Realia et naturalia

DISSERTATIONES
MATHEMATICAE
UNIVERSITATIS
TARTUENSIS
84

JEVGENI KABANOV

Towards a more productive Java EE ecosystem

TARTU ÜLIKOOL · UNIVERSITAS TARTUENSIS · 1632

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
84

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
84

JEVGENI KABANOV

Towards a more productive Java EE ecosystem

UNIVERSITY OF TARTU
PRESS
1632

Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

This dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (Ph.D.) in Informatics on February 26th, 2013, by the Council of the Faculty of Mathematics and Computer Science, University of Tartu.

Supervisor:

Prof. PhD.	Varmo Vene Institute of Computer Science University of Tartu, Tartu, Estonia

Opponents:

Prof. PhD.	R. Nigel Horspool Department of Computer Science University of Victoria, Victoria, Canada
Prof. PhD.	Tanel Tammet Department of Computer Science Tallinn University of Technology, Tallinn, Estonia

Commencement will take place on April 5th, 2013 at 14:15 in Liivi 2–404.

The publication of this dissertation was financed by Institute of Computer Science, University of Tartu.

ISSN 1024-4212
ISBN 978-9949-32-231-2 (print)
ISBN 978-9949-32-232-9 (pdf)

University of Tartu Press
www.tyk.ee
Order No. 63

Contents

List of Original Publications (included in the thesis)

I Oleg Mürk, **Jevgeni Kabanov**. **Aranea: web framework construction and integration kit**. In Proc. of 4th Int. Conf. on Principles and Practice of Programming in Java, PPPJ 2006 (Mannheim, Aug./Sept. 2006), v. 178 of ACM Int. Conf. Proc. Series, pp. 163-172. ACM Press, 2006.

II **Jevgeni Kabanov**, Michael Hunger, Rein Raudjärv. **On designing safe and flexible embedded DSLs with Java 5.** Sci. of Comput. Program., v. 76, n. 11, pp. 970-991, 2011.

III **Jevgeni Kabanov**. **Method and arrangement for re-loading a class**, US Patent nr 20080282266.

IV **Jevgeni Kabanov** and Varmo Vene, 2012, **A Thousand Years of Productivity: The JRebel Story**, Software: Practice and Experience, *to appear*.

List of Other Publications

I **Jevgeni Kabanov**, Varmo Vene. **Recursion schemes for dynamic programming.** In T. Uustalu, ed., Proc. of 8th Int. Conf. on Mathematics of Program Construction, MPC 2006 (Kuressaare, July 2006), v. 4014 of Lect. Notes in Comput. Sci., pp. 235-252. Springer, 2006.

II **Jevgeni Kabanov**, Rein Raudjärv. **Embedded typesafe domain specific languages for Java.** In Proc. of 6th Int. Conf. on Principles and Practice of Programming in Java, PPPJ 2008 (Modena, Sept. 2008), v. 347 of ACM Int. Conf. Proc. Series, pp. 189-197. ACM Press, 2008.

III Aivar Annamaa, Andrei Breslav, **Jevgeni Kabanov**, Varmo Vene. **An interactive tool for analyzing embedded SQL queries.** In K. Ueda, ed., Proc. of 8th Asian Symp. on Programming Languages and Systems, APLAS 2010 (Shanghai, Dec. 2010), v. 6461 of Lect. Notes in Comput. Sci., pp. 131-138. Springer, 2010.

IV **Jevgeni Kabanov**. **JRebel tool demo**. In D. Pichardie, ed., Proc. of 5th Wksh. on Bytecode Semantics, Verification, Analysis and Transformation, BYTECODE 2010 (Paphos, March 2010), v. 264, n. 4 of Electron. Notes in Theor. Comput. Sci., pp. 51-57. Elsevier, 2011.

Abstract

We started our investigations in 2005. Our goals were to address some of the more gaping holes in the Java ecosystem and bring it on par with the languages touted as more productive. The first effort was to design a better web framework, called "Aranea". At that point of time Java had more than thirty actively developed web frameworks, and many of them were used simultaneously in the same projects. We decided to focus on two key issues: ease of reuse and framework interoperability.

The next issue that we focused on was the data access layer. We believed that SQL is the right way to access the data in a relational database, as it expresses exactly the data that is needed without much overhead. Instead of embedding it into the strings, we decided to embed it using the constructs of the Java language, thus creating an embedded domain-specific language (DSL). As one of the goals was to provide extensive compiler-time validation, we made extensive use of Java Generics and code generation to provide maximum possible static safety.

Our work on the SQL DSL made us believe that building typesafe embedded DSLs could be of great use for the Java community. We embarked on building two more experimental DSLs, one for generating and manipulating Java classes on-the-fly and the other for parsing and generating XML. These experiments exposed some common patterns, including restricting DSL syntax, collecting type safe history and using type safe meta-

data. Applying those patterns to different domains helps encode a truly type safe DSL in the Java language.

Our final and largest effort concentrated on a major disadvantage of the Java platform as compared to the dynamically-typed language platforms. Namely, while in PHP or Ruby on Rails one could edit any line of code and see the result immediately, the Java application servers would force one to do "build" and "deploy", which for larger applications could take minutes and even tens of minutes.

We came up with a novel and practical way of reloading code on the JVM, which we developed and released as the product "JRebel". We made use of the fact that Java bytecode is a very high level encoding of the Java language, which is easy to modify during load time. This allowed us to insert a layer of indirection between the call sites, methods and method bodies which was versatile enough to manage multiple versions of code and redirect the calls to the latest version during runtime.

Today, seven years after we began our efforts, it is clear that our approach to web framework development did not bear a lot of fruit. The framework itself has been released as open source in 2005. The typesafe DSL effort was more successful, as it influenced both further research on the topic and some changes inspired by this and similar efforts made its way into the design of the latest JPA specification. Our dynamic code reloading solution is in wide use today in the Java community, with over 3000 organizations using it day to day.

Disclaimer

Part I

Overview

Chapter 1

Introduction

> "The fact is, reality is complicated, and not amenable to the *one large idea* model of problem solving. The only way that problems get solved in real life is with a lot of hard work on getting the details right. Not by some over-arching ideology that somehow magically makes things work."
>
> Linus Torvalds

Since the Tim Berners-Lee famous proposal of World Wide Web in 1990 [Bern 90] and the introduction of the Common Gateway Interface in 1993, the world of online web applications has been booming. In the nineties the Java language and platform became the first choice for web development in and out of the enterprise. But by the mid-aughts the platform was in crisis – newcomers like PHP, Ruby and Python have picked up the flag as the most productive platforms, with Java left for conservative enterprises.

However, this was not because those languages and platforms were significantly better than Java. Rather, the issue was that innovation in the Java ecosystem was slow, due to the ways the platform was managed. Large

vendors dominated the space, standards were designed by committees and the brightest minds were moving to other JVM languages like Scala, Groovy or JRuby.

In this context we started our investigations in 2005. Our goals were to address some of the more gaping holes in the Java ecosystem and bring it on par with the languages touted as more productive. The first effort was to design a better web framework, called "Aranea". At that point of time Java had more than thirty actively developed web frameworks, and many of them were used simultaneously in the same projects. We decided to focus on two key issues: ease of reuse and framework interoperability.

To solve the first issue we created a self-contained component model that allowed the construction of both simple and sophisticated systems using a simple object protocol and hierarchical aggregation in style of the Composite design pattern. This allowed one to capture every aspect of reuse in a dedicated component, be it a part of framework functionality, a repeating UI component or a whole UI process backed by complex logic. Those could be mixed and matched almost indiscriminately subject to rules expressed in the interfaces they implemented.

To solve the second issue we proposed adapters between the component model and the various models of other frameworks. We later implemented some of those adapters both in a local and remote fashion, allowing one to almost effortlessly capture and mix different web application components together, no matter what the underlying implementation may be.

The next issue that we focused on was the data access layer. At that point in the Java community the most popular ways of accessing data was either using embedded SQL strings or an Object-Relational Mapping tool "Hibernate". Both approaches had severe disadvantages. Using embedded SQL strings exposed the developers to typographical errors, lack of abstraction, very late validation and dangers of dynamic string concatena-

tion. Using Hibernate/ORM introduced a layer of abstraction notorious for the level of misunderstanding and production performance issues it caused.

We believed that SQL is the right way to access the data in a relational database, as it expresses exactly the data that is needed without much overhead. Instead of embedding it into strings, we decided to embed it using the constructs of the Java language, thus creating an embedded DSL. As one of the goals was to provide extensive compiler-time validation, we made extensive use of Java Generics and code generation to provide maximum possible static safety. We also built some basic SQL extensions into the language that provided a better interface between Java structures and relational queries as well as allowing effortless further extension and enabling ease of abstraction.

Our work on the SQL DSL made us believe that building type safe embedded DSLs could be of great use for the Java community. We embarked on building two more experimental DSLs, one for generating and manipulating Java classes on-the-fly and the other for parsing and generating XML. These experiments exposed some common patterns, including restricting DSL syntax, collecting type safe history and using type safe metadata. Applying those patterns to different domains helps encode a truly type safe DSL in the Java language.

Our final and largest effort concentrated on a major disadvantage of the Java platform as compared to the *dynamically-typed* language platforms. Namely, while in PHP or Ruby on Rails one could edit any line of code and see the result immediately, the Java application servers would force one to do "build" and "deploy", which for larger applications could take minutes and even tens of minutes.

Initial investigation revealed that the claims of fast code reloading were not quite solid across the board. Dynamically-typed languages would typically destroy state and recreate the application, just like the Java application servers. The crucial difference was that they did it quickly and the

productivity of development was a large concern for language and framework designers.

However as we investigated the issue deeper on the Java side, we came up with a novel and practical way of reloading code on the JVM, which we developed and released as the product "JRebel". We made use of the fact that Java bytecode is a very high level encoding of the Java language, which is easy to modify during load time. This allowed us to insert a layer of indirection between the call sites, methods and method bodies which was versatile enough to manage multiple versions of code and redirect the calls to the latest version during runtime.

There have been over the years some basic developments in the similar fashion, but unlike them we engineered JRebel to run on the stock JVM and to have no visible impact on application functional or non-functional behaviour. The latter was the hardest, as the layer of indirection both introduces numerous compatibility problems and adds performance overhead. To overcome those limitations we had to integrate deeply on many levels of the JVM and to use compiler techniques to remove the layer of indirection where possible.

1.1 Publications and Contributions

This dissertation is based on four papers, which are listed below.

- **Publication 1: Aranea: web framework construction and integration kit.**
  - This paper describes our efforts at developing a self-contained component object model that would allow assembling web frameworks and would serve to ease application interoperability. The author lead the design and development of "Aranea" for several years and participated in the initial design of the core model and most extensions.

- **Publication 2: On designing safe and flexible embedded DSLs with Java 5**

  - This paper describes the common patterns in designing a safe and flexible embedded domain-specific language on the Java 5 platform, using features like generics, enums, foreach and code generation. It includes descriptions of three prototype example DSLs that give rise to the patterns. The Bytecode DSL was designed and prototyped by the author, and the SQL DSL was developed from his descriptions and under his direct supervision. The patterns were derived by the author with help from other contributors.

- **Publication 3: Method and arrangement for re-loading a class**

  - This patent describes the underlying basic ideas that went into the development of the dynamic code reloading solution "JRebel". The author has developed the initial version of JRebel and written the patent largely alone.

- **Publication 4: A Thousand Years of Productivity: The JRebel Story**

  - This paper relates the experience of creating "JRebel" from both an academic and and industry perspective. It describes the state-of-the art before and after the development, describes the technical challenges in developing the product in the existing and complex ecosystem and measures the impact of the tool on the ecosystem. The challenges and measurements described have been either done by the author or under his direct supervision. The author is the primary contributor of the paper.

1.2 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 corresponds to publication I and describes our efforts at creating the component object model, reference web framework, interoperability module and other extensions of the "Aranea" project. It also relates some practical experiences and further developments outside of the scope of the paper.

Chapter 3 corresponds to publication II and describes the patterns in designing the typesafe embedded domain-specific languages on the Java 5 platform as well as the reference embedded DSLs we created to study the challenges and derived the patterns from. It also relates some further developments outside of the scope of the paper.

Chapter 4 corresponds to publications III and IV and describes the development of the "JRebel" code reloading solution as well as its research and industry implications. It gives a brief technical description of the core ideas to help understand the patent and provides some context outside of the scope of the paper or patent.

Chapter 5 relates the previously discussed developments to the industry and research ecosystem and summarizes the improvements of the state-of-the-art as well as discussing possible further developments in these directions.

Chapter 2

Aranea: Web Framework Construction and Integration Kit.

> "For the last 10 years it has been all about giant Java servers that poop HTML."
>
> Cameron Purdy

2.1 Motivation

In 2005, when we started work on the "Aranea" project the Java web framework ecosystem was both rich and fragmented. Over 30 open source Java web frameworks were actively developed, not including commercial products or other platforms like .NET and numerous dynamic languages. This is also not to mention in-house corporate frameworks that never saw public light. Many different and incompatible design philosophies were used, but even within one approach there were multiple frameworks that had small implementation differences and were consequently incompatible with each other.

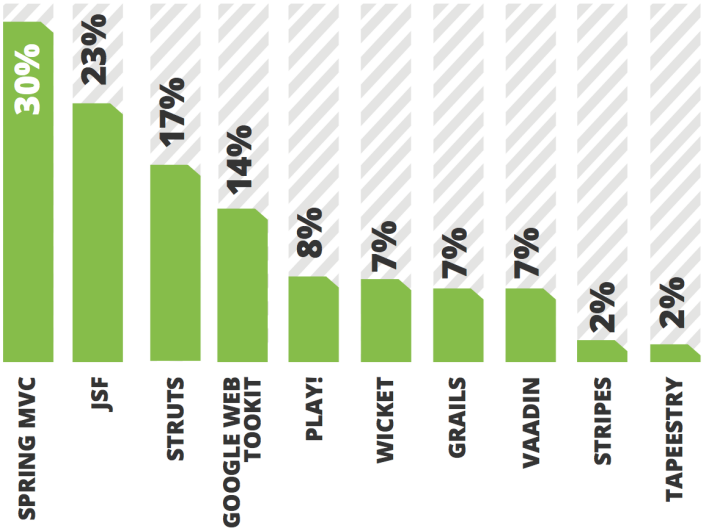

Figure 2.1: Web Framework Popularity in 2012 [Kaba 11a]

The advantage of such a situation is that different approaches and ideas are tried out. Indeed, many very good ideas have been proposed during these years, many of which we incorporated into the project. Although today there is still no clear winner in the web framework market, a lot of consolidation has taken place, but as one can see in figure 2.1 [Kaba 11a] only 8 web frameworks have over 5% of the market.

The goal of the "Aranea" project was to create a simple yet versatile platform that could be used to express the various approaches, experiment with new framework features and provide and interoperability layer between different web frameworks and applications. It would provide researchers with a good starting point, framework developers with a common integration platform and a reference implementation, and users with a framework that takes component reusability to a new level to accommodate complex UI requirements as shown in figure 2.2.

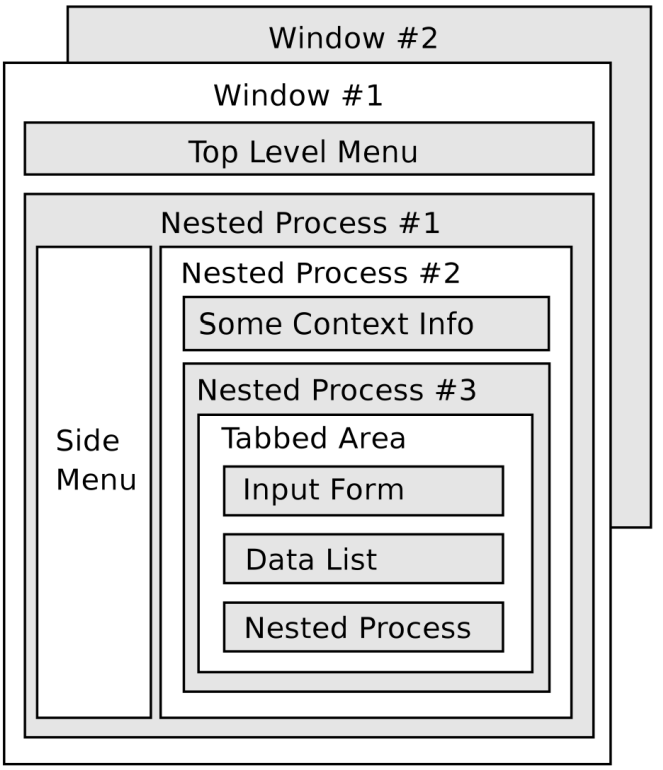

Figure 2.2: Aranea Complex UI

2.2 Background

Aranea draws its ideas from multiple frameworks such as Struts, WebWork, JavaServer Faces, ASP.NET, Wicket, Tapestry, WASH, Cocoon, Spring Web Flow, and RIFE.

Although we were not aware of Seaside [Duca 04] when developing this framework, we have to acknowledge that the rich UI programming interface of widgets and flows is almost identical with the programming interface of Seaside, but the design of Seaside differs a lot and it is not intended as a component model for web framework construction and integration.

2.3 Contribution

The Aranea component object model is built around two basic types of server-side components with a well-defined lifecycle and dependencies—stateless unsynchronized `Service`s and stateful synchronized `Widget`s. Both must implement the basic `Component` interface that provides a lifecycle, dependencies in the `Environment` and messaging via self-routing `Message`s.

```
interface Component {
  void init(Environment env);
  void enable();
  void disable();
  void propagate(Message msg);
  void destroy();
}
```

The first basic component is a `Service`. Since the service can be both stateless and re-entrant, it provides only one `action()` method that is responsible for all logic, interaction and rendering the component may do. `Path`, `Input` and `Output` are abstractions over HTTP protocol request and

response that allow one to build better abstractions and hide irrelevant implementation details.

```
interface Service extends Component {
    void action(
           Path path,
           InputData input,
           OutputData output
           );
}
```

The second basic component is a `Widget` that is assumed to be stateful and non-reentrant. It has separate request-response cycle phases: `update()` is called on some or all widgets irrespective of where the request is directed, `event()` is called on the widget that originated the request, `process()` is called on all widgets that will render and `render()` is an idempotent method that may be called as many times as necessary.

```
interface Widget extends Service {
    void update(InputData data);
    void event(Path path, InputData input);
    void process();
    void render(OutputData output);
}
```

These descriptions may cause one to think that the Aranea core hosts and manages these components and their lifecycle. Actually, the Aranea core is meta-cicular in the sense that the Aranea core consists of the same components as the application, thus developing application and framework features require the same mental model and set of skills. Figure 2.3 illustrates the key components in the hierarchy that are necessary to host the root application widget. A typical Aranea assembly will contain dozens of components in the basic hierarchy and include dozens more ready made

components that capture various aspects of UI functionality and are intended to be used in the application, including flow containers, form containers, paged list containers and so on.

The most interesting side effect of this model is that almost everything can be expressed as a first class abstraction. Here are some concepts that are hard to express in most other web frameworks:

Flows A very typical need in a business application is a multi-stage process wherein some information is collected, some choices are made, some data is updated and then the result is returned to the caller. This is complicated to implement without a decent state management and hard to reuse without a ubiquitous component model. In Aranea, we created a `FlowContainerWidget` that captures and manages the abstraction of state management and navigation, which can be hosted by any other widget. This allows one to arbitrarily call flows, nest flows and even run parallel flows on the same page independently with guaranteed isolation.

Contexts Another common requirement is to have a portion of the application operating within a specific context, e.g. think medical application with a particular *patient* selected. Aranea allows one to easily nest that whole part of the application in a particular `Widget`, that both renders the necessary information and provides the context in the `Environment` of all underlying widgets.

Some typical web frameworks aspects are also different in the presence of a hierarchical component model:

Configuration Unlike other frameworks that typically require custom configuration mechanism, Aranea can be wired together by a Dependency Injection container like Spring or Guice and its individual components and aspects are configured using the same mechanism.

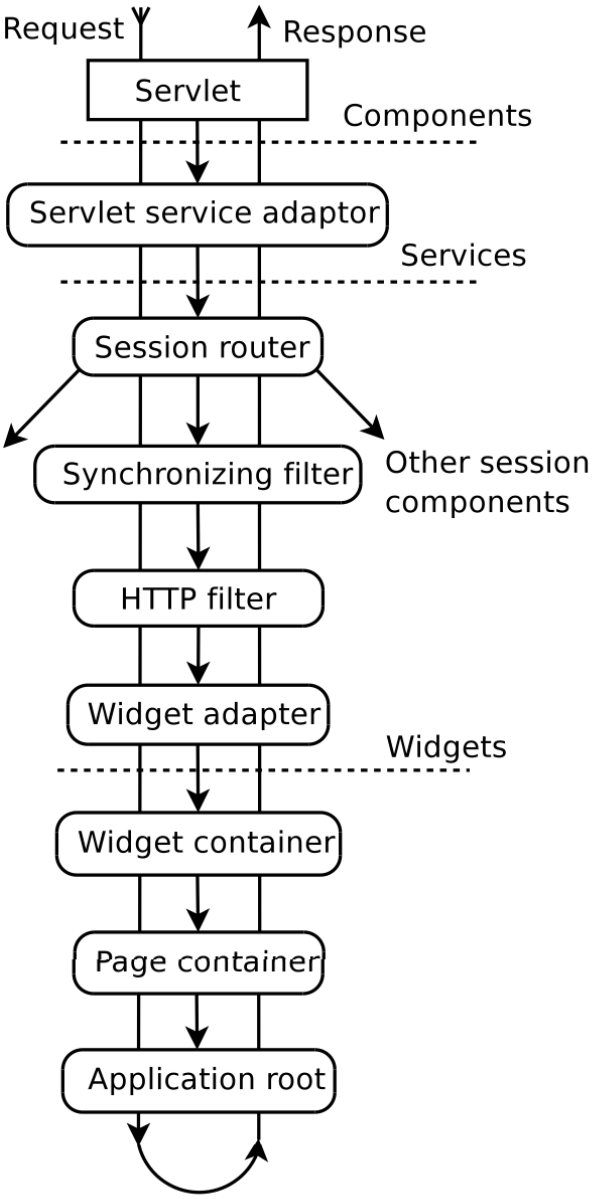

Figure 2.3: Aranea Component Hierarchy

Security There is no need to protect particular URLs, as the application code defines all allowed navigation. Thus the only protection needed is largely at the level of menus and menu items.

Error Handling The beauty of a hierarchical component model is that exceptions can occur and be processed at any level of the hierarchy. E.g. the `FlowContainerWidget` processes the exceptions on its own and the rest of the widgets are isolated from these exceptions and render without issue.

Concurrency As widgets are always synchronized and stateful, there is no need to worry about mutable state or concurrency, instead one can focus on the actual application requirements.

2.4 Conclusions and Further Developments

Several further developments have been implemented since the publishing of the article and documented in various bachelor's and master's theses.

Blocking Calls/Continuations Implemented by Rein Raudjärv [Raud 07] as a bachelor's thesis, this is an implementation of continuation-style programming, inspired by Scheme's call/cc, in the context of the Aranea component model and specifically the flow navigation pattern. This was implemented using load-time bytecode processing that captures the call stack and resumes execution on the next request.

Ajax Implemented by Alar Kvell [Kvel 07] as a bachelor's thesis, this brings contextual rendering and update regions, so that only the necessary widgets are rendered and updated on each user interaction. Along with a Javascript library that made it trivial to call widget events as well as simplified a number of other UI interactions. Extremely rich UI applications have been built using this development.

Dynamic XML Schema-Based Web Forms in Java Implemented by Rein Raudärv [Raud] in his master's thesis, it provides for a declarative XML-based DSL for describing UI forms and mapping to a persistent XML data model. This has been used to build applications for organizations that have a lot of user-generated data.

Lightweight Web Integration Implemented by Maxim Boiko [Boik 08] in his master's thesis, it describes and implements the local and remoting interoperability features of Aranea, as envisioned in the paper. Implementation for JSF, Struts and Remote Portlets was included.

History Navigation Mechanisms and Web Application State Implemented by Taimo Peelo [Peel 08] as a bachelor's thesis, it provides advanced server-side and client-side state management as well as state versioning and branching in the VCS style.

Java Web Applications on Desktop Implemented by Priit Liivak [Liiv 08] as a bachelor's thesis, this allows users to deploy Aranea web applications on the desktop, allowing applications to function offline as well as providing access to desktop APIs like tray icons and notifications. It includes a database for offline caching and operations and foresees a lot of later functionality of Google Gears and HTML5.

It is our belief that although Aranea did not catch either a lot of market or a lot of mindshare, it has performed well as a platform for experimentation and research. It is also running in multiple complex production web applications to this day.

Chapter 3

Embedded Typesafe DSLs on Java 5

> "I went into this knowing very little about ORM, and even very little about databases."
>
> Gevin King on creating Hibernate

Domain-specific language usually refers to a small sublanguage that has very low overhead when expressing domain-specific data and behavior. DSL is a broad term [Deur 00, Bent 99] and can refer both to a fully implemented language and a specialized API that looks like a sublanguage [Huda 96], but still written using some general-purpose language. Such DSLs in the latter meaning have been introduced by both the functional [Brin 04] and dynamic language communities [Cuad 07]. Both these communities (especially functional) took advantage of function composition and operator overloading to build combinator-based languages that look nothing like the host one. The functional community also strongly supports the notion of type safety; therefore DSLs they create are usually statically typed.

The main motivation for using DSLs (whether embedded or external) is threefold. First of all, the key feature of DSLs is encoding domain-specific data and behavior with low overhead. This means that the code is both easier to comprehend and easier to maintain. Secondly, thanks to the low overhead the DSL text should also be understandable by the domain expert. This makes it easier to collaborate with the expert on encoding the domain-specific logic. Finally, with embedded DSLs one can make use of the compiler advanced features to ensure type safety on the level of DSL constructs, thus eliminating certain types of errors already during compilation.

In the Java community DSLs are becoming increasingly popular. Unfortunately published work in the area is very rare and most of the innovation is done in an ad hoc way by various members of the Java community. Almost the only paper in the area was published by Freeman et al [Free 04] and describes the lessons learnt from designing the jMock embedded DSL. Another example is the Hibernate Criteria [Baue 05]. Those and some folklore examples introduced a technique for writing embedded DSLs using method call chaining that was coined Fluent Interface by Martin Fowler [Fowl 05].

Unfortunately most of the current DSLs do not use the advanced features of the language introduced in Java 5. Although quite a few clever tricks can be found in the wild, designing a safe and flexible DSL is still a challenge. In our paper we show how to make use of Generics, Enums, static imports and other Java 5 language capabilities to significantly improve the resulting DSL's flexibility and safety. We introduce several novel patterns that make designing a Java 5 DSL an easier task. It is our goal that the paper would serve as a starting point for someone designing an embedded DSL that takes full advantage of the Java 5 features.

3.1 DSLs

To illustrate our motivation consider the following example. SQL is commonly embedded in dynamically concatenated Java strings. This example includes multiple mistakes, which may be pretty hard to detect in this form.

```
ResultSet rs = SqlUtil.executeQuery(
  "SELECT name, height, birthday " +
  "FORM person" +
  "WHERE heigth >= " + 170);
while (rs.next()) {
   String name = rs.getString("name");
   Integer height = rs.getInt("height");
   Date birthday = rs.getDate("birthday");
   System.out.println(
     name + " " + height + " " + birthday);
}
```

The following example rewrites the same query in the statically type-safe embedded SQL DSL. All of the mistakes in the previous example would cause a compilation error in this one, including both syntactical and type errors. In fact, in a sufficiently advanced IDE, like Eclipse, the autocomplete suggestion will be precisely the syntactically and type correct entries.

```
Person p = new Person();

List<Tuple3<String, Integer, Date>> rows =
  new QueryBuilder(datasource)
    .from(p)
    .where(gt(p.height, 170))
    .select(p.name, p.height, p.birthday)
    .list();
for (Tuple3<String, Integer, Date> row : rows) {
  String name = row.v1;
```

```
  Integer height = row.v2;
  Date birthday = row.v3;
  System.out.println(
    name + " " + height + " " + birthday);
}
```

Next one can see a "Hello, World!" program in Java.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

The same program in compiled form is expressed via class structurs (constructors, methods, fields) and Java bytecode. This form can be derived from the compiled class using the `javap` tool shipped with the JVM.

```
public class HelloWorld {
  public <init>()V
    ALOAD 0
    INVOKESPECIAL Object.<init>()V
    RETURN

  public static main([LString;)V
    GETSTATIC System.out : LPrintStream;
    LDC "Hello, World!"
    INVOKEVIRTUAL PrintStream.println(LString;)V
    RETURN
}
```

Typically, a need to generate or manipulate Java bytecode arises, it's a fairly complex and error-prone procedure. The leading generators are ASM [Brun 02] and Javassist [Chib 98]. Both use strings and untyped

arguments in lieu of any statically verifiable constructs. We propose that a typesafe DSL could significantly improve productivity and reduce the defect rate in such complex undertakings.

```
new ClassBuilder(
  cw, V1_4, ACC_PUBLIC, "HelloWorld", "Object", null)
  .beginStaticMethod(
    ACC_PUBLIC | ACC_STATIC,
    "main", void.class, String[].class)
  .getStatic(System.class, "out", PrintStream.class)
  .push("Hello, World!")
  .invokeVirtualVoid(
    PrintStream.class, "println", String.class)
  .returnVoid()
  .endMethod();
```

The DSL that we propose not only tracks the types of all the arguments, but, Java bytecode being a stack-based language, also records and tracks all the slots and types on the stack, to make sure that an incorrect expression would not compile.

The bytecode DSL is not fully typesafe, as some arguments like method names are expressed as strings. Unfortunately, unlike SQL, which operates againsta predefined schema, bytecode DSL 1) operates in an open world, where some of the classes may not be available at the type and 2) even if all classes were predefined there are just too many of them to create a viable typesafe metadata dictionary.

Finally, we also prototyped an XML-based DSL, which like the bytecode DSL is not entirely statically typesafe. However, in that context we also explored later verification, where some of the constraints are expressed by the Java metadata attributes and can be verified in full by the runtime.

```
final Tag<PERSON> person = xml.tag(PERSON.class, NAME, "Michael")
   .attr(BIRTHDAY,"dd.MM.yyyy",birthDay) // uses SimpleDateFormat
```

```
.attr(SIZE, "#000.00", 1.82) // uses DecimalFormat
.add().tags(
  xml.tag(ADDRESS.class, CITY, "Dresden")
    .attr(ZIP, "01309"),
    .attr(COUNTRY, Country.DE) // Country is Enum
    .text("This is the main address"),
  xml.tag(ADDRESS.class, CITY, "Berlin")
    .attr(STREET, "Gleimstr.")
    );
```

3.2 Patterns

Let us now take a step back and look at the patterns showing up in the design of the DSLs we have described. We describe the patterns in an informal way, but try to illustrate in the paper with the examples of use and a discussion of impact. It is our hope that these patterns will simplify the future design and development of complex typesafe Java DSLs.

A brief summary of patterns follows.

Restricting Syntax *At any moment of time the DSL builder should have precisely the methods allowed in the current state.* This is important both from the point of validating syntax and providing a great IDE autocompletion experience to the user.

Type History *Type history can be accumulated as a type list and use it to reject actions that do not fit with that history* The most advanced use of this pattern is in the Bytecode DSL, where the whole stack is incrementally recorded and then validated at compile-time against the expressions.

Typesafe Metadata *Metadata used by the DSL should include compile-time type information.* This was best expressed in the SQL DSL,

where the database schema is used to generate its full representation in the typesafe Java expressions, less so in Bytecode and XML DSLs.

Unsafe Assumptions *Allow the user to do type unsafe actions, but make sure he has to document his assumptions.* Good design should always allow escape hatches.

Hierarchical Expressions *Use method chaining when context is needed and static functions when hierarchy and extensibility are needed.* There used to be a lot of pushback against using static functions in such manner, but now that functional languages are making a strong comeback the situation has reversed.

Closures *Use closures to escape the method chaining for control flow and reuse.* Combining method chaining with control flow is almost impossible without closures. With Java 8 project Lambda we are excited about opportunities to build even better DSLs.

3.3 Further Developments

Several further developments have been implemented since the publishing of the article.

Typesafe DSL for relational data manipulation in Java Juhan Aasaru [**?**] has expanded on our prototype of the typesafe SQL DSL and did a full blown implementation with tests and documentation, as well as experimented with multiple extensions to our original ideas.

Squill Building on Juhan Aasaru's work, the authors of the paper have developed the SQL DSL further as the Open Source project "Squill" [**?**]. We implemented different database dialects, practical array of database functions and other necessary functionality. Despite that,

to our knowledge, Squill has never been used in a production application.

Our work on the typesafe DSLs in Java became one of the seminal papers in that area and is cited in most further developments. There are numerous interesting ideas coming out in this research area and it is our belief that our initial goal of creating a foundation for later developments has been achieved.

As far as the industry is concerned, due to our work in this area we have been invited into discussions of various DSLs, including the JPA 2.0 Criteria JSR standard and some of the influence of our work can be seen in the latest standards.

Chapter 4

JRebel: Dynamic Application Reloading on the JVM

> "[changing the JVM] to allow the addition of methods and to allow some method attribute changes is currently planned for [Java 7]"
>
> Sun JVM Bug 4910812, 21st Mar 2006

One of the common ways for enterprises to build web applications is by using the Java Enterprise Edition platform. This platform introduces the concept of an application server that manages the HTTP client connections and provides services like security, transactional execution and network clustering. Each application can the focus on the specific business logic and needs to be *deployed* on the application server to function. The application itself is a ZIP archive with a *.war* or *.ear* file extension with internal structure corresponding to the Java EE specification [Shan 06].

The application needs to be *compiled* from the sources, *packaged* into the archive and *deployed* to the application server before it can begin executing.

Such a procedure was conceived to make deploying the application to a production environment, where it will begin servicing the users, to be safe and predictable. In development, however, this causes multiple delays every day, as developers make GUI changes need to validate them in a running application. As the time it takes to build and deploy an application is in general proportional to its size, this causes unnecessary waste of time and frustration in the development of Java EE applications.

An effort to skip the deployment step was also made, when Dmitriev et al [Dmit 01a] introduced the HotSwap mechanism in 2001. HotSwap allowed one to substitute the method bodies of Java classes in the running application, thus changes to the code could be propagated directly to the application server over the wire. Although this improved things somewhat, the limitation to method bodies severely reduced its usefulness.

By 2007 it was not uncommon to have a 15 minutes long build & deploy cycle for large enterprise applications even on the best available hardware. In this setting we released the first version of our software, JRebel [Kaba 08a]. In the beginning, JRebel was developed as an improvement over the HotSwap mechanism, allowing methods, fields and classes to be added in addition to changing the method bodies. As of 2012, the tool supports a much wider array of changes, allowing the developers to almost always skip both packaging and deployment steps for all Java EE applications with the few exceptions discussed later in the paper. As a result, the tool grew from its humble beginnings in 2007 to a major time-saver in the Java EE ecosystem, with tens of thousands of users benefiting every day.

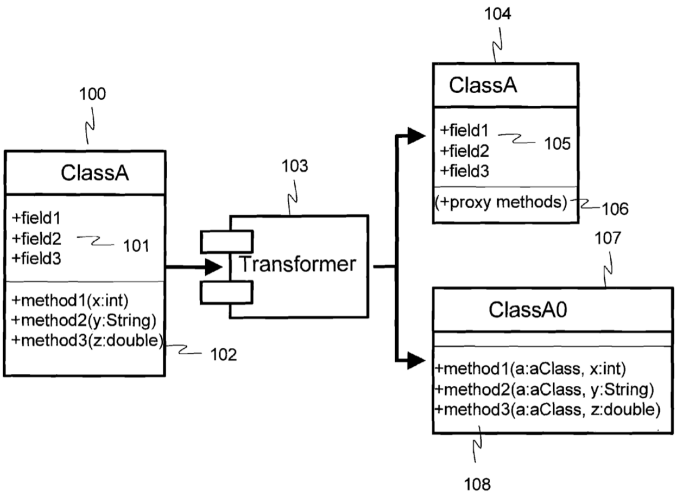

Figure 4.1: JRebel Class Transformation

4.1 Contribution

JRebel works by instrumenting the application and JVM classes to create a layer of indirection. When an application class is loaded, it is instrumented to replace the method bodies with a redirection to the runtime redirection service as shown on figure 4.1. The runtime redirection service manages and loads the class and method versions using supplementary anonymous classes, that are loaded per each class or method version. Additionally, the invocations in the method bodies are rewritten to account for added methods and fields as well as changes in the method override hierarchy.

Although ZeroTurnaround does not disclose the details of JRebel internals beyond the filed patents [Kaba 08a], we will describe the underlying concepts using a series of examples below.

We start with a simple class C with two methods `method1()` and `method2()`.

```
public class C extends X {
  int y = 5;
```

```
  int method1(int x) {
    return x + y;
  }
  void method2(String s) {
    System.out.println(s);
  }
}
```

During load-time JRebel will instrument that class. The class signature will be preserved, but the bodies of all methods will be replaced with a runtime redirection call, that includes all of the information about the call, including name, signature and arguments. Arguments are packed into an `Object` array to allow for any number and type of arguments to be passed to the runtime.

```
public class C extends X {
  int y = 5;

  int method1(int x) {
    Object[] o = new Object[1];
    o[0] = x;
    return Runtime.redirect(this, o, "C", "method1", "(I)I");
  }

  void method2(String s) {
    Object[] o = new Object[1];
    o[0] = s;
    return Runtime.redirect(
      this, o, "C", "method2", "(Ljava/lang/String;)V");
  }
}
```

JRebel will also load a version 0 of the class C, which will be named C_0 or similarly incorporate the version identifier in the class name. It

includes all of the methods, but they are converted to `static` and take the instance of class C as the first argument. The bodies of these methods are instrumented to go through the runtime for the method calls and field lookups of methods and fields.

```
public abstract class C0 {
  public static int method1(C c, int x) {
    int tmp1 = Runtime.getFieldValue(c, "C", "y", "I");
    return x + tmp1;
  }

  public static void method2(C c, String s) {
    PrintStream tmp1 =
      Runtime.getFieldValue(
        null, "java/lang/System", "out", "Ljava/io/PrintStream;");

    Object[] o = new Object[1];
    o[0] = s;

    Runtime.redirect(
      tmp1, o, "java/io/PrintStream;", "println",
        "(Ljava/lang/String;)V");

  }
}
```

Now let's say that the user has change the class C by adding a new method `z()` and calling it from `method1()`.

```
public class C {
  int y = 5;

  int z() {
```

```
    return 10;
  }

  int method1(int x) {
    return x + y + z();
  }
  ...
}
```

JRebel detects the change and loads a new version of class C called C_1. This class adds a static version of method `z()` and uses the new body of `method1()`.

```
public class C1 {
  public static int z(C c) {
    return 10;
  }

  public static int method1(C c, int x) {
    int tmp1 = Runtime.getFieldValue(c, "C", "y", "I");
    int tmp2 = Runtime.redirect(c, null, "C", "z", "(V)I");

    return x + tmp1 + tmp2;
  }
  ...
}
```

As the runtime will always route the `Runtime.redirect` call to the latest version of the class, calling `new C().method1(10)` will return 15 before the change and 25 after the change.

Of course such naive implementation is both missing a lot of necessary details and is incredibly inefficient. In reality most of the redirection calls in JRebel are optimized away completely and the ones that still take place

are completely inlined either in call sites, destination methods or generated intermediary methods.

4.2 Impact

JRebel has been around since 2007, but it gained traction over time. Today over 3000 organizations are customers of ZeroTurnaround. But how much is the software actually used? Does it measurably save time for the users? If yes, then how much? And what are the savings in dollar equivalent, if any?

These are the questions that need to be answered in order to understand what is the actual impact of the software on our users. Of course, measuring the productivity gain of the software is always challenging. However in the case of JRebel there are some estimates that are pretty easy to put together.

1. JRebel itself gathers statistics on the numbers of updates it does and reports the figures back to us, when possible, using the "Homecalling" functionality.

2. JRebel Social is our free product for non-commercial use that requires a tie-in to an online account and thus provides more accurate reporting.

3. During the years we have run several surveys that allow us to estimate the time wasted on reloading changes in the Java EE ecosystem as well as other key productivity metrics.

The first results are from homecalling data. From a sample of 1000 random users, the average number of redeploys prevented yearly is *2226*. To count the number of redeploys prevented per hour we assume 240 working days a year and 5 coding hours a day. Thus *9.28* redeploys are prevented per day and *1.86* redeploys per coding hour.

The data from JRebel Social is summarized in table 4.1.

Item	Description
Average Redeploy Time	139.7 sec / 2:20 mins
Total Redeploys Prevented Per User	2,516
Total Redeploys Prevented	16,290,165
Total Time Saved	72 years 60 days

Table 4.1: JRebel Social Data

The data from our 2010 survey, answered by 1027 respondents with only 28% being ZeroTurnaround customers, is summarized in table 4.2.

Item	Average	StdDev
Redeploy time	3.1 mins	2.8
Redeploy frequency in an hour	4	3.2
Time spent redeploying in an hour	10.5 mins	9.8
Coding time spent redeploying	17.5%	16.3
Time spent redeploying per year	5.3 work-weeks	4.9

Table 4.2: Redeploy Survey Results

Applying the average redeploy time of *3.1 minutes* to the results from the previous section we would get an average of *5.77 minutes* an hour spent redeploying. This constitutes *9.6%* of coding time. The time saved per day would be *28.77 minutes* and the time per working week of five days is *3.23 hours*. Finally, the time saved per year would be *115.01 hours* or *2.88 40-hour work weeks*.

Whereas there are some difference across the three datasets, we can derive good lower and upper bound for the actual time saved, which is from *2.88* to *5.3* 40-hour work-weeks a year.

Using those bounds we can also estimate the amount of directly measurable time saved among all of the JRebel users over the lifetime of the product. Assuming a number of users over $15,000$ the total time saved falls between *900* and *1666* man-years a year.

4.3 Further Developments

Since its initial release in September of 2007 JRebel has gone through four more major releases, where each added a wealth of important functionality.

1.0 Initial release, support for class reloading and basic Java SE integration.

2.0 Support for packaged deployment with virtual packaging through the "rebel.xml" functionality.

3.0 Deep integration with Java SE and Java EE to support code and configuration updates for EJB, JSP, JSF, CDI and JPA standards as well Proxies, enums, static fields and so on.

4.0 Shipped with 50+ framework and container plugins for code and configuration reloading in the ecosystem.

5.0 Included JRebel Remoting, that allows updating code remotely as seamlessly as locally.

Our current focus goes largely is largely on providing the same quick and easy updates in distributed production applications as are provided by JRebel for development. "LiveRebel 1.0" had the following major differences from JRebel:

Server and application management LiveRebel can deploy, undeploy and update applications on any subset of the running servers.

Distributed orchestration LiveRebel either applies update on all application servers or on none. A failed update is rolled back automatically.

Structural diff Before an update, LiveRebel compares the current and the previous version and identifies if there are any changes that cannot be applied by the hotpatching engine derived from JRebel.

Request buffering Ensures that no requests are lost during the update and that hotpatching is applied on an application without load.

Changeset distribution Only tne difference between versions is uploaded to the server nodes.

However after gathering feedback from users we understood that hotpatching is just a small piece of the puzzle and focused on supporting the whole application lifecycle. LiveRebel 2.0 included the following changes:

Distributed load balancer LiveRebel starts a load-balancing daemon in front of every server and those are used to redistribute requests among servers when necessary.

Multiple update strategies LiveRebel would fall back to rolling restarts and then to offline restarts if hotpatching wasn't available.

Configuration management LiveRebel allows scripts to be shipped in the artifacts and executed at different points of the application lifecycle.

Our further work is connected largely to application lifecycle orchestration, configuration management and expansion to platforms beyond Java.

Chapter 5

Conclusions and future work

> "The current version is always just a beta, the next one will rock your world!."
>
> Cameron Purdy

Today, seven years after we began our efforts, it is clear that our approach to web framework development did not bear a lot of fruit. The framework itself has been released as open source in 2005. Some research has referred Aranea, some further experimentation has been done on the platform under our supervision and several production application are running on it, but the world of web application integration moved instead to RESTful services and browser-based UI integration.

The typesafe DSL effort was more successful, as it influenced both further research on the topic and some changes inspired by this and similar efforts made its way into the design of the latest JPA specification. The DSLs described in the paper never made it past the prototype phase and were never used in a production application.

Our dynamic code reloading solution is in wide use today in the Java community, with over 3000 organizations using it day to day. JRebel suc-

cess has motivated further research in similar technologies as well as some competing efforts, all beneficial for the community and ecosystem.

All in all, our contribution has been a drop of improvement in the ocean of challenges, but thanks to the efforts of other good folks the Java EE platform of today is significantly better than 7 years ago.

Our further efforts are now focused on application maintainance, life-cycle and delivery, which remains a very promising area of study, as huge inefficiencies are easy to come by. In particular we plan to investigate application state migration, change and release workflow process automation, database schema versioning, configuration and environment as code as well massive long-running online system evolution.

References

[Aasa 08] Juhan Aasaru. *Typesafe DSL for Relational Data Manipulation in Java*. Master's thesis, University of Tartu, 2008.

[Ande 00] J. Andersson and T. Ritzau. **Dynamic code update in JDrums**. In: *Proceedings of the ICSE00 Workshop on Software Engineering for Wearable and Pervasive Computing*, Limmerick, Ireland, 2000.

[Ande 98] J. Andersson, M. Comstedt, and T. Ritzau. **Runtime support for dynamic Java architectures**. In: *ECOOP98 Workshop on Object-Oriented Software Architectures, Brussels*, 1998.

[Anna 10] Aivar Annamaa, Andrey Breslav, Jevgeni Kabanov, and Varmo Vene. **An Interactive Tool for Analyzing Embedded SQL Queries**. In: Kazunori Ueda, editor, *APLAS*, pp. 131–138, Springer, 2010.

[Atki 00] M. Atkinson and M. Jordan. **A review of the rationale and architectures of PJama: a durable, flexible, evolvable and scalable orthogonally persistent programming platform**. *Sun Microsystems, Inc. Mountain View, CA, USA*, 2000.

[Bail 01] B.P. Bailey, J.A. Konstan, and J.V. Carlis. **The effects of interruptions on task performance, annoyance, and anxiety in the user interface**. In: *Proceedings of IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 593–601, Amsterdam, The Netherlands, 2001.

[Baue 05] C. Bauer and G. King. *Hibernate in action*. Manning, 2005.

[Bent 99] J.L. Bentley and J. Bentley. *Programming Pearls*. Addison-Wesley Professional, 1999.

[Bern 90] T. Berners-Lee and R. Cailliau. **World-Wide Web: Proposal for a Hypertext Project**. *European Particle Physics Laboratory (CERN)*, 1990.

[Bess 10] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. **A few billion lines of code later: using static analysis to find bugs in the real world**. *Communications of the ACM*, Vol. 53, No. 2, pp. 66–75, 2010.

[Boik 08] M. Boiko. *Lightweight Web Integration*. Master's thesis, University of Tartu, 2008.

[Brin 04] B. Bringert, A. Höckersten, C. Andersson, M. Andersson, M. Bergman, V. Blomqvist, and T. Martin. **Student paper: HaskellDB improved**. pp. 108–115, ACM Press New York, NY, USA, 2004.

[Brun 02] E. Bruneton, R. Lenglet, and T. Coupaye. **ASM: a code manipulation tool to implement adaptable systems**. *In Proceedings of the ASF Journees Composants (JCÓ2): Adaptable and Extensible Component Systems*, Vol. 30, 2002.

[Burn 05] E. Burnette. *Eclipse IDE Pocket Guide*. O'Reilly Media, Inc., 2005.

[Chib 98] S. Chiba. **Javassist: a reflection-based programming wizard for Java**. In: *Proceedings of OOPSLA98 Workshop on Reflective Programming in C++ and Java*, p. 174, 1998.

[Cuad 07] J.S. Cuadrado and J.G. Molina. **Building Domain-Specific Languages for Model-Driven Development**. *IEEE Software*, Vol. 24, No. 5, pp. 48–55, 2007.

[Deur 00] A. van Deursen, P. Klint, and J. Visser. **Domain-specific languages: an annotated bibliography**. *ACM SIGPLAN Notices*, Vol. 35, No. 6, pp. 26–36, 2000.

[Dmit 01a] M. Dmitriev. **Towards flexible and safe technology for runtime evolution of java language applications**. In: *Proceedings of the Workshop on Engineering Complex Object-Oriented Systems for Evolution*, pp. 14–18, Citeseer, 2001.

[Dmit 01b] Mikhail Dmitriev. **Safe Class and Data Evolution in Large and Long-Lived Java[tm] Applications**. Tech. Rep., Sun Microsystems, Inc., Santa Clara, CA, USA, 2001.

[Dmit 99] M. Dmitriev. **The first experience of class evolution support in PJama**. In: *Advances in persistent object systems: proceedings of the Eighth International Workshop on Persistent Object Systems (POS-8) and the Third International Workshop on Persistence and Java (PJAVA-3), August 30-September 4, 1998, Tiburon, California*, p. 279, Morgan Kaufmann Pub, 1999.

[Duca 04] Stéphane Ducasse, Adrian Lienhard, and Lukas Renggli. **Seaside–a multiple control flow web application framework**. In: *Proceedings of European Smalltalk User Group Conference*, pp. 231–257, 2004.

[Fowl 05] Martin Fowler and Eric Evans. **FluentInterface at** `http://www.martinfowler.com/bliki/FluentInterface.html`. 2005.

[Free 04] S. Freeman, T. Mackinnon, N. Pryce, and J. Walnes. **jMock: supporting responsibility-based design with mock objects**. *Conference on Object Oriented Programming Systems Languages and Applications*, pp. 4–5, 2004.

[Gamm 95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[Gitz 06] Ralf Gitzel, Markus Aleksy, and Martin Schader, editors. *Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java, PPPJ 2006, Mannheim, Germany, August 30 - September 1, 2006*, ACM, 2006.

[Grau 01] P. Graunke, S. Krishnamurthi, S. Van Der Hoeven, and M. Felleisen. **Programming the web with high-level programming languages**. *Programming Languages and Systems*, pp. 122–136, 2001.

[Greg 08] A.R. Gregersen and B.N. Jorgensen. **Module Reload through Dynamic Update-The Case of NetBeans**. In: *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR'2008)*, pp. 23–32, IEEE, Athens, Greece, 2008.

[Greg 09] A.R. Gregersen, D. Simon, and B.N. Jørgensen. **Towards a dynamic-update-enabled JVM**. In: *Proceedings of the Workshop on AOP and Meta-Data for Software Evolution*, p. 2, ACM, 2009.

[Hewl 05] Hewlett-Packard. **Abuse of Technology Can Reduce UK Workers Intelligence**. *HP Technical Report,*, Vol. 22, pp. 129–69, April, 2005.

[Hieb 94] Robert Hieb, R. Kent Dybvig, and Claude W. Anderson, III. **Subcontinuations**. *Lisp Symb. Comput.*, Vol. 7, No. 1, pp. 83–110, Jan. 1994.

[Huda 96] Paul Hudak. **Building domain-specific embedded languages**. *ACM Comput. Surv.*, Vol. 28, No. 4, p. 196, 1996.

[Jack 02] T. Jackson, R. Dawson, and D. Wilson. **Case Study: evaluating the effect of email interruptions within the workplace**. In: *Conference on Empirical Assessment in Software Engineering, Keele University, EASE*, pp. 3–7, Citeseer, 2002.

[Jack 03] T. Jackson. **Instant messaging implications in the transition from a private consumer activity to a communication tool for business**. In: *New Approaches to Software Quality*, p. 319, British Computer Society, Software Quality Management, Canterbury, 2003.

[Kaba 06] Jevgeni Kabanov and Varmo Vene. **Recursion Schemes for Dynamic Programming**. In: Tarmo Uustalu, editor, *MPC*, pp. 235–252, Springer, 2006.

[Kaba 08a] E. Kabanov. **METHOD AND ARRANGEMENT FOR RELOADING A CLASS**. May 2008. US Patent nr 20080282266.

[Kaba 08b] Jevgeni Kabanov and Rein Raudjärv. **Embedded typesafe domain specific languages for Java**. In: Luís Veiga, Vasco Amaral, R. Nigel Horspool, and Giacomo Cabri, editors, *PPPJ*, pp. 189–197, ACM, 2008.

[Kaba 11a] Jevgeni Kabanov. **Java EE Productivity Report**. 2011. http://www.zeroturnaround.com/java-ee-productivity-report-2011/.

[Kaba 11b] Jevgeni Kabanov. **JRebel Tool Demo**. *Electr. Notes Theor. Comput. Sci.*, Vol. 264, No. 4, pp. 51–57, 2011.

[Kaba 11c] Jevgeni Kabanov, Michael Hunger, and Rein Raudjärv. **On designing safe and flexible embedded DSLs with Java 5**. *Sci. Comput. Program.*, Vol. 76, No. 11, pp. 970–991, 2011.

[Kaba 12] Jevgeni Kabanov and Varmo Vene. **A thousand years of productivity: the JRebel story**. *Software: Practice and Experience*, 2012.

[Kicz 01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. **An Overview of AspectJ**. In: *Proceedings of the 15th European Conference on Object-Oriented Programming*, pp. 327–353, Springer-Verlag, London, UK, UK, 2001.

[Kim 08] Dong Kwan Kim and Eli Tilevich. **Overcoming JVM HotSwap constraints via binary rewriting**. In: *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*, pp. 5:1–5:5, ACM, New York, NY, USA, 2008.

[Kvel 07] A. Kvell. **Aranea Ajax**. 2007. Bachelor's Thesis, University of Tartu.

[Lian 98] Sheng Liang and Gilad Bracha. **Dynamic class loading in the Java virtual machine**. *SIGPLAN Not.*, Vol. 33, pp. 36–44, October 1998.

[Liiv 08] P. Liivak. **Java Web Applications on Desktop**. 2008. Bachelor's Thesis, University of Tartu.

[Lind 99] T. Lindholm and F. Yellin. *Java virtual machine specification*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[Mala 00] Scott Malabarba, Raju Pandey, Jeff Gragg, Earl Barr, and J. Fritz Barnes. **Runtime Support for Type-Safe Dynamic Java Classes**. In: Elisa Bertino, editor, *ECOOP 2000*

Object-Oriented Programming, pp. 337–361, Springer Berlin / Heidelberg, 2000.

[Murk 06] Oleg Mürk and Jevgeni Kabanov. **Aranea: web framework construction and integration kit**. In: Ralf Gitzel, Markus Aleksy, and Martin Schader, editors, *PPPJ*, pp. 163–172, ACM, 2006.

[Peel 08] T. Peelo. **History Navigation Mechanisms and Web Application State**. 2008. Bachelor's Thesis, University of Tartu.

[Pete 59] L. Peterson and M.J. Peterson. **Short-term retention of individual verbal items.** *Journal of Experimental Psychology*, Vol. 58, No. 3, p. 193, 1959.

[Puka 08] Mario Pukall, Christian Kastner, and Gunter Saake. **Towards Unanticipated Runtime Adaptation of Java Applications**. In: *15th Asia-Pacific Software Engineering Conference (APSEC'2008), Beijing, China*, pp. 85–92, IEEE Computer Society, Los Alamitos, CA, USA, 2008.

[Quei 00] Christian Queinnec. **The influence of browsers on evaluators or, continuations to program web servers**. *SIGPLAN Not.*, Vol. 35, No. 9, pp. 23–33, Sep. 2000.

[Raud] R. Raudjärv. **Dynamic XML Schema Based Web Forms in Java**. Master's thesis, Faculty Of Mathematics And Computer Science, Institute Of Computer Science, University Of Tartu, Estonia. 2010. Software and thesis available from: `http://code.google.com/p/xsd-web-forms`.

[Raud 07] R. Raudjärv. **Blocking Calls in Java**. 2007. Bachelor's Thesis, University of Tartu.

[Rome 06] T. Römer. **Web Deployment Unit**. 2006. Bachelor's Thesis, University of Tartu.

[Shan 06] B. Shannon. **JavaTM Platform, Enterprise Edition (Java EE) Specification, v5**. *Sun Microsystems*, 2006.

[Soli 98] R. van Solingen, E. Berghout, and F. van Latum. **Interrupts: just a minute never is**. *IEEE Software*, Vol. 15, No. 5, pp. 97–103, 1998.

[Subr 09] Suriya Subramanian, Michael Hicks, and Kathryn S. McKinley. **Dynamic Software Updates: A VM-centric Approach**. In: *PLDI '09: Proceedings of the 2009 ACM SIGPLAN conference on Programming Language Design and Implementation*, pp. 1–12, ACM, Dublin, Ireland, 2009.

[Susa 01] H.R. Susarla, M. Garg, and E. Sandhya. **Dynamic class reloading mechanism**. 06 2001. US Patent App. 09/895,287.

[Ueda 10] Kazunori Ueda, editor. *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, Springer, 2010.

[Uust 06] Tarmo Uustalu, editor. *Mathematics of Program Construction, 8th International Conference, MPC 2006, Kuressaare, Estonia, July 3-5, 2006, Proceedings*, Springer, 2006.

[Veig 08] Luís Veiga, Vasco Amaral, R. Nigel Horspool, and Giacomo Cabri, editors. *Proceedings of the 6th International Symposium on Principles and Practice of Programming in Java, PPPJ 2008, Modena, Italy, September 9-11, 2008*, ACM, 2008.

[Wurt 10] T. Würthinger, C. Wimmer, and L. Stadler. **Dynamic code evolution for Java**. In: *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, pp. 10–19, ACM, 2010.

Acknowledgements

First of all I'd like to thank the various Tallinn cafeteria where this thesis was written. Thank you, Cafe More, Cafe Komeet, Cafe Mmuah & Cafe Kadriorg Gourmet. I wouldn't have done it without you!

My lovely wife kept me sane enough to work through this and my cutie one year-old daughter made sure that I didn't oversleep my duties.

My grandmother was the main source of motivation and an unending source of inspiration. It is mainly due to her weekly reminders that this work was finished.

Thanks to Mark Fishel and Konstantin Tretjakov for helping debug the thesis as well for the pleasure of their company for now over 20 years.

Thanks to ZeroTurnaround's employees and investors, who were understanding enough to let the CEO take the time to finish this undertaking.

The main and special gratitude goes to my advisor, Varmo Vene, who, during the 11 years I spent studying at Tartu University, has been a continuous source of inspiration, knowledge and good advice.

Kokkuvõte (Summary in Estonian)

Produktiivsema Java EE ökosüsteemi poole

Alates Java programmeerimiskeele loomisest 1995. a. on selle üks kõige olulisemaid kasutusvaldkondi veebirakenduste programmeerimine. Java populaarsuse põhjuseks ei olnud ainult keeledisainilised omadused nagu objektorienteeritus ja range tüübisüsteem, vaid ennekõike platvormist sõltumatus ja standardiseeritud teekide rohkus, mis tegi tavaprogrammeerijatele veebirakenduste programmeerimise jõukohaseks. Kümme aastat hiljem oli olukord muutunud märkimisväärselt. Java oli kaotamas oma liidripositsiooni uutele, nn. dünaamilistele keeltele nagu PHP, Ruby ja Python. Seejuures polnud põhjuseks mitte see, et need keeled ise oleksid tunduvalt Javast paremad, vaid Java ökosüsteemi areng oli väga konservatiivne ja aeglane.

Antud kontekstis alustasime aastal 2005 oma uuringuid eesmärgiga parandada suurimad probleemid Java ökosüsteemis ja viia see vähemalt samale tasemele ülalmainitud keeltega. Käesaolevas dissertatsioonis on esitatud vastavate uuringute tulemused. Dissertatsioon põhineb neljal publikatsioonil – kolmel eelretsenseeritud teadusartiklil ja ühel patendil.

Esimeseks katseks oli uue veebiraamistike integreerimisraamistiku "Aranea" loomine. Antud hetkel oli Javas üle kolmekümne aktiivselt arendatavat veebiraamistikku, mistõttu otsustasime fokuseeruda kahele

võtmeprobleemile: raamistike taaskasutatavuse lihtsus ja koostöövõime. Selleks töötasime välja uudse komponentmudeli, mis võimaldab kirjeldada süsteemi teenus- ja kasutajaliideskomponentide hierarhilisi seoseid, ja realiseerisime eri raamistike adapterid komponentmudelisse sobitumiseks. Uuringute tulemused on esitatud artiklis [Murk 06].

Järgmise probleemina käsitlesime andmete haldamiskihi kirjeldamist. Lähtusime eeldusest, et relatsioonilistes andmebaasides on SQL kõige enamlevinud andmete kirjelduskeel, ja efektiivne admehalduskiht peab võimaldama Javas lihtsasti esitada SQL päringuid, samas garanteerima konstrueeritavate päringute süntaktilise korrektsuse. Senised lahendused baseerusid reeglina SQL päringute programsel konstrueerimisel sõnedena, mistõttu päringute korrektsuse kontroll oli raskendatud. Lahenduseks töötasime välja nn. rakendispetsiifilise keele (i.k. *domain-specific language*, *DSL*) SQL päringute esitamiseks kasutades Java keele tüübisüsteemi vahendeid nende korrektsuse kompileerimisaegseks valideerimiseks. Töö käigus identifitseerisime üldised tarkvara disainimustrid, mis lihtsustavad analoogiliste tüübikindlate DSLide loomist, ja kasutasime neid kahe uue eksperimentaalse tüübikindla DSLi loomisel — Java klasside täitmisaegseks loomiseks ja manipuleerimiseks ning XMLi parsimiseks ja genereerimiseks. Uuringute tulemused on esitatud artiklis [Kaba 11c].

Kolmanda ülesandena pühendusime ühele olulisemale Java platvormi puudusele võrreldes dünaamiliste keeltega. Kui PHP's või Ruby's saab programmi koodi otseselt muuta ja tulemust koheselt näha, siis Java rakendusserverid nõuavad rakenduse "ehitamist" (i.k. *build*) ja "paigutamist" (i.k. *deploy*), mis suurte rakenduste korral võib võtta mitmeid või isegi kümneid minuteid. Probleemi lahenduseks töötasime välja uudse ja praktilise meetodi koodi ümberlaadimiseks Java platvormil, mille põhjal arendasime ja lasime välja toote "JRebel". See kasutab Java baitkoodi laadimisaegset modifitseerimist koos spetsiaalse ümbersuunamiskihiga kutsekohtade, meetodite ja meetodikeha vahel, mis võimaldab hallata koodi

erinevaid versioone ning täitmisajal suunata väljakutsed viimasele versioonile. Uuringute tulemused on esitatud patendis [Kaba 08a] ja artiklis [Kaba 12].

Täna, rohkem kui seitse aastat pärast uuringute algust, tuleb tõdeda, et meie töö veebiraamistikega lõi küll eduka platvormi erinevate eksperimentaalsete ideede uurimiseks ja katsetamiseks, kuid reaalses tarkvaratööstuses ei ole leidnud laialdast kasutust. Töö tüübikindlate DSLidega oli edukam, sest see mõjutas otseselt edaspidiseid uuringuid antud teemal ning selle elemendid leidsid rakendust viimases JPA standardi spetsifikatsioonis. Kõige suurem mõju tarkvaratööstusele on meie dünaamiline koodiümberlaadimise lahendus, mis on tänapäeval Java kogukonnas laialdaselt kasutusel ning mida kasutavad igapäevaselt rohkem kui 3000 erinevat organisatsiooni üle maailma.

Part II

Papers

Curriculum vitae

Name: Jevgeni Kabanov
Date of Birth: 28.04.1983
Citizenship: Estonian

Education:

2006–2013	University of Tartu, Faculty of Mathematics and Computer Science, doctoral studies, Specialty: computer science
2005–2007	University of Tartu, Faculty of Mathematics and Computer Science, master studies, Specialty: computer science
2001–2005	University of Tartu, Faculty of Mathematics and Computer Science, bachelor studies, Specialty: computer science
...–2001	Tallinn High School of Humanities

Work experience:

01/2007 - ...	OÜ ZeroTurnaround, CEO
01/2005 – 01/2007	University of Tartu, programmer
06/2003 – 01/2007	AS Webmedia, R&D lead

Elulookirjeldus

Nimi: Jevgeni Kabanov
Sünniaeg: 28.04.1983
Kodakondsus: Eesti

Haridus:

2006–2013	Tartu Ülikool, Matemaatika-informaatikateaduskond, doktoriõpe, Eriala: informaatika
2005–2007	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, Eriala: informaatika
2001–2005	Tartu Ülikool, Matemaatika-informaatikateaduskond, bakalaureuseõpe, Eriala: informaatika
...–2001	Tallinna Humanitaargümnaasium

Teenistuskäik:

01/2007 – ...	OÜ ZeroTurnaround, CEO
01/2005 – 01/2007	Tartu Ülikool, programmeerija
06/2003 – 01/2007	AS Webmedia, R&D juht

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.

43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.

64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.
74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
75. **Nadežda Bazunova.** Differential calculus $d^3 = 0$ on binary and ternary associative algebras. Tartu 2011, 99 p.
76. **Natalja Lepik.** Estimation of domains under restrictions built upon generalized regression and synthetic estimators. Tartu 2011, 133 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
80. **Marje Johanson.** $M(r, s)$-ideals of compact operators. Tartu 2012, 103 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
82. **Vitali Retšnoi.** Vector fields and Lie group representations. Tartu 2012, 108 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.