

# Higher-Order Process Engineering

Dissertation  
zur Erlangung des Grades eines  
Doktors der Naturwissenschaften  
der Technischen Universität Dortmund  
an der Fakultät für Informatik

von  
Johannes Neubauer

Dortmund  
2014



---

Tag der mündlichen Prüfung:  
Dekan: Prof. Dr.-Ing. Gernot A. Fink

Gutachter:  
Prof. Dr. Bernhard Steffen  
Prof. Dr. Mike G. Hinchey



# Acknowledgements

First of all I want to thank Prof. Bernhard Steffen and Prof. Tiziana Margaria for their guidance and company in the past years. Thank you for the discussions, the support, the challenges, the motivations, and for preparing the ground that made this thesis possible. Special thanks go to Prof. Mike Hinchey for acting as my referee. Furthermore, I thank my fiancée and my family for their patience and support.

I am very grateful to my office mate Stephan Windmüller for a great time and an outstanding teamwork. I will always remember our special way of pair programming. Furthermore a great thanks goes to the complete staff of the Chair for Programming Systems, TU Dortmund for the warm and encouraging atmosphere as well as for countless helpful discussions.

Last but not least, I want to thank all those who spent their time for proofreading this thesis, namely, Malte Isberner, Stephan Windmüller, Stefan Naujokat, Oliver Bauer, Dr. Anna-Lena Lamprecht, and Dr. Michael Neubauer.



# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Research Problems and Related Work . . . . .	3
1.2. Contributions . . . . .	5
1.3. Conventions . . . . .	6
1.4. Organisation . . . . .	6
<b>2. Preliminaries</b>	<b>7</b>
2.1. Extreme Model-Driven Design . . . . .	7
2.1.1. Adapter Pattern . . . . .	9
2.1.2. Hierarchy & Reuse . . . . .	10
2.2. ChainReaction . . . . .	11
<b>3. Higher-Order XMDD</b>	<b>13</b>
3.1. Dynamic Service Binding . . . . .	14
3.2. Structure of HOPE . . . . .	15
3.3. jABC4 . . . . .	16
3.4. Higher-Order Process Modeling . . . . .	18
3.5. Domain Preparation . . . . .	24
3.5.1. SLG Libraries . . . . .	25
3.5.2. IO SIB Libraries . . . . .	26
3.6. Codegenerator . . . . .	27
3.6.1. Generating the Graph Interface for Game Strategies . . . . .	28
3.6.2. Generating the Graph Implementation for a Game Strategy . . . . .	29
3.6.3. Embedding Generated Game Strategies into ChainReaction . . . . .	30
<b>4. Projects</b>	<b>33</b>
4.1. Risk-Based Testing via Active Continuous Quality Control . . . . .	33
4.2. Learning-based cross-platform conformance testing . . . . .	33
4.3. Dynamic Web Application . . . . .	34
4.4. Model-driven Reengineering of the Business Logic of Java applications . . . . .	34
4.5. Property-Driven Benchmark Generation . . . . .	35
4.6. Java and Scala Codegenerator . . . . .	35
4.7. Capturing and Processing of Biomedical Data . . . . .	36
<b>5. Conclusion and Future Work</b>	<b>37</b>
<b>A. Selected Publications</b>	<b>55</b>
<b>B. Comments on My Participation</b>	<b>57</b>

**C. Other Publications**

**59**



# List of Figures

2.1. SIB adapter pattern. Source: publication I . . . . .	10
2.2. A screenshot of the open source computer board game ChainReaction in retro-look . . . . .	11
2.3. Course of a chain reaction resulting in a win situation for the blue player . . . . .	12
3.1. Dynamic SIB pattern. Source: publication I . . . . .	14
3.2. Coarse-grained structure of the higher-order process engineering (HOPE) approach. . . . .	14
3.3. Screenshot of the jABC4 main window showing a process model of a simple game strategy (GS) for the computer board game ChainReaction (cf. publication VII). . . . .	17
3.4. Higher-order process engineering in action, illustrating a snapshot while conducting a ChainReaction game with two game strategies. . . . .	19
3.5. The service graph for iterating through an arbitrary collection. . . . .	21
3.6. Both tabs of the settings dialog for service browsers. Source: publication VII. . . . .	23
3.7. A view of a service browser offering some services of the JRE. Source: publication VII. . . . .	23
3.8. Interface graph for game strategies in the ChainReaction scenario. Source: publication VII . . . . .	24
3.9. Example IO browser for the configuration graph shown in Fig. 3.10. Source: publication VII . . . . .	24
3.10. Example for an interface graph enhanced with preconfigured activities for the ChainReaction scenario. Source: publication VII. . . . .	26
3.11. Class diagram of the generated graph interface for game strategies and the generated graph implementation for the example game strategy shown in Fig. 3.3 . . . . .	29
3.12. Statistics for the automatic tournament at the end of the project week	31



# 1. Introduction

“With great power comes great responsibility”

— Voltaire

Business process modeling solutions aim at tightly involving the application expert in the software development process to avoid common and costly ‘communication accidents’ during requirements engineering and to decrease time to market. Their popularity, in particular of the recent standard BPMN 2.0, clearly indicates the need for this new involvement. At the same time there are unexpected hurdles when it comes to dealing with integration, (runtime) variability, and interoperability. *Higher-order process engineering* (HOPE) elegantly overcomes these hurdles as even in its simplicity-oriented version it allows for a powerful plug&play fashion, where processes and services can be moved around just like data. This is reminiscent to standardization efforts known from hardware like the *universal serial bus* (USB) [AD01].

Computer-supported processes increasingly influences our daily life: be it for shopping, travel planning, travelling itself, tax declaration, or as part of our business life, we are confronted all the time with different computer/web applications of enormously varying quality and flexibility. Indeed, there are very different solutions for almost the same problem, even by the same provider, depending on the particular profile of a particular user/customer situation. The variance concerns not just the look-and-feel but also the required user process, and, at the technical level<sup>1</sup>, the application programming interfaces (APIs) for the interoperation between systems. As a consequence, users continuously have to fight with this historically grown but unintended technical diversity, and producers with exploding maintenance costs.

Increased interest in business process modeling came with BPMN 2.0 [All09], due to its promise to make application-level business processes directly executable. It looked like an almost universal cure, as it seemed to close the semantic gap, the main source of misunderstandings between business and IT. Looking closer, this dream becomes true only for very specific scenarios. In particular, it still does not support variability, as it lacks means to manage process variation and variant-rich systems.<sup>2</sup> All current BPM standards like BPEL [Pas05], BPMN [RM06] and even the recent BPMN 2.0 are constrained to fixed bindings between business activities in a model and the services or substructures they refer to. Thus each variant has

---

<sup>1</sup>The business level and the technical level are distinguished: the technical level encompasses the IT aspects and the implementation, while the business level focuses on the essence of the application, i.e., it comprises all the levels above the implementation that do not require programming expertise.

<sup>2</sup>The need for variant-rich systems is often addressed via static approaches in general subsumed by the notion of *product-lines* [VdLSR07] in the literature.

to be modeled explicitly via decision points that are modeling pendants to *if*- and *switch*-clauses, in comprehensive but very large models, or maintained individually, as a collection of separate entities. Both alternatives cause severe problems: whereas the comprehensive modeling leads to unmanageably large models, the maintenance and support of many structurally similar individual variant models is a nightmare. This is far away from the ideal of simplicity that made business process modeling so appealing [MS10, MS11].

Enhancing BPM with a *simplicity-oriented* version of *higher-order process passing* breaks the spell. Being higher-order, it supports a very flexible form of (type-based) process integration. It also introduces a new discipline of variability modeling [JLM<sup>+</sup>12] that captures variants comprising functionality that was still unknown at the process' start. Already deployed and even running processes can be seamlessly enhanced with new functionality, without touching their code base.

In fact, based on the higher-order concepts, (new) services and (component) processes can be selected, modified, constructed and then safely passed *as if they were data*. Though unlike data they may be plugged into activities and executed (played) dynamically. This *plug&play* approach allows one to add new services, components, and processes without the need to change the system or interrupt the running processes. Even if this flexibility is not essential and all functionalities are known in advance, it leads to drastic size reductions and better understandability of the models.

Controlling variability means controlling the exploding wealth of combinations (cf., e.g., [LNS13]), a concern orthogonal to computational aspects. Thus in particular at the level of (business) process modeling, variability concerns and computational aspects should be separated in favor of simplicity and understandability [MS10, MS11].

Thus, simplicity-orientation in this regard refers to the user-centeredness – focused on application experts – of the approach hiding the complexity going hand in hand with achieving compatibility between components essential for *plug&play* semantics. In this regard two different types of abstraction layers support a strict enforcement of *separation of concerns* [HL95]: *hierarchical modeling* and *preconfiguration of activities*.

The need for hierarchical modeling is owed to the fact that on the one hand systems are getting more and more complex, resulting in the need to break the specifications down into manageable pieces in a *divide and conquer* fashion. On the other hand, with the increasing globalization of processes and systems, e.g. end-to-end-processes that naturally comprise various management levels, the number of participants involved in the development process is also steadily increasing. This results in a complex network structure for the participants, where the technical expert on one level is at the same time the application expert on the next lower level – in terms of getting more technical – and so forth. Preconfiguration of activities further helps to guard the modeler from the complexity of data-flow handling as well as from doing repetitive work.

Compatibility is achieved via sophisticated (type-aware) interface descriptions of activities leading to compositionality. Data-dependencies are consequently modeled

via a common execution context sharing resources between activities. The models are hierarchical starting on a domain-specific level, leading down to a technical level. The process components on the technical level base on structures of a conventional programming language (i.e. the *target language*). The idea is to give more power to application experts as well as avoiding *technological breaks*, leading to more complete process models which

- are a better basis for communication with technical experts,
- allow for more concise validations on the modeling level, and
- need no round-trip engineering as they are based on semantically well-defined structures of a target language and are therefore directly executable.

The major challenge is to prevent becoming overly complex. The key to this is to consequently support the paradigm of *separation of concerns*.

## 1.1. Research Problems and Related Work

This section sketches the research problems addressed in this thesis and relates them to the state-of-the-art in the literature. In a nutshell, the main question of this thesis is

*How can (business) process modeling be enabled for variability?*

To answer this, the following three general questions regarding variability and how state-of-the-art approaches deal with them have to be answered:

1. *What* can be variable?
2. *How* is variability realized?
3. *When* does the variability take effect?

Current business process modeling approaches mainly focus on being simple, as they aim at involving application experts throughout the complete development life-cycle. Therefore, variability is not in their main concern. Some BPM solutions, however, allow variability via interactive ad-hoc modeling which is either done at design-time [WKM<sup>+</sup>10] or at run-time [De10, KSKS09]. This is in general used to create ad-hoc workflows (design-time), e.g. for ‘in silico’ experiments in biology, or to react to one-time situations, e.g. implementing an exceptional case, via manual intervention (run-time). The different variants are not reflected in these models. Moreover, especially in the BPMN 2.0 [OMG11] context, the service integration process is dominated by scripting. This may be abused to realize run-time variability [Act12, Red12]. Scripting activities can execute methods on arbitrary objects offering runtime variability, but the method call is written as an expression. Input parameters and return value are defined in the expression, i.e. the model is neither aware of the type of the service instance, its input parameters or the return values nor which resources are accessed. Furthermore, these approaches do not allow processes as first-class citizens.

Web services ideally offer a high dynamicity as their strength lies in *service discovery*, focusing on fault-tolerance and availability properties of systems as well as *business to business* (B2B) integration. The compatibility between services should be given by static interface descriptions in the *web service description language* (WSDL) [W3C07], non-functional properties like *quality of service* (QoS), and further semantic annotations [MPW07, We07, LMS09a]. A matching web service is looked up at runtime and executed [Men02, ZBN<sup>+</sup>04, AP05, CDPEV05, JMG05, CAkH05]. The decision, which service should be executed [BCT06, BSBM05, LZS<sup>+</sup>05, PBH07, KOT13], is in general made for each execution. An exception is [AP07], where the authors consider also stateful web services that may be reused at several points of execution. Unfortunately, these approaches usually suffer from the bad realization of web service repositories in the real world.

A research field, where the environment may impact the behavior of a system, so that it exploits at run-time the variability added at design-time, is adaptive or even more self-adaptive software [MSKC04, VH10]. Here the *business logic* is separated from the *adaption logic* in order to decrease complexity as well as increase reuse due to decoupling from a specific environment [cDL03, ZC06, BCJO12, MPT12]. Separation of concerns [KLM<sup>+</sup>97], computational reflection [FF04, Mae87], and component-based design [PPBG09, Gon13, ACN02, C<sup>+</sup>02] are the enabling technologies for self-adaptive systems. Adaptive software in essence reacts locally on the environment via some conditions in the context of their given capabilities and therefore does not reconfigure the system globally.

The *autonomic system specification language* (ASSL) [VH12] is used for *autonomous systems* in the context of space missions [VH13a], in different case studies like developing a control-software for the wide-angle camera carried on board NASA's Voyager II Spacecraft [VH12], or the BepiColombo mission [VH13b]. The underlying concept goes one step further as it captures requirements as goals (like mission objectives) via *autonomy requirements engineering* (ARE). At run-time, new requirements are created based on *self-diagnosis* [VH13c] and realized via *self-adaption*. These approaches mainly focused on embedded systems and the technical realization of self-adaptivity. They do not explicitly address involving the application experts.

In the context of variant-rich systems there are different design-time variability approaches and methodologies [VdLSR07, PBL05, WKM<sup>+</sup>10] using aspect-orientation and model transformation [VG07, JLM<sup>+</sup>12]. These approaches use the commonality within a portfolio of a product family and are limited to creating products from an a priori determined set of variants.

Planning-based approaches [MR02, BM06, MS07, MMK<sup>+</sup>09, JLM<sup>+</sup>12] are goal-oriented, similar to the work in the context of ASSL [VH10]. They adapt the model-structure itself with techniques like process model synthesisis, in general a design-time technique using temporal logics, and planning, employing heuristic search primarily at run-time.

There exists foundational work on higher-order approaches like process calculi of communicating systems such as CHOCS [Tho93] and the  $\pi$ -calculus [Mil99]. More-

over, conceptual studies have been carried out for autonomic computing [BFR07, AK09] reminiscent to chemical reactions, where data and operations are interpreted as molecules and chemical reactions.

*Extreme model-driven design* (XMDD) [MS12] generalizes both BPMN 2 as well as related approaches and supports domain-specific *business activities*. The latter offer reusability and a sufficient abstraction from technological detail in a way that allows for agile process development involving non-programmers. A systematic study [DS12] revealed that only the *Java application building center* (jABC) [SMN<sup>+</sup>06] with its underlying XMDD approach supports business activities that are both *easy to integrate* and *easy to use*. Moreover, XMDD unites service discovery [KMWS07], integration of web services [KMK<sup>+</sup>08, LMS09a], ad-hoc modeling [Lam13], process model synthesis [LNMS10, NLS12], and planning [MMK<sup>+</sup>09], although the service integration [DS12] is its unique selling point. In this thesis XMDD is extended by type-safe data-flow modeling culminating in higher-order semantics allowing to model variants in a concise, comprehensible, and manageable fashion. Systems can even integrate completely new behavior at run-time.

## 1.2. Contributions

The main contribution of this thesis is to extend the in essence *control-oriented* XMDD paradigm with *data-orientation*, adapting well-known paradigms from programming languages in a *simplicity-first* fashion, ranging from features of object-orientation [KA90] to functional programming [Ses12] (cf. publication VII). The new approach is type-aware, so that type-safety can be validated at design-time. The central achievement is, however, the introduction of higher-order semantics by treating services and processes as *first-class citizens*. They may be moved around *just like data* and *plugged and played* into activities at runtime, thus enabling *higher-order process engineering* (HOPE).

The realization has been carried out minimally invasive on top of the current reference implementation of XMDD – the modeling environment jABC3 [SMN<sup>+</sup>06, SM08] – facilitating its simplicity-oriented plugin framework (cf. publication IV). The universality of the jABC allowed to realize the new kinds of activities with only six generic components following the design-pattern of jABC3 (cf. publication VII). This way the plugin mechanism and already existing plugins, e.g. for execution [KM06, SMC<sup>+</sup>96] (i.e. interpretation), full-code generation [JMS08], global verification via model checking [BMRS07, BMRS09] as well as local validations [Neu07] could be reused and even components developed via existing and new (HOPE) concepts may coexist in the new framework allowing a fluent migration process. Thus, one could focus on adding newly available data-flow and type information incrementally in the jABC3 framework and its plugins.

The HOPE approach has been validated in different scenarios where the dynamic exchange of processes, services, and service implementations is essential:

- dealing with the *combinatorial explosion* regarding variant rich systems as well as the flexibility needed to be able to react to requirements or environmental changes at run-time (cf. publications II and V),
- applying the approach to *active automata learning* [Ang87, SHM11, NMS13, NSB<sup>+</sup>12, WNS<sup>+</sup>13, HSM11] and risk-based testing [FR14, GT02] (cf. publications I and VI) integrating process models into the active automata learning framework LearnLib [RSBM09, MSHM11] via full-code generation [Jö13, JS12],
- run-time enabling *process model synthesis* (cf. publication III), i.e. loading tailored, synthesized processes at run-time and plug them into running processes as appropriate, and
- several Bachelor and Master theses as well as research projects ranging from reverse engineering, over modeling of dynamic databases, benchmark generation, automata learning, to modeling game strategies.

### 1.3. Conventions

The following conventions will be used throughout this document:

*new notion*

New notions will be written emphasized.

“label”

Labels of activities, branches, and context variables will be presented in quotation marks. The differentiated types of labels will be disambiguated in the text.

**C**ClassName, **I**InterfaceName, and **E**EnumerationName

Simple as well as full qualified names of Java types will be prefixed with a corresponding icon, and written in a type writer font.

**G**ServiceGraphName and **G**InterfaceGraphName

Simple as well as full qualified names of graph types will be prefixed with a corresponding icon, and written in a type writer font.

### 1.4. Organisation

The structure of this thesis reflects my contributions as follows: In Chap. 1 the research questions, related work and my contributions are described. Chapter 2 introduces the extreme model driven design (XMDD) paradigm and one thing approach (OTA) as well as the accompanying example for this thesis. The concepts of HOPE are highlighted in Chap. 3 by means of the accompanying example. Chapter 4 summarizes projects validating the HOPE approach. Chap. 5 concludes the thesis and gives an outlook to future work in this research area.



## 2. Preliminaries

This chapter will

- briefly outline the technological and conceptual basis in Sec. 2.1 as well as point out the new requirements that led to HOPE and
- introduce the accompanying example for this thesis in Sec. 2.2: pupils model strategy-based computer opponents for the computer board game ChainReaction.

### 2.1. Extreme Model-Driven Design

The basis for higher-order process engineering (HOPE) is the extreme model-driven design paradigm (XMDD) [MS04, MS06, MS09a, MS12] which embodies ideas from

1. service orientation [MSR05],
2. model-driven design [VG07], and
3. the end-user-centeredness advocated in extreme programming [BA04].

Combining these strands enables application experts to control the design and evolution of processes during the whole life-cycle according to their own level of technical competence and business responsibility. The *one thing approach* (OTA) [SN07, MS09b] provides the conceptual modeling infrastructure for XMDD that enables all the stakeholders (application experts, designers, component experts, implementers, quality assurers, ...) to closely cooperate in the design process.

In particular it enables *immediate user experience and feedback* and thereby *seamless acceptance*: all stakeholders know, refine, and modify one and the same “thing”, without duplications or need to juggle with different modeling languages or paradigms. It allows them to observe the progress of the development and the implications of decisions at their own level of expertise, which is a central trait of the one thing approach.

The language for this comprehensive model where all the information converges are executable process models called *service logic graphs* (SLGs). Operationally, the process models are similar to control flow graphs: the nodes represent activities and the branches describe how to continue the execution depending on the result of the previous activity. Following the terminology of telecommunication systems [SMC<sup>+</sup>96], these activities are called *service-independent building blocks* (SIBs).

SIBs may represent a single functionality (i.e., a service) or a whole subgraph (i.e., another SLG) introducing hierarchy [SMBK97], thus serving as a macro that hides more detailed process models. SIBs are parameterizable and communicate resources

via shared *execution contexts*, a hierarchical concept. The application expert is equipped with a collection of SIBs, which forms the available domain of reusable, configurable processes and components shaping a kind of *domain specific language (DSL)*.

SLGs are also directly formal models: they are semantically interpreted as Kripke Transition Systems (*KTS*), a generalization of both Kripke structures (*KS*) and labeled transition systems [MSS99] (*LTS*) that allows labels both on nodes and edges.

**Definition 2.1.** A *KTS* over a finite set of atomic propositions  $AP$  is a structure  $KTS = (S, s_0, Act, R, \mathcal{I})$ , where

- $S$  is a finite set of *states*.
- $s_0 \in S$  a dedicated *start state*.
- $Act$  is a finite set of *actions*.
- $R \subseteq S \times Act \times S$  is a total *transition relation*.
- $\mathcal{I} : S \rightarrow 2^{AP}$  is an *interpretation function*.

In publication VII the underlying formal model is incrementally enhanced for better presentation of the new concepts presented here. Therefore, an alternative formal definition of the core elements in an SLG is introduced, providing a more natural (canonical) interpretation. In a *KTS* the action labels are interpreted as the active part and the states are idle. In SLGs the activities are the active part (cf. the *states* in a *KTS*), and decide which branch (cf. the *actions* in a *KTS*) is followed after execution to find the successor activity, leading to the following definition:

**Definition 2.2.** A *dKTS* (say *dual KTS*) over a finite set of atomic propositions  $AP$  is a structure  $dKTS = (\mathcal{A}, a_0, \mathcal{B}, \delta, \mathcal{I})$ , where

- $\mathcal{A}$  is a finite set of *activities* (instantiations of SIBs).
- $a_0 \in \mathcal{A}$  a dedicated *start activity*.
- $\mathcal{B}$  is a finite set of branching labels denoted by *branches*.
- $\delta : \mathcal{A} \times \mathcal{B} \rightarrow \mathcal{A}$  is a *transition function*.
- $\mathcal{I} : \mathcal{A} \rightarrow 2^{AP}$  is an *interpretation function*.

A *dKTS* is called *dual KTS*, since edge labels and nodes swap their semantics. Moreover, in order to have a clearer separation of notions, the term service independent building block will be used for templates of *activities*, i.e., an activity is an instantiation of a SIB component.

The *Java application building center* (jABC) [SMN<sup>+</sup>06] in its current version 3 as well as its predecessors starting with the MetaFrame [SM99] tool have been the

technical incarnation of the XMDD and OTA approach representing SLGs as graph visualizations. A systematic inquiry of the status quo in business process modeling [DS12] resulted in the insight that jABC is the only environment for model-driven design [MBD<sup>+</sup>12] supporting proper integration of services in a service-oriented fashion [MBD<sup>+</sup>12]: so called *domain-specific business activities*.

Originally the framework dealt in particular with two communication paradigms regarding the execution context:

**Pipelining** The *electronic tool integration* platform (ETI) [SMB97], its Java based ancestor jETI [KMFS06, KMSN07, KMSN08], and Bio-jETI [LMS08b, LMS08a, LMS<sup>+</sup>08c, LMS09a, LMS09b] – a specialisation focusing on applications in the biology context – use direct pipelining for the data transfer between activities.

**Call Context** Modeling telecommunication services [SM99, HMN<sup>+</sup>01, HHNS02] naturally resort to a global fixed data model, since the structure of the underlying hardware self-evidently suggests it.

In both cases there was either

- no issue regarding the context handling, as it could be realized via model checking and temporal logic formulas, or
- it could be handled via the configuration universe induced by user-defined abstract types over taxonomies.

In combination with the declaration of *gen*, *mod*, and *kill* properties on SIB parameters, this was an appropriate abstraction for introducing PROPHETS [NLS12] facilitating process synthesis.

Later on XMDD has been employed to different fields and projects [MNS05, MKS08, LNMS11, HMM<sup>+</sup>08, JSM11, NSM<sup>+</sup>01, NMS13], so that the framework has been extended to fit the increasing needs, e.g., a hierarchical context with the scopes *local*, *declared*, *parent*, and *global* has been added. Variability had either been hidden beneath the SIB adapter, thus buried on the code-level (cf. Sec. 2.1.1), via design-time process model synthesis, or via ad-hoc modeling.

### 2.1.1. Adapter Pattern

The integration of services into the jABC3 framework bases on the *adapter pattern* (cf. Fig. 2.1): Each SIB is represented by a Java class, which defines the parameters, branches, documentation, icon (for representation as a node), SIB adapter, and other meta-information of the SIB (like which validations regarding correct usage of a SIB should be conducted during modeling) shaping its interface to the modeling level (i.e., the SLGs).

A *SIB adapter* may be realized in any target language and encapsulate the service call that the SIB represents. It takes the execution context as well as the SIB parameters as input, invokes the service and stores the result into the context again. In general there exists at least an adapter implemented in Java for each SIB, since the jABC is based on Java and comes with an integrated interpreter.

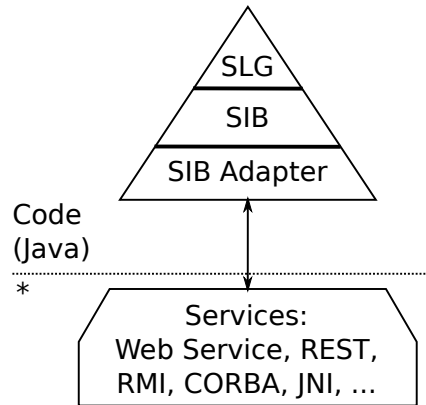


Figure 2.1.: SIB adapter pattern. Source: publication I

In a SIB class there is no differentiation between input and output parameters and for which branch which parameters are used as outputs, information that is missing for modeling-level validations like data-flow analysis. Thus, it is defined in the SIB class whether a parameter should be entered as a constant in the modeling environment or whether it should be connected to a variable in the context. This may lead to a lot of SIB classes for the same service, just to model that some parameters are constants (i.e., *static*) and others are read from the context (i.e., *dynamic*).

Since a SIB receives the complete hierarchical execution context as input it may access the data – reading as well as writing – on the current and on any hierarchy level above, including the global context, which is accessible from everywhere. This enables to write SIB libraries that need nearly no configuration on the modeling level as the communication with the execution context is defined almost entirely in the SIB adapter, but at the modeling level there is no track of this, and therefore it cannot be validated easily. In addition it is a hard task to interoperate between different SIB libraries, when the communication definition is buried on the coding level.

Adding new services to jABC3 requires the implementation at least of a SIB class and an adapter class, which depend on the jABC framework themselves. Hence this mixes the concerns, as a technically versed person has to implement them. Therefore – for some domains – generators have been developed, creating these classes automatically from *web-service description language* (WSDL) descriptions [KMSN08] or a wizard for importing services from *enterprise resource planning* (ERP) systems [MBD<sup>+</sup>12]. In this thesis this concept is generalized to dynamic service binding (cf. Sec. 3).

### 2.1.2. Hierarchy & Reuse

The input/output parameterization of an SLG is modeled via model parameters and model branches. A modeler marks parameters and branches of the activities in an SLG as model parameters (branches) in order to export them to the next hierarchy level.

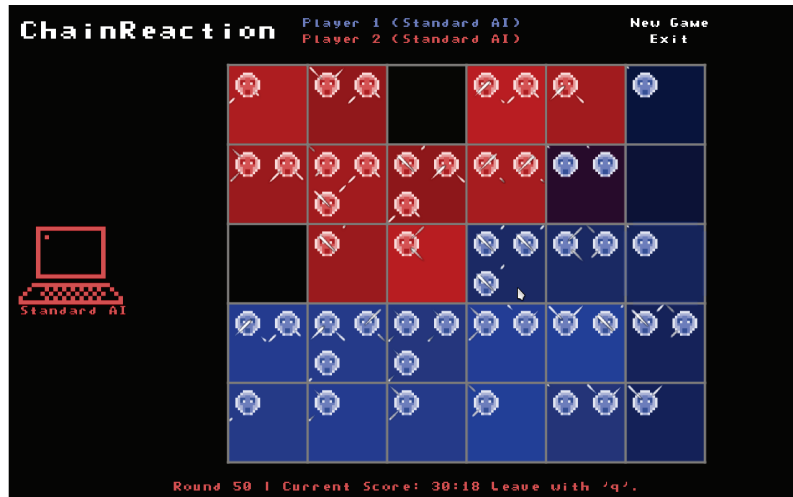


Figure 2.2.: A screenshot of the open source computer board game ChainReaction in retro-look

An SLG is represented by a *graph SIB*, which retrieves the model parameters and branches from the submodel and dynamically adds corresponding parameters and branches to its instantiated activity. The model parameters can be configured in the graph SIB just as it would be the case for the SIBs the parameters come from. Furthermore it is possible to bundle parameters and branches to one model parameter (or model branch respectively).

If an activity referencing a sub model is executed, it will create a new (empty) execution context and invoke the start activity of the SLG. The model parameters are evaluated as the control-flow reaches the corresponding SIB. If an exported branch is followed, the execution context will be removed and the corresponding branch on the invoking graph SIB will be followed (one hierarchy level above).

The wiring between a graph SIB and its sub model is – as for services – static. I.e., it is set at modeling time and is not meant to be changed at runtime. Additionally, a graph SIB may choose a sub model from the current jABC project, only. Reuse beyond project boundaries is merely subject to SIBs in the jABC3 framework. SIBs, on the contrary, may be bundled into SIB libraries and imported to jABC projects.

## 2.2. ChainReaction

In order to show the impact of HOPE, a accompanying example will be used, which bases on a project week for pupils regularly held at events at the university (TU Dortmund) and at schools to attract pupils for computer science. The pupils have been asked to create a game strategy (GS) representing a computer opponent in the open source computer board game *ChainReaction*<sup>1</sup> with jABC4.

ChainReaction (cf. Fig. 2.2) is a two player computer game with a  $6 \times 5$  board. The opponents place one token – denoted by “atom” – in a cell turn-by-turn. The owner of a cell is the player with at least one atom in the cell. A player may place

<sup>1</sup><http://cr.freewarepoint.de>

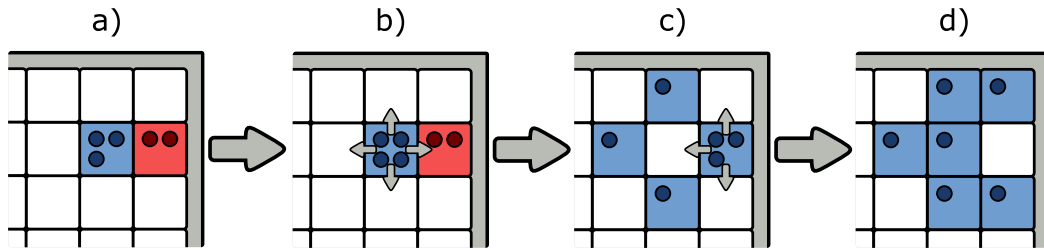


Figure 2.3.: Course of a chain reaction resulting in a win situation for the blue player

an atom either in his own or in empty cells. Each cell has a capacity depending on its horizontal and vertical neighbors. Hence a corner cell has a capacity of two, an edge cell a capacity of three, and a center cell a capacity of four atoms. If a cell reaches its capacity, it will explode and each atom spreads to one neighbor cell. A spreaded atom assimilates all atoms of its new cell, no regard who owned the cell before or how many atoms were already in that cell. The exploded cell will be empty and thus is no longer owned by the player. If one of the assimilated neighbor cells reaches its capacity because of the new atom, it will explode, too. The name of the game comes from the resulting – for the player hard to predict – chain reactions which have the potential to change the complete board. The goal of the game is to reach a board configuration where the opponent owns no more cells.

Fig. 2.3 shows the course of an exemplary chain reaction. In board situation a) it is the blue player's turn. He could place it on any cell but the right neighbor of his own cell, because that one is owned by the red player. Player blue places an atom in his sole cell as it is critical, i.e., only one atom is lacking until the capacity is reached. As shown in board situation b) it explodes, resulting in board situation c). A winning situation for the blue player has been reached, but there is a cell which has reached its capacity so that it explodes, leading to the final board situation d).

The task for the pupils was to create a game strategy which evaluates a given cell in a given board configuration regarding the benefit to place an atom there. The HOPE technology turned out to be easy to learn for the pupils, and led intuitively to surprisingly good solutions.

### 3. Higher-Order XMDD

The *higher-order process engineering* (HOPE) approach is a consequent evolution of the XMDD approach. The main goal of XMDD still holds for HOPE: involve the application expert, who knows the requirements on an application, into the system design process. However, application experts are not necessarily well-versed in technical realization, so that typically technical experts implement the requirements. On the contrary, technical experts in general lack expertise in the application domain. According to concepts like extreme programming, lean design, and agile computing, this is tackled by strengthening the communication between technical experts and application experts. Unfortunately, the state-of-the-art solutions to this are either:

1. based on informal communication [SS06, STA05], which is error-prone and puts forward a *semantic gap* between the participants in terms of terminology and experience,
2. use a formal, executable language like BPEL+BPMN [AAA<sup>+</sup>07], but it burdens the modeler to juggle with technical details like web service endpoints, or
3. use the formal description language BPMN 2.0 [OMG11] which is less technical, but introduces a *semantic gap* between the description and its realizations, as the standards as well as their current realizations have no proper support for the integration of business activities in a service-oriented fashion [DS12].

In the XMDD approach and its reference implementation, the jABC framework, *immediate user experience and feedback* as well as *seamless acceptance* have been realized introducing a hierarchical coordination layer, i.e. the SLGs, with its components, i.e. the SIBs, which are decoupled from the service implementations beneath. Hence the application experts design the application behavior according to the requirements in coarse-grained, easy-to-understand process models, and the technical experts implement the needed SIBs and services. Furthermore, the role of *domain experts* is situated in between application experts and technical experts: they are responsible for bundling SIBs from technical experts to libraries and present them to the application experts tailored to a specific domain [MS06].

The HOPE approach extends XMDD with runtime variability capabilities, in order to meet the growing requirements on (business) process modeling in terms of supporting system evolution beyond the state-of-the-art of design-time variability like product-lining and variability modeling with the flexibility to safely add new functionality at runtime. The weapon of choice hereof is going higher-order by treating process instances and service objects as first-class citizens and add data-orientation to a control-oriented approach. This leads to the flexibility of adding new entities

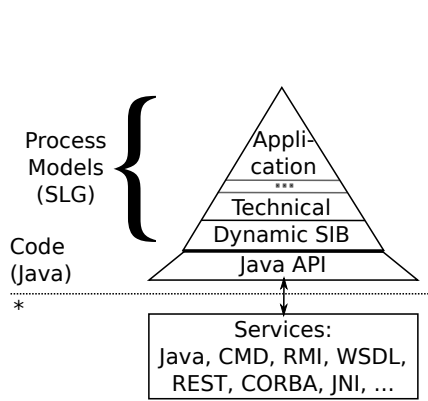


Figure 3.1.: Dynamic SIB pattern. Source: publication I

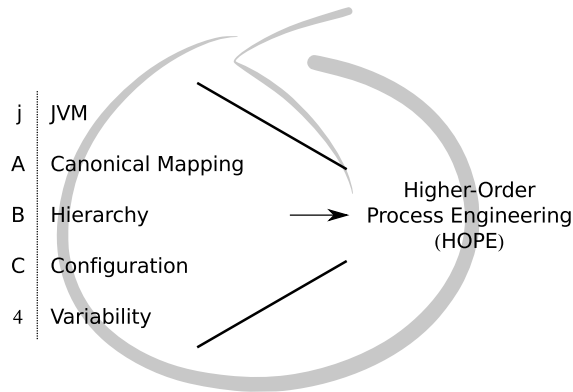


Figure 3.2.: Coarse-grained structure of the higher-order process engineering (HOPE) approach.

to a running system seamlessly as well as results in very concise and understandable models as the variants do not have to be captured explicitly. Hence there is no explosion in terms of model size or number of variants. A process model can be completely unaware at design-time of the implementations that will be executed for the constituent activities at runtime (cf. publication V). To guarantee executability it is just necessary that the implementations adhere to required interfaces, which is achieved by a strict higher-order type discipline. Moreover, the process models may be generated to executable code maintaining the flexibility of the modeling approach of HOPE (cf. publication VI).

In the following, Sec. 3.1 will sketch how service objects are lifted to first-class citizens. Sec. 3.2 illustrates the structuring of the development process in the HOPE approach, whereas Sec. 3.3 briefly introduces the user interface of the jABC4 framework. In Sec. 3.4 the impact of higher-order process passing is demonstrated by means of the accompanying example. Sec. 3.5 shows how domain-specific business activities are realized in the HOPE approach. Finally, Sec. 3.6 outlines HOPE's full-code generation capabilities.

### 3.1. Dynamic Service Binding

The decoupling of modeling and implementation level had been realized via the adapter pattern (cf. Sec. 2.1.1) in the XMDD paradigm, which ensures executability, readiness for full-code generation, and independence of the implementation, so that the modeling can take place even when the service implementations are not (yet) available.

The challenge has been to further on support *domain-specific business activities* that are easy-to-integrate and easy-to-use (after adding data-orientation to the XMDD approach). This is tackled by elevating the design of SIBs (i.e. the templates for activities) to the modeling level (cf. Fig 3.1). Since the SIB adapter is a static wrapper around service objects, it is replaced by dynamic service binding in technical SLGs, where a method of the target language is bound to an activity. This enables



second-order semantics by treating objects and their methods as first-class citizens (cf. publication I). Thus, these services (i.e. methods) had to be made available in the modeling environment so that a modeler can find and use them easily in process models. This has been tackled via *service browsers* in the user interface of jABC4. A service browser shows methods in a tree-view sorted by class and package and offers to filter them by various properties.

SIBs are now realized via service logic graphs (SLGs) and published in libraries to be reused by application experts. A user may access them via the *graph browser* in jABC4. Additionally, application experts should be shielded from the complexity of the complete communication between activities which had been encapsulated by the SIB adapter before. This issue is resolved by preconfiguring activities accordingly to create new, tailored SIB libraries from existing ones. The latter represent domain-specific business activities in the HOPE approach. As they in particular differentiate between input and output parameters over their predecessors (i.e. the SIBs), they are called *IO SIBs* and are accessible via *IO SIB browsers* in the user interface.

### 3.2. Structure of HOPE

HOPE partitions the development process into different layers (cf. Fig. 3.2) in order to support *separation of concerns* to face complexity. The basis are standard APIs in a target language, i.e. Java [Blo08] or Scala [OSV10] in the reference implementation, denoted by *Java Virtual Machine* (JVM). Java or more generally JVM based languages are a good choice here as Java is a largely platform-independent language. Further on, almost every technology or method can be wrapped into a Java method as it provides generators producing WSDL and *representational state transfer* (REST) stubs, interpreters for scripting languages like Groovy, Jython, and JRuby as well as the *Java native interface* (JNI) for accessing platform dependent functions implemented, e.g., in C, Objective-C, C#, or C++. However, Java is only used on the tooling/realization level. The concepts are quite general and can be implemented in and for different general programming languages.

On top of that, a layer of technical process models is situated, which directly binds methods of the target language dynamically to activities (cf. layer “Canonical”). The underlying target language provides well-defined semantics guaranteeing executability. This way, the hard technological break is moved behind a Java API, where only technical experts operate, supporting the principle, “simple for the many, difficult for the few” [MS05].

Technical details are encapsulated via hierarchical modeling (cf. layer “Hierarchy”). Moreover, the HOPE approach is component-based, it enforces sophisticated input/output parameterization supporting parametric polymorphism<sup>1</sup> of process models. Together with the already available type system of the underlying target language this enables to model explicitly the (type-aware) data-flow information of all components in addition to the control-flow information.

---

<sup>1</sup>Realizations of parametric polymorphism are often referred to as “generics” in the corresponding target languages [NW06].

The complexity being a consequence thereof is tackled via preconfiguration of activities and their input/output parameterization (cf. layer “Configuration”). Furthermore, *inversion of control* (IoC) is used to *inject* components into processes, reminiscent to component models like Java EE [Gon13]. Global or more specifically non-local resources are almost inevitable in big software systems. *Dependency injection* (DI) is a modern concept that offers a fine-granular management of which components access which resources. This decoupling is often desirable as selecting resources is a concern orthogonal to the business-logic.

Consider a web application where the access to a shared database is regulated via transactions<sup>2</sup>. With dependency injection, the environment provides components, for accessing the database attached to the correct transaction, to each process instance. This prevents from accidental misuse of transactions.

The process models become flexible and stay comprehensible by adding a higher-order flavor: services and process instances are treated as *first-class citizens*. Implementations of an activity may be exchanged at runtime (cf. layer “Variability”).

These layers may be iteratively applied by diverse participants in the development process to create a complete hierarchy of domain-specific, preconfigured components, and make them available in libraries to be reused as activities on the respective next layer of abstraction.

The system uses the available type information to check whether the use of services and process instances passed is correct with regard to the formal parameters of the underlying API or graph interface and the actual arguments. This type-checking mechanism intersects the technical and business worlds and narrows the semantic gap between the complex network of application experts and technical experts.

As an example for higher-order process passing, consider a shopping order process (cf. publication II). It is finalized with a payment step. There are different ways to pay, like by credit card, direct debit, online payment services, et cetera. Still from the point of view of an application expert it is simply the act of paying a given amount to a given company which should be modeled via a business activity called “payment”. Lateron, at runtime, a concrete payment service has to be selected. This can be done via an arbitrary lookup/discovery mechanism depending on information like user preference (i.e. configuration), or direct manual selection in an online form. When the activity “payment” is executed, it should reproducibly invoke the selected process. If a new payment service becomes available, it should even be possible to use it for the activity “payment” without a redeployment of the shopping order process.

### 3.3. jABC4

The user interface of the modeling environment jABC4 (i.e. the incarnation of HOPE) is shown in Fig. 3.3. It is divided in three main areas:

---

<sup>2</sup>A transaction encompasses a unit of work on a database and has to fulfill the properties *atomic*, *consistent*, *isolation*, *durable* (ACID).

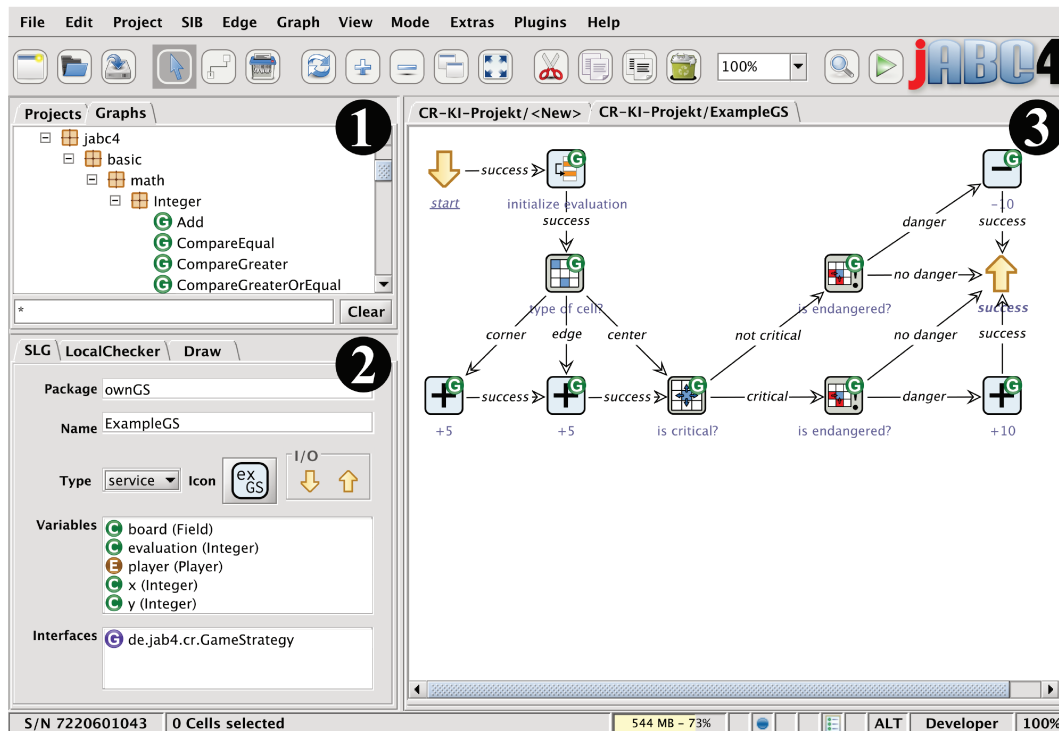


Figure 3.3.: Screenshot of the jABC4 main window showing a process model of a simple game strategy (GS) for the computer board game ChainReaction (cf. publication VII).

1. The *browser area* consists of different tabs with tree-views for resources, namely the *project browser*, *graph browser*, *service browsers*, and *IO SIB browsers* (cf. Sec. 3.1).
2. The *graph canvas* is the main modeling area, where the SLGs are depicted. New activities are created by drag&drop from the browser area and control-flow is provided via adding edges with branch names.
3. The *inspector area* contains panels like the *SLG-*, *Draw-*, and *LocalChecker* inspector, which display meta-information regarding the current SLG or selected activities and provide editors for changeable attributes.

In Fig. 3.3 the graph canvas shows an example game strategy for ChainReaction. Currently, no service- or IO SIB browser is available, as – in the spirit of separation of concerns – the visibility of components depends on the level of technical expertise of a user. In addition, the graph browser presents some basic SLGs for simple arithmetic operations.

The *SLG inspector* depicts meta-information like *package*, *name*, the SLG's *variables* shaping its execution context, and the *interface* it implements. All game strategies, e.g., implement the (graph) interface `GameStrategy`. The icons left of the variables in the inspector classify its type like `C` for a Java class, `I` for a Java interface, `E` for a Java enum, `G` for a graph implementation, and `G` for a graph interface. Further on, the type's name without package (i.e. the *simple name*) is

situated behind the name in parantheses. The user can change the name and package of the graph, add, delete, and rename context variables, edit their type, as well as drag&drop between variables and activities wiring data dependencies.



### 3.4. Higher-Order Process Modeling

First evidence of the impact and ease-of-use of the HOPE approach has been gained by applying it to non-programmers. Pupils in secondary education with little technical expertise have been encouraged to create game strategies, shaping the behavior of computer opponents for the game ChainReaction (cf. Sec. 2.2) with jABC4.

Fig. 3.4 shows multiple hierarchy levels of a higher-order process execution that tests an example strategy **GExampleGS** (cf. Fig. 3.3) against the reference strategy named **GStandardGS**. Both strategies are graph implementations (called *service graphs*) of the graph interface **GGameStrategy**. The example strategy prefers corner cells (bonus of +10), then edge cells (bonus of +5). Moreover, it considers whether a cell is critical (i.e. the cell will explode, if one more atom is added) and will add a bonus of +10, if the cell is endangered, i.e. there are critical neighbor cells of the opponent, since the reaction will be carried further via these cells. The strategy will give a malus of -10, if the cell is not critical but endangered, since the opponent can occupy this cell (and assimilate all atoms) when it is his turn.

#### Challenge 1: Graph Interfaces

Two types of SIBs have been introduced to define graph interfaces: the *input-* and *output SIB*. The input SIB is used as start activity declaring the formal input parameters and an output SIB represents an end activity declaring a branch and its formal output parameters. An *interface graph* analogously uses input- and output SIBs to declare the input/output parameterization for all its implementations (comparable with a method signature), but does not contain any control-flow (comparable with a method body). Although HOPE adds programming language features to process modeling, these are intentionally introduced in a *tamed* manner in order to keep it simple and manageable for non-programmers (cf. publication III). In this spirit the inheritance relation is kept flat: a service graph may implement exactly one interface graph, and there is no inheritance relation between service graphs or between interface graphs.

The top-level SLG in Fig. 3.4 is created by the pupils (i.e. the application experts) and instantiates new process instances via the *constructor activities* “ExampleGS” and “StandardGS” for the two game strategies. Constructor activities are highlighted via the green “\*” overlay-icon  in the top-right of the activities’ icon. The thick grey arcs represent the data-flow. The process instances of the constructor activities are written to the context variables “player 1 GS” and “player 2 GS”, respectively. They are used as inputs for the *abstraction activity* labelled “start game”. Abstraction activities introduce hierarchy and the small green “G” overlay-icon  illustrates that it references a fixed graph implementation, which is depicted below

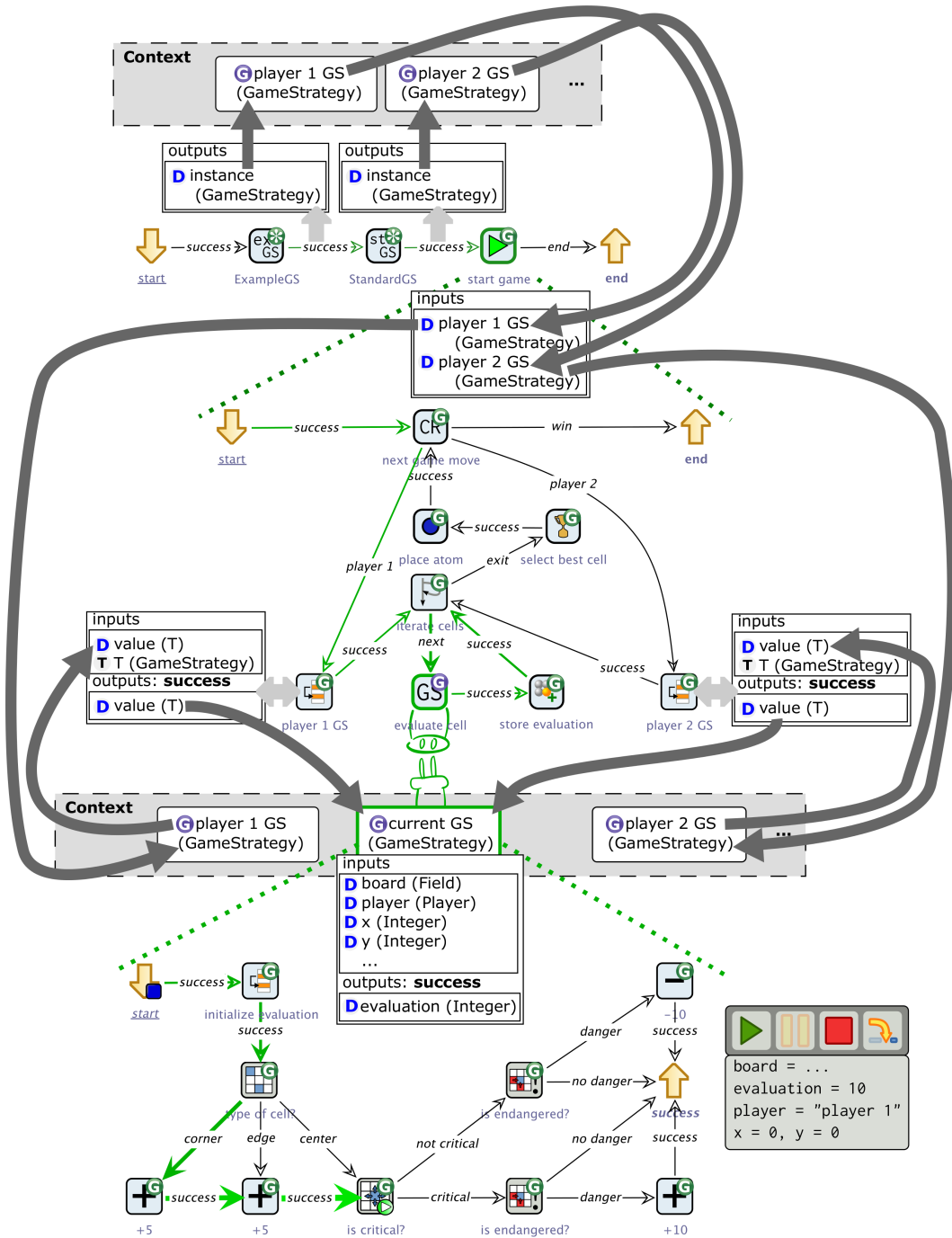


Figure 3.4.: Higher-order process engineering in action, illustrating a snapshot while conducting a ChainReaction game with two game strategies.

(on the next hierarchy level). The corresponding process model handles the interaction with the game, so that the three-node top-level SLG named *starter graph* is all the pupils had to prepare for testing their strategies.

This kind of (higher-order) modeling is reminiscent to functional programming: One or more functions (processes) may be passed to a higher-order function (process), handing off the details of applying the former. Although functional programming languages like Haskell [Mar10] and ML [Ull94] have the reputation to be complex and hard to learn, concepts like higher-order functions simplify programming tasks as they decouple *what* should be done from details of *how* it is realized. Therefore, constructs like  $\lambda$ -expressions [Ste90] basing on the  $\lambda$ -calculus [Bar84] enter into more-and-more object-oriented languages like C# or Java 8 [War14].

The service graph named **GStartCR** on the second-hierarchy level – referenced by activity “start game” of the top-level graph – retrieves the two game strategies as inputs and stores them to the context variables “player 1 GS” and “player 2 GS” (on the second hierarchy level). A technical expert (i.e. a Bachelor student) implements the respective process model **GStartCR**, serving as a bridge between jABC4 and the game. In essence,

1. it invokes for each turn the corresponding strategy for every placeable cell resulting in a cell evaluation in form of an integer,
2. chooses randomly from the cells with the same highest evaluation, and
3. places an atom in the chosen cell.

Hence, the game strategies of the pupils just have to deal with evaluating one cell in a divide-and-conquer fashion, whilst the bridging graph takes care of the recurring task to interact with the game, iterate through the board, and decide which cell should be chosen. Additionally, the bridging graph **GStartCR** is completely unaware which game strategies will be executed as long as they fulfill the graph interface **GGameStrategy**.

#### **Challenge 2: Compositionality of SLGs**

Higher-order process passing requires compositionality of processes. In this regard, besides graph interfaces, the handling of scopes for variable declarations is essential. In [WS73] the authors consider global variables as harmful – in the spirit of [Dij68] where Dijkstra considers the ‘goto’-statement harmful – as it leads to side effects which are undesired especially in functional programming. Therefore, in HOPE all context variables are local and an SLG cannot access any variable higher (or lower) in the call context, consistent with [WS73] which proposes a more cautious treatment of scopes. As non-local resources are almost inevitable and often desirable, a modern approach is to decouple the primary (i.e. functional) business-logic from the logic of the selection and retrieval of resources by introducing dependency injection. Hence, the ‘consuming’ process hands over the control regarding the selection and retrieval of resources to the environment of a process instance (cf. Sec. 3.2). This allows the *container*<sup>3</sup> to tailor resources to the needs of the current process instance, based

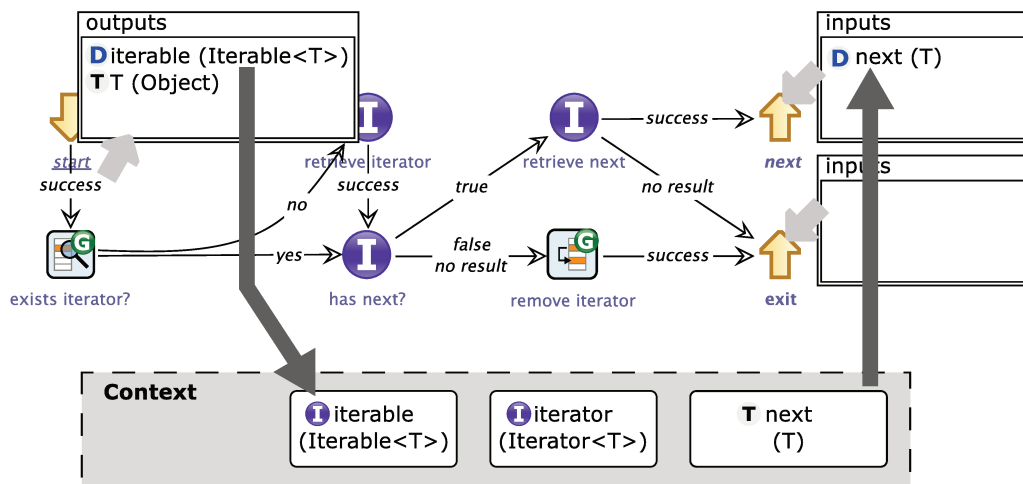




Figure 3.5.: The service graph for iterating through an arbitrary collection.

on the system state, the caller of the process, and the global configuration of the system.

For instance, in the context of product-lining, the environment decides (e.g. on the basis of configuration), which concrete product is selected and injects the respective resources at runtime. For publication I and VI this has been used to execute the same high-level tests and inject different implementations of the system under test (SUT), in order to be able to compare the results later on. As proposed in Sec. 5 the logic of the dependency injection can be described in process models, too.

In detail, the activity “next game move” evaluates after each turn whether it is the first or second player’s turn (cf. branches “player 1” and “player 2”), or the game has ended because the last turn led to a winning situation (cf. branch “win”). If it is the first player’s turn, the corresponding game strategy stored in the variable “player 1 GS” is written to variable “current GS” via the activity “player 1 GS”. Analogously, the game strategy of the second player is written to variable “current GS” if it is his turn. In both cases the same (in terms of identical) activity “iterate cells” is used to iterate through the board and the same activity labelled “evaluate cell” is used to execute either the game strategy of the first or second player as it invokes the process instance in context variable “current GS”. The purple “G” overlay-icon  of activity “evaluate cell” highlights, that it references an interface graph dynamically and may therefore invoke any compatible graph implementation at runtime. The service graph StartCR is concise and reusable as it can both

1. handle arbitrary game strategies and
2. self-adapt at run-time according to the current game situation (i.e. which player’s turn it is).

In complex domains, a network of participants in the development process originates, so that the technical expert on one level becomes the application expert on

the next one by means of getting more technical. For accessing the API of the game ChainReaction, a technical expert builds a library of SLGs tailored to evaluating the worthiness of a game cell (cf. Sec. 3.5) and provides it to the application experts (i.e. the pupils). But then again he could rely on existing basic SLGs for dealing with standard operations regarding context handling (e.g. the SLG  $\textcircled{G}$ PutToContext for activities “player 1 GS” and “player 2 GS”) or standard Java APIs like the collections framework.

In activity “iterate cells” the service graph  $\textcircled{G}$ Iterate<T>, e.g., is used to iterate through the cells of the game board. The implementation of the underlying graph is shown in Fig. 3.5. It is a generic process model with the (graph) type parameter T for the elements of the collection to be iterated. Hence, highly reusable process models are realized via introducing generics (i.e. type parameters) for graphs.

#### Challenge 3: Representation and Editing of Types

A context variable may represent a Java type like a class, an interface, or an enum as well as a graph type like a service or interface graph. For the former the Java reflection API [FF04] could be used, the latter had to be represented separately as no Java types exist for graphs in the modeling environment. Moreover, each of these types can have type arguments. The Java reflection API does not offer creating parameterized types at runtime dynamically, so that this had to be provided in order to edit type arguments in the modeling environment. Finally, a process model like  $\textcircled{G}$ Iterate<T> is itself generic and in the scope of the graph (i.e. its execution context and activities), the type parameter may be used as a type or as a type argument both for parameterizable Java and graph types.

$\textcircled{G}$ Iterate<T> uses *atomic activities* (i.e. they bind to a Java method and not to a sub graph like abstraction activities) to

- retrieve the iterator from the collection the first time it is executed,
- check each time it is executed whether it has still elements left, and
- if yes returns the next element via the branch “next”, or
- if not leaves via the branch “exit” without a return value.

#### Challenge 4: Service & SLG Selection

The atomic activities in a service graph may represent an arbitrary public method of a Java type on the classpath. These are made available in service browsers, which can be added to and configured in a jABC4-project. The settings dialog is shown in Fig. 3.6 and allows to select entries from the classpath as well as filter them by package and name. In the corresponding service browser as illustrated in Fig. 3.7, methods are the leaves in a tree-view. They may be dragged-and-dropped to the graph canvas to create a corresponding activity (cf. publication I). For abstraction



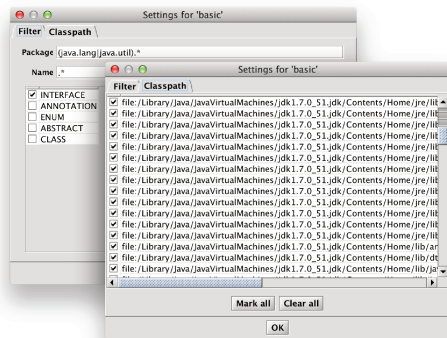


Figure 3.6.: Both tabs of the settings dialog for service browsers.  
Source: publication VII.

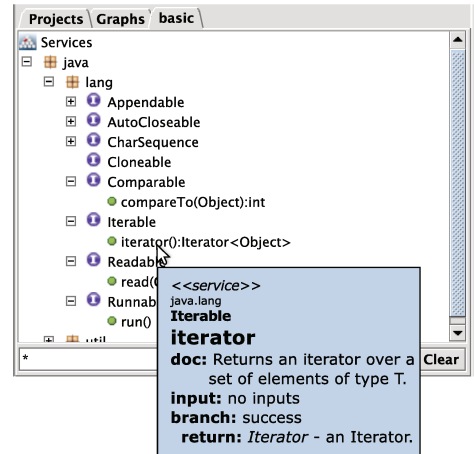


Figure 3.7.: A view of a service browser offering some services of the JRE.  
Source: publication VII.

activities, a graph browser scans the classpath for graph types and presents them analogously in a tree-view sorted by package and name (cf. the browser area in the top-left of Fig. 3.3).

Back to graph **G**StartCR in Fig. 3.4: the activity “iterate cells” invokes process model **G**Iterate<Cell> to iterate through the cells of the board. The only prerequisite is that the Java class **C**Field, which represents the board, implements the standard Java interface **I**Iterable<Cell> where **C**Cell is the implementation for the cells of the board. All collections in the Java collection framework implement **I**Iterable and therefore the service graph **G**Iterate can be used for any collection. Hence, the reusability of process models can be enhanced significantly by exploiting inheritance in combination with subtype polymorphism of the target language.

#### Challenge 5: Adapting Type Arguments for Activities

Both atomic and abstraction activities are created via drag&drop from a corresponding resource browser. They are configured automatically with the input/output parameterization of the underlying structure (i.e. a method or sub model) during instantiation. Type parameters are initially set to its upper bound. For the SLG **G**Iterate<T> this is **C**Object. If a context variable is connected to the input parameter “iterable” (cf. Fig. 3.5), the graph type parameter T has to be parameterized according to the type argument of the context variable. In case of a class implementing **I**Iterable<Cell>, the class **C**Cell is identified and set. The output parameter of the branch “next” obtains the type, too.

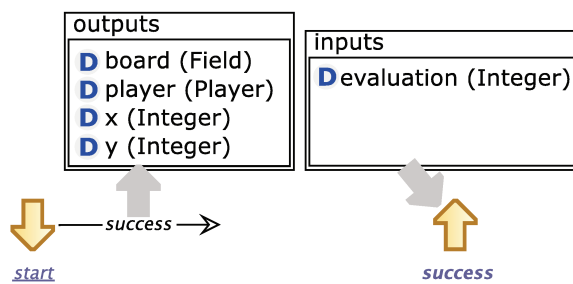


Figure 3.8.: Interface graph for game strategies in the ChainReaction scenario. Source: publication VII

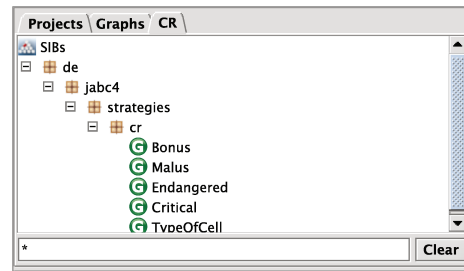


Figure 3.9.: Example IO browser for the configuration graph shown in Fig. 3.10. Source: publication VII

Further on the iterator has to be reused with its current state for a complete iteration through the board, so that the branch “next” is not followed infinitely often for the first element. This is done by storing the process instance of the graph  $\text{GIterate}<\text{Cell}>$  – together with its current state – in a context variable.

Fig. 3.4 shows a snapshot of an execution. The green control-flow edges highlight the history of execution. The green “play”-symbol in the bottom-right of the icon of activity “is critical?” in the actual game strategy depicts that the execution is paused before executing the very activity. As the green arrows suggest, it is the strategy of the first player and the game strategy is the same as shown in Fig. 3.3. The interpreter of the jABC can be configured to stop at a given activity via breakpoints, which are highlighted by a small purple rectangle as shown in the bottom-right of the start activity of the game strategy in Fig. 3.4. The interpreter further on offers debugging features like step-by-step execution and shows the current status of the execution context as shown in the debugger window right of the game strategy.

### 3.5. Domain Preparation

As mentioned before, a game strategy may be tested in the jABC4 environment with a simple *starter graph* (cf. top-level hierarchy in Fig. 3.4) which instantiates the strategy-processes and passes them to the higher-order process  $\text{GStartCR}$ , provided by technical experts. For designing new game strategies there exists a graph interface  $\text{GGameStrategy}$  (cf. Fig. 3.8) that has as input parameters the board, the current player, and the coordinates of the cell to be evaluated. It declares one output branch “success” with a single return parameter “evaluation” expecting an integer.

The pupils are able to create a new game strategy as well as a starter graph using ready-made templates further easing their creation. For a game strategy, e.g., the input- and output SIBs suiting the graph interface  $\text{GGameStrategy}$  are already instantiated and the input and output parameter are connected to corresponding context variables. In addition, a  $\text{GPutToContext}$  activity initializes the cell evaluation with the value 0. This is already a valid game strategy, which can directly be used in a starter graph. Since it evaluates every cell with 0, the graph  $\text{GStartCR}$  would choose a cell completely random.

Hence the pupils refine an always executable game strategy, starting with a strategy that chooses a cell randomly, to make more sophisticated cell evaluations. For this purpose the domain is prepared with business activities tailored to the task of creating strategies as described in the following.

### 3.5.1. SLG Libraries

ChainReaction offers a Java API to access the current board situation via the class **CField** and to simulate triggering actions in the game. Technical experts, or more precisely domain experts of the game, used the corresponding methods as atomic activities to create service graphs encapsulating a task like identifying whether a given cell is (cf. activities in Fig. 3.4):

- an edge, corner, or center cell (cf. activity “type of cell?”),
- critical, i.e., the cell will explode, if one more atom is added (cf. activity “is critical?”), or
- endangered, i.e., a neighbor cell of the opponent is critical (cf. activity “is endangered?”).

The resulting SLGs may be offered as SIBs to application experts, enabling them to model game strategies.

#### Challenge 6: Dependency Management

The enhanced library concept in the HOPE approach involved the consideration of managing dependencies of a jABC4-project. Therefore Apache Maven<sup>4</sup> support has been added, so that if a corresponding configuration file is in the root folder of a project, the dependencies defined there will be added to the class path of the project. The different SIB browsers may then rely on this enhanced class path for finding services (i.e., Java methods) and SLGs<sup>5</sup>. The latter may be delivered as SLG libraries each library bundled in a standard Java archive via maven artifacts. The maven files can be packaged along with jABC4 project properties for a given domain, so that the user is shielded from the complexity of doing the configuration on his own.

For simple tasks like adding a bonus to the cell evaluation (or subtract a malus) existing SLGs performing arithmetic operations may be used. But if a modeler uses these SIBs as activities, the complete communication with the execution context has to be defined manually. Moreover, an activity that has to be manually configured to take a constant value, add it to the context variable “evaluation”, and store the result back into the context variable, is not as easy to use as an activity **GAddBonus** where the modeler simply adjusts the bonus as needed. The latter uses the same graph **GAdd**, but its affiliation to a library for game strategies as well as its name, package, and icon are domain-specific, the first argument (of the addition) is set to the static value 1, the second argument is preconfigured to the current evaluation,

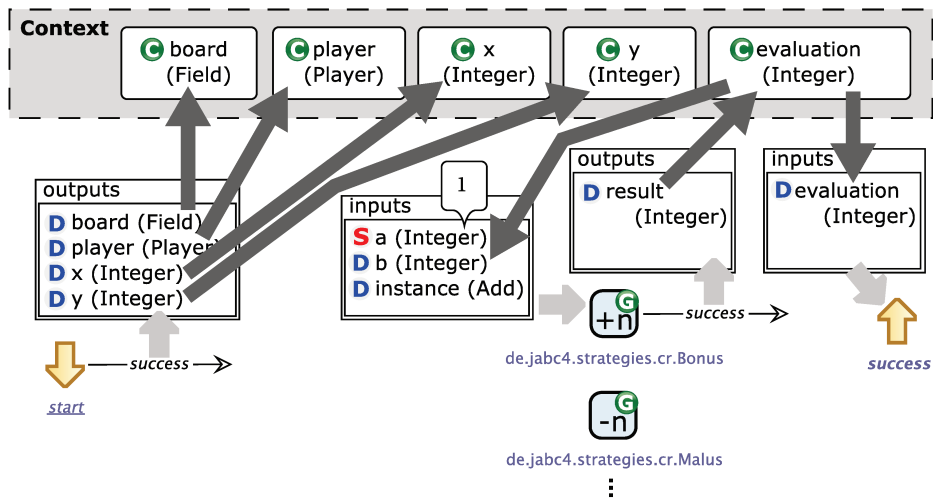


Figure 3.10.: Example for an interface graph enhanced with preconfigured activities for the ChainReaction scenario. Source: publication VII.

and the sum is analogously preset to be stored to the variable “evaluation”. Thus, preconfiguration of activities provides a remedy to support domain-specific IO SIBs as described in Sec. 3.5.2.

### 3.5.2. IO SIB Libraries

Supporting domain-specific business activities enables an application expert to think in the terms of his or her application domain, as they provide the corresponding *domain-specific language* (DSL) [MHS05]. This is not a new pattern, but XMDD and HOPE are the only approaches with true support for it [DS12]. Business activities are distinguished from technical or scripting activities in that they are organized in libraries containing a set of templates (i.e. the SIBs) for executable, easy-to-use activities tailored to a given domain regarding their name, taxonomy, configuration, level of expertise, and competence [DS12]. This implies that the components of such a library are easy to track, and the parameterization confines to the bare necessities. In the spirit of service-orientation it should furthermore be possible to reuse components selectively from different libraries and arrange them in a new tailored library for a given domain.

In the HOPE approach, libraries of business activities (i.e. IO SIBs) are represented by so called *IO SIB libraries*. Such a library is realized via a *configuration graph* (cf. publication VII). Each IO SIB is declared by an activity, the *preconfigured activities*, in the graph be it an atomic or abstraction activity. All preconfigured activities of a configuration graph (shaping an IO SIB library) are presented in an *IO SIB browser* in jABC4 (cf. Sec. 3.3). The IO SIBs serve as templates for new activities being instantiated via drag&drop to the graph canvas. All information is copied from a configuration activity to a newly created instance activity. This comprises settings like:

- package and name,

- icon,
- data-dependencies,
- whether an input parameter is *dynamic* (i.e. reads from a context variable) or *static* (i.e. a constant),
- if the input parameter is set as a constant: its value, and
- the type argument for (graph) type parameters.

All these preconfigured information can be changed in the respective instance activity without affecting the template activity lateron.

Regarding the ChainReaction scenario, a domain expert enhanced the interface graph **GGameStrategy** via additional activities, i.e. instantiations of arbitrary libraries as, e.g., the aforementioned basic SLGs as well as the domain-specific SLGs described in Sec. 3.5.1. This hybrid form is called *configuration interface graph*. A resource browser (i.e. an IO SIB browser) for this library is depicted in Fig. 3.9 and an excerpt of the corresponding configuration interface graph is shown in Fig. 3.10.

The IO SIB with the simple name “Bonus”, e.g. is created by preconfiguring an activity that adds two integers and stores the result in the context. The preconfigured activity may be dragged from the browser and dropped on to the graph canvas. As preconfigured in **GGameStrategy**, the first input parameter will be static and set to a bonus of 1. The second argument reads from the context variable “evaluation”. If the variable does not exist yet it will be created and the type is automatically set to **GInteger**. It will be reused if another activity of the library, that reads from or writes to the respective variable, is instantiated in the current graph. Analogously, the output parameter “result” is associated to variable “evaluation”. The modeler does not have to set anything, but adjust the bonus appropriately.

Other IO SIBs in the ChainReaction library (cf. Fig. 3.9) like “Endangered”, “Critical”, or “TypeOfCell” do not preconfigure activities from a basic library for arithmetic operations, but from the SLG library described in Sec. 3.5.1. The underlying SLGs have input parameters for the board, the current player, and the coordinates of the current cell, in order to retrieve the corresponding information. These data-dependencies are preconfigured in **GGameStrategy** (indicated by the dots in Fig. 3.10). Thus there is no need at all to configure these activities, they can just be dropped to the graph canvas and connected regarding the control-flow. Hence, they work out-of-the-box.

## 3.6. Codegenerator

In the jABC3 framework, full-code generation of SLGs is supported via the Genesys plugin [JMS08]. Genesys produces code for several target languages and platforms and has different kinds of generators that:

- use the interpreter of the jABC to execute SLGs,
- generate the control-flow directly into structured code, or

- into an adjacency data-structure, which is traced at runtime.

In this thesis a new codegenerator for jABC4 (cf. publication VI) is introduced, a consequent evolution of the Genesys codegenerators. The new generator uses additional information available in jABC4 SLGs, i.e.:

- type information for context variables and business activities,
- input/output parametrization of SLGs,
- second-order context facilitating to move services and (sub-) models around *just like data* (cf. publications III and V),
- graph interfaces for SLGs enabling to decide safely at runtime which implementation of an SLG is to be executed, and
- access to external SLG libraries supporting reuse of generated SLGs via *separate compilation* [LF79].<sup>6</sup>

With this information, type-safe Java code is generated from an SLG independently from the framework. A service graph is represented by a Java class and an interface graph by a Java interface. In the following, the code generator used for publication VI will be sketched along the ChainReaction scenario, using an adjacency data-structure to represent the control-flow.

#### 3.6.1. Generating the Graph Interface for Game Strategies

A Java interface generated from a graph interface is resemblant to a *functional interface* [War14], a design-pattern for defining a function signature. The resulting input/output parameterization is used for function objects (e.g. anonymous functions like  $\lambda$ -expressions [Ste90]) that may be passed as arguments to methods and therefore introduce higher-order functions in object-oriented languages [Kü95]. Functional interfaces declare a single method, representing the function as well as its formal parameters and return type.

A *generated graph interface* similarly declares a method named `execute()` dedicated to invoke the underlying process. In the top-left of Fig. 3.11, a class diagram for the Java interface `GameStrategy` representing the equally named interface graph is shown. The parameter list of the method `execute()` reflects the input parameters defined in the input SIB of the SLG, i.e. the *board*, *player*, and the *x* and *y* coordinates of the current cell (cf. Fig. 3.8). The return type of `execute()`-methods for all SLGs is `String` as it reflects the branch to follow after the execution, a concept not available in Java.

Unlike functional interfaces, a generated graph interface further on declares one method per output SIB (declaring an output branch) that is named by convention `get<BranchName>Result()`,<sup>7</sup> representing the output for the returned branch.

<sup>6</sup>Separate compilation is a programming language feature that allows to compile dependent language entities (e.g. classes and interfaces where one invokes methods on the other) independently.

<sup>7</sup>If necessary, the name of the branch will be transformed to a valid Java identifier (e.g. space characters are replaced by underscores).

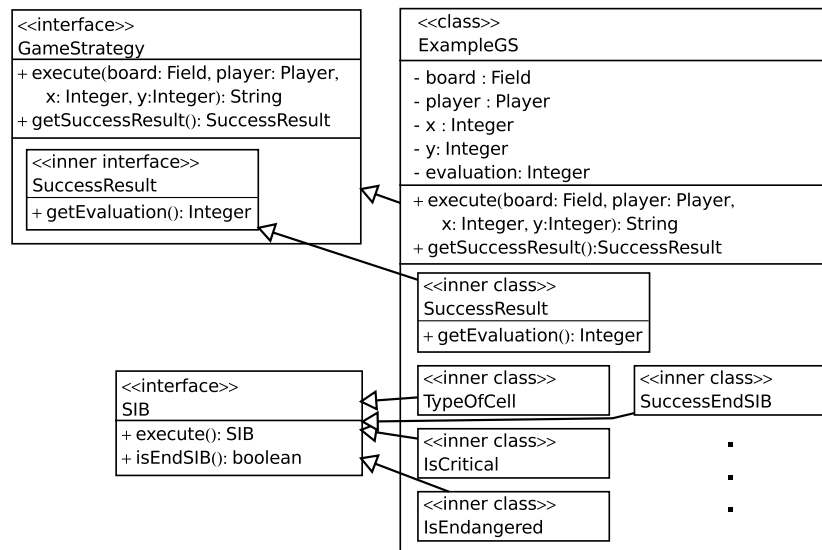


Figure 3.11.: Class diagram of the generated graph interface for game strategies and the generated graph implementation for the example game strategy shown in Fig. 3.3

Ⓒ **GameStrategy** declares only one branch named “success”. So, the respective method is named `getSuccessResult()`. A branch may declare arbitrary many output parameters, but Java allows for methods a single return type, only. Hence, the return type is realized via an inner interface, with one “getter”-method per output parameter. For graph Ⓒ **GameStrategy**, this is Ⓓ **SuccessResult**, with the method `getEvaluation()` returning an integer.

### 3.6.2. Generating the Graph Implementation for a Game Strategy

The graph implementations of the interface graph Ⓒ **GameStrategy** are each generated to a Java class that implements the interface as shown on the right of Fig. 3.11 for graph Ⓒ **ExampleGS** (cf. Fig. 3.3).

The local variables of the class are the context variables, whilst the activities are realized via inner classes named *SIB container* implementing the interface Ⓓ **SIB**. The method `execute()` of the class Ⓒ **ExampleGS** stores the input arguments (i.e. the board, player, and x and y coordinate) to the corresponding local variables as defined in the model and executes the first activity after the input SIB via its SIB container.

The method `execute()` of a SIB container invokes either a Java method in case of an atomic activity or another generated SLG in case of an abstraction activity. Inner classes have access to the members of their surrounding class, so that the method `execute()` has access to the context variables for reading the process or service instance, reading input parameters, and writing the output parameters of an activity. The value of an input parameter declared as *static* is directly generated into the method. Moreover, a SIB container evaluates the branch to follow after the

execution of an activity, retrieves the successor SIB container from the adjacency list (not shown in the class diagram), and returns it.

The activities are executed one after another in a loop until an end activity (i.e. an output SIB) is reached, and the corresponding branch name is returned. For graph **GExampleGS** the branch is named “success” and the method `getSuccessResult()` returns an instance of the inner class **CSuccessResult**, implementing the corresponding, equally named inner interface. Via the method `getEvaluation()` it accesses and returns the current value of the context variable `evaluation`.

### 3.6.3. Embedding Generated Game Strategies into ChainReaction

Both the generated interface graph and the example game strategy **CExampleGS** are independent from the jABC4 framework. The next step is to integrate the pupils’ game strategies into the game ChainReaction outside of jABC4, i.e. without a starter graph and the bridging graph **GStartCR** (cf. Sec. 3.4). A game strategy in ChainReaction represents a computer opponent and is called an *artificial intelligence* (AI). An AI has to implement the interface shown in Listing. 3.1:

---

```
1 public interface AI {
2     public void doMove();
3     // ...
4 }
```

---

Listing 3.1: The interface for ChainReaction AIs

The method `doMove()` is called each time it is the AI’s turn. An excerpt of a generic implementation wrapping an arbitrary generated game strategy implementing the interface **IGameStrategy** is shown in Listing. 3.2:

---

```
1 public void doMove() {
2     EvalField evalResults = new EvalField(game.getField());
3     for (Cell cell: game.getField()) {
4         if (isPlacementPossible(cell, game.getCurrentPlayer())) {
5             // Execute the generated game strategy
6             result = gameStrategy.execute(
7                 copyField(game.getField()), cell.getX(), cell.getY(),
8                 game.getCurrentPlayer());
9             if (result.equals("success")) {
10                evalResults.setValueAt(cell,
11                    gameStrategy.getSuccessResult().getEvaluation());
12            }
13        }
14    }
15    game.selectMove(chooseBestCell(evalResults));
16 }
```

---

Listing 3.2: A generic wrapper for generated jABC4 game strategies

It iterates through the cells of the game board and if the current player is allowed to place an atom in the cell, in ll. 6-8 the generated game strategy (a member of the wrapper class) will be executed. Afterwards, the resulting branch is evaluated, the evaluation value of the game strategy is retrieved, and cached to the evaluation



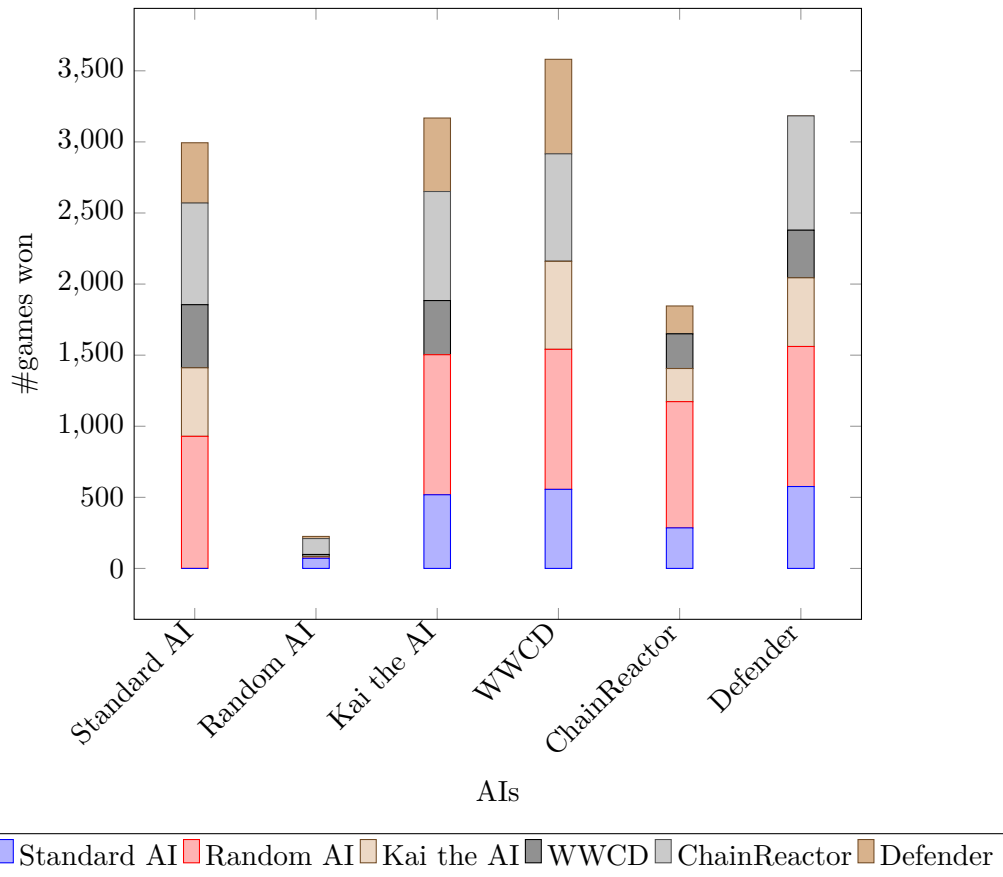


Figure 3.12.: Statistics for the automatic tournament at the end of the project week

results (cf. ll. 10f.). Finally, as the iteration is finished, one of the cells with the highest rating is chosen randomly (cf. l. 15).

Such an AI can directly be used as a computer opponent in ChainReaction and the pupils organized a small tournament at the end of the project week, where they competed against their own strategies. Furthermore, the wrapped game strategies have been assessed via a small tournament application, that conducts a given amount of games, where the AIs play against each other. The results (i.e. which AI has won/lost against which other AI) are presented live in stacked bar charts which “grow” during the tournament, in order to arouse the pupils’ suspense.

Fig. 3.12 shows the statistics for the automatic tournament. For reference, a standard strategy manually implemented in Java and a random strategy have partaken. Every stacked bar shows how many games the corresponding AI has won against the other AIs. Apparently, the AI named “WWCD” of the pupils has beaten all the others including our reference implementation, although the pure modeling part in the project week adds up to a few hours, only.



## 4. Projects

The HOPE approach has been applied in several projects as well as Bachelor and Master theses for very different environments and matters. The following sections sketch the respective scenarios and the impact of applying HOPE. For the pupils project ChainReaction please refer to Sec. 2.2.

### 4.1. Risk-Based Testing via Active Continuous Quality Control

Active Continuous Quality Control (ACQC) [WNS<sup>+</sup>13] employs active automata learning technology [Ang87, SHM11, KV94, HSM11] to automatically maintain test models along the whole life-cycle of an application. The approach has been enhanced to involve risk analysts (cf. publication VI) to prioritize critical aspects of a *system under test* (SUT) tailoring ACQC's model extraction to support risk-based testing [FHBM12]. Risk analysts have been provided with an abstract modeling level tailored to design test components (i.e. *concrete* learning symbols), that encompass data-flow constraints reflecting a given risk profile. Technically, HOPE has been applied for both abstracting from calls to a concrete implementation of the SUT, in terms of *system migration*, and pure *functional evolution* and modeling of data dependencies between the learning symbols. Via the jABC4 codegenerator, both have been generated to code: the alphabet models and a transformation process model. The latter takes a sequence of *abstract* learning symbols (i.e. the control-flow of a test case provided by the learning algorithm) and a generated alphabet model (providing data-flow information and concretizing abstract learning symbols) as inputs and transforms them to an executable test case model.

The transformed models may be seamlessly integrated in the ACQC setup, which uses the LearnLib framework [RSBM09, MSHM11] for automata-learning, as the generated code from the jABC4 codegenerator is flexibly applicable in distinct environments. This was already sufficient to steer the ACQC process in a fashion that increases the risk coverage while at the same time the testing effort is radically reduced. The approach has successfully been applied to several case studies with Springer's Online Conference Service (OCS) illustrating the impact of combining risk prioritization via HOPE and ACQC, employing risk-based regression testing tailored for system migration and for pure functional evolution.

### 4.2. Learning-based cross-platform conformance testing

Learning-based cross-platform conformance testing (LCCT) is an approach specifically designed to validate successful system migration. HOPE has been employed to

combine adequate user-level system abstraction via process-models and the higher-order integration of executable test-blocks with learning-based automatic model inference and comparison [SHM11, MSHM11, BGJ<sup>+</sup>05].

An API layer abstracting from different implementations of the system under test (SUT) has been introduced and used in the process models via dynamic service binding (cf. publication I). The respective implementation is provided to the jABC4 processes via dependency injection at runtime. The impact of LCCT has been illustrated along the migration of Springer’s Online Conference Service (OCS) from a browser-based implementation to using a RESTful web service API [Mar12].

### 4.3. Dynamic Web Application

In a nutshell, with HOPE data-orientation and type awareness accrued to process modeling and with the *dynamic web application* (DyWA) the corresponding domain model in terms of types and their associations is supplied. This helps to narrow the semantic gap even more, as the application experts are now involved in domain modeling, too.

The DyWA [Fro13] is a prototype-driven approach to the development of process-oriented web applications. Key to this approach is to combine business process modeling with DyWA a web-based prototype, that accompanies the development right from the beginning. The DyWA offers a new, simple definition facility integrated into its web interface for application domains in terms of type schemata. It captures the data types and their associations. Based on the defined types and corresponding CRUD operations, a generator automatically creates corresponding Java classes. These may directly be used in jABC4 facilitating the dynamic service binding of HOPE.

Application experts are thus able to model data according to their knowledge and understanding, and act upon this data in easy to compose business processes in jABC4, which are directly executable within DyWA. Hence, jABC4 and DyWA complement each other. This way of proceeding bootstraps the process design from the (much simpler) understanding of the data structures and their relations, speeding up the creation of running prototypes, and making this creation accessible also to application experts.

As every step is automated via a corresponding code generator, no manual programming is required. This opens the whole development process, including the domain modeling, to the application expert, who can validate and check the design at any time by ‘playing’ with the executable prototype.

### 4.4. Model-driven Reengineering of the Business Logic of Java applications

This Master thesis [Tom13] focuses on the semiautomatic generation of executable process models from existing source code. A user chooses the program parts and the import granularity. The transformation from structures of the programming

language to SLGs is done automatically. Every construct, e.g. a method call, an if-then clause, or an arithmetic expression, is translated to activities in the model. The edges represent the control-flow and the data-flow (i.e. the usage and assignment of variables) is realized via the execution context.

The approach benefits from the ability to dynamically bind method calls directly to activities in HOPE, enabling a canonical mapping between source code and process models. Moreover, using HOPE the data-flow can straightforwardly be translated to the type-safe context variables and their read/write operations regarding the activities. For structures in Java, that do not have a direct counterpart in jABC4 (like arithmetic expressions), a small library of methods has been developed and considered during the transformation. This is reminiscent of the language Scala [OSV10] where every operator is realized as a method.

In the course of the Master thesis a prototypic implementation named *Code Importer* has been created and applied in a case study to an application for quality management in short-range transit. The resulting SLGs were comprehensible for non-technical employees in the cooperating company, directly executable, and could be generated back to code via the jABC4 codegenerator (cf. Sec. 3.6). A comprehensive test-suite has been applied successfully to the reengineered application.

## 4.5. Property-Driven Benchmark Generation

In [SIN<sup>+</sup>13] a systematic approach is introduced for automatic generation of platform-independent benchmarks. The generator is adjustable in complexity in order to evaluate verification tools for reactive systems. In essence, a tool chain transforms a set of automatically generated *linear-time logics* (LTL) properties into source code, in several property-preserving steps comprising LTL synthesis, model checking, property-oriented expansion, path condition extraction, theorem proving, SAT solving, and code motion. The approach supports various formats, platforms, and competition scenarios. The jABC4 has been used in [Ges13] to graphically model a higher-order process defining the general workflow of a benchmark generator, which may then be configured with property preserving transformation processes in a service-oriented fashion.

Different communities should be addressable via a growing set of programming languages, tailored sets of programming constructs, and different notions of observation that may be integrated seamlessly into the process models. Since jABC4 addresses non-programmers, it is envisaged that the community will develop their own benchmark generators.

## 4.6. Java and Scala Codegenerator

In the course of this thesis (cf. Sec. 3.6 and publications VI and VII), codegenerators following the full-code generation principle [Jö13] have been developed for the target languages Java [Blo08] and Scala [OSV10]. These generators have been implemented as jABC4 process models that are executed in a first step via the jABC interpreter to generate themselves to code in a bootstrapping process. Lateron the generators

are executable Java or Scala classes that can be used in a second step to generate arbitrary jABC4 SLGs to code.

The jABC4 dependency injection feature and its runtime variability capabilities enabled to implement the codegenerators as a product-line, since the generators for different target languages have a lot in common. For instance, traversing the jABC4 SLG that serves as source for the generation as well as retrieving the necessary information from activities and context variables, is independent from the target language. Therefore, both generators base on a common SLG library. At several points in the general processes, output for the given target language has to be created. These are realized via abstraction activities representing a corresponding interface graph taking the extracted information necessary to produce the respective code snippet as input. These activities are marked as *injected*. Each code generator provides implementations (i.e. service graphs) for the graph interfaces defined in the common library, producing code for the respective target language. At runtime, the general SLG for generating an SLG to code is executed and at every *injection point* the corresponding graph implementation of the selected codegenerator is executed.

Adding new codegenerators for other programming languages or environments requires just to create a new product by implementing the language specific process models. Since the SLG for starting a code generation is in the common SLG library, even the API for using the new codegenerator stays the same.

### 4.7. Capturing and Processing of Biomedical Data

DyWA and jABC4 have been applied for capturing and processing biomedical data in the context of the PROBRAL project no. 54388776 of the German Academic Exchange Service (DAAD) in cooperation with the cancer metabolism research group in the institute of biomedical sciences of university of Saõ Paulo in Brazil, the chair for information systems and management of the Orfalea college of business in Cal Poly – San Luis Obispo, the German institute of human nutrition (DIfE) in Potsdam Germany, and the chair service and software engineering in the Universität Potsdam in Germany. The project is concerned with the phenomenon of systemic inflammation in cachectic cancer patients.

The combination of these tools bringing together the dynamic and easy-to-use data modeling capabilities of the DyWA and the seamless integration of this domain into jABC4's flexible process modeling environment help to involve the experts in biomedicine in both

1. to create and refine a domain model and
2. to operate on the data structures in process models.

This project has many participants (mainly non-technical people), and more and more institutes will partake in this joint venture in the near future, so that it is (and will be) predestined as a case study for HOPE.

## 5. Conclusion and Future Work

This thesis tackles the growing requirements on (business) process modeling in terms of supporting system evolution beyond the state-of-the-art of design-time variability like product-lining and variability modeling with the flexibility to safely add new functionality at runtime. Key to this intent is to enhance the in essence *control-oriented* XMDD paradigm with *data-orientation* and to adapt well-known paradigms from programming languages in a *simplicity-first* fashion [MS10, MS11], ranging from features of object-orientation [KA90] to functional programming [Ses12] (cf. publication VII). The new HOPE approach is type-aware, so that type-safety can be validated at design-time. But the central achievement is the introduction of higher-order semantics by treating services and processes as *first-class citizens*. They may be moved around *just like data* and *plugged and played* into activities at runtime, thus enabling higher-order process engineering. This leads to comprehensible and concise models still manageable for application experts which are in general non-programmers.

Moreover, a complex network of participants in the development process arises from the increasing globalization of processes and systems, e.g. end-to-end-processes naturally comprise various management levels. This trend potentially introduces a semantic gap on every ‘level’, since the technical expert on one level is at the same time the application expert on the next lower level – in terms of getting more technical – and so forth. This has been tackled via hierarchical modeling in combination with the deployment of SLG libraries serving as domain-specific business activities in a service-oriented fashion. Preconfiguration of activities further helps to guard the modeler from complexity of data-flow handling as well as doing repetitive work.

In several projects it has been shown (cf. Chapter 4) that the fields of application for process modeling are not constrained to describing high-level workflows of business applications and scientific workflows anymore, as it has been employed in areas ranging from reverse engineering, over modeling of dynamic data bases, benchmark generation, automata learning, to modeling game strategies.

This is still just the tip of the iceberg since adding more and more *tamed* programming language features, as well as components and services for a great deal of domains has the potential to involve the application experts into the whole (development) life-cycle of any application. The according future work can be categorized into enhancements to the modeling environment, the (modeling) language, and the ecosystem:

**Modeling Environment** The current implementation of jABC4 is prototypically realized on top of the jABC3 with its powerful plugin concept, which made extending the framework very comfortable (cf. publications IV and VII). But jABC3 is an

in-house development, so that it takes a lot of strength to maintain. Therefore, the new CINCO SCCE Meta Tooling Suite basing on the eclipse framework [MLA10] is currently in development at the chair for programming systems at TU Dortmund. The new concepts facilitate the meta-modeling capabilities of the *eclipse modeling framework* (EMF) [SBPM08, Gro08] to support a user in describing a modeling environment for arbitrary model types like petri nets, class diagrams, or SLGs. Tailored view and editor components are generated from the descriptions. These components integrate into the eclipse ecosystem and hence are allowed to benefit from existent features like version control and dependency management support as well as the community that impels the development of the platform. Hence, a migration from the jABC3-based HOPE framework to a solution in the Eclipse environment is a natural step.

Besides a complete technological shift to another platform, some enhancements to the development environment itself might increase the acceptance by application experts. In the context of the ChainReaction project, e.g., a simple jABC plugin for creating preconfigured SLGs from templates has been developed. This simplified the process of creating new process models for the pupils immensely. Hence, preparing and offering templates of SLGs, jABC projects (including the configuration of the necessary dependencies for a domain), wizards for common tasks, and resource browsers (cf. Sec. 3.1) for a specific domain should be an integral feature of jABC4.

Further on, the HOPE approach structures the development process into several layers (cf. Sec. 3.2) taking into account that the development of big software projects entails a complex network of participants with different technical expertise and competence. Accordingly, the different roles of participants should be considered to create tailored views on the entities of the development process. In the spirit of the *one thing approach* (OTA) [SM08] the models should contain all the information to generate the corresponding application, but every participant should only see the information that is necessary for his task and that he or she can handle.

Validation and verification of SLGs has always been an essential part of the XMDD approach [SMCB96, RSM08]. The jABC4 approach adds new information to SLGs like the input output parameterization and type-awareness. There are already some basic checks using the facilities of the Java type system to validate type-safety as well as employing model checking for some data-flow analysis as proposed in [Ste91, Ste93, SCK<sup>+</sup>95, LMS06]. But the potential for adding (and checking) more sophisticated and domain-specific constraints is not fully exploited, in particular since the dynamic service binding via computational reflection enables to extract information from the underlying services like validation information (e.g. *bean validation* (BV)<sup>1</sup> constraints like the `@NotNull` annotation).

Moreover, jABC4 already has some type inference capabilities for the type parameters of activities (cf. Challenge 5) and prefiltering of types when choosing a type for a context variable or a parameter (cf. publication VII). Where possible, this should be enhanced to fully infer the types of context variables and parameters, in order to release the modeler from the task to edit types, like in Scala [OSV10].

---

<sup>1</sup><http://beanvalidation.org/1.1/>



---

**Language** The HOPE approach offers sophisticated type-aware interface descriptions for graphs with named input parameters, branches, and output parameters. These are declared via input and output SIBs. But currently exceptional branches have to be defined similar to the normal branches. This leads either to starlike patterned SLGs (with a high amount of edges), if nearly every activity has some exceptional branches that have to be connected to a corresponding end activity, or the exceptions are ignored and – following the current default behavior – thrown up to the highest hierarchy-level to be handled in the calling logic. For atomic activities (i.e. activities representing a method invocation) the exceptions explicitly thrown of a method are automatically translated to branches and for all other cases two general branches handling exceptions and errors respectively are added. Similarly, a better default behavior for exceptions and implicitly added branches for exception handling on every hierarchy-level should be accrued.

A context variable may be marked as *injected*, so that at runtime an implementation is provided. On the one hand it should be possible to add some information regarding the requirements on the expected resource if the type itself is not sufficient. For instance, some library families, as e.g. the *Java API for XML processing* (JAXP)<sup>2</sup>, have different implementations with varying feature sets. At runtime several implementations may be available. Thus, when loading a component, the required features have to be requested, so that an implementation that fulfills the requirements can be chosen and provided by the container. Moreover, although the selection of resources is a concern orthogonal to the business-logic (cf. Sec. 3.2), it may be beneficial to be able to define the corresponding discovery logic in a separate process model. This approach bases on the concept of *producer methods*, e.g. in the *context and dependency injection* (CDI)<sup>3</sup> specification, which allows an implementer to provide injection logic for resources.

Further orthogonal concerns like access control, parameter validation, transaction handling, monitoring, and logging should not be mixed with the business logic. Consequently, *aspect oriented programming* (AOP) [KLM<sup>+</sup>97] should be integrated into the HOPE approach, too. An activity could be marked as *intercepted* with a corresponding *interceptor graph*. At runtime the interceptor graph is executed before the activity is invoked. At some point the intercepted activity may be invoked or it is not executed at all, e.g. if the caller does not have access privileges. If the intercepted activity has been called, the interceptor graph is able to evaluate the outcome of the execution afterwards, e.g., in order to commit or rollback a transaction.

**Ecosystem** Right now, there exists exactly one Java and one Scala generator. Both use an adjacency data-structure for the control-flow, that is mutable in order to enable runtime adaptations of the control-flow (cf. publication VI). Further on, each process instance – no matter whether it is injected or not – is retrieved via a CDI container, because only managed instances (of the container) can have injected resources. These features cost a lot of performance, but are not necessary for all SLG libraries. Therefore it should be configurable, whether the graphs of a library

---

<sup>2</sup><https://jaxp.java.net>

<sup>3</sup>JSR-299: <https://jcp.org/en/jsr/detail?id=299>

can be generated with structured, fixed code or need the flexibility to be adjustable at runtime and whether there is a need for dependency injection or other container managed features like the aforementioned interceptors graphs. This way, basic SLG libraries can be generated to highly optimized code and the domain-specific high-level SLGs offer the flexibility and a rich feature set (including DI and AOP).

The combination of the HOPE approach with DyWA has turned out to be very effective as it narrows the semantic gap between application experts and technical experts in two dimensions: control-flow and data-flow. Currently, the integration of the tools is realized conveniently via generators, but the DyWA is a web application and the jABC4 a desktop application. Thus, a modeler has to switch between tools. As the web is a very promising platform (even for development environments), a web version of the jABC4 should be developed in the spirit of the jABC3 WebABC [MS13] and it should be aimed at a seamless integration with the DyWA. At the time of writing, several Bachelor theses attend to the realization of the different facets of this objective.

In order to enter industrial practice on a larger scale, the HOPE approach needs a profound set of SLG libraries for wide-ranging domains and APIs (like Microsoft Office or leading *enterprise resource planning* (ERP) systems), including enhanced support for long-running processes, event-handling, and concurrency. At present, a Bachelor thesis works accordingly on migrating the existing extensive SIB libraries of the jABC3 framework to jABC4.

# Bibliography

- [AAA<sup>+</sup>07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C.K. Liu, R. Khalaf, Dieter Koenig, M. Marin, V. Mehta, S. Thatte, D. Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>, 2007.
- [ACN02] J. Aldrich, Craig Chambers, and D. Notkin. Archjava: connecting software architecture to implementation. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 187–197. IEEE, May 2002.
- [Act12] Activiti Team. Activiti BPM Platform, 2012. <http://www.activiti.org/>.
- [AD01] Don Anderson and Dave Dzatko. *Universal Serial Bus System Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001.
- [AK09] Oana Andrei and Hélène Kirchner. A higher-order graph calculus for autonomic computing. In Marina Lipshteyn, Vadim E. Levit, and Ross M. McConnell, editors, *Graph Theory, Computational Intelligence and Thought*, pages 15–26. Springer-Verlag, Berlin, Heidelberg, 2009.
- [All09] T. Allweyer. *BPMN 2.0-Business Process Model and Notation*. Bod, 2009.
- [Ang87] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
- [AP05] Danilo Ardagna and Barbara Pernici. Global and local qos guarantee in web service selection. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, volume 3812, pages 32–46. Springer, 2005.
- [AP07] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, 33(6):369–384, 2007.
- [BA04] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

- [Bar84] Hendrik Pieter Barendregt. *The lambda calculus*, volume 3. North-Holland Amsterdam, 1984.
- [BCJO12] Joakim Bjørk, Dave Clarke, Einar Broch Johnsen, and Olaf Owe. A type-safe model of adaptive object groups. In *Proceedings of the 11th International Workshop on Foundations of Coordination Languages and Self Adaptation*. arxiv, 2012.
- [BCT06] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3):327–357, sep 2006.
- [BFR07] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Programming self-organizing systems with the higher-order chemical language. *International Journal of Unconventional Computing*, 3(3):161–177, 2007.
- [BGJ<sup>+</sup>05] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the Correspondence Between Conformance Testing and Regular Inference. In Maura Cerioli, editor, *FASE05*, volume 3442 of *LNCS*, pages 175–189. Springer, Apr 2005.
- [Blo08] J. Bloch. *Effective Java*. Java Series. Pearson Education, 2008.
- [BM06] Jorge A. Baier and Sheila A. Mcilraith. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, AAAI'06*, pages 788–795. AAAI Press, 2006.
- [BMRS07] Marco Bakera, Tiziana Margaria, Clemens Renner, and Bernhard Steffen. Verification, Diagnosis and Adaptation: Tool-supported enhancement of the model-driven verification process. In *Revue des Nouvelles Technologies de l'Information (RNTI-SM-1)*, page 85–98. Dec 2007.
- [BMRS09] Marco Bakera, Tiziana Margaria, Clemens Renner, and Bernhard Steffen. Tool-supported enhancement of diagnosis in model-driven verification. *Innovations in Systems and Software Engineering*, 5:211–228, 2009.
- [BSBM05] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible? In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *Technologies for E-Services*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Springer Berlin Heidelberg, 2005.
- [C<sup>+</sup>02] T. Coupaye et al. *The Fractal Composition Framework (Version 1.0)*. The ObjectWeb Consortium, Jul 2002.
- [CAkH05] Daniela Barreiro Claro, Patrick Albers, and Jin kao Hao. Selecting web services for optimal composition. In *IN PROCEEDINGS OF THE 2ND*

- 
- INTERNATIONAL WORKSHOP ON SEMANTIC AND DYNAMIC WEB PROCESSES (SDWP 2005)*, pages 32–45. Springer, 2005.
- [cDL03] Pierre charles David and Thomas Ledoux. Towards a framework for self-adaptive component-based applications. In *In DAIS'03, volume 2893 of LNCS*, pages 1–14. Springer-Verlag, 2003.
- [CDPEV05] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani. Qos-aware replanning of composite web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 121–129 vol.1. IEEE, Jul 2005.
- [De10] Peter Dadam and et.al. From ADEPT to AristaFlow BPM Suite: A Research Vision Has Become Reality. In et. al. Rinderle-Ma, editor, *Business Process Management Workshops*, volume 43 of *LNBIP*, pages 529–531. Springer Berlin Heidelberg, 2010.
- [Dij68] Edsger W. Dijkstra. Letters to the editor: Go to statement considered harmful. *Commun. ACM*, 11(3):147–148, Mar 1968.
- [DS12] Markus Doedt and Bernhard Steffen. An Evaluation of Service Integration Approaches of Business Process Management Systems. In *Proc. of the 35th Annual IEEE Software Engineering Workshop (SEW 2012)*. IEEE, 2012.
- [FF04] Ira R. Forman and Nate Forman. *Java Reflection in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.
- [FHBM12] Michael Felderer, Christian Haisjackl, Ruth Breu, and Johannes Motz. Integrating manual and automatic risk assessment for risk-based testing. In Stefan Biffl, Dietmar Winkler, and Johannes Bergsmann, editors, *Software Quality. Process Automation in Software Development*, volume 94 of *Lecture Notes in Business Information Processing*, pages 159–180. Springer Berlin Heidelberg, 2012.
- [FR14] Michael Felderer and Rudolf Ramler. A multiple case study on risk-based testing in industry. *STTT-RBT*, 2014. Under Review.
- [Fro13] Markus Theo Frohme. Agile Domänenmodellierung für prozessgesteuerte Webanwendungen. Bachelor thesis, TU Dortmund, Feb 2013.
- [Ges13] Maren Geske. Property-specific Generation of Benchmarks for the Validation of Reactive Systems. Bachelor thesis, TU Dortmund, Apr 2013.
- [Gon13] Antonio Goncalves. Context and dependency injection. In *Beginning Java EE 7*, pages 23–66. Apress, 2013.
- [Gro08] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, Boston, MA, USA, 2008.

- [GT02] Paul Gerrard and Neil Thompson. *Risk Based E-Business Testing*. Artech House, Aug 2002.
- [HHNS02] Andreas Hagerer, Hardi Hungar, Oliver Niese, and Bernhard Steffen. Model generation by moderated regular extrapolation. *LNCS*, pages 80–95, 2002.
- [HL95] Walter L Hürsch and Cristina Videira Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, Massachusetts, 1995.
- [HMM<sup>+</sup>08] Martina Hörmann, Tiziana Margaria, Thomas Mender, Ralf Nagel, Bernhard Steffen, and Hong Trinh. The jabc approach to rigorous collaborative development of scm applications. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 724–737. Springer Berlin Heidelberg, 2008.
- [HMN<sup>+</sup>01] Andreas Hagerer, Tiziana Margaria, Oliver Niese, Bernhard Steffen, Georg Brune, and Hans-Dieter Ide. Efficient regression testing of CTI-systems: Testing a complex call-center solution. *Annual review of communication, Int. Engineering Consortium (IEC)*, 55:1033–1040, 2001.
- [HSM11] Falk Howar, Bernhard Steffen, and Maik Merten. Automata Learning with Automated Alphabet Abstraction Refinement. In *Twelfth International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 2011.
- [JLM<sup>+</sup>12] Sven Jörges, Anna-Lena Lamprecht, Tiziana Margaria, Ina Schaefer, and Bernhard Steffen. A Constraint-based Variability Modeling Framework. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(5):511–530, 2012.
- [JMG05] Michael C. Jaeger, Gero Mühl, and Sebastian Golze. Qos-aware composition of web services: An evaluation of selection algorithms. In *Proceedings of the 2005 Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part I, OTM’05*, pages 646–661. Springer-Verlag, Berlin, Heidelberg, 2005.
- [JMS08] Sven Jörges, Tiziana Margaria, and Bernhard Steffen. Genesys: service-oriented construction of property conform code generators. *Innovations in Systems and Software Engineering*, 4(4):361–384, 2008.
- [JS12] Sven Jörges and Bernhard Steffen. Exploiting Ecore’s Reflexivity for Bootstrapping Domain-Specific Code-Generators. In *Proc. of 35th Software Engineering Workshop (SEW 2012)*, pages 72–81. IEEE, 2012.

- 
- [JSM11] Sven Jörges, Bernhard Steffen, and Tiziana Margaria. Building Code Generators with Genesys: A Tutorial Introduction. In JoãoM. Fernandes, Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering III*, volume 6491 of *Lecture Notes in Computer Science*, pages 364–385. Springer Berlin Heidelberg, 2011.
- [Jö13] Sven Jörges. *Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach*, volume 7747 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Germany, 2013.
- [KA90] Setrag Khoshafian and Razmik Abnous. *Object Orientation: Concepts, Languages, Databases, User Interfaces*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [KLM<sup>+</sup>97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP'97 — Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997.
- [KM06] Martin Karusseit and Tiziana Margaria. Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service. *Electronic Notes in Theoretical Computer Science*, 157(2):101 – 118, 2006.
- [KMFS06] Christian Kubczak, Tiziana Margaria, Arno Fritsch, and Bernhard Steffen. Biological LC/MS Preprocessing and Analysis with jABC, jETI and xcms. In *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006): 15-19 November 2006, Paphos, Cyprus*, pages 308–313. IEEE Computer Society, 2006.
- [KMK<sup>+</sup>08] C Kubczak, T Margaria, M Kaiser, J Lemcke, and B Knuth. Abductive Synthesis of the Mediator Scenario with jABC and GEM. In *Semantic Web Services Challenge: Proceedings of the 2008 Workshops*, pages 52–63. 2008.
- [KMSN07] Christian Kubczak, Tiziana Margaria, Bernhard Steffen, and Stefan Naujokat. Service-oriented Mediation with jETI/jABC: Verification and Export. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT Workshop*, pages 144–147. IEEE Computer Society Press, Silicon Valley, California, USA, Nov 2007.
- [KMSN08] Christian Kubczak, Tiziana Margaria, Bernhard Steffen, and Ralf Nagel. *Service-oriented Mediation with jABC/jETI*, pages 71–99. Springer, 2008.

- [KMWS07] Christian Kubczak, Tiziana Margaria, Christian Winkler, and Bernhard Steffen. An approach to Discovery with miAamics and jABC. In *Proc. of 2007 International Conference on Web Intelligence and Intelligent Agent Technology (IEEE/WIC/ACM 2007) - WI-IAT Workshop*, pages 157–160. 2007.
- [KOT13] K. Klai, H. Ochi, and S. Tata. Formal abstraction and compatibility checking of web services. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 163–170. IEEE, June 2013.
- [KSKS09] Yeondae Kwon, Yasumasa Shigemoto, Yoshikazu Kuwana, and Hideaki Sugawara. Web API for biology with a workflow navigation system. *Nucleic Acids Research*, 37(suppl\_2):W11–16, Jul 2009.
- [KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MITP, Cambridge, MA, USA, 1994.
- [Kü95] Thomas Kühne. Higher order objects in pure object-oriented languages. *SIGPLAN OOPS Mess.*, 6(1):1–6, Jan 1995.
- [Lam13] Anna-Lena Lamprecht. *User-Level Workflow Design - A Bioinformatics Perspective*, volume 8311 of *Lecture Notes in Computer Science*. Springer, 2013.
- [LF79] Richard J. LeBlanc and Charles N. Fischer. On implementing separate compilation in block-structured languages. In *Proceedings of the 1979 SIGPLAN Symposium on Compiler Construction*, SIGPLAN '79, pages 139–143. ACM, New York, NY, USA, 1979.
- [LMS06] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. Data-Flow Analysis as Model Checking Within the jABC. In Alan Mycroft and Andreas Zeller, editors, *Compiler Construction*, volume 3923 of *Lecture Notes in Computer Science*, pages 101–104. Springer Berlin Heidelberg, 2006.
- [LMS08a] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. Seven Variations of an Alignment Workflow - An Illustration of Agile Process Design and Management in Bio-jETI. In *Bioinformatics Research and Applications*, volume 4983 of *Lecture Notes in Bioinformatics*, page 445–456. Springer, Atlanta, Georgia, 2008.
- [LMS08b] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. Supporting Process Development in Bio-jETI by Model Checking and Synthesis. In *Semantic Web Applications and Tools for Life Sciences (SWAT4LS 2009)*. *CEUR Workshop Proceedings*, volume 435. CEUR-WS.org, 2008.
- [LMS<sup>+</sup>08c] Anna-Lena Lamprecht, Tiziana Margaria, Bernhard Steffen, Alexander Sczyrba, Sven Hartmeier, and Robert Giegerich. GeneFisher-P: varia-



- 
- tions of GeneFisher as processes in Bio-jETI. *BMC Bioinformatics*, 9 Suppl 4:S13, 2008.
- [LMS09a] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. Bio-jETI: a framework for semantics-based service composition. *BMC Bioinformatics*, 10 Suppl 10:S8, 2009.
- [LMS09b] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. From Bio-jETI Process Models to Native Code. In *14th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2009, Potsdam, Germany, 2-4 June 2009*, pages 95–101. IEEE Computer Society, Jun 2009.
- [LNMS10] Anna-Lena Lamprecht, Stefan Naujokat, Tiziana Margaria, and Bernhard Steffen. Synthesis-Based Loose Programming. In *Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal*, pages 262–267. IEEE, sep 2010.
- [LNMS11] Anna-Lena Lamprecht, Stefan Naujokat, Tiziana Margaria, and Bernhard Steffen. Semantics-based composition of EMBOSS services. *Journal of Biomedical Semantics*, 2(Suppl 1):S5, 2011.
- [LNS13] Anna-Lena Lamprecht, Stefan Naujokat, and Ina Schaefer. Variability Management Beyond Feature Models. *IEEE Computer*, 46(11):48–54, 2013.
- [LZS<sup>+</sup>05] Fangfang Liu, Liang Zhang, Yuliang Shi, Lili Lin, and Baile Shi. Formal analysis of compatibility of web services via ccs. In *Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on*, page 6 pp. IEEE, Aug 2005.
- [Mae87] Pattie Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, Dec 1987.
- [Mar10] Simon et. al. Marlow. Haskell 2010 language report. *Available online <http://www.haskell.org/onlinereport/haskell2010>*, 2010.
- [Mar12] Janina Kim Marks. Vergleich von Präsentations- und Geschäftslogikschicht prozessorientierter Webanwendungen auf Basis regulärer Extrapolation. Bachelor thesis, TU Dortmund, Jul 2012.
- [MBD<sup>+</sup>12] Tiziana Margaria, Steve Boßelmann, Markus Doedt, Barry D. Floyd, and Bernhard Steffen. Customer-Oriented Business Process Management: Visions and Obstacles. In Mike Hinchey and Lorcan Coyle, editors, *Conquering Complexity*, pages 407–429. Springer London, 2012.
- [Men02] D. Menasce. Qos issues in web services. *Internet Computing, IEEE*, 6(6):72–75, Nov 2002.

- [MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, Dec 2005.
- [Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.
- [MKS08] Tiziana Margaria, Christian Kubczak, and Bernhard Steffen. Bio-jeti: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. *BMC Bioinformatics*, 9(S-4), 2008.
- [MLA10] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2nd edition, 2010.
- [MMK<sup>+</sup>09] Tiziana Margaria, Daniel Meyer, Christian Kubczak, Malte Isberner, and Bernhard Steffen. Synthesizing Semantic Web Service Compositions with jMosel and Golog. In *The Semantic Web - ISWC 2009*, volume 5823 of *LNCS*, pages 392–407. Springer Berlin/Heidelberg, 2009.
- [MNS05] Tiziana Margaria, Ralf Nagel, and Bernhard Steffen. jETI: A Tool for Remote Tool Integration. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440/2005 of *LNCS*, page 557–562. Springer Berlin/Heidelberg, 2005.
- [MPT12] Emanuela Merelli, Nicola Paoletti, and Luca Tesi. A multi-level model for self-adaptive systems. In *Proceedings of the 11th International Workshop on Foundations of Coordination Languages and Self Adaptation*. arxiv, 2012.
- [MPW07] D. Martin, M. Paolucci, and M. Wagner. Towards Semantic Annotations of Web Services: OWL-S from the SAWSDL Perspective. In *OWL-S Experiences and Future Developments Workshop at ESWC 2007*. 2007.
- [MR02] Mihhail Matskin and Jinghai Rao. Value-added web services composition using automatic program synthesis. In *Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES 2002*, pages 213–224. Springer, 2002.
- [MS04] Tiziana Margaria and Bernhard Steffen. Lightweight coarse-grained coordination: a scalable system-level approach. *Software Tools for Technology Transfer*, 5(2-3):107–123, 2004.
- [MS05] Tiziana Margaria and Bernhard Steffen. Second-Order Semantic Web. In *Proc. of 29th Annual IEEE/NASA Software Engineering Workshop*, pages 219–227. IEEE Computer Society, Los Alamitos, CA, USA, 2005.
- [MS06] T. Margaria and B. Steffen. Service engineering: Linking business and it. *Computer*, 39(10):45–55, Oct 2006.

- 
- [MS07] Tiziana Margaria and Bernhard Steffen. LTL-Guided Planning: Revisiting Automatic Tool Composition in ETI. In *Proc. of the 31st Annual IEEE / NASA Software Engineering Workshop (SEW 2007)*, Columbia, MD, USA, pages 214–226. IEEE Computer Society, 2007.
- [MS09a] Tiziana Margaria and Bernhard Steffen. Agile IT: Thinking in User-Centric Models. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 490–502. Springer Berlin/Heidelberg, 2009.
- [MS09b] Tiziana Margaria and Bernhard Steffen. Business Process Modelling in the jABC: The One-Thing-Approach. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on Business Process Modeling*. IGI Global, 2009.
- [MS10] Tiziana Margaria and Bernhard Steffen. Simplicity as a Driver for Agile Innovation. *Computer*, 43(6):90–92, 2010.
- [MS11] T. Margaria and B. Steffen. Special session on "simplification through change of perspective". In *Software Engineering Workshop (SEW), 2011 34th IEEE*, pages 67–68. IEEE, Jun 2011.
- [MS12] Tiziana Margaria and Bernhard Steffen. Service-orientation: Conquering complexity with xmdd. In Mike Hinchey and Lorcan Koyle, editors, *Conquering Complexity*. Springer, 2012.
- [MS13] Maik Merten and Bernhard Steffen. Simplicity driven application development. *Journal of Integrated Design and Process Science (SDPS)*, 16, 2013.
- [MSHM11] Maik Merten, Bernhard Steffen, Falk Howar, and Tiziana Margaria. Next Generation LearnLib. In *Proc. of 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011)*, Saarbrücken, Germany, pages 220–223. Springer, 2011.
- [MSKC04] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, Jul 2004.
- [MSR05] Tiziana Margaria, Bernhard Steffen, and Manfred Reitenspieß. Service-Oriented Design: The Roots. In *Proc. of the 3rd Int. Conf. on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands, volume 3826 of *LNCS*, pages 450–464. Springer, 2005.
- [MSS99] Markus Müller-Olm, David Schmidt, and Bernhard Steffen. Model-Checking - A Tutorial Introduction. In *Proceedings of the 6th International Symposium on Static Analysis (SAS '99)*, pages 330–354. Springer, 1999.

- [Neu07] Johannes Neubauer. Localchecker plugin for the jabc. *TU Dortmund, Germany*, 2007. <http://hdl.handle.net/2003/30118>.
- [NLS12] Stefan Naujokat, Anna-Lena Lamprecht, and Bernhard Steffen. Loose Programming with PROPHETS. In Juan de Lara and Andrea Zisman, editors, *Proc. of the 15th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2012), Tallinn, Estonia*, volume 7212 of *LNCS*, pages 94–98. Springer Heidelberg, 2012.
- [NMS13] Johannes Neubauer, Tiziana Margaria, and Bernhard Steffen. Design for Verifiability: The OCS Case Study. In *Formal Methods for Industrial Critical Systems: A Survey of Applications*, chapter 8, pages 153–178. Wiley-IEEE Computer Society Press, Mar 2013.
- [NSB<sup>+</sup>12] Johannes Neubauer, Bernhard Steffen, Oliver Bauer, Stephan Windmüller, Maik Merten, Tiziana Margaria, and Falk Howar. Automated continuous quality assurance. In *Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA), 2012*, pages 37–43. Springer, 2012.
- [NSM<sup>+</sup>01] Oliver Niese, Bernhard Steffen, Tiziana Margaria, Andreas Hagerer, Georg Brune, and Hans-Dieter Ide. Library-based design and consistency checking of system-level industrial test cases. In Heinrich Hussmann, editor, *Fundamental Approaches to Software Engineering*, volume 2029 of *Lecture Notes in Computer Science*, pages 233–248. Springer Berlin/Heidelberg, 2001.
- [NW06] M. Naftalin and P. Wadler. *Java Generics and Collections*. O’Reilly Media, 2006.
- [OMG11] OMG. Business Process Model and Notation (BPMN) Version 2.0, 2011. <http://www.omg.org/spec/BPMN/2.0/>.
- [OSV10] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. Artima Series. Artima, Incorporated, 2010.
- [Pas05] J. Pasley. How BPEL and SOA are changing Web services development. *Internet Computing, IEEE*, 9(3):60–67, 2005.
- [PBH07] J. Pathak, S. Basu, and V. Honavar. On context-specific substitutability of web services. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 192–199. IEEE, Jul 2007.
- [PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [PPBG09] Jérémy Philippe, Noël Palma, Fabienne Boyer, and Olivier Gruber. Self-adapting service level in java enterprise edition. In JeanM. Bacon and BrianF. Cooper, editors, *Middleware 2009*, volume 5896 of *Lecture*

- 
- Notes in Computer Science*, pages 143–162. Springer Berlin Heidelberg, 2009.
- [Red12] RedHat Software - JBoss. jBPM Website, 2012. <http://www.jboss.org/jbpm>.
- [RM06] J.C. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In *The 18th CAiSE. Proceedings of Workshops and Doctoral Consortium*, pages 521–532. Namur University Press, 2006.
- [RSBM09] Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. LearnLib: a framework for extrapolating behavioral models. *International Journal on Software Tools for Technology Transfer (STTT)*, 11(5):393–407, 2009.
- [RSM08] Harald Raffelt, Bernhard Steffen, and Tiziana Margaria. Dynamic Testing Via Automata Learning. In *Proc. of the Haifa Verification Conference 2007 (HVC '07)*, volume 4899 of *LNCS*, pages 136–152. Springer, 2008.
- [SBPM08] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley, Boston, MA, USA, 2008.
- [SCK<sup>+</sup>95] Bernhard Steffen, Andreas Claßen, Marion Klein, Jens Knoop, and Tiziana Margaria. The Fixpoint-Analysis Machine. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 72–87. Springer Berlin Heidelberg, 1995.
- [Ses12] Peter Sestoft. Higher-order functions. In *Programming Language Concepts*, volume 50 of *Undergraduate Topics in Computer Science*, pages 77–91. Springer London, 2012.
- [SHM11] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to Active Automata Learning from a Practical Perspective. In Marco Bernardo and Valérie Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer Berlin Heidelberg, 2011.
- [SIN<sup>+</sup>13] Bernhard Steffen, Malte Isberner, Stefan Naujokat, Tiziana Margaria, and Maren Geske. Property-Driven Benchmark Generation. In *International SPIN Symposium on Model Checking of Software (SPIN2013)*, volume 7976 of *LNCS*, pages 341–357. Springer, 2013.
- [SM99] Bernhard Steffen and Tiziana Margaria. Metaframe in practice: Design of intelligent network services. In *Correct System Design - Correct System Design, Recent Insight and Advances*, volume 1710 of *Lecture Notes in Computer Science*, pages 390–415. Springer, 1999.

- [SM08] B. Steffen and T. Margaria. Business process modelling in the jabc: The one-thing approach. In *Handbook of Research on Business Process Modeling*. IGI Global, 2008.
- [SMB97] Bernhard Steffen, Tiziana Margaria, and Volker Braun. The Electronic Tool Integration platform: concepts and design. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):9–30, 1997.
- [SMBK97] Bernhard Steffen, Tiziana Margaria, Volker Braun, and Nina Kalt. Hierarchical Service Definition. *Annual Review of Communications of the ACM*, 51:847–856, 1997.
- [SMC<sup>+</sup>96] Bernhard Steffen, Tiziana Margaria, Andreas Claßen, Volker Braun, Manfred Reitenspieß, and Helmut Wendler. Service Creation: Formal Verification and Abstract Views, 1996.
- [SMCB96] Bernhard Steffen, Tiziana Margaria, Andreas Claßen, and Volker Braun. Incremental Formalization: A Key to Industrial Success. *Software - Concepts and Tools*, 17(2):78–95, 1996.
- [SMN<sup>+</sup>06] Bernhard Steffen, Tiziana Margaria, Ralf Nagel, Sven Jörges, and Christian Kubczak. *Model-Driven Development with the jABC*, volume 4383 of *LNCS*, pages 92–108. Springer Berlin/Heidelberg, 2006.
- [SN07] Bernhard Steffen and Prakash Narayan. Full Life-Cycle Support for End-to-End Processes. *IEEE Computer*, 40(11):64–73, 2007.
- [SS06] August-Wilhelm Scheer and Kristof Schneider. Aris — architecture of integrated information systems. In Peter Bernus, Kai Mertins, and Günter Schmidt, editors, *Handbook on Architectures of Information Systems*, pages 605–623. Springer Berlin Heidelberg, 2006. 10.1007/3-540-26661-5\_25.
- [STA05] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. *Process Modeling using Event-Driven Process Chains*, pages 119–145. John Wiley & Sons, Inc., 2005.
- [Ste90] G.L. Steele. *Lambda-Expressions*. HP Technologies Series. Digital Press, 1990.
- [Ste91] Bernhard Steffen. Data Flow Analysis as Model Checking. In *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, page 346–365. Springer-Verlag, 1991.
- [Ste93] Bernhard Steffen. Generating Data Flow Analysis Algorithms from Modal Specifications. *Science of Computer Programming*, 21(2):115–139, 1993.
- [Tho93] Bent Thomsen. Plain chocs: A second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59, 1993.

- 
- [Tom13] Vera Tomskikh. Modellgetriebenes Reengineering der Geschäftslogik von Java-Applikationen. Master thesis, TU Dortmund, Dec 2013.
- [Ull94] J.D. Ullman. *Elements of ML Programming*. An Alan R. Apt book. Prentice Hall, 1994.
- [VdLSR07] F. Van der Linden, K. Schmid, and E. Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer Berlin Heidelberg, Germany, 2007.
- [VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *Proceedings of the 11th International Software Product Line Conference, SPLC '07*, pages 233–242. IEEE Computer Society, Washington, DC, USA, 2007.
- [VH10] E. Vassev and M. Hinchey. The challenge of developing autonomic systems. *Computer*, 43(12):93–96, Dec 2010.
- [VH12] Emil Vassev and Mike Hinchey. The assl approach to specifying self-managing embedded systems. *Concurr. Comput.: Pract. Exper.*, 24(16):1860–1878, Nov 2012.
- [VH13a] Emil Vassev and Mike Hinchey. Autonomy requirements engineering: A case study on the bepicolombo mission. In *Proceedings of the International C\* Conference on Computer Science and Software Engineering, C3S2E '13*, pages 31–41. ACM, New York, NY, USA, 2013.
- [VH13b] Emil Vassev and Mike Hinchey. Autonomy requirements engineering: A case study on the bepicolombo mission. In *Proceedings of the International C\* Conference on Computer Science and Software Engineering, C3S2E '13*, pages 31–41. ACM, New York, NY, USA, 2013.
- [VH13c] Emil Vassev and Mike Hinchey. Knowledge-based self-adaptation. In *Proceedings of the 6th Latin-American Symposium on Dependable Computing (LADC 2013)*, pages 11–18. SBC – Brazilian Computer Society Press, Rio de Janeiro, Brazil, Apr 2013.
- [W3C07] W3C. Web Services Description Language (WSDL) Version 2.0, 2007. <http://www.w3.org/TR/2007/REC-wsd120-20070626/>.
- [War14] R. Warburton. *Java 8 Lambdas: Pragmatic Functional Programming*. O'Reilly Media, 2014.
- [We07] K Wolstencroft and et.al. The (my)Grid ontology: bioinformatics service discovery. *International Journal of Bioinformatics Research and Applications*, 3(3):303–325, 2007.
- [WKM<sup>+</sup>10] David Withers, Edward Kawas, Luke McCarthy, Benjamin Vandervalk, and Mark Wilkinson. Semantically-guided workflow construction in Taverna: the SADI and BioMoby plug-ins. In Tiziana Margaria and

- Bernhard Steffen, editors, *4th International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA 2010) - Volume Part I*, volume 6416 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin/Heidelberg, Oct 2010.
- [WNS<sup>+</sup>13] Stephan Windmüller, Johannes Neubauer, Bernhard Steffen, Falk Howar, and Oliver Bauer. Active Continuous Quality Control. In *16th International ACM SIGSOFT Symposium on Component-Based Software Engineering, CBSE '13*, pages 111–120. ACM SIGSOFT, New York, NY, USA, 2013.
- [WS73] W. Wulf and Mary Shaw. Global variable considered harmful. *SIG-PLAN Not.*, 8(2):28–34, feb 1973.
- [ZBN<sup>+</sup>04] Liangzhao Zeng, B. Benatallah, A. H H Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, May 2004.
- [ZC06] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, pages 371–380. ACM, New York, NY, USA, 2006.



## A. Selected Publications

### I Learning-Based Cross-Platform Conformance Testing

by Johannes Neubauer, and Bernhard Steffen. In Submission, 2014, *Software Testing, Verification and Reliability*, John Wiley & Sons, Inc..

### II Second-Order Servification

by Johannes Neubauer, and Bernhard Steffen. In Georg Herzworm, and Tiziana Margaria, editors: *International Conference on Software Business*, 2013, *Lecture Notes in Business Information Processing* 150, Springer, pp. 13–25, doi: 10.1007/978-3-642-39336-5\_2.

### III Higher-Order Process Modeling: Product-Lining, Variability Modeling and Beyond

by Johannes Neubauer, Bernhard Steffen, and Tiziana Margaria. *Electronic Proc. in Theoretical Computer Science*, vol. 129, 2013, pp. 259-283.

### IV Simplicity-First Model-Based Plug-In Development

by Stefan Naujokat, Johannes Neubauer, Anna-Lena Lamprecht, Bernhard Steffen, Sven Jörges, and Tiziana Margaria. In D. Garbervetsky and S. Kim, editors, Special Issue: Developing Tools as Plug-ins: *TOPI 2012*, ser. *Software: Practice and Experience*, 2013, John Wiley & Sons, Ltd.

### V Plug&Play Higher-Order Process Integration

by Johannes Neubauer, and Bernhard Steffen, *IEEE Computer*, vol. 46, no. 11, 2013.

### VI Risk-Based Testing via Active Continuous Quality Control

by Johannes Neubauer, Stephan Windmüller, and Bernhard Steffen. In Michael Felderer, Ina Schieferdecker editors, Special Issue: Risk-Based Testing, *Software Tools for Technology Transfer*, 2014, Springer. To appear.

## Technical Report

### VII Higher-Order Process Engineering: The Technical Background

by Johannes Neubauer. Technical Report, *TU Dortmund, Germany*, 2014, Eldorado.



## B. Comments on My Participation

### - **Publication I**

I am the main author of the paper. I developed the concepts and implemented the enhancements to the jABC, i.e., a canonical mapping of methods to business activities as well as introducing a second-order context for services. I designed and described the case study and participated in conducting the experiments.

### - **Publication II**

I am the main author of the paper. I developed the concepts and implemented the enhancements to the jABC, i.e., introducing type-safe interfaces for process models as well as a second-order context for process instances. I designed and described the running example.

### - **Publication III**

I am the main author of the paper. I developed the concepts and implemented the enhancements to the jABC, i.e., type-safe stacked second-order process contexts for higher-order process modeling. The concepts and contents have been discussed amongst all authors. I designed and described the running example.

### - **Publication IV**

I am one of the two main authors of the paper. The concepts and contents have been discussed amongst all authors. I was incorporated in writing nearly every section (excluding 4.3 and 4.4), whilst I was the main author of sections 2 and 6. The paper illustrates how well the concepts developed in the context of this thesis integrate in the overall jABC framework.

### - **Publication V**

I am the main author of the paper. The concepts and contents have been discussed amongst all authors. I designed and described the running example.

### - **Publication VI**

I am the main author of the paper. The concepts and contents have been discussed amongst all authors. I participated in carrying out the case study as well as planning and writing of all sections. I am solely responsible for sections 4 and 5.



## C. Other Publications

- **Design for Verifiability: The OCS Case Study**  
by Johannes Neubauer, Tiziana Margaria, Bernhard Steffen. *Formal Methods for Industrial Critical Systems: A Survey of Applications*, 2011, pp. 151–177, John Wiley & Sons, Inc..
- **Simplified Validation of Emergent Systems through Automata Learning-Based Testing**  
by Bernhard Steffen, and Johannes Neubauer. *Software Engineering Workshop (SEW), 2011 34th IEEE*, 2011, pp. 84–91, IEEE.
- **Reusing System States by Active Learning Algorithms**  
by Oliver Bauer, Johannes Neubauer, Bernhard Steffen, Falk Howar. *Eternal Systems*, 2012, Springer.
- **Automated Continuous Quality Assurance**  
by Johannes Neubauer, Bernhard Steffen, Oliver Bauer, Stephan Windmüller, Maik Merten, Tiziana Margaria, and Falk Howar. *Software Engineering: Rigorous and Agile Approaches (FormSERA), 2012 Formal Methods in Industrial Critical Systems*, 2012, pp. 37–43, IEEE.
- **Active Continuous Quality Control**  
by Stephan Windmüller, Johannes Neubauer, Bernhard Steffen, Falk Howar, and Oliver Bauer *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, 2013, pp. 111–120, ACM.