A Standardized Approach to Tool Integration

David Skoglund
Abstract

A Standardized Approach to Tool Integration

David Skoglund

This thesis examines the standard OSLC, for sharing resources, as a standardized integration platform. The previously most common way of integrating tools has been to implement a middleware which translates communication between different tools written in different programming languages. OSLC uses common concepts found on the web to form a language independent platform with no logic attached to its structure. Instead of logic OSLC relies on a metadata structure, enabling communication across different languages and platforms. A practical implementation between a debug analyzer (Optima OSE) and a test-harness (Farkle) was implemented to show some of the core OSLC functionality. The results show that integration is indeed possible. The analysis of the implementation showed that even though simple in the sense of concepts and standards used OSLC is far from mature enough for widespread adoption, mostly due to a somewhat overworked specification.
Dedication

I would like to dedicate this thesis to my parents for all their love, support and for lending me their car! I would also like to thank my thesis supervisors Barbro Claesson, Detlef Scholle at Xdin AB and Philipp Ruemmer at Uppsala University for providing me with feedback on my report and practical work.
# Contents

1 **Introduction** 4  
   1.1 Motivation ................................................. 4  
   1.2 Problem Statement ......................................... 5  
   1.3 Outline ..................................................... 6  
   1.4 Delimitations ............................................... 6  

2 **Background** 7  
   2.1 iFEST ...................................................... 7  
   2.2 Short Introduction to Lifecycle Models ....................... 7  
   2.3 Definition of the term Tool ................................ 8  
   2.4 Tool Integration ........................................... 9  
      2.4.1 Levels of Integration ................................... 9  
      2.4.2 Platform based Frameworks ............................. 11  
      2.4.3 Conclusion ............................................ 12  
   2.5 The Web as a Platform for Integration ....................... 12  
      2.5.1 The Semantic Web .................................... 12  
      2.5.2 Representational State Transfer ....................... 13  
   2.6 Modern Approaches to Integration ........................ 14  
      2.6.1 Component Based Architectures ....................... 14  
      2.6.2 TOOLBUS ........................................... 16  
      2.6.3 Model Based Integration .............................. 16  
      2.6.4 Conclusion ........................................... 17  

3 **Open Services for Lifecycle Collaboration** 21  
   3.1 Introduction to OSLC ...................................... 21  
   3.2 OSLC Technical Foundation ................................ 22  
      3.2.1 Metadata ............................................. 23  
      3.2.2 CRUD ................................................ 24  
      3.2.3 Containers & Serviceprovider ........................ 25  
      3.2.4 Conclusion ........................................... 25
4 Implementation 27

4.1 Optima Overview ................................. 28
4.2 Farkle Overview .................................. 28
  4.2.1 Alternative ways of implementation .......... 28
  4.2.2 Overview of Farkle’s OSLC Provider Adapter .... 29
  4.2.3 Resource Sharing ............................... 30
  4.2.4 Overview of Optima’s OSLC Consumer Adapter .. 30
  4.2.5 Flexibility of the Design ...................... 31
4.3 In depth description of the Farkle adapter .......... 31
  4.3.1 TestMechanics ................................. 31
  4.3.2 ResourceStorage ............................... 31
  4.3.3 Web Interface ................................. 32
  4.3.4 Optima OSLC consumer adapter and GUI ........ 32
4.4 Test Cases ........................................... 33
  4.4.1 Basic Functionality ............................ 33
  4.4.2 Verifying OSLC Web Structure .................. 33
  4.4.3 Verifying Database Concurrency ............... 33

5 Conclusion, Discussion and Future Work 34

5.1 Conclusion .......................................... 34
5.2 Discussion ......................................... 36
5.3 Future Work ........................................ 37
Bibliography ........................................... 39
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>SDLC according to the <em>waterfall model</em></td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>ALM illustration</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Wassermans integration stack</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>A CORBA request from a client to the actual object implementation, mediated by the ORB</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>CORBA IDL interface example</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Simplified illustration of the TOOLBUS architecture</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>A metamodel based transformation, as seen in[5]</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>OSLC relational diagram</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>XML/RDF syntax example</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>A visual representation of the OSLC provider structure [16]</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>A user chooses and executes a test case, if the test were to fail an EventTrace is set to target and the test is reexecuted.</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Optima and OSE target relation</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Pros and cons of using different methods of implementing OSLC[16]</td>
<td>30</td>
</tr>
<tr>
<td>4.4</td>
<td>Illustration of the overall design</td>
<td>32</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Software developers and carpenters alike depend on their respective tools in order to fully practice their craft. Albeit the tools used by the two professions are different their importance to the individual using them is very much the same.

When developing software numerous tools are used to optimize, support or in other ways enhance the developers work. The number of tools used in a development process strongly relates to the current software complexity or size of a given project. This is an issue which severely affects the work process if the tools used are not aligned or compatible. While in theory integrating all of the related tools may create a more optimized work process the task of integration is complex in itself.

Integrating tools from different software vendors have long been troublesome from a life cycle point of view. The time period in which different developers support their tools may vary from months to decades while the tools themselves have a much longer shelf life. The technical challenge of integration is arguably even more daunting. Different programming languages, operating systems and data storage are some of the aspects to be taken into account.

Another key aspect of integration is that of flexibility and longevity. While simple integration customized for a current set of tools might work at the given moment future changes might require adding new tools or replacing old. Using general ways of integration relieves the developer from the burden of having to create a new custom integration specialized for the new tools, with the ability to reuse old code.

The traditional method of achieving such a integration has in the past been to implement a middleware logic or a "broker" to facilitate the interaction between software. While this indeed makes the task of integration easier the "broker" logic itself is very complex and could limit the ultimate
functionality of the resulting integration. Examples of such systems are CORBA[13], TOOLBUS[21], DCOM[19].

This thesis sets out to explore a method which instead of a middleware relies on meta data structures and concepts which commonly are associated with the WEB, and the practical implications in implementing such a method. The standard is called OSLC and is defined by a joint venture of participants from the corporate as well as the private sector. What sets OSLC apart from some of its competitors is that it completely lacks any computational logic. Instead OSLC provides specifications which specify structures for how to enable sharing of data between tools and the minimum frameworks and standards needed to be implemented in order to achieve this. OSLC mandated the use of XML encoded in RDF[3] for the meta data structure and REST[8] for communication between those who intend to use a tools resources or share their own.

The work has been carried out at XDIN AB2, a participant in the iFEST project. iFEST3 is a research project funded by the European Union which aims to develop a framework for embedded system tools in order to cut engineering costs and reduce the time it takes for a product to reach the market, while minimizing the efforts of replacing or migrating tools. An open standard for data exchange between tools could meet many of the requirements specified by iFEST.

1.2 Problem Statement

The thesis is divided equally into two parts, one of which is an in-depth academic study about integration theory and an open tool chain approach to integration, OSLC and the possible limitations of a standardized tool integration approach. The academic study centers around previous research in the area of integration presented in various published articles.

The second part of the master thesis is devoted to the practical implementation of an integration between the the Integrated development environment Optima4 and the test harness Farkle. The two tools are used in a test/debug context targeting embedded hardware running the real time operating system ENEA OSE5. The intended purpose of the integration is to provide Farkle with the debug functionality available in Optima, or extending Optima with the test execution capabilities of Farkle. Thus making it possible to investigate possible reasons for test cases failing execution.

The thesis serves as a pilot study for further work in the area of tool integration. The following questions were formulated as a starting point:

---

1Open Services for Life cycle Collaboration, http://open-services.net/
2XDIN AB, http://www.xdin.se/
4http://www.enea.com/optima/
Question 1: *How should tools broadcast their services?*

Question 2: *How do different nodes (tools) discover one another?*

Question 3: *Does OSLC help in identifying artifacts likely to be shared?*

Question 4: *Does the OSLC standard limit the developer with respect to information being shared?*

Question 5: *In what format should said artifacts be presented according to OSLC?*

Question 6: *To what extent does a system relying on an open-tool-chain standard need complementary software to function (e.g.: databases, servers)?*

1.3 Outline

The remainder of this thesis is structured as follows:

**Chapter 2** The second chapter is intended to provide the necessary background knowledge needed for discussions regarding concepts and ideas within the area of integration. This chapter also presents some of the platforms used today and in the past. It also provides information about the typical stakeholders involved when developing software.

**Chapter 3** This chapter describes OSLC at its core and the technology behind it.

**Chapter 4** This chapter is devoted to the design and consequent implementation of the practical work, an OSLC adapter.

**Chapter 5** The last chapter encompasses conclusion, discussion and future work.

1.4 Delimitations

Different ways of implementing an open tool-chain type integration is only discussed in the academic part of the thesis. The practical implementation is limited to the intended functionality of the integration and features such as security(oAuth) and query capabilities which are defined in the OSLC standard have not been implemented.
Chapter 2

Background

We seldom contemplate the difficulties that come with the increase of computing power, but as this also increases the demand for more capability and versatility from our systems, developers face the problem of uniting the system as a whole. The different demands of the stakeholders involved add more complexity to the issue[1]. Users want a seamless experience where their data is processed in an optimal way regardless of software restrictions while developers desire simple and troublesome free support for integrating their product with other software vendors.

This chapter provides an introduction to the field of integration with integration theory as a starting point. Followed by modern integration methods and the underlying technology which supports it.

2.1 iFEST

This thesis project has been carried out as a part of the iFEST\(^1\) research project. The iFEST project started in 2010 with an estimated project duration of 36 months and belongs to the joint-venture initiative ARTEMIS\(^2\). The main purposes of the project is to promote standardization of integration technologies, focusing specifically on open tool chain solutions for hardware-software co-design and life cycle support for systems with high life expectancy (decades).

2.2 Short Introduction to Lifecycle Models

The way in which software used to be created, in the early days of computing, depended highly on the opinions and thoughts of the software developer in charge. This way of developing and maintaining software was doable as

\(^1\)industrial Framework for Embedded System Tools

\(^2\)ARTEMIS, http://www.artemis-ju.eu/
long as the software complexity was low and the intended use did not include any safety critical tasks. Nowadays software is used in sectors in which human lives depend on it, e.g. healthcare, and thus the requirements on how a piece of software is made, tested and supported far surpasses those in the early days of computing. The notion of software being ”done” once its coded and tested is no longer sufficient practice.

The goal of lifecycle models are to provide a framework which leads to well documented and maintainable software with a slightly more predictable nature than the previous ad-hoc development methods used. The main focus of this thesis OSLC specifies different metadata structures based on categories adhering to the ALM lifecycle model, which will be more thoroughly discussed in a later chapter.

The Software Development Lifecycle (SDLC)

An SDLC model is a framework which specify the activities performed, or should be performed, in the development process and the life thereafter, and more importantly the ongoing process of managing the software throughout the governance [18]. Most of theses models are arranged in different geometric forms, but are all mostly linear with a touch of iteration from a sequential point of view, see figure 2.1 for the SDLC^3 model.

Application Lifecycle Management(ALM)

ALM is said to be a fusion of software engineering and business management. While both ALM and the SDLC cover the development aspects of an application ALM is focused on the project surrounding the application instead of just the development phase. This includes variables such as stakeholder interests and resource availability, in other words anything that could or will affect the project. The author of [18] states that ALM consists of the three categories Governance, Development and Operations. SDLC, which centers around the development phase, is therefore considered to be a subset of ALM. As seen in fig2.2 the governance aspect runs from start to finish and is thought to represent the business management needs throughout the project. The operation phase starts just before the upcoming software launch and specifies the operations needed to maintain, deploy updates and the actual release itself.

2.3 Definition of the term Tool

Understanding the term tool is necessary for contextual purposes. While no strict definition exists we choose to define Tools as being software which facilitates, analyses, modifies or in other ways extends the core functionalities

^3Software Development Life Cycle
of other programs or systems. A set of tools used to develop or maintain an application is called a toolchain, if the tools are "chained" to exchange data with each other.

2.4 Tool Integration

While we could settle, in terms of correctness, by defining tool integration as the process in which different tools are combined to form a whole, we require a more context specific definition. According to Ian Thomas and Brian A Nejmeh the relationship between the tools is what should be the primary focus and thus extends the previous definition by stating that [1]:

...tool integration is about the extent to which tools agree.

The authors continue to state that:

The subject of these agreements may include data formats, user-interface conventions, use of common functions

The use of the word may indicates that integration is more about combining similarities than it is merging all of a the different functionalities of tools with one another.

2.4.1 Levels of Integration

In [2] the author Anthony Wasserman identifies four fundamental stages of integration. Wasserman’s study, as a whole, is regarded as a foundation in the field of IDE\(^4\) and has paved way for many other concepts and ideas in the field of integration. A previously cited study [1] extends Wassermans levels in order to further specify integrational relationships.

\(^4\)Interactive Development Environments
Figure 2.2: The aspects of ALM illustrated throughout the application life-time, as seen in [18]. The lines represent the time in which work is conducted in the respective aspects

The following section will explain Wasserman’s integration levels as seen in [2] while also presenting the additional properties suggested in [1].

**Presentation Integration**

Perceptual aspects should be shared amongst tools in order to achieve a common look and feel from a user perspective. This is important to minimize unnecessary time spent on learning the GUI\(^5\) instead of the actual tool function. It needs to be emphasized that the functionality should in no way be affected by the presentational integration since it only concerns the visual elements.

**Data Integration**

Sharing data, or rather presenting data in a agreed format, is likely one of the most important aspects of a tool. There are two main principles of data integration, *data format* and *data communication*. *Data format* specifies in what way data should be shared, while there does not exist one true way of sharing data, a general consensus within the computer science community advocates the use of shared repositories such as databases. But could be implemented as easily as a shared file written in an agreed format. *Data communication* specifies how data is shared directly between software with-

---

\(^5\)Graphical User Interface
out the use of storage. IPC\textsuperscript{6}, such as pipes\textsuperscript{7} and message passing is the traditional choice when the applications reside on the same system.

**Control Integration**

The ability to share notifications about current states and upcoming events is a critical aspect of any integration. These abilities bring completely new functionality to the tools being used. This and the ability to activate tools via other tools is what is referred to as control integration. The level of control tools have over each other is said to be either a loose or tight coupling, while a loose coupling might be just broadcasting current information a tight coupling refers to having a high level of influence over other tools via message passing or remote procedure calls.

**Process Integration**

While the previously stated integration types closely relate to the tool itself process integration refers to how tools interact according to a pre-defined process, on a organizational level. In reality this means that tools devoted to process management needs to be integrated with development based tools.

**Tool Categorization**

Being able to categorize tools according to function and identifying a set of base integration levels from that category can help a developer in planning out how and what to integrate. The \cite{2} study divides tools into either \textit{vertical} or \textit{horizontal} tools. As seen in fig2.3 the vertical tools need less integration with other vertical tools while underlying horizontal tools has a bigger surface area and should be integrated to several vertical tools. The Vertical tools are described as tools which are used during a certain phase, for example a test phase, while the horizontal tools are used during all the phases together, for example an entire development process.

**2.4.2 Platform based Frameworks**

The term framework can be described as a way of structuring or standardizing an arbitrary task or event. In software engineering a framework is a way of conforming to a certain standard or protocol to guarantee a certain amount of base functionality. Depending on a single software suite or an IDE\textsuperscript{8} may not seem like a drawback as long as said platform delivers all required functionality but if additional functionality is needed the original

\textsuperscript{6} Inter Process Communication
\textsuperscript{7} A form of IPC especially used in UNIX
\textsuperscript{8} Integrated Development Environment
integration problem re-arises. The problem described is called "vendor-lock-in" and limits the developers freedom of choice, since it forces the use of specific platform compliant tools. In such cases developers rely on several IDEs to overcome the shortcomings of one platform[5], but this obviously adds unnecessary complexity to an already complex environment.

Changes within the industry suggests that many platforms are evolving into more open Tool-Chain friendly frameworks[6],[7]. Eclipse’s SDK⁹ Lyo is one such example.

2.4.3 Conclusion

There are several aspects of integration to be aware of. Control integration and data integration may be the ones which often promote the integration itself, in promise of some kind of optimization, but this should not discourage one from integrating on other levels as well such as presentation, which has been suggested to be vital for software to function in an intended way[23]. The theoretical part of integration is well documented but due to the complex nature of integration the actual task of integration often limits developers as to what to integrate and to what extent. By categorizing software and mapping the categories to a model, see fig 2.3, one can determine which levels of integration should be prioritized over another.

2.5 The Web as a Platform for Integration

Using technology niched towards the Web for integration purposes might not seem like the most obvious choice for those new to the area, especially when the goal is to integrate software found locally on the same machine. However the technology used to facilitate communication and data sharing on the Web can just as easily be implemented to facilitate integration on a local level. Furthermore the Web does not discriminate against different platforms and operating systems, in that context the Web could be regarded as independent platform for integration in itself.

The following subchapter describes OSLC’s Web heritage, while leaving out OSLC specific for the upcoming chapter.

2.5.1 The Semantic Web

For the last decade there has been a growing desire to reshape the way we represent published data on the World Wide Web. While hyper-links create bonds between different documents, Web pages, no real bonds used to exist between data. This is, in part, said to be one of the factors which helped popularise the Web in its infancy, since there were no strict restrictions on

⁹Software Development Kit
how data should be formatted[3]. While being able to link data creates a lot of interesting possibilities the biggest benefit stems from linked data being machine-interpretable[4].

The Semantic web is a collaborative movement led by the W3C\textsuperscript{10} which advocates the use of agreed formats to extend the traditional hyper-link based Web to a web of data. The actual term Linked data was coined by Tim Berners-Lee, one of the authors of [3] and director of the W3C, and describes a concrete way, or rather methods to achieve, the aforementioned web of data.

**Linked data**

As previously stated the key units in the traditional hypertext based Web are documents, in HTML\textsuperscript{11} format. The HTML documents are linked together via un-typed links where as Linked data depends on documents containing data in RDF\textsuperscript{12} syntax, the link itself is created through typed statements about the data. The typed statements made about the data is said to describe “things” rather than data, which could be easier to comprehend from an educational standpoint [3], as well as being readable both by machines and human beings. Tim Berners-Lee created these four rules, or guidelines, for linking data[3] :

1. **Use URIs as names for things**
2. **Use HTTP URIs so that people can look up those names.**
3. **When someone looks up a URI, provide useful information, using the standards (RDF*)**
4. **Include links to other URIs, so that they can discover more things.**

### 2.5.2 Representational State Transfer

Using custom interfaces for published content did not matter during the time when the Web was still in its infancy. However as the number of content requests increased so did the demand for a uniform standard to reduce developer induced latency, a result of developers not conforming to a specific standard and instead relying on ad-hoc interfaces.

Roy Fielding and Richard Taylor defines REST\textsuperscript{13} as a coordinated set of architectural constraints that attempts to minimize the latency and network communication, while at the same time maximizing the independence and scalability of component implementation[8]. In more simplified terms REST

\textsuperscript{10}World Wide Web Consortium,http://www.w3.org

\textsuperscript{11}HyperText Markup Language

\textsuperscript{12}Resource Description Framework

\textsuperscript{13}Representational State Transfer
is defined as a web service or "middle ware" technology that uses protocols to allow applications to exchange data across the Web.

REST distinguishes the following three key architectural elements[9]:

Data elements, or resources, are identified through the simple precondition of being nameable, which implies that only the semantics of a resource remain static. In a web context resources are identified using URIs\textsuperscript{14}.

Connectors are used to facilitate the network communication between components. Different connector types are used in different situations e.g: The client connector type is used for sending requests or receiving responses and The server connector is used for listening for requests and sending responses.

Components are the abstract entities which are identified by their role within an application, the different roles include user agent, proxy, gateway and origin server. An example of a user agent is a Web browser, which is an application for retrieving and presenting information resources on the World Wide Web.

If a platform or framework conforms to the specified REST constraints it is said to be "RESTful".

\section*{2.6 Modern Approaches to Integration}

This sub chapter will discuss universal and non universal framework approaches that are non platform dependent, or in other words methods of integration which separates the tools being integrated from the framework used to facilitate the integration[5]. Some leeway will be given to architectures that are semi independent in regard to their operating system dependencies.

\subsection*{2.6.1 Component Based Architectures}

The complex issue of developing software has long been known to be troublesome. The term "software crisis" was popularised at the NATO summit in 1968, held in Germany, to describe the difficulties in hardware evolution in regards to software quality. Douglas MCllroy’s, one of the attendees, proposed solution was to reshape the development process to use components to aid the developers in their quests for better software quality. The so called components were reusable modules or library’s with high versatility with the intent to promote a modular development process[20].

\textsuperscript{14}Uniform Resource Identifier
Douglas McIlroy might have been slightly ahead of his time in the sense that component oriented programming became popular first 20 years later. Component based architectures are interesting from an integration perspective since they use a standardized way of inter process communication. In practical terms this means that the process of data integration becomes a more transparent task since the finite set of components are known even though source code specific information may be lacking, the below subsections aims to describe some of the most used architectures.

Common Object Request Broker Architecture

CORBA is arguably the most successful attempt at creating a universal inter-system communication framework. It was founded by the OMG\textsuperscript{15}, also known for creating specifications such as UML, in 1991 and still remains relevant to the point of continuous updates of the specification[14].

Figure 2.4 illustrates CORBA’s most central concept, requests for distributed objects. The ORB, or request broker as it is also called, is a distributed middleware which facilitates or ”translates” requests between different systems. The mechanics behind such requests are independent of location which means that a request is handled in the same way whether it be between systems in the same LAN\textsuperscript{16} or between systems separated by a large distance[13]. One of CORBA’s weaknesses is that of memory management, or rather the lack of memory management. It has been suggested that this weakness leads to ad-hoc solutions in regards to avoiding memory leaks[21].

Interface Definition Language (IDL)

Prior to making a object request necessary information about possible object actions are needed. An object’s interface specifies just that and is written in OMG’s Interface Definition Language, see example 2.6.1 for a short interface stub. The language and syntax is made to be recognisable for most experienced developers, bearing syntax similarities with languages such as C and Java interfaces. But IDL isn’t a programming language, defined interfaces are merely object information specified in a standard syntax form. To enable different programming languages to use these interfaces OMG has defined mappings to the most common and widely used programming languages.

A mapping between IDL and an arbitrary programming language specifies a static translation of attributes and variables in IDL to said language, since the translation is precise the mapping will yield the same result independent of number of translations[13].

\textsuperscript{15}Object management group, http://www.omg.org/
\textsuperscript{16}Local Area Network
Component Object Model (COM)

COM is a language independent interface, a named collection of abstract operations that represent an arbitrary functionality, standard for inter process communication developed by Microsoft\(^{17}\) which was introduced in 1993. The Component Object model is often referred to as being a binary standard, to further underline its platform and language independence, meaning that the standard applies after the program has been translated into binary code. While that statement is mostly true COM requires languages used for implementing objects to support function pointers\(^{17}\), somewhat limiting the actual choice of programming language.

COM bares many similarities with the previously mentioned CORBA. Both of the standards uses interface definition languages for multi-language support. But where as COM is strictly used for intercommunication between processes located on the same system CORBA has no restrictions in regards to the actual locations of the processes.

DCOM & XPCOM

DCOM was created to extend the COM platform to work in a distributed fashion, for situations when created objects needs to be placed at remote system separated from that of the client. DCOM further resembles CORBA.

2.6.2 TOOLBUS

TOOLBUS is a software coordination architecture relying on a process algebra\(^{18}\) based scripting language for inter process communication. As seen in fig 2.6 tools communicate with one another via process, written in the scripting language, within the toolbus abstraction. The connection to the toolbus is handled by language specific adapters, illustrated as a grey extensions of the tool squares in fig 2.6, which converts necessary information from the tools to the common TOOLBUS data format called aTerm\(^{19}\).

The processes within the toolbus abstraction send requests and events to the tools, one could arguably say that the processes ”controls” the tools. Several processes can be mapped to one or more tools but its desirable to keep a one to one mapping since it is desirable to to minimize unnecessary complexity\(^{21}\).

2.6.3 Model Based Integration

A quite different approach to integration is that of model based methods. Abstracting virtual and physical aspects of the development process has

\(^{17}\)Microsoft, www.microsoft.com
\(^{18}\)Process algebra can be described as ways of modeling system behavior through math
\(^{19}\)Annotated Term
become increasingly popular during the past decade to raise the overall software quality, but also as an attempt to make the development process comprehensible for the increasing number of individuals who might lack the semantic or technical knowledge needed to fully comprehend the process. UML, is to a large extent, credited for much of the gained popularity and is today a natural part of the development process [5].

When using modeling, or rather meta models, in the context of integration it provides a visual basis for the common framework or artifacts to be shared. The meta model acts like a descriptive blueprint on how to transform data between tools. To carry out the actual transformation a semantic parser/translator is needed [5] which transforms the actual visual model representation to runnable programming code. Fig 2.7 illustrates a framework which takes input and output models from unrelated tools and by the help of a translator generator and a general mapping model creates a customized broker for each new integration, presented in [5].

2.6.4 Conclusion

Whether it be TOOLBUS or CORBA, the main approach to integration seem to be that of dedicated middleware. Brokers or translators work to translate semantics from one process to another, sometimes from language to language and sometimes using architecture specific syntax as a middle translator. These types of architectures rely to a large extent on the implementation of the translator, or the interface dictionary, and less on the actual developer using the service.

---

20 Unified Modelling Language
Figure 2.3: Wassermans integration stack as seen in [1]. The previously unmentioned Platform Integration refers to the set of system services that provide network and operating systems transparency to the tools above. Process Integration is left out from the stack due difficulties how to best integrate it with other tools.
Figure 2.4: A CORBA request from a client to the actual object implementation, mediated by the ORB

```java
interface Account {
    // Attributes to hold the balance and the name
    // of the account's owner.
    attribute float balance;
    readonly attribute string owner;

    // The operations defined on the interface.
    void makeDeposit(in float amount,
                     out float newBalance);
    void makeWithdrawal(in float amount,
                        out float newBalance);
};
```

Figure 2.5: CORBA IDL interface example
Figure 2.6: The TOOLBUS architecture. The picture illustrates how two tools communicate through the processes in the TOOLBUS via their individual adapters. Figure taken from [21].

Figure 2.7: A metamodel based transformation, as seen in [5].
Chapter 3

Open Services for Lifecycle Collaboration

3.1 Introduction to OSLC

In the previous chapter I made a short resume of some of the leading architectures used for data integration today. They all have in common that some sort of middleware, a translator or broker, is used to facilitate the interaction between processes or software. While the middleware makes the integration possible it also adds to the overall complexity of the interactions, which is unwanted.

A far less technical solution to the integration and cross-platform conundrum would be to create a common standard which could be used without the help of middleware software and usable on most systems without ad hoc solutions. While standardizing software communication sounds like the best solution "on paper" its implementation is far from easy and rather naive, in that it requires many users to adopt or conform to work properly. It requires either creating common grounds between the systems, or programming languages, by creating new protocols or finding common grounds to build the standard from. While being used constantly it is not often that the Web is recognized as a "universal" standard, or rather the technology facilitating it.

OSLC is a community, made up by software developers, which aims to use existing protocols to standardize the way tools share data with one another, specifically tools belonging to the Application lifecycle management sector. Founded in 2008 the initiative works in small work groups which focuses on creating specifications regarding a specific phase of the lifecycle, also called domains, see fig 3.1[16]. To ensure concurrency across all the domains the workgroups need to follow the core specification. At its current state the OSLC core specification is version 2.0, and so are the majority of the ALM group specifications. It should be noted that OSLC also specifies
standards for PLM\textsuperscript{1}, ISM\textsuperscript{2}, SPM\textsuperscript{3} [16] as well. This chapter explains the OSLC core and the technology used to facilitate it.

Figure 3.1: The figure illustrates different tools belonging to different ALM categories and how they are linked together within the category and to other categories via Linked data. [16]

### 3.2 OSLC Technical Foundation

In chapter 2 I briefly described Linked data and its core principles. Linked data is the foundation of OSLC. Everything that could be considered sharable, form an OSLC perspective, is called an artifact, e.g., a test case or a development plan. Each artifact is identified, and addressable, through its own unique URL. Artifacts can thus be called HTTP resources, and these can be manipulated using the standard HTTP actions GET, PUT, POST, DELETE. The third rule of Linked data reads:

\begin{itemize}
\item \textsuperscript{1}Product lifecycle Management
\item \textsuperscript{2}Integrated Service Management
\item \textsuperscript{3}Software Project Management
\end{itemize}
When someone looks up a URI, provide useful information, using the standards (RDF*)

OSLC uses RDF/XML to describe resources, which is also the most popular RDF notation. Lastly resources within the same domain reference each other, which adheres to the forth and final linked data rule[3][16]. OSLC separates those who want a certain resource and those exposing resources, Producers and consumers [16].

3.2.1 Metadata

In a previous chapter the issue with data exchange was briefly discussed. Metadata is a key concept in many computer science areas and is often explained as information about data. The W3C Director Tim Berner-Lee even goes so far as to say that, through a Web perspective, data without mapped metadata is "garbage" since it is impossible to index, because it is not machine readable [3]. In regards to integration, information about data is necessary to provide the context in which it was generated, IDL’s or the RDF standard are examples of ways of using metadata to facilitate integration and independence from platforms and programming languages[3][19].

XML/RDF

XML4 is a text encoding standard originally created by the W3C in attempts to meet the growing problem of publishing content on the Web. In short XML could be described as a way of standardizing organized data, like address books or financial records, to make web publication easier. Since XML is just another way of structuring data its not bound to any specific platform or software which makes it perfect for integration purposes, or sharing data between applications. XML is not a programming language, the syntax resembles that of HTML, but it should be noted that XML enforces strict syntax rules which if broken renders the XML document worthless5[11].

        While XML is a angle-bracket-and-slashes notation it is also a data model. Since XML is a way to organise structured data it can quite easily be illustrated graphically, which will in such cases create a tree like structure, with among other things, entities as nodes. XML provides no semantic meaning by itself and does not support relations outside the XML document itself. XML/RDF on the other hand is the RDF datamodel implemented in XML syntax. RDF represents a quite different datamodel than that of XML. In RDF typed statements are made about resources so called triples. The triples consists of a subject, predicate and object which combined together

---

4 Extensible Markup Language
5 The W3C specification "forbids" XML parsers to overlook syntax rules
form a statement. From a datamodel perspective these produces a relational graph. RDF uses URIs for all names of entities (resources), the names are global in scope, meaning they are references are the same regardless of which document they appear in [22].

```xml
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ok="http://www.banana.com/rdf/">
  <rdf:Description rdf:about="http://www.banana.com/"/>
  <ok:title>banana.com</ok:title>
  <ok:author>Curious George</ok:author>
</rdf:RDF>
```

Figure 3.2: A short example of XML/RDF: The first line is mandatory for XML documents and specifies the xml specification version used. The second line specifies the rdf namespace as the referenced url, this is mandatory when using RDF. Line 5 specifies our own created namespace "ok" while lines 7 and 8 specifies the title name and author name. If we were to parse this document our statement would output the result shown in graphical form.

### 3.2.2 CRUD

OSLC follows, as previously mentioned, REST and is based upon existing Web-architectures such as HTTP\(^6\). One of the most basic concepts of HTTP is that of CRUD. CRUD is an acronym for Create, Read, Update and Delete, four basic functions of so called persistent storage. In HTTP they are known as POST, GET, PUT, DELETE and in OSLC they are the primary means of manipulating or getting resources. The commands/requests are defined in the messages sent [8].

\(^6\)Hyper Text Transfer Protocol
3.2.3 Containers & Serviceprovider

Another previously unmentioned aspect, in regards to artifacts, is that of containers. All created artifact belong to a certain container, unrelated to which domain the tool producing the artifact adheres to. The containers offer a way to organize the artifacts into groups consisting of its peers.

OSLC organizes its resources in a ServiceProvider. It can be thought of as a container/artifact administrator, or as the container representation, which handles requests from consumers on how to find a certain resource or requests to create new ones, the ServiceProvider is a resource in itself [16]. Multiple ServiceProviders make up the ServiceProviderCatalog, fig 3.3 shows the the structure of a OSLC provider.

![Diagram of OSLC provider structure]

Figure 3.3: A visual representation of the OSLC provider structure [16]

3.2.4 Conclusion

OSLC is a different take on integration, from what has previously been discussed in this thesis. It does not use any middleware acting like a broker, instead it specifies standards for exposing resources. It is based on familiar architectures and concepts used on the Web, such as Linked data, REST(HTTP). Building OSLC from existing protocols has the advantage of
making it attainable by almost all users and easy to understand since OSLC
does not add any new protocols or syntax, other than specifying structures
and attributes of certain OSLC “actors”. However it should be mentioned
that tools need to support OSLC natively in order to work middleware free
[16], for other cases ad hoc adapters are needed, a highly developer depen-
dent task from a quality perspective.
Chapter 4

Implementation

The following chapter describes the task of implementing, designing, identifying resource sharing and the subsequent integration between the tools Farkle and Enea Optima using OSLC. Farkle is categorized as a Test Execution tool while Optima is a debugger for the real-time operating system OSE. The integration provides means for users to execute tests and depending on the outcome analyze target parameters or use debug functionality to find and correct errors. The data being shared by the tools are test execution results. Figure 4.1 illustrates a typical usecase.

Figure 4.1: A user chooses and executes a test case, if the test were to fail an EventTrace is set to target and the test is reexecuted.
4.1 Optima Overview

Optima is a Tool Suite targeting systems running the ENEA OSE realtime operating system. It is based on the Eclipse Platform and the Eclipse CDT (C/C++ Development Tools\(^1\)).

Optima contains an OSE-aware command-line source code debugger, based on the GNU debugger\(^2\), which has been adapted by adding special features for the Enea OSE Real-Time Operating system. The GDB for OSE is also used by the Optima Eclipse plugins. Optima contains a set of Eclipse plugins that provide support for launching and run-mode debugging OSE core modules and load modules. Optima supports viewing, profiling, and manipulating of OSE target systems. In order to perform any debugging, Optima needs a run-mode monitor and an OSE Gateway server on target.\(^{[24]}\)

See fig 4.2 for a flow chart on the how optima communicates with an OSE target from a host system.

4.2 Farkle Overview

Farkle is a Test Harness tool which runs on a host machine, Linux/Unix, for testing applications running on a target machine, running ENEA OSE. Farkle tests are written in the programming language Python and execute just like ordinary Python unit tests would, with the difference that Farkle test scripts make use of the Farkle library and functions which enables the proxy execution\(^{[25]}\).

4.2.1 Alternative ways of implementation

The three main approaches to implementing OSLC are called Native, Plugin and Adapter. While the plugin and the adapter approach require a certain amount of knowledge when it comes to the tools source code the adapter approach only requires knowledge of the tools API. For the scope of this thesis only the adapter approach has been used. The adapter approach could be described as middleware between tools API and a OSLC front end. A negative aspect of using the adapter approach is that a developer is limited to the functionality provided through a tools API, which depending on the tool could be a bad thing if necessary functionality is not supported. Arguably the biggest benefit with choosing the adapter approach when integrating tools is as previously mentioned the amount of freedom the developer has over the design, table 4.3 gives a short summary over the different ways of implementing OSLC and their respective advantages and disadvantages.

\(^{1}\)Eclipse CDT, http://www.eclipse.org/cdt/
\(^{2}\)Gnu Project Debugger, http://sources.redhat.com/gdb/
the relationship between the host system running Optima and the OSE target as well as the relationship between the native eclipse kernel and plugins together with the Optima tool suite plugins. Taken from [24].

4.2.2 Overview of Farkle’s OSLC Provider Adapter

The Farkle Adapter is divided into the three separate areas ResourceStorage, TestMechanics and Web interface.

- **TestMechanics**: This area comprises functionality which is related to interaction with the test tool Farkle.

- **ResourceStorage**: Since Farkle by itself is nothing more than a command-line based tool which takes and input and gives an output there is a need to create a context which gives more meaning to the resources, other than being test files. This area encapsulates results in order to create that context while also producing a persistent way of storage.

- **Web interface**: An OSLC provider is in short a REST server which lets OSLC consumers, REST clients, request its resources via http requests. The web interface comprises all of the named functionality.
<table>
<thead>
<tr>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| Native approach  | • Gives you complete control over the quality implementation  
                  • Good approach for tool vendors shipping products with OSLC support | • You need control over the code if you want to do this  
                  • Not a good approach for customers who want to add OSLC support to a vendor’s products  
                  • You need to learn product’s language and platform |
| Plugin approach  | • Uses established and supported mechanism to extend product and add OSLC support | • Limitations on plugins may limit quality of OSLC implementation  
                  • You need to learn product’s language and platform |
| Adapter approach | • Can be implemented without modifying product being OSLC-enabled  
                  • You can use your favorite platform and language | • Limitations of product’s API may limit quality of OSLC implementation  
                  • May introduce redundant URL for product resources, e.g. adapter provided dub URLs must be used instead of native Bugzilla bug URLs |

Figure 4.3: Pros and cons of using different methods of implementing OSLC[16].

4.2.3 Resource Sharing

OSLC specifies resource types within all of its specifications. The only resource available for sharing is the output from an executed test, a test result. The ALM category Quality Management specifies the fundamental principles of a TestCaseResults resource, which maps to the needs of Farkle’s test output.

4.2.4 Overview of Optima’s OSLC Consumer Adapter

Push and Pop are two concepts related to server client communication. The two concepts describes the role that the server plays in relation to the client. Push communication relies on the server to send data to clients while Pop based communication relies on clients requesting data from the server. OSLC only supports server client communication of type Pop.

The design of the Optima adapter is limited to a single java class named OptimaConsumer.java which handles both the communication with the Farkle
rest server as well as invoking appropriate response from the Optima API.

4.2.5 Flexibility of the Design

As with all OSLC implementations there are no hard dependencies meaning that new tools need only to configure the URL’s to which they need to connect in order to request a resource. Some connection may require extra information or attributes about the available resources, this will however not affect any other connected tools. Furthermore, since OSLC uses nothing but widely adopted architectures such as REST and XML there are no restrictions in regards to the type of frameworks used to accomplish the connection as long as it follows the given standard.

4.3 In depth description of the Farkle adapter

The following section specifies in a detailed fashion the different components in the actual implementation.

4.3.1 TestMechanics

This category consists of 3 main classes: Execution, TestResource and interface linker (.java). The Execution class creates a runtime object based on a given command which is then executed, it is initially also used to parse python test cases and later to execute a selected test case. The TestResource class is the structure of a test case(-result) resource, these objects are created once a python unit test file has been parsed and then later updated after the execution result is done. The interface linker class serves as a middleman between the integration GUI and the execution class, appending in a correct way before execution.

4.3.2 ResourceStorage

TestCaseDB.java together with an Apache Derby database\(^3\) comprises this category. The TestCaseDB class provides basic SQL functionality through a java database driver. The database holds only one table which carries the attributes of the testcases parsed and executed. The database is made available trough a network interface and can be connected to via URL.

---

\(^3\)Apache Derby, http://db.apache.org/derby/
4.3.3 Web Interface

The Web Interface category consists of Apache Jetty webserver\(^4\) and three servlets with corresponding Java Service Pages\(^5\) (.jsp) files. Users connecting to the webserver, or rather the web app \texttt{http://localhost:8080/frakleadapter/}
will be redirected to the serviceprovidercatalog servlet, mapped to baseurl/-catalog. The servlet dispatches its corresponding .jsp file which executes its embedded scripts before being handed over to the client. The serviceprovidercatalog holds necessary information about how to reach the serviceprovider which in turn holds information about the test result resources themselves.

4.3.4 Optima OSLC consumer adapter and GUI

Optima’s OSLC adapter is fairly simple. It uses a framework for rest services named Jersey\(^6\). By creating a Jersey REST client the optima consumer can connect to the farkleadapter webapp and the test case resource descriptions. The test descriptions status attribute, which denotes if the executed test case failed or passed determines if the Optima functionality should be used. If the testcase failed a method which uses the Optima plugin API connects to the target and enables a traceaction to provide information when rerunning the test.

---

\(^4\)Apache Jetty, http://jetty.codehaus.org/jetty/

\(^5\)Java Service Pages (jsp) lets one include java code in a xml or html document to automatically create dynamic pages for a user.

\(^6\)Jersey, http://jersey.java.net/
4.4 Test Cases

A collection of tests were specified and verified in order to validate functionality.

4.4.1 Basic Functionality

The basic functionality was tested by conducting the series of actions needed to execute a testcase. After failing the test execution the user chooses to reexecute the same testcase but with an eventtrace enabled, courtesy of the Optima plugin API. While some optima functionality still is left to implement the test shows that the fundamental integration works. Several smaller tests were also constructed in order to verify the test execution itself. These tests involved executing python unittest cases/methods but also whole python files, theses were all successfully executed.

4.4.2 Verifying OSLC Web Structure

To verify the total structure of the Farkle adapter a testcase was constructed which consisted of a series of user driven actions. The user enters the root URL and should through the given information in XML/RDF form be able to make his way down to the actual final resource. This testcase was successfully executed by a user.

4.4.3 Verifying Database Concurrency

The implementation uses a database to create a resource context, instead of just using plain files which has no real relationship with the test tool. Three different tests were constructed to verify database concurrency. The tests involved making database transactions by executing testcases and parsing test files at the same time as the optima consumer tried to request the same resources from the database. The test results were successful.
Chapter 5

Conclusion, Discussion and Future Work

5.1 Conclusion

OSLC may seem simple and lightweight but when the integration only includes two tools the overhead of using many different frameworks and proprietary technology creates a lot of work which could have been done in other more direct ways. A positive thing about the adapter approach to integration is how modular the solution is. Both of the adapters created in this thesis could be stripped down and serve as templates for other tools, independent of the domain.

In a more practical sense OSLC should be enforced when dealing with medium to large scale size tools. It should be noted that OSLC is not a method of integration rather a concept for making resources available in an efficient way. This way of reasoning helps one decide if using OSLC is really a must. In other words is there a need for integrating features or is there a high enough demand for a specific resource which requires OSLC sharing.

The following questions were presented in the introductory chapter as the starting point for the thesis, these are the answers derived from writing the thesis.

Question 1:  *How should tools broadcast their services? & How do different nodes (tools) discover one another?*

Services or rather OSLC resources are grouped together in containers by their respective use, this simplifies the task of finding similar resources. As Serviceprovider catalogs are collections of many service providers it is fairly simple to traverse into other domains and discover other resources.

Question 2:  *Does OSLC help in identifying artifacts likely to be shared?*  OSLC specifies a number of artifacts and their respective
base structures within each published specification. While not helping the developer directly it might help indirectly to map already specified resources to an actual tool resource.

Question 3: **Does the OSLC standard limit the developer with respect to information being shared?**

The meta structures defines a core set structure which one must follow in order to conform to OSLC in a correct way, but this does not hinder developers from extending the structure to include details specific to their software. Since OSLC defines actual resources and one could argue that there are restrictions as to which resources actually may be shared.

Question 4: **In what format should artifacts be presented according to OSLC?**

OSLC dictates the minimum use of XML/RDF but puts no restrictions on additional meta notations such as JSON1, Turtle2.

Question 5: **To what extent does a system relying on an open-tool-chain standard need complementary software to function (eg : databases, servers)?**

From a minimal dependency point of view implementing an OSLC producer requires a Web server which houses the servlets which dispatches the actual meta structures in XML/RDF format. A REST client is all that is needed at the consumer side. As for general open-tool-chain standards which relies on a broker/middleware needs a server node which handles translation between the different software.

---

1 JSON, http://www.json.org/
5.2 Discussion

In this section I present some of my own thoughts and ideas regarding OSLC based on my own experience from this master thesis.

In its current state all of the ALM related OSLC specifications are finalized, and have been for almost 8 months. There does however still exist a shortage in regards to implementations and specifically of the open source or reference type. An OSLC SDK\(^3\) is in the works but is currently, as of June 2012, behind schedule. This has indirectly lead to very few implementations, no where is this more evident than in OSLC’s own software section. Most of the current implementations could be attributed to either IBM Jazz\(^4\), Task top Synch\(^5\) or KOVAIR omnibus\(^6\), three big companies which all provide big ALM PLM systems with little transparency [16]. While this is a big issue for developers which currently are trying to conform to OSLC standards one needs to think about the nature of the project, which is essentially a open standard with no really incentives such as a monetary gain.

It’s evident that neither the SDK or the other OSLC specifications is being rushed, one could however question if they are overworked. The amount of namespaces and details related to a single domain, or resource, is quite intimidating. I would suggest that the specification should be reworked in order to minimize boilerplate code\(^7\), especially code belonging to the ServiceProvider. I would also suggest that the OSLC primer should be extended with a more detailed overview of the relationship between the ServiceProvider and the ServiceProviderCatalog while also including real life system references which could aid novice developers.

It is also interesting to note that OSLC is scenario driven in the way it has been specified. This essentially means that scenarios outside the current scope are not supported, or rather developers finding themselves outside the scope will have to develop workarounds in order to successfully conform to the standard.

\(^3\)Software Development Kit
\(^5\)Task top sych, http://tasktop.com/support/new/index-sync20
\(^7\)Boilerplate code: Code which have to be included in many places with little or no alteration
5.3 Future Work

This thesis has focused on point-to-point integration, via OSLC, with a clearly specified goal. It would be of interest to further research how different tools from different domains would work together. Specified but not yet implemented is the area of automation, or as described in the previous theory chapter as control. As control is one of the essential core features of any type of integration further research in how control is implemented through http requests and the implications this has would have would possibly be an attractive topic of any future work.

As for the implementation done in this master thesis more work could be done to finalize the integration itself, namely regarding the presentation of the eventtrace which is started together with a failed test. As for OSLC features there are several which has been left unimplemented, due to lack of documentation and time constraints. OSLC query capabilities and secure authentication via OAuth\(^8\), to mention a few.

A complete redesign of the adapter to instead be automated and thus not require any connection from a GUI, since operations then are handled through the requests themselves. The functionality these tools share might be a bit to farfetched for a use-case implementation such as this. Connecting a bug reporting system to the test tool could provide more understandable results, or rather more evident.

It should also be noted that the current implementation suffers from an array of bugs which is a continuous area of future work.

\(^8\)OAuth, http://oauth.net/
Bibliography


