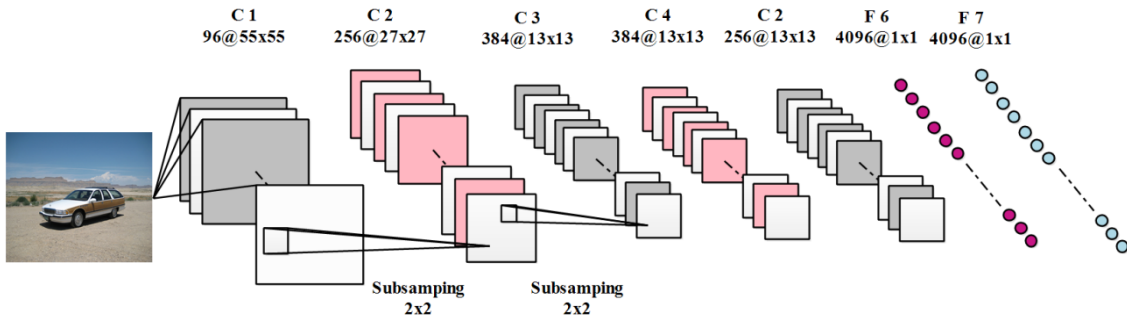
$f = \sum_{w \in W} v^{(w)}$

end for

over pre-trained topical feature model. In implementation of this algorithm we firstly use OnlineLDA [6] to train topic feature model. OnlineLDA fits topic models with online stochastic optimization thus it can easily apply to large scale corpus. In this research, we extend OnlineLDA to multi-level for fitting topics models from different semantic level. Finally we got 336 topic models and we generate the top-50 words for each topic. We here list some 5 example topics with top-8 words.

Table 1: example topics learned from Wikipedia

Topic 0	war, force, prime, royal, navy, commander, killed, squadron
Topic 1	science, back, silver, audience, scientific, knowledge, shooting, christmas
Topic 2	best, directed, festival, short, girl, drama, special, series
Topic 3	river, western, south, state, australia, greek, running, williams
Topic 4	county, district, village, population, town, province, rural, towns
...	...
Topic 336	southern, natural, black, established, map, history, animals, iowa

**Figure 3:** deep CNN architecture

4.2 Image Representation

In order to learn grounded feature from images, we use the state-of-art approach for extracting features. In this paper we concentrate on features that learned from deep conventional neuron network which realized by A. Krizhevsky [8]. But we removed the last layer in feature extraction, in that way, each image can be represented as 4096-dimension vector. For each input image, it was resized to 256×256 pixels before image processing. The first convolutional layer filters it with 64 feature maps whose size is 55×55 pixels. The subsampling operation summarizes the neighborhood pixels of center pixel with size 2×2 . Subsampling results as input of C2, which has 256 feature maps, each feature map is 27×27 pixels. And C3, C4, C5 same feature size 13×13 pixels and have 384, 384 and 256 feature maps respectively. F6 and F7 is full connection layer, both of them have 4096 feature vector. The output of F7 is the features that we aim to extract from image.

Table 2: configurations for model training

Modality	Configuration	Training time
Image model	ubuntu 12.04, 4 cpus, nvidia 780 GPU	8 days
Topic models	ubuntu 12.04, 4 cpus,	9 days for 3 levels

5 Ongoing Experiments

5.1 Datasets

In this work, we used dataset from different modalities for training visual and textual model respectively. For building visual model we use ImageNet [3], which is a large scale manually labeled image database. Our dataset is from ILSVRC2012, a subset of ImageNet and consists of 1.2 million training image and 50,000 validation images. Those images are assigned to 1000 categories according to the center meaning of image. With respect to modeling text data, we use Wikipedia dataset, on one hand, we used 3.3 million Wikipedia [12] articles to train 336 topic models from 3 level. For comparison reason, we use the same dataset as used in [11, 14, 15]. This dataset contains 2886 documents and each document is image-text pair, it is more convenient for us to examine the effectiveness of our approach and compare with others' work.

5.2 Training Procedure

In this subsection, we describe the experiments of feature learning for text and image separately. Firstly, we give our configuration information for training image model and text topic models as follows: It takes about 8 days for obtaining visual models and 9 days for textual model, in which we conducted our topic training experiments for each topic level serially. The accuracy of visual model is verified in image classification tasks. Through fine tuning layer "conv", "conv 4" and "conv 5" and learning rate to adapt our 64-batch size. The final test accuracy is about 53.4% after around 1.4 million iterations.

As we mentioned before, in multi-level topic training, we trained 16 topics, 64 topics and 256 topics respectively. In fitting topics to 3.3 million Wikipedia articles, we configured training settings as recommend in [6]. The training configurations and iteration times are: (1) in level-16, 35516 iterations with batch size 64, (2) level-64, 25521 iterations with batch size 64 and (3) level-256 have 3220 iterations with batch size 1024. To compare with different topic levels, we selected one topic which is mainly about "music" in three different topic levels.

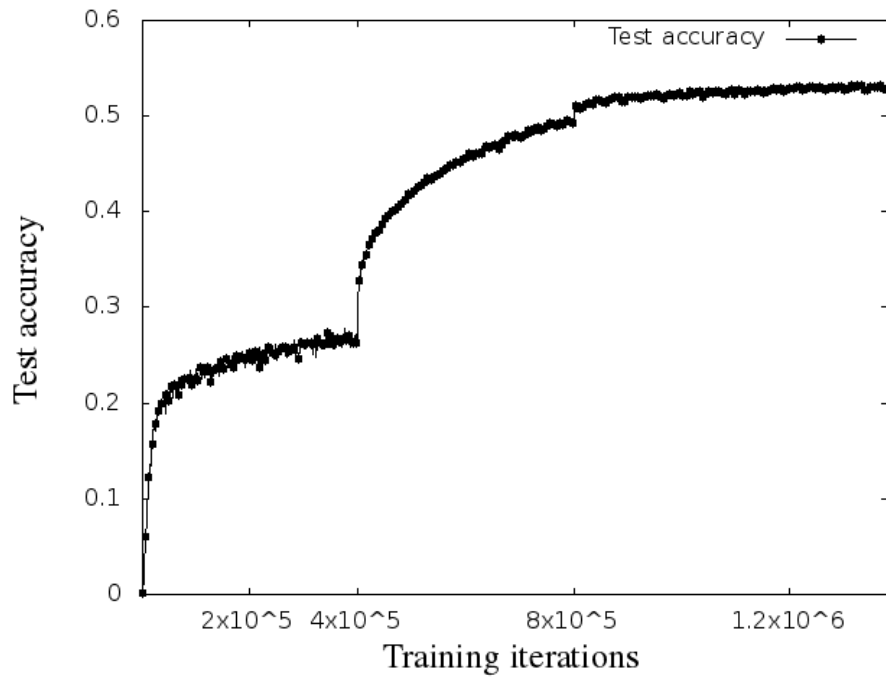


Figure 4: The procedure of visual model training

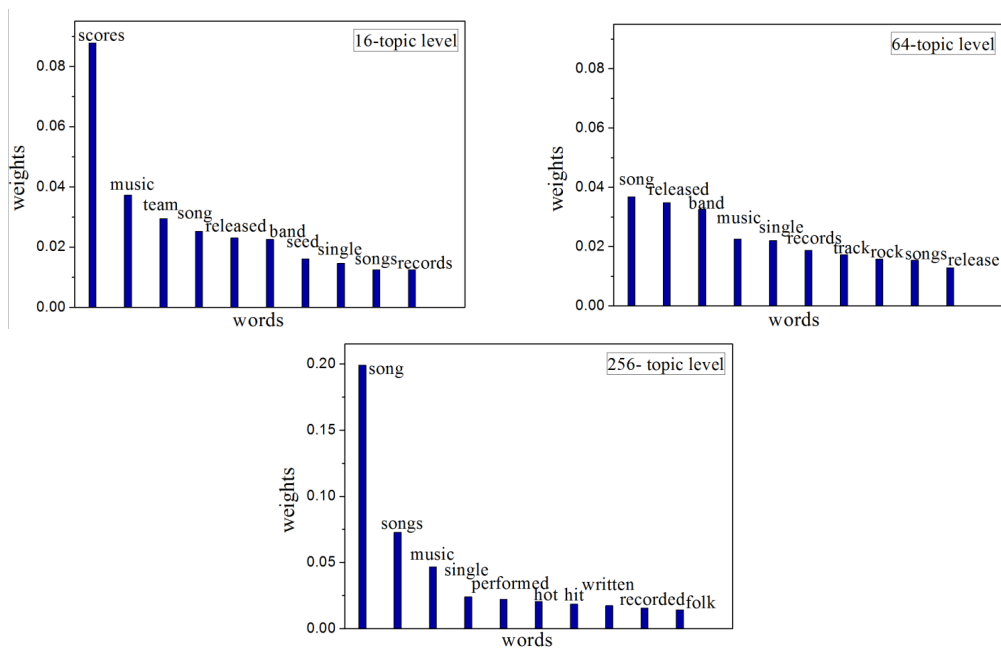


Figure 5: "Music" topic in three different levels (top-10 words are displayed)

6 Conclusion and outlook

In this report, an approach for modeling multimodal data is proposed, visual features are learned by pre-trained visual model and textual features are learned with pre-trained topical feature map, which is trained by using proposed multi-level on-line LDA. Our future work will focus on mapping visual data to textual and mapping textual data to visual with deep neural network. And apply cross representation to multimedia retrieval for improving retrieval performance with extracted semantic features.

References

- [1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3 (2008), pages 346–359.
- [2] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation". In: *the Journal of machine Learning research* 3 (2003), pages 993–1022.
- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pages 248–255.
- [4] J. Fan, A. K. Elmagarmid, X. Zhu, W. G. Aref, and L. Wu. "ClassView: hierarchical video shot classification, indexing, and accessing". In: *Multimedia, IEEE Transactions on* 6.1 (2004), pages 70–86.
- [5] X. He, W.-Y. Ma, and H.-J. Zhang. "Learning an image manifold for retrieval". In: *Proceedings of the 12th annual ACM international conference on Multimedia*. ACM. 2004, pages 17–23.
- [6] M. Hoffman, F. R. Bach, and D. M. Blei. "Online learning for latent dirichlet allocation". In: *advances in neural information processing systems*. 2010, pages 856–864.
- [7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pages 1097–1105.
- [9] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pages 91–110.

- [10] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. “Multimodal deep learning”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pages 689–696.
- [11] N. Rasiwasia, J. Costa Pereira, E. Coviello, G. Doyle, G. R. Lanckriet, R. Levy, and N. Vasconcelos. “A new approach to cross-modal multimedia retrieval”. In: *Proceedings of the international conference on Multimedia*. ACM. 2010, pages 251–260.
- [12] A. H. Razavi and D. Inkpen. “Text Representation Using Multi-level Latent Dirichlet Allocation”. In: *Advances in Artificial Intelligence*. Springer, 2014, pages 215–226.
- [13] N. Srivastava and R. Salakhutdinov. “Multimodal learning with deep boltzmann machines”. In: *Advances in neural information processing systems*. 2012, pages 2222–2230.
- [14] K. Wang, R. He, W. Wang, L. Wang, and T. Tan. “Learning coupled feature spaces for cross-modal matching”. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pages 2088–2095.
- [15] J. Yu, Y. Cong, Z. Qin, and T. Wan. “Cross-modal topic correlations for multimedia retrieval”. In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE. 2012, pages 246–249.
- [16] M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014*. Springer, 2014, pages 818–833.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
94	978-3-86956-319-0	N/A	N/A
93	978-3-86956-318-3	ecoControl : Entwurf und Implementierung einer Software zur Optimierung heterogener Energiesysteme in Mehrfamilienhäusern	Eva-Maria Herbst, Fabian Maschler, Fabio Niephaus, Max Reimann, Julia Steier, Tim Felgentreff, Jens Lincke, Marcel Taeumel, Carsten Witt, Robert Hirschfeld
92	978-3-86956-317-6	Development of AUTOSAR standard documents at Carmeq GmbH	Regina Hebig, Holger Giese, Kimon Batoulis, Philipp Langer, Armin Zamani Farahani, Gary Yao, Mychajlo Wolowyk
91	978-3-86956-303-9	Weak conformance between process models and synchronized object life cycles	Andreas Meyer, Mathias Weske
90	978-3-86956-296-4	Embedded Operating System Projects	Uwe Hentschel, Daniel Richter, Andreas Polze
89	978-3-86956-291-9	openHPI: 哈索•普拉特纳研究院的 MOOC (大规模公开在线课) 计划	Christoph Meinel, Christian Willems
88	978-3-86956-282-7	HPI Future SOC Lab : Proceedings 2013	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Bernhard Schulzki (Hrsg.)
87	978-3-86956-281-0	Cloud Security Mechanisms	Christian Neuhaus, Andreas Polze (Hrsg.)
86	978-3-86956-280-3	Batch Regions	Luise Pufahl, Andreas Meyer, Mathias Weske
85	978-3-86956-276-6	HPI Future SOC Lab: Proceedings 2012	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Bernhard Schulzki (Hrsg.)
84	978-3-86956-274-2	Anbieter von Cloud Speicherdiensten im Überblick	Christoph Meinel, Maxim Schnjakin, Tobias Metzke, Markus Freitag
83	978-3-86956-273-5	Proceedings of the 7th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering	Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch (Hrsg.)

ISBN 978-3-86956-320-6
ISSN 1613-5652