

# **Bachelor Thesis**

## **Building a Distributed Rule Translator System as a Web Service**

## **Entwicklung eines verteilten Regelübersetzungssystems als Webservice**

**Marco Pehla**

---

# **Bachelor Thesis: Building a Distributed Rule Translator System as a Web Service: Entwicklung eines verteilten Regelübersetzungssystems als Webservice**

Marco Pehla

Copyright © 2007 Brandenburg University of Technology at Cottbus, Faculty of Computer Science, Chair of Internet Technology

## **Abstract**

R2ML is a XML based markup language developed for the rule interchange. In this thesis, on each other depending technologies are introduced, that allow to build a distributed rule translator system as a Web Service. With the help of the Model Driven Architecture (MDA) approach, the whole system is modeled in UML 2.0 class, use-case and sequence diagrams. In order to fulfill the complete process of the Model Driven Architecture (MDA) approach, one specific implementation of the system is explained in detail in this thesis. Therefore the Java Enterprise Edition 5 was chosen as middleware platform. In this particular case, the JBoss application server with Enterprise JavaBeans 3.0 support, is the concrete middleware platform. Stateless Enterprise JavaBeans 3.0 are playing one central role in the implementation of the system and were chosen to deploy the system as a Web Service. In the introduced implementation are XSLT 2.0 files used to translate between the rule languages. The integration of Saxon 8, as XSLT processor that supports XSLT in version 2.0, was therefore necessary and is explained in detail. Moreover, a web interface for the developed Web Service, written in PHP 5, is introduced as example client as well.

## **Zusammenfassung**

R2ML ist eine XML basierende Abbildungssprache für den Austausch von Regeln. In dieser Arbeit werden voneinander abhängige Technologien vorgestellt, die es ermöglichen ein verteiltes Regelübersetzungssystem als Webservice zu entwickeln. Mit der Hilfe des Ansatzes der modelgetriebenen Entwicklung (MDA), ist das komplette System in UML 2.0 Klassendiagrammen, Geschäftsprozessdiagrammen (use-case Diagrammen) und Sequenzdiagrammen modelliert. Um den kompletten Prozess des modelgetriebenen Entwicklungsansatzes zu vervollständigen, ist eine spezielle Implementierung des Systems in dieser Arbeit detailliert erläutert. Dafür wurde die Java Enterprise Edition 5 Plattform als Middleware ausgewählt. In diesem speziellen Fall ist der JBoss Applikationsserver mit Unterstützung für Enterprise JavaBeans 3.0 die konkrete Middleware Plattform. Zustandslose (stateless) Enterprise JavaBeans 3.0 spielen eine große Rolle in der Implementierung des Systems und wurden ausgewählt um das Regelsprachenübersetzungssystem als Webservice zur Verfügung zu stellen. In der vorgestellten Implementierung werden XSLT 2.0 Dateien zur Übersetzung zwischen den Regelsprachen benutzt. Die Integration von Saxon 8 als XSLT Prozessor, welcher XSLT in Version 2.0 unterstützt, war notwendig und ist daher detailliert erklärt. Des weiteren wird ein Beispielclient in Form einer dynamischen Webseite für den entwickelten Webservice eingeführt. Dieser wurde in PHP 5 geschrieben.

---

---

---

---

# Dedication

This thesis is dedicated to my family for their patience with my study.

---

# Table of Contents

Acknowledgements .....	viii
Foreword .....	ix
1. Conventions Used in This Document .....	ix
1. Plan of the Thesis .....	1
1. Task Description .....	1
2. Solution Approach .....	1
2. Introduction .....	3
1. Rule Languages .....	3
2. Web Services .....	7
3. SOAP .....	8
3. Design .....	12
1. Safeguard the System .....	12
2. Source Control System .....	13
3. Distribution of Responsibilities .....	14
4. Model Driven Architecture .....	16
4.1. Domain Model .....	17
4.2. Design Model .....	19
5. Use Cases .....	24
6. Sequence Diagrams .....	26
6.1. Translation of a Rule .....	27
6.2. Reception of all Source Languages .....	29
6.3. Reception of all Target Languages .....	30
4. Implementation .....	32
1. Java Programming Language .....	32
2. Application Server .....	33
3. Implementation Model .....	35
3.1. Enterprise JavaBeans 3.0 Technology .....	39
3.2. Enterprise JavaBean Container .....	39
3.3. Choice of the Interchange Language .....	40
3.4. Management of Translations .....	43
3.5. Explanation of the Source Codes .....	46
5. Web Interface .....	74
1. Design Hints .....	75
2. Using Web Services with PHP .....	79

2.1. SOAP Exception Handling .....	82
2.2. SOAPClient Bugs .....	83
6. Conclusion .....	85
1. Extensions of the System .....	85
A. ....	87
B. ....	89
Bibliography .....	90

---

## List of Figures

2.1. R2ML Tree View: Ancestor Relation .....	7
3.1. Distribution in the Rule Translator System .....	15
3.2. Domain Model: Rule Translator System .....	18
3.3. Design Model: Rule Translator System .....	20
3.4. Illustration of Source and Target Languages .....	22
3.5. Illustration of Interchange Languages .....	23
3.6. Use Case Diagram: Client (Dynamic Website) .....	25
3.7. Use Case Diagram: Web Service .....	26
3.8. Sequence Diagram: Translation of a Rule .....	28
3.9. Sequence Diagram: Reception of all Source Languages .....	30
3.10. Sequence Diagram: Reception of all Target Languages .....	31
4.1. Multitiered Applications [BCE+] .....	34
4.2. Implementation Model: Web Service .....	37
4.3. Packaging of the Translator Web Service .....	60
4.4. Packaging of an Enterprise Archive .....	61
4.5. ANT Integration in the Eclipse IDE .....	64
4.6. ANT Console Output in the Eclipse IDE .....	65
5.1. R2ML Translator Web Interface .....	75
5.2. Pop-up Blocker Resistant Message Window .....	77

---

## List of Tables

5.1. Displays and Resolutions .....	76
-------------------------------------	----



---

## List of Examples

2.1. Simple if-then Rule .....	3
2.2. Rule in Object Oriented Pseudo Code .....	3
2.3. F-Logic Rule: Ancestor Relation .....	4
2.4. SOAP Request Message .....	9
2.5. SOAP Response Message .....	10
2.6. SOAP Fault Message .....	11
4.1. Lack Markup in RuleML .....	40
4.2. Strict Markup with SWRL Built-Ins and XPath Functions .....	41
4.3. Strict Markup in R2ML: "previous year" .....	42
4.4. Strict Markup in R2ML: "customer spending a minimum of 5000 previous year" .....	43
4.5. XML: Translation Descriptor .....	44
4.6. XML Schema: Translator Descriptor .....	45
4.7. Java Annotations: Remote Interface .....	46
4.8. Stateless Bean Web Service .....	48
4.9. Java Annotation: @WebService .....	49
4.10. Java Annotations: Alternative Business Interface Declaration .....	50
4.11. Java Annotation: @WebMethod .....	51
4.12. Java Annotation: @WebParam .....	51
4.13. Java Annotation: @SOAPBinding .....	53
4.14. Java Annotation: @Resource .....	55
4.15. Deployment Descriptor: ejb-jar.xml .....	57
4.16. throwing of a SOAPException .....	59
4.17. ANT build.xml File .....	62
4.18. R2ML ReferencePropertyAtom .....	66
4.19. XSLT Snippet of the R2ML to F-Logic Translation .....	66
4.20. Usage of xsl:message .....	69
4.21. Implementation of the translateDirect() Operation .....	70
5.1. CSS for HTML textarea element .....	76
5.2. Pop-up Blocker Resistant Message Window .....	78
5.3. PHP Web Service Call: getSourceLanguages() .....	79
5.4. WSDL: Response Type Declaration .....	81
5.5. PHP Web Service Call: getTargetLanguages() .....	82
5.6. PHP: SOAP Exception Handling .....	83

5.7. WSDL: Parameter Mapping of an Operation .....	84
A.1. R2ML Rule: Ancestor Relation .....	87

---

# Acknowledgements

First and foremost, I would like to thank Dr. Adrian Giurca, for his invaluable directions and support throughout my research efforts towards this thesis. I would like to thank as well Prof. Dr. Gerd Wagner for his good support and permission to write this thesis in English language and all other people, who always believed and trust in me and my abilities.

---

# Foreword

## 1. Conventions Used in This Document

The following typographical conventions are used in this document.

`Constant width` is used for source code examples, fragments, XML elements and tags.

*Emphasis* is used to mark special things in the content and for the quotation of reference titles.

---

# Chapter 1. Plan of the Thesis

## 1. Task Description

In rule base applications, business rules are expressed with the help of executable rule languages like JBoss Rules or Oracle Business Rules. One and the same rule has a different form in every rule language. In case of a rule platform change, it takes huge efforts to translate all existent rules manually to the new rule language. In this thesis a system based on the rule markup language R2ML should be developed, that supports the translation of the most important rule languages. This system should be implemented as a Web Service.

Geschäftsregeln werden in einer regelbasierten Applikation mit Hilfe einer ausführbaren Regelsprache wie z.B. JBoss Rules oder Oracle Business Rules ausgedrückt. Dieselbe Regel nimmt dadurch in unterschiedlichen Sprachen unterschiedliche Formen an und es ist sehr aufwendig, z.B. bei einem Wechsel der Regelplattform alle Regeln manuell in die Sprache der neuen Plattform übersetzen zu müssen. In dieser Arbeit soll ein Regelübersetzungssystem, das die wichtigsten Regelsprachen unterstützt, auf der Basis der Regelauszeichnungssprache R2ML entwickelt und als Webservice implementiert werden.

## 2. Solution Approach

The main goal during the development of the rule language R2ML was not only to develop another new rule language, the main goal was moreover to develop a language for the rule interchange. Therefore R2ML need to cover all important informations of other rule languages. Thus makes R2ML the ideal language for the interchange of rules in a translator system.

Web Services are self-describing and platform independent functionalities, that can be used over a network. They are easy to integrate in already existing applications and a good choice for a public rule translation system interface.

The solution of the rule translation system, introduced in this thesis, use a the Java Enterprise Edition 5 middleware platform to deploy the Web Service with the help of Enterprise JavaBeans 3.0. The Model Driven Architecture (MDA) approach is used to

model the complete system. Use case and sequence diagrams helping to describe the most important parts of the system with appropriate diagrams.

---

# Chapter 2. Introduction

The Chair of Internet Technology at the Brandenburg University of Technology at Cottbus is member of the REVERSE<sup>1</sup> project. Inside of the work package I1, one goal was to develop a rule interchange language based on XML<sup>2</sup>. To prove that a new technology fulfil the purpose for which it was developed, a good opportunity is to build systems which take advantage of it. Therefore, the Chair of Internet Technology felt the decision to build a rule translator system that use the new rule interchange language R2ML. In my thesis I want to describe what it need to *Building a distributed Rule Translator System as a Web Service*.

## 1. Rule Languages

Rules are used to specify behaviour. Every rule has a condition and conclusion part. In computer programs, rules are often written with the help of if-then statements.

The Example 2.1, “Simple if-then Rule” is taken from the Product Derby presentation paper of the UServ case study delivered during the 9th International Business Rules Forum in Washington, DC in 2005. The introduced UServ case study is focused on vehicle insurance products and their modelling with the help of business rules.

### Example 2.1. Simple if-then Rule

**If** the car's price is greater than \$45,000, **then** the car's potential theft rating is high.  
[BRF 05]

In object oriented programming languages, the same rule can be expressed like in Example 2.2, “Rule in Object Oriented Pseudo Code”.

### Example 2.2. Rule in Object Oriented Pseudo Code

```
if(car.getPrice() > 45000) {  
    car.setPotentialTheftRating = "high";  
}//end-if
```

---

<sup>1</sup><http://www.reverse.net>

<sup>2</sup>eXtensible Markup Language

In the paper of the UServ case study, you can find far more complex rules, grouped in rule set like *Automobile Eligibility*. All rules in this rule set, establish the eligibility for a car. Another set of rules define the eligibility categories for drivers and more other things.

In order to express such rules or rule sets, it is not easy to define them in programming languages. These languages are simply not designed to markup business rules. This is laying in the duty of rule languages. Rule languages give us the opportunity to express business rules in a syntax specified on rule markup. When the content of business rules is marked up well in a rule language, then this rule is understandable by both, machines and humans.

Common used rule languages are e.g. F-Logic, Jess, Jena, JBoss Rules, RuleML or Oracle Business Rules.

### **Example 2.3. F-Logic Rule: Ancestor Relation**

```
FORALL ?X, ?Y, ?Z ?X[ancestor->?Y] <- ?X[father->?Z]
                                AND ?Z[ancestor->?Y].
```

In natural language the Example 2.3, “F-Logic Rule: Ancestor Relation” could be written in one simple sentence. *If X has a father Z and this Z has ancestors Y, then X has ancestors Y.* The rule language F-Logic is used for this example . The original example was published in the tutorial *How to Write F-Logic Programs* [ONTOPRISE 04]. According to the presentation of Martin Weindel, *F-Logic Forum: Results and Open Issues left* [WEINDEL 06], the syntax of F-Logic has changed a little bit. The question mark sign should now be written in front of variables in order to distinguish them. All double arrows are dropped from the syntax. Multi-values, previously written with the two arrow operator, are now written with the single operator. As you can see, the Example 2.3, “F-Logic Rule: Ancestor Relation” , is already written in the new syntax.

To create a rule you need to know the semantics of all components of the language. Therefore we break down the Example 2.3, “F-Logic Rule: Ancestor Relation” to her components and try afterwards to build the same rule in another rule language with these components step by step from scratch.

### **Break Down into Components**

- All variables that appear in the rule are X, Y and Z. As mentioned before, they are now written in the new syntax of F-Logic, with a question mark sign in front of their



name. This makes it pretty easy to find and distinguish them. Usually in object oriented languages and in the most other rule languages, we would call the F-Logic variables not variables, we would call them objects. Therefore we call F-Logic's variables in the following *object variables*.

- The object variable `X` has a property `ancestor`. This property value is at a short gaze the object variable `Y`. But when we think object oriented, then it is impossible that the object variable `Y` itself is the value of a property. Therefore the value of the property `ancestor` have to be the *reference* to the object variable `Y`. We will see later why this makes an important difference.
- The left orientated arrow `<-` symbolize that statement left at his side is the conclusion of the F-Logic rule.
- Next to the right side of him are the condition statements written. In this case the built-in AND express that both conditions need to be evaluated to true when the rule should apply.
- The object variable `X` has a property `father` with a reference to the object variable `Z`.
- The object variable `Z` has a property `ancestor` with a reference to the object variable `Y`.

### **Building the R2ML Rule from Scratch**

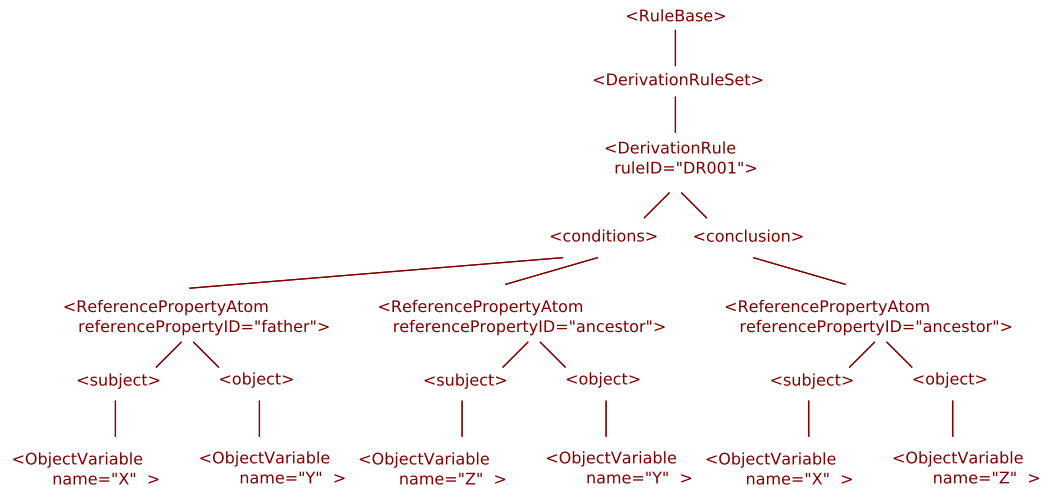
1. Every R2ML rule has a root element called `RuleBase`.
2. Elements of the abstract type `RuleSet` can be added to the `RuleBase` element. Since the object variable `X` need to derive a reference property in the rule later, we choose here the type `DerivationRuleSet`. It is also possible to add more and other kinds of `RuleSet` types to the `RuleBase` element. A `DerivationRuleSet` is a set of rules of the type `DerivationRule`.
3. As already explained, we add an element of the type `DerivationRule` to the `DerivationRuleSet`. The value of the `ruleID` attribute of the `DerivationRule` element is `DR001`, since it is the first derivation rule in the set.
4. Every derivation rule has a `condition` and a `conclusion` element.

5. We remember we had three reference properties in the original rule. Two in the condition and one in the conclusion part. Every previously called statement is called an atom in R2ML. We use here the `ReferencePropertyAtom` element for the markup. Since in R2ML are several different kinds of atoms possible at this point, we need to chose the right kind of atom here in order to markup the rule in the right way. We add the `ReferencePropertyAtom` with the attribute `referencePropertyID` and the value `father` and a second `ReferencePropertyAtom` with the value `ancestor` as child elements to the condition element. We add as well a `ReferencePropertyAtom` with the value `ancestor` for the `referencePropertyID` attribute as only possible child to the conclusion element.
6. Every `ReferencePropertyAtom` has the nested elements `subject` and `object`.
7. The `subject` element of the `ReferencePropertyAtom`, which is a child element of conditions, with the value `father` as `referencePropertyID` need to have an `ObjectVariable`, with the attribute value `X` as name, as child element. The `object` element has the `ObjectVariable` with the name `Y` as child element.

The `subject` element of the `ReferencePropertyAtom`, which is a child element of conditions, with the value `ancestor` as `referencePropertyID` need to have an `ObjectVariable`, with the attribute value `Z` as name, as child element. The `object` element has the `ObjectVariable` with the name `Y` as child element.

The `subject` element of the `ReferencePropertyAtom`, which is a child element of conclusion, with the value `ancestor` as `referencePropertyID` need to have an `ObjectVariable`, with the attribute value `Z` as name, as child element. The `object` element has the `ObjectVariable` with the name `Z` as child element.

The structure of an XML file is very similar to the structure of a tree. The Figure 2.1, “R2ML Tree View: Ancestor Relation” shows in a simplified tree the absolute necessary elements that build the R2ML rule. Every single step of the build process correspond to one hierarchy layer in the tree view. The complete R2ML rule in XML format can be found in the Appendix A, of this document.

**Figure 2.1. R2ML Tree View: Ancestor Relation**

Different rule languages use often a similar concepts to markup the content. This gives us the possibility to translate between rule languages.

For further informations about F-Logic, the tutorial *How to Write F-Logic Programs* [ONTOPRISE 04] is worth reading. R2ML is a rule interchange language and could be used for both, as an intermediate format and as a standalone rule language. What the attributes of an interchange language really are and how this language is used by the translator system is explained in Chapter 3, *Design*.

## 2. Web Services

Web services are not services provided by the web, which is often the synonym for the world wide web also known as the Internet. Web services does not describe websites where people use services to buy or sell things or inform about topics in an Internet lexicon. The the World Wide Web Consortium (W3C) published in the document *Web Services Architecture* [BHM+ 04] following definition.

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL<sup>3</sup>). Other systems interact with the Web service in a manner prescribed by its description using SOAP<sup>4</sup> messages, typically conveyed using

<sup>3</sup>Web Service Description Language

<sup>4</sup>Simple Object Access Protocol

HTTP<sup>5</sup> with an XML<sup>6</sup> serialization in conjunction with other Web-related standards.” [BHM+ 04]

An imaginary web service may provide a weather forecast. You send him your location and receive the forecast as text. Another web service may provide you the actual exchange rate to your local currency when you send him the currency of your favoured country.

Every web service is completely described by his web service description catalogue (WSDL). The web service description catalogue (WSDL) “...defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. In essence, the service description represents an agreement governing the mechanics of interacting with that service.” [BHM+ 04]

Web services are accessible by their endpoint URL<sup>7</sup>. The web service description catalogue of the web service, introduced later in this document, is accessible with an ordinary Internet browser. This web service use the HTTP protocol for the transfer of SOAP messages to communicate with clients of the web service.

### 3. SOAP

SOAP is the message format used for the communication with the web service, introduced later in this document. Especially, SOAP 1.1 messages are used to communicate with the clients. The the World Wide Web Consortium (W3C) published in the document *Simple Object Access Protocol (SOAP) 1.1* [DEK+ 00] the following description.

“SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially

---

<sup>5</sup>Hypertext Transfer Protocol

<sup>6</sup>eXtensible Markup Language

<sup>7</sup>Unified Resource Locater

be used in combination with a variety of other protocols; [...] SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC<sup>8</sup>.” [DEK+ 00]

In Example 2.4, “SOAP Request Message” you see the SOAP message that is send from a client to a web service. The SOAP message consist of a envelop that hold a header and an body element. In the following SOAP messages the header elements are not used and are therefore empty.

### Example 2.4. SOAP Request Message

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:ns1='http://r2ml/jaws'>
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:getTargetLanguages xmlns:ns1='http://r2ml/jaws'>
      <ns1:sourceLanguage xmlns:ns1='http://r2ml/jaws'>
        F-LogicXML
      </ns1:sourceLanguage>
    </ns1:getTargetLanguages>
  </SOAP-ENV:Body></SOAP-ENV:Envelope>
```

The operation wrapped in this SOAP request message can be written in a pseudo code language as `getTargetLanguages("F-LogicXML");`. The character string `F-LogicXML` is the content, or child text node, of the element `<ns1:sourceLanguage>`. He is as well the only parameter of the operation `getTargetLanguages`, which is also the parent element.

The SOAP response message to the request you see in Example 2.5, “SOAP Response Message”.

---

<sup>8</sup>Remote Procedure Call

**Example 2.5. SOAP Response Message**

```
<env:Envelope
  xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
  <env:Header/>
  <env:Body>
    <ns1:getTargetLanguagesResponse
      xmlns:ns1='http://r2ml/jaws'
      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
      <ns1:result>F-Logic</ns1:result>
      <ns1:result>JBossRules</ns1:result>
      <ns1:result>Jena2</ns1:result>
      <ns1:result>Jess</ns1:result>
      <ns1:result>R2ML</ns1:result>
      <ns1:result>RuleML</ns1:result>
    </ns1:getTargetLanguagesResponse>
  </env:Body>
</env:Envelope>
```

The response are many `<result>` elements. The text child nodes representing the content. In this case, the response of the operation `getTargetLanguages("F-LogicXML")`; are the result string characters `F-Logic`, `JBossRules`, `Jena2`, `Jess`, `R2ML` and `RuleML`.

In Example 2.6, “SOAP Fault Message” you see the a SOAP fault message which is the response from the web service to a request of the client. According to the message content, the client forgot to specify input data.

**Example 2.6. SOAP Fault Message**

```
<env:Envelope
  xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
  <env:Header/>
  <env:Body>
    <env:Fault
      xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
      <faultcode>env:Client</faultcode>
      <faultstring>
        r2ml.TranslationException: [ERR0001] No input data.
      </faultstring>
      <detail>
        <ns1:TranslationException
          xmlns:ns1='http://r2ml/jaws'
          xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
          <ns_message:message
            xmlns:ns_message='http://soap.xml.javax/jaws'>
            [ERR0001] No input data.
          </ns_message:message>
        </ns1:TranslationException>
      </detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

The content of the `<faultcode>` element shows that the client is responsible for the error. The content of the `<faultstring>` shows here the reason. A more detailed information can be found by looking at the `<detail>` element. The child element of `<detail>` is here a `<TranslationException>` element. This element has again a child element `<message>`, which has text content. All this tells us that the client created a fault, moreover a translation exception with the message `[ERR0001] No input data..`

A complete description of the SOAP 1.1 message format can be found in the document *Simple Object Access Protocol (SOAP) 1.1* [DEK+ 00].

---

# Chapter 3. Design

A good design is *the* crucial factor for developing professional software applications. In this phase you need to consider many different things and these depend mainly on the kind of application that need to be build. For the rule translator system the decision felt to separate the translations from the application. One reason for this decision was to shield the application for unauthorized access. In contradiction to the translations, the application needs rarely further development. Translations need, at least as often modifications as the languages for they are written for.

## 1. Safeguard the System

To shield applications for unauthorized access from outside should be pretty obvious and should always require the highest priority. That means you need to protect the system with a firewall when it is connected to an outside network like the Internet. Of course you need to change all default passwords of software that is accessible by others. It is also sometimes useful to prevent every access to administrator functionality from the Internet, when it is not absolutely necessary. There is also a need to shield the application from unauthorized access from inside your own network. Everybody who has access to parts of the system where he is not working on, should not become access.

I want to explain the last sentence more precisely with a little story. In the previous translator system which was build, every translation file lay in the same folder with a JavaServer Page. Actually this was not a good idea. The JavaServer Page was responsible to create a dynamic web appearance and to provide a translator functionality for each translation. One problem was that a colleague, which should work on a translation file, thought he might could change something in all JavaServer Page files as well. The system was designed in a way that every single JavaServer Page has a connection to a single JavaBean. This JavaBean has the mission to translate rule languages. Since the system was designed without access control, every member in the team could access everything. The game went so far, that the colleague copied the source code of the JavaBean code to a new file and changed afterwards the implementation. Then he deleted all Java Server Pages and used instead new files in the JavaServer Pages XML format. Nobody recognized all the changes inside of the source control system. With the next necessary and urgent build, no translation worked. Of course this day was the developer of the



system absent and nobody did made a backup of the source codes and of the last working translator application before.

It was doubtless not the intention of the colleague to destroy the translator system. The problem was he had less information about the system and furthermore *he got the possibility to access everything*. After he spoke with the developer of the system, it took him several hours to revert all changes and to repair the system. One big problem for instance was that he deleted all JavaServer Pages from the source control system when he created the new files in the JavaServer Page XML format. He was simply not aware of the consequences. When files inside of the source control system are deleted, they could not be restored. They are dropped from the underlying database. This is a very bad situation and negates completely the main idea of using a source control system. There is no chance to revert everything. In our case all JavaServer Pages needed to be restored from their not working XML version. During this process we found the reason for the malfunction of the translation system. The private development server of the team member was a different build and had another configuration. Our older to the Internet connected productivity server was not configured to support the newer XML format of Java Server Pages. The colleague that did the fateful changes wrote also some bugs in his changed implementation.

This little story illustrates very good that we need to think about roles and access control mechanisms when parts of the application are growing, are always under development and many people work on different parts with different responsibilities. It is often the situation that only the system developer really knows how all parts are related and depending on each other. Other team members that are responsible for translations do not need to know how the complete system work, they should only concentrate their translations to archive the best possible result. This is a kind of hierarchy where every team member has a role with another important responsibility. The system developer is responsible for a working system and reasonable interfaces for the translations. The translation developer has the responsibility for the translation and her correct mapping.

After the experience with the first translator system, I thought about how to avoid these kind of problems when some people doing other people's job.

## 2. Source Control System

For the development of software applications it is always a good idea is to use a source control system. But what is a source control system?

“Versioning, the most basic feature of a source-control system, describes the ability to store many versions of the same source file. [...] Once the file has been checked out, the user performs her edits on the file. When the user is happy with her changes (or just wants to checkpoint them), the file is then *committed* (or *checked in*) and updated in the repository. At this point, the repository knows about two versions of the file. Each version has some kind of unique identifier, called *revision number*. When a user checks out the file again, she is given the last revision.” [HENDERSON 06]

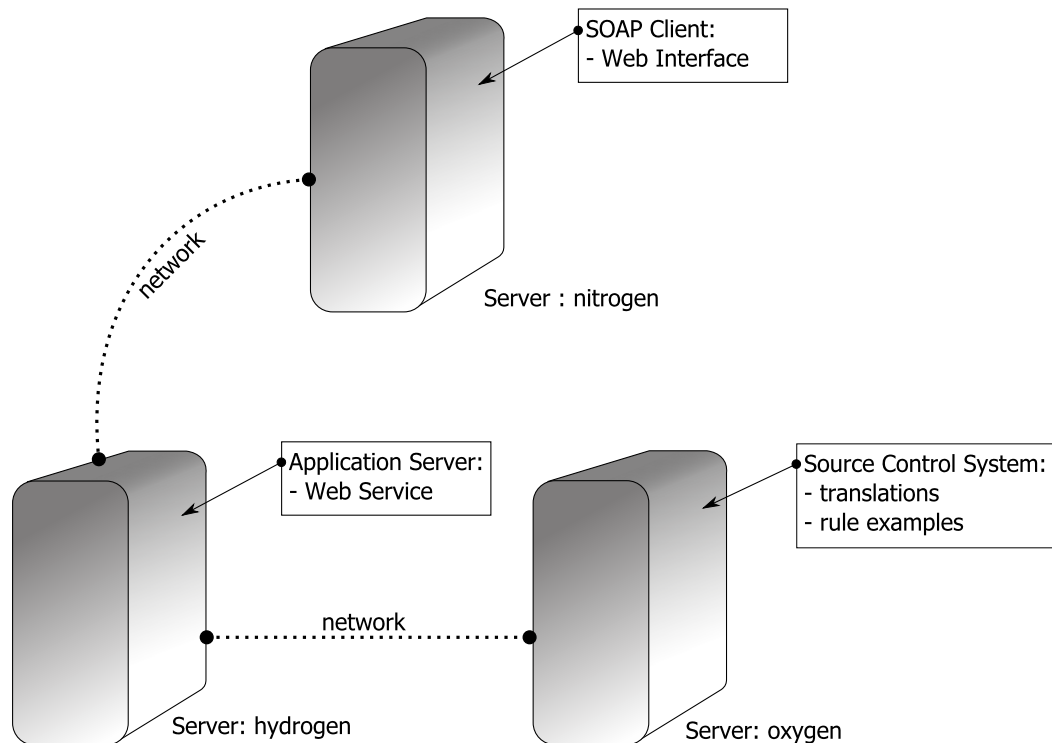
The source control system is an own independent system. Access rights could be assigned for each group or user. The folder access from a network e.g. the Internet is controlled by the source control system itself.

For the rule translator system a source control system is only used for the translation files and rule examples. This is a good idea since translations are changing sometimes and are developed by several users in contradiction to the translator itself. There is no access to the translator from the source control system possible. This gives translation developers the opportunity to concentrate on her assigned work and minimize the possibility to destroy the translator application by accident.

In the root of the repository inside of the source control system is a file that describes all re-checked and proper working translations. The translator only need to know where he can find the descriptor file. When the translator need to perform a translation he read the descriptor and knows afterwards where he can find and get the latest translation from the source control system in order to fulfil his task.

### **3. Distribution of Responsibilities**

Distribution of responsibilities to different servers is an approach which should be considered in the design phase. In the case of the distributed translator system the decision felt to distribute translations and their translator application. One reason was to strictly detach both things. That means there is an productivity server where the final applications are running and another development server were an source control system is installed. This development server could be used for the translations and other files which are in the development process. Even the source codes for the translator application itself can be managed in another branch of the source control system and secured by restriction of users or groups.

**Figure 3.1. Distribution in the Rule Translator System**

The distribution of responsibilities has the benefit to avoid an overloading of one single server. A machine where servers running for JavaServer Pages, Enterprise JavaBeans, PHP or a source control system could faster become overloaded than most people think. *How much memory are all server instances really need in a worst case? Has the processor enough speed? How can we scale up the system in future?* When responsibilities are distributed on different servers you fast find your bottleneck. Then you can investigate to speed up only the responsible server.

But it is not always necessary or even possible to use one dedicated machine for one server. Virtualisation has become a big impact in the area of informatics in the last years. It means to emulate many computers on a single machine. Every emulated computer is called a virtual machine and act like a real computer. Virtual machines have their own network cards and can communicate with other virtual or real machines. Today all modern computer processors have a hardware instruction set to support virtualisation in hardware. The chip vendor Intel call her hardware virtualisation technology Vanderpool or VT. His competitor AMD call it Pacifica or AMD-V. That all means a virtual machine

is not emulated and slow anymore, she is running with performance close to native hardware.

Another big trend today are multi-core processors. This makes it possible to assign processor cores to virtual machines by a mouse click. Thus makes a separation of services to virtual servers running on a single dedicated machine an opportunity, because you can specify how much memory and processor cores should be used for this very virtual machine.

Today, it is often the case that applications need to fulfil purposes for they where not laid-out before in there development stage. Since the translation of rules is an important topic in the research area of informatics, it is a good idea to dispatch a possibility to scale up the whole system in future if there is a need for it. An application server, which supports clustering is a possibility to easy scale up the whole system. It is obvious that only clustering of application servers which are installed on real machines make sense. To scale a virtual machine it is more reasonable to assign more resources like processor cores or memory. If this is not sufficient, the next step is the migration to a own dedicated machine.

## 4. Model Driven Architecture

From the website of the Object Management Group Inc. (OMG), the inventor of the Model Driven Architecture, comes the following definition.

“The MDA is a new way of developing applications and writing specifications, based on a platform-independent model (PIM) of the application or specification's functionality and behaviour. A complete MDA specification consists of a definitive platform-independent base model, plus one or more platform-specific models (PSM) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support.” [OMG]

Dr. Jon Siegel, the vice president of technology transfer at Object Management Group Inc. (OMG) explains in his article *Making the Case: OMG's Model Driven Architecture* [SIEGEL 02] the MDA development process in more detail.

John Hogg from IBM wrote on a slide in his presentation *Brass Bubbles: An Overview of UML 2.0 (and MDA)* [HOGG 03] the following statement. “Software has the rare

property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods! => This ensures perfect accuracy of software models; since the model and the system that it models are the same thing ” The most important conclusion of Mr. Hogg was that “The model is the implementation”. Hogg called models useful if they fulfil five characteristics.

### **Characteristics of Useful Models**

- **Abstract**

Emphasize important aspects while removing irrelevant ones

- **Understandable**

Expressed in a form that is readily understood by observers

- **Accurate**

Faithfully represents the modelled system

- **Predictive**

Can be used to derive correct conclusions about the modelled system

- **Inexpensive**

Much cheaper to construct and study than the modelled system

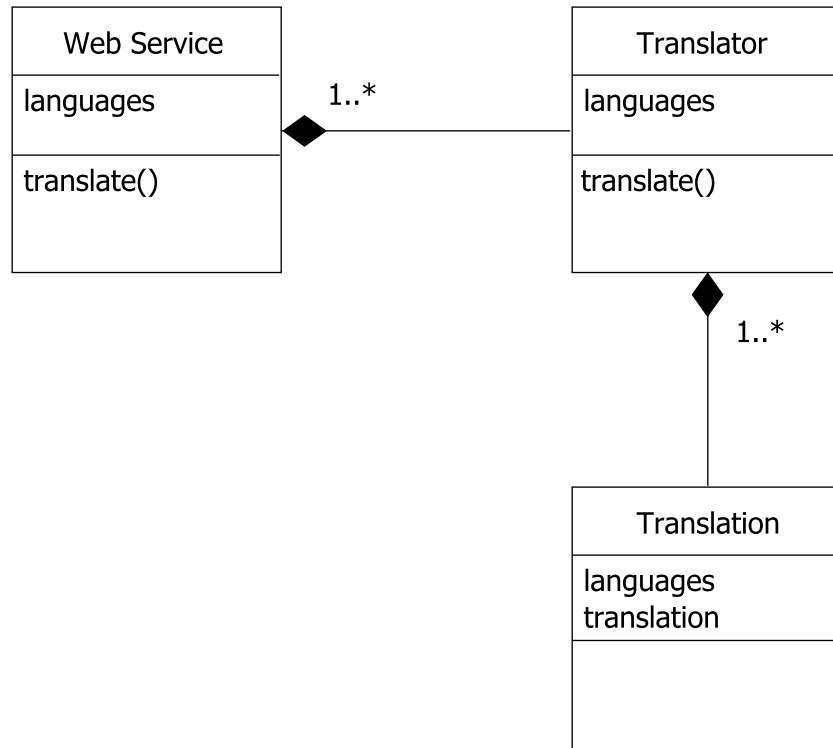
In the next sections you find two platform-independent models (PIM) in form of a simple domain and more specific design model. Moreover, several use case diagrams are used to describe the most important processes. In three sequence diagrams are all necessary operations and their behaviour modelled. With the collection of all these different kind of diagrams, the complete distributed rule translator system is modelled. In Chapter 4, *Implementation* one platform-specific (PSM) model and the implementation in the Java programming language, is introduced and explained with snippets of the most important parts of the source code.

## **4.1. Domain Model**

The domain model is called a computation-independent model (CIM) in terms of the Model Driven Architecture. It shows only the absolute necessary operations and

attributes without data types. For the distributed rule translator system is this model simple and straightforward. You can see in Figure 3.2, “Domain Model: Rule Translator System” three classes are building the basis of the system.

**Figure 3.2. Domain Model: Rule Translator System**



Web services are always stateless. This means, every call from a client creates a new web service object. The class `Web Service` is an interface and acts as a wrapper for the class `Translator`. Therefore his properties and operations are the same than in the `Translator` class. Both classes have the property `languages` and the operation `translate`. To keep the domain model as simple as possible, `languages` is written in plural and stand for the source language of the rule and the target language of the translation result. The association between the `Web Service` and the `Translator` is called an composition association. A composition is a stronger kind of aggregation and express that one class is an absolute necessary part of another class. The `Translator` is the part that helps the `Web Service` to provide an `translate` operation. The translator itself is the class with the real functionality. Since the web service create a new translator object for every client, he has not to keep an existent translator instance. This explains the cardinality of one or more translators. The association between the classes

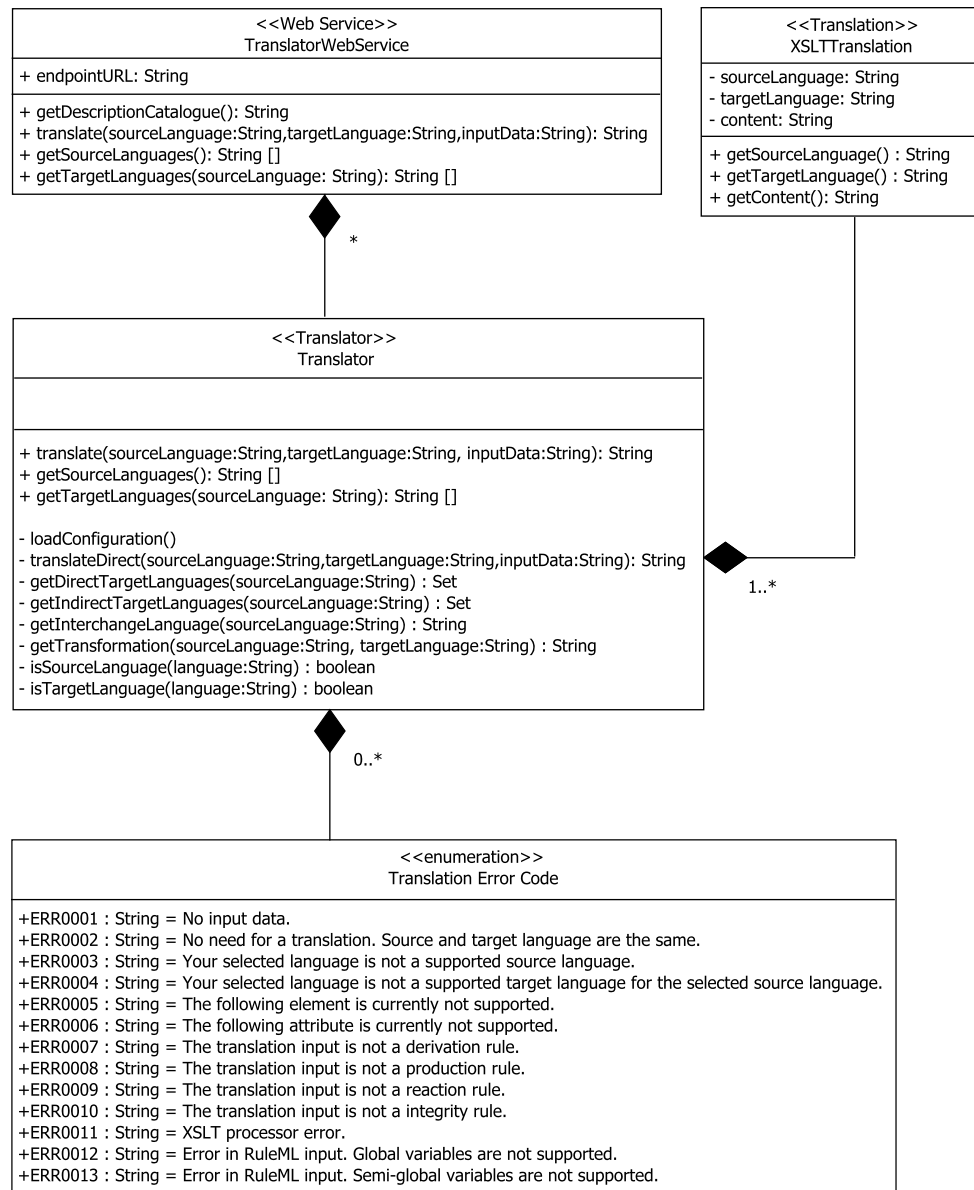
`Translator` and the `Translation` is also a composition. At least one translation is absolute necessary for the translator. Important properties of the class `Translation` are their `languages`, which stands again for the source and target language, and their property `translation` which stands for the mapping description between these languages.

There is one main reason why the translator is modelled as a separate class and not together with the web service. Reusability. Imagine we want later develop an application that use not a web service as interface for the translation of rules. In this case we just drop the class `Web Service`. Our business logic, the already implemented and established translator system, remains and could be used without changes to build our new application. We do not need to reinvent the wheel and save a lot of time.

## 4.2. Design Model

A more specific model than the domain model is the design model. According to the model driven architecture (MDA) approach, the design model is an platform-independent model (PIM). It can be derived from the domain model by adding data types and more specific functions. All cardinalities and associations stay in the same condition than they were in the domain model.

Figure 3.3. Design Model: Rule Translator System



### 4.2.1. The Web Service

Every web service has an endpoint URL, where he receive his operation calls. The class `TranslatorWebService` represent this with her property `endpointURL`. This property is public because clients need to access it in order to use the web service. Since URL's are in the strict sense only strings with a specific meaning, the property got the type `String`. Since this model is platform-independent and there are maybe programming languages where no type URL exist, we are with this markup on the safe side.



To describe his interface, a web service use a XML descriptor file called the web service description catalogue (WSDL). Since XML is a text format and in this case transmitted over a network, the operation `getDescriptionCatalogue()` returns a `String` that include the web service description XML content. To provide an web service description catalogue (WSDL) is in the duty of the environment in where the web service is deployed. But this is a platform-specific issue. The operation simply models that a web service provide his description catalogue.

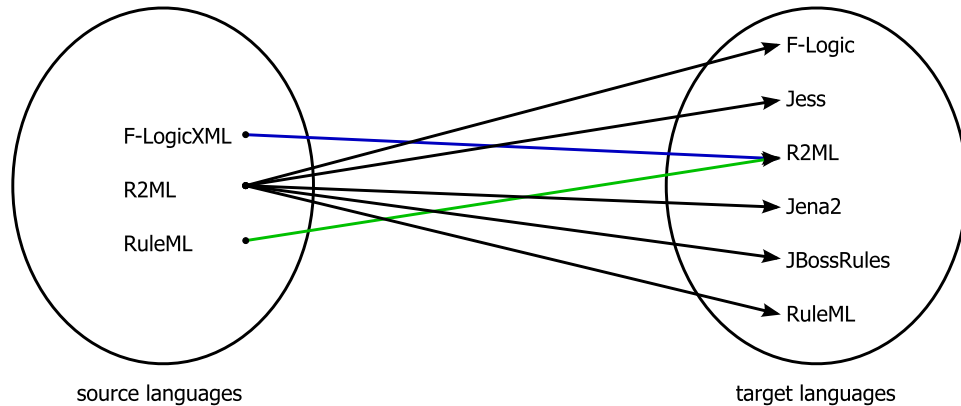
Before any translation could be made, it is necessary to know what source and target languages are supported or are currently available. Maybe a translation in one direction is supported (also known as unidirectional), but no translation backward is available or even possible. Therefore we need to specify the source language when we want to know what kind of target languages are available. An operation like `getLanguages()` would be just reasonable if all translations are bidirectional. When we are able to translate from A to B and back again. But this is not always the case.

## 4.2.2. The Translation

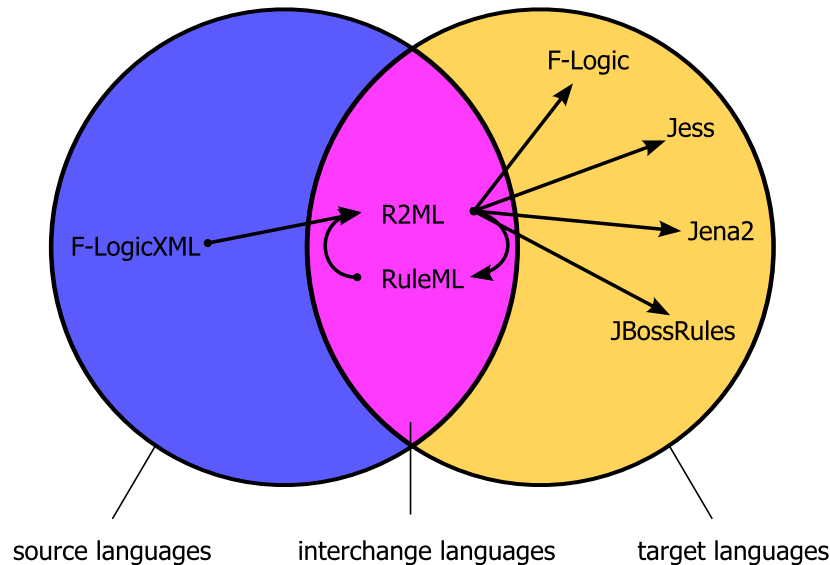
Every translation need properties that specify the source and target language. The property `content` in the class `XSLTTranslation` describes the translation itself, but could later also implemented as a pointer to a translation file somewhere else. The `get` operations symbolise a read only access. Operations to modify translations from inside of the system, are not provided.

## 4.2.3. The Translator

The class `Translator` is the part of the application, that provides the functionality to convert input data from a source to a target language format, with the help one or more translations. He need to know about all available translations and her source and target languages. With this information he can combine translations. Inside of the translator we need to distinguish between direct and indirect translations.

**Figure 3.4. Illustration of Source and Target Languages**

Direct translations translate direct from the source to the target language. In contradiction to indirect translations which use an intermediate format or an interchange language. Interchange languages are both, source and target language. They allow the translator for instance to translate from `F-LogicXML` to `Jess` even if there is no direct translation available as you can see in Figure 3.4, “Illustration of Source and Target Languages”. The benefit of an interchange language is enormous. It depends directly on the number of translations from the interchange language to other languages. Imagine you write only one new translation from a new source language to the interchange language. Then the translator is able to provide you more than one target language. Roughly speaking, the new translation to a interchange language inherit all target languages of the interchange language. At this point it is important to mention that an interchange language need to be able to capture all important informations of all languages she need to interchange. Such a language is often strict and complex, but any lost of information could not be tolerated and is not acceptable. The quality of indirect translations depend mainly on the choice of the interchange language. This need to be considered carefully. Not every language is appropriate for this purpose. In the range of rule languages e.g. `RuleML` would not be the best choice. An explanation for that thesis follow in the implementation chapter in Section 3.3, “Choice of the Interchange Language”.

**Figure 3.5. Illustration of Interchange Languages**

The choice to use sets for internal operations with languages in the class `Translator` has two simple reasons. Sets do not have duplicates and do not have a fixed size, this helps to avoid programming mistakes. Arrays in contradiction need a fixed size, they can have duplicates entries and can run out of bounds. With sets we are on the safe side and do not need to care much about these things. The second and most important reason for the usage of sets is, when we create a intersection of the sets of the source and target languages, we receive a set of all interchange languages that could be used for indirect translations. This is a simple and powerful example which shows the importance of choosing the right data types. This design issue maybe avoid programming mistakes later in the implementation phase.

A detailed explanation of operations follows in Section 5, “Use Cases” and Section 6, “Sequence Diagrams” .

#### 4.2.4. The Translator Error Codes

When the Translator need to fire an exception, which is transported to the client, then it is good idea to have a previous defined set of error cases and a corresponding explanation for them. This is the purpose of the `Translator Error Code` enumeration. It is a kind of contract of defined and expected error cases. Everyone in the developer team use this error codes to describe errors in his part of the application. Errors can not only happen in the translator application, they can also appear inside the translation during

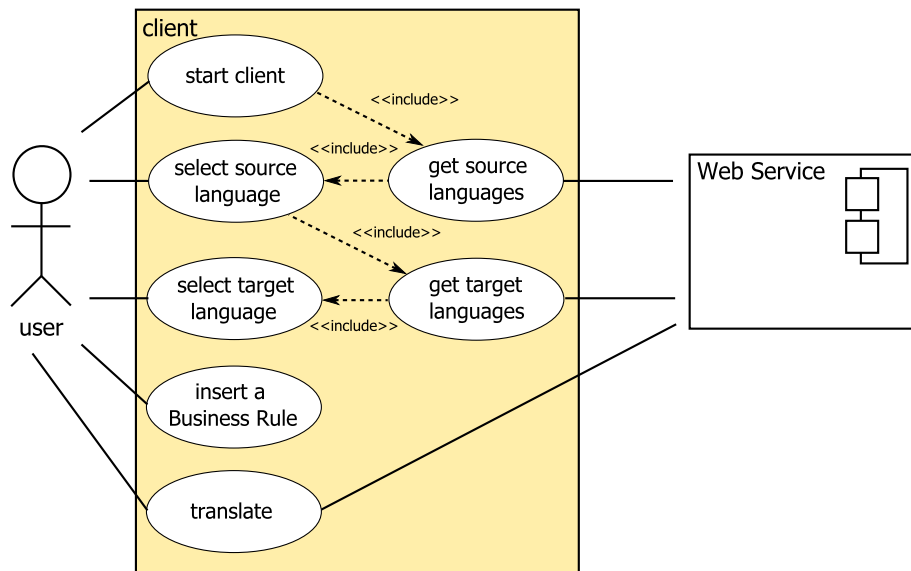
the translation process. Thus the translation developer need to know about these kinds of available error codes.

When a client want to use the web service, but did not specify the necessary translation input, an exception is fired with the error code `ERR0001`. As mentioned before errors can also appear inside of a translation. For instance, is it not always possible to translate a rule from R2ML to JBoss Rules. Only production rules could be translated. If the translation input is another kind of rule e.g. a derivation rule, the translation need to create an error. The translation use therefore the error code `ERR0008` to explain why the translation is not possible. To provide an extra document with all listed error cases and description on a website is a good solution. Of course only the main developer of the web service should be allow to add new error codes. Of course it is not a good behaviour to change the meaning of existent error codes. A better solution is to use new codes and keep the old ones, in order to be somehow backward compatible. That means error codes grow with the development of the application and their translations.

## 5. Use Cases

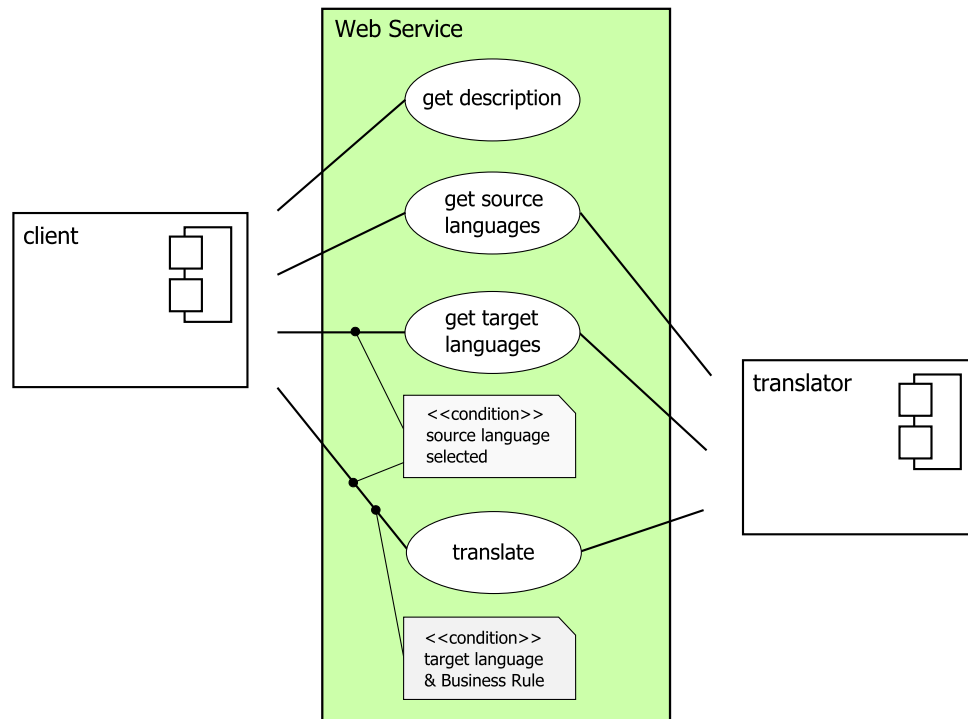
Use case diagrams explaining business processes in a easy understandable way. The use cases concept of was invented by the Swedish computer scientist Dr. Ivar Jacobson. In an interview in 1999 he said “I identified the use case concept in 1986, and when I had found that concept I knew I found something that solved many problems to me, because I could use this concept for everything that systems did, and for every kind of system. It helped me a lot to create a systematic methodology.” [JACOBSON 99]

Both, the client and the web service are involved in three use cases. These are the use cases for receiving the source languages, the target languages and to translate a rule. In the Figure 3.6, “Use Case Diagram: Client (Dynamic Website)” a dynamical website take the role of a client. This example scenario that helps us to imagine the use cases.

**Figure 3.6. Use Case Diagram: Client (Dynamic Website)**

Imagine a user open his web browser and type in the address of the website that act as client. Always the first use case is the start procedure of the client when the website is build for the browser. To fill the complete website with meaningful informations, the client need to call the web service. He need to receive all available source languages and has to select one automatically, maybe the first one. Afterwards he performs another call to the web service in order to receive all target languages for the previously selected source language. One of the target languages is been selected by him again. After the previously performed use cases to building the website, the user can now change for instance the source or target language and activate with his action the corresponding use case. When the user insert a rule e.g. with copy and paste, he could click on the translate button and perform with his action a translation with the help of the web service.

All use cases of the web service are shown in Figure 3.7, “Use Case Diagram: Web Service”. Almost all are pretty simple, because they use directly the functionality of the translator. Only one use case lay in the duty of the web service. He need to provided his description, that means clients should be able to read the web service description catalogue (WSDL). A client could call a web service to receive source and target languages. When he is asking for a target language also a source language need to be specified. Additional information is also necessary when a client is asking for a translation. The source, the target language and a rule is in this case mandatory to fulfil this business process.

**Figure 3.7. Use Case Diagram: Web Service**

Internally, the translator does almost everything to translate a rule. The translator is the environment in where the translation is made. Therefore the structure of the translator and his operations are explained in the next section with the help of sequence diagrams more precisely.

## 6. Sequence Diagrams

In 1987 adopted Dr. Ivar Jacobson sequence diagrams for the object oriented software development. “In the design phase, sequence diagrams are used to gain a more detailed specification for operations.” [BALZERT 01] The benefit of creating one sequence diagram for every use case is, that we retrieve a complete set of necessary operations and her belongings to the classes. This information helps us later in the implementation phase to avoid programming mistakes. The programmer do not need to think about to solve a problem during the implementation, this work is already done.

In sequence diagrams could easily been seen whether a operation need to have public or private access. Public access means that other classes need this operation. Private operations are needed only by the class itself. Arrows between objects of classes denoting

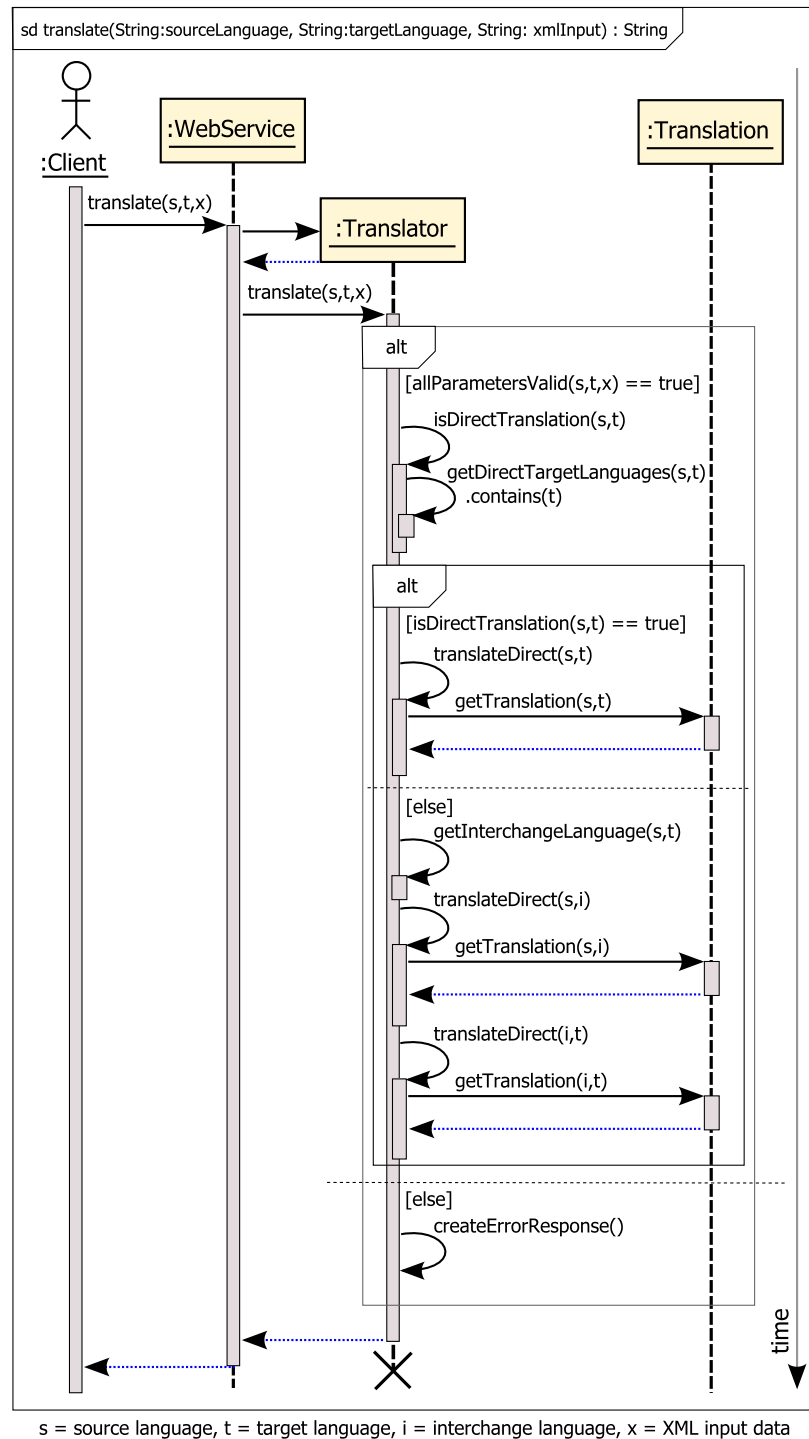
public access. Contrary to arrows which pointing to the same object and denoting private access. To indicate the construction of an object from a class, an arrow point to the name of the class. When a object is not constructed it is assumed that it already existed before. The destruction of the object is indicated by a big X at the end of the her lifeline. A very good and detailed explanation of sequence diagrams could be found in Donald Bell's document *UML's Sequence Diagram* [BELL 04].

In the rule translator system are three important use cases, where the client and the web service both are involved. The most interesting use case is the translation of rules. Use case diagrams are usually modelled in a high grade of abstraction. They give us no useful information about the participating operations to fulfil the task. But a really well-defined look gives us the sequence diagram. A correct and useful sequence diagram could be constructed by going through the use cases, step by step and in combination with the design model, operation by operation. If there are operations missing in the design model, this could easily found out in the design phase before starting any implementation.

## 6.1. Translation of a Rule

During the translation of a rule, several classes with different operations are used. By looking at Figure 3.8, “Sequence Diagram: Translation of a Rule”, the following description could be verbalised and easy retraced.

Figure 3.8. Sequence Diagram: Translation of a Rule



When a client call the `translate(s,t,x)` operation of the `:Web Service`, the web service create a new object `:Translator`. The parameter `s` stand for the source, `t` for the target language and `x` for the XML input data. After creating



the instance, the `:WebService` call the `translate(s,t,x)` operation of the `:Translator`. The `:Translator` need at first to check if all parameters are valid. If the source and the target language are supported and the XML input data is not empty and the result of the operation `allParametersValid(s,t,x)` equals `true` the next operation can be executed. The `:Translator` checks now if he need to perform a direct or indirect translation. He could survey this by calling his `isDirectTranslation(s,t)` operation. These operation call again the `getDirectTranslations(s,t)` operation. If the result of this operation `contains(t)` a target language for the given source language, then he know that he has to make a direct translation.

In the diagram the term `alt` is an abbreviation for the word alternation. When for instance the result of the operation `isDirectTranslation(s,t)` equals `true` then all operations in the first casket are executed. If not, all operations in the small box under the `else` keyword are executed. In the first case, the `:Translator` calls his `translateDirect(s,t)` operation. The operation itself calls the `getTranslation(s,t)` operation of the `:Translation` object to receive the appropriate translation description. With this translation the `:Translator` translates now directly from the source to the target language the XML input data.

When the result of the operation `isDirectTranslation(s,t)` equals not `true` then the all operations in the second casket under the `else` keyword are executed. To translate indirect the `:Translator` need to retrieve the interchange language by calling his `getInterchangeLanguage(s,t)` operation with the source and target language as parameters. With the interchange language he is now be able to translate directly from this source to the interchange language and with the translation result as input data from the interchange to the target language.

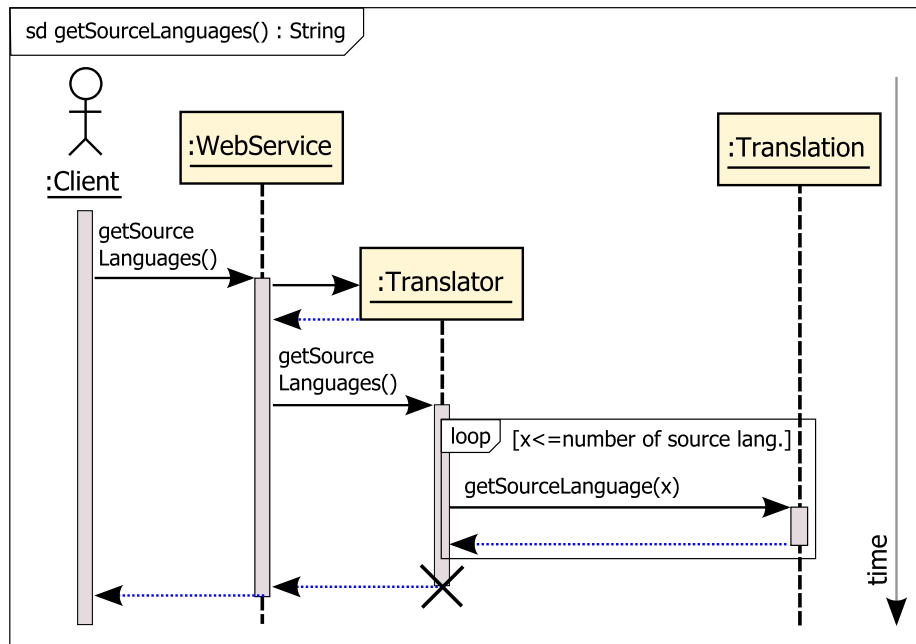
In the case that the `allParametersValid(s,t,x)` operation do not return `true`, because of some unexpected parameter values, the last `else` box in the diagram is been executed. A error response message for the `:Client` is created here as result of the translation. When the translation is finished the `:Translator` returns the result to the `:WebService`, he returns it to the client and destroys the `:Translator` object.

## 6.2. Reception of all Source Languages

When a `:Client` calls the `:WebService`, this call is forwarded to the `:Translator`. The `:Translator` calls the `:Translation` for source languages as

long as there is one left available. Afterwards he returns all source languages to the `:Translator` and he return them to the `:Client`.

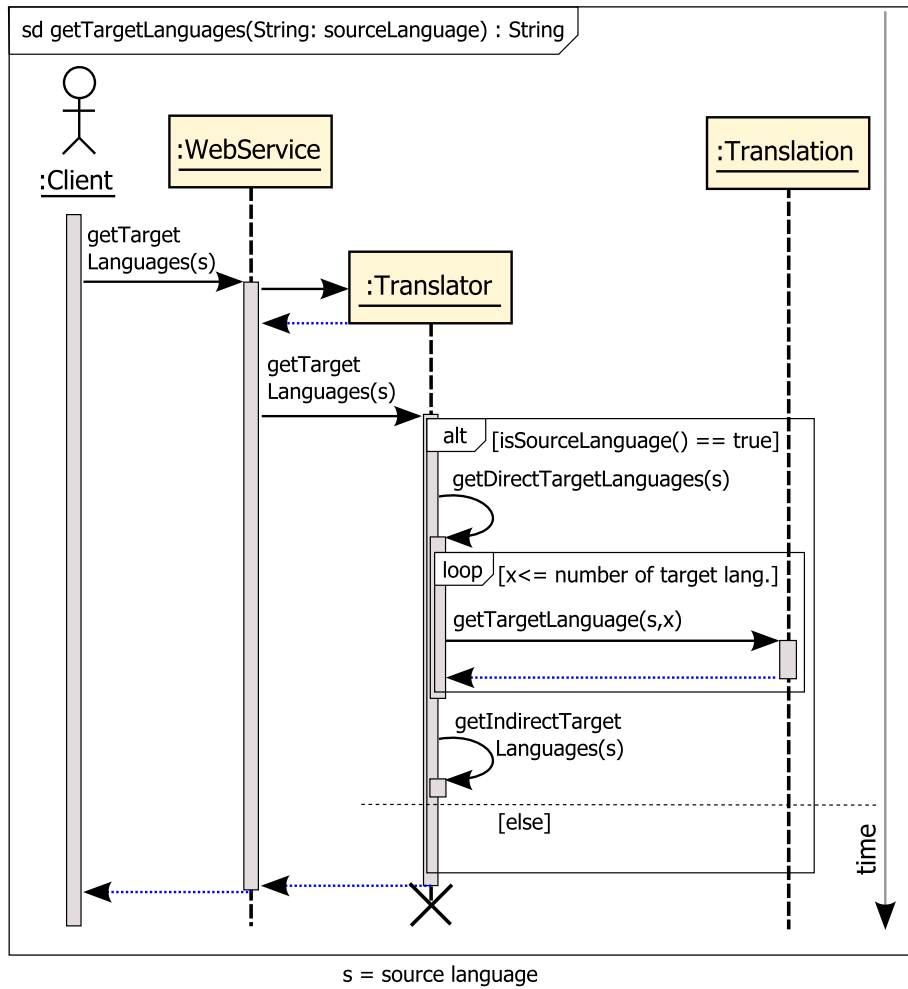
**Figure 3.9. Sequence Diagram: Reception of all Source Languages**



## 6.3. Reception of all Target Languages

The `:Client` call the `:WebService` to receive all target languages for one source language. The `:WebService` forward his call to the `:Translator` where the `:Translator` first check if the parameter `s` is a supported source language. If this is the case, he calls his `getDirectTargetLanguages(s)` operation with the source language as parameter. This operation receive in a loop every target language for the specified source language from the `:Translation`. After the following `getIndirectTargetLanguages(s)` operation call, and an union of both target languages sets, the `:Translator` return the result to the `:WebService` and from him he return it to the `:Client`. For the case that the parameter `s` of the operation `getTargetLanguages(s)` is not a source language and the else branch is executed, simply an empty result is returned. That means that no target language is available for that source language.

**Figure 3.10. Sequence Diagram: Reception of all Target Languages**



---

# Chapter 4. Implementation

In this chapter the platform-specific model (PSM) is introduced. In order to finish the Model Driven Architecture (MDA) process to build an application, one concrete implementation follows. The middleware platform used for the translator web service is an Java Enterprise Edition application server with support of the Enterprise JavaBeans 3.0 technology.

## 1. Java Programming Language

Java is a programming language that it is independent of the underlying hardware platform. Every Java application runs without changes on almost every kind of computer. This is possible because Java programs are compiled to an intermediate form called Java byte-code. To run a Java application this byte-code is interpreted by a virtual machine, so to say a Java processor, and an environment of libraries. For almost every available computer system Java virtual machines are available. Java applications created on a Windows machine, which usually have a CISC<sup>1</sup> processor architecture, can also run on a Macintosh PowerPC machine which has a completely different RISC<sup>2</sup> processor architecture. But the behaviour of applications is still the same. Thus makes Java programs not really platform-independent, because their platform is the Java programming language, but at least hardware-independent.

To explain why the Java Enterprise Edition was chosen, we need to know first which other Java versions are available and what could be done with them. There are three versions of the Java programming language, each for a different purpose, currently available on the market. The Java Standard Edition (Java SE) is the basis of the Java language and an environment for any kind of standard computer application. We distinguish here between the Java Runtime Environment (JRE), that provides only an environment to run already compiled Java byte-code and the Java Standard Developer Kit (SDK), which allows to compile Java source code into byte-code. The Java Runtime Environment (JRE) is always included in the Java Standard Developer Kit (SDK) and allows for instance the Internet browser to execute Java applications inside of websites. These kind of applications are also known as Java Applets.

---

<sup>1</sup>Complex Instruction Set Computer

<sup>2</sup>Reduced Instruction Set Computer

The Java Micro Edition (Java ME) is an environment for personal computers that allows the development of Java application for mobile devices. Today almost every new mobile phone or PDA<sup>3</sup> has a runtime environment for Java Micro Edition (Java ME) applications. This environment includes also a virtual mobile phone that allows to run and debug Java ME applications on a computer without a real mobile phone. The Java Micro Edition has some differences compared to the standard edition. For instance the way of graphical presentation of user interfaces is not the same. This is mainly caused by the reduced screen size of mobile devices and the disability of cheap and old devices to show coloured graphics. Today in 2007, games are the biggest part of available application written in the Java Micro Edition (Java ME). But as we know from the past, the possibilities of the applications increase along with the power of their hardware platform.

The Java Enterprise Edition (Java EE) is more an extension of the Java Standard Edition (Java SE). This edition is specialised on enterprise and business processes. The *Java™ Platform, Enterprise Edition 5 Specification* [JSR244] describes the complete environment in detail. Another worth reading source that explains the Java Enterprise architecture is *The Java™ EE 5 Tutorial* [BCE+].

One important difference to the Java Standard Edition (Java SDK) are Enterprise JavaBeans (EJB).

“The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification.” [JSR220]

“The benefit to application developers is that they can focus on writing the business logic necessary to support their application without having to worry about implementation the surrounding framework.” [BM 06]

## 2. Application Server

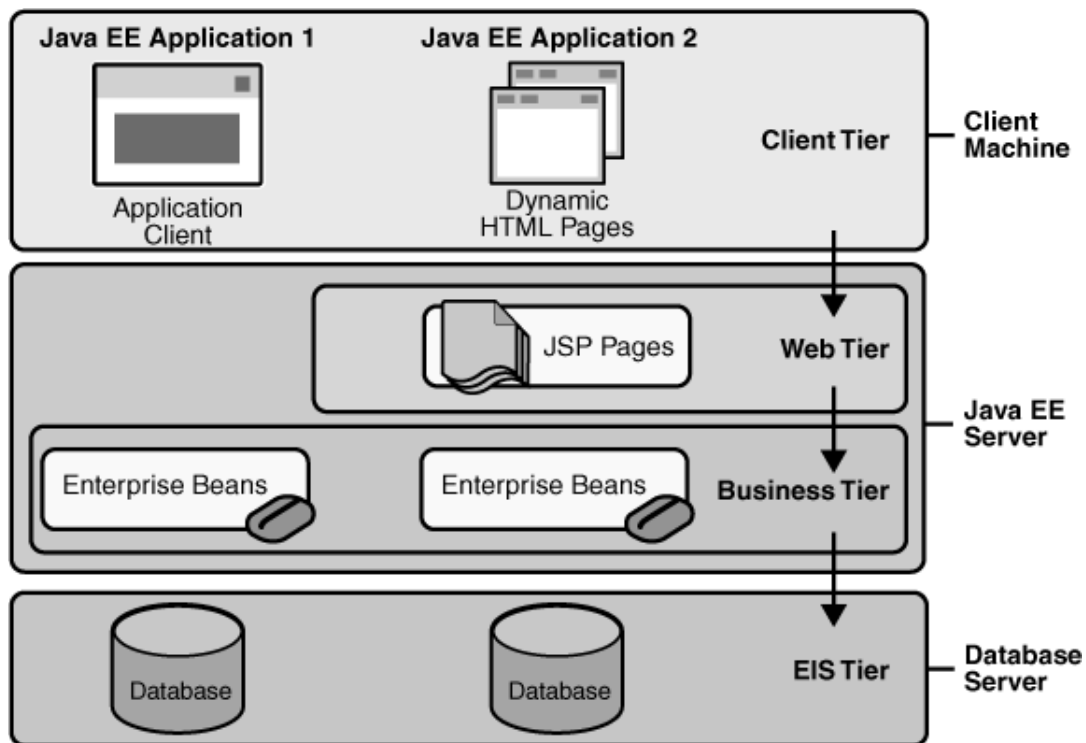
“A Java EE server is a server application that the implements the Java EE platform APIs and provides the standard Java EE services. Java EE servers are sometimes called

---

<sup>3</sup>Personal Digital Assistant

application servers, because they allow you to serve application data to clients, much as how web servers serve web pages to web browsers. Java EE servers host several application component types that correspond to the tiers in a multi-tiered application. [...] In a multi-tiered application, the functionality of the application is separated into isolated functional areas, called tiers. Typically, multi-tiered applications have a client tier, a middle tier, and a data tier (often called the enterprise information systems tier). The client tier consists of a client program that makes requests to the middle tier. The middle tier's business functions handle client requests and process application data, storing it in a permanent datastore in the data tier. Java EE application development concentrates on the middle tier to make enterprise application management easier, more robust, and more secure." [YFC]

**Figure 4.1. Multitiered Applications [BCE+]**



On the market are several application server available that implement the Java Enterprise Edition and the Enterprise JavaBeans 3.0 specification. For almost every case is an open source solution absolute sufficient. The free WebSphere Community Edition<sup>4</sup>

<sup>4</sup><http://www.ibm.com/software/webservers/appserv/community/>

application server is a good choice if you want to use it along with IBM's free version of the DB2 database and already plan to migrate later to the commercial versions of both products. The WebSphere application server is based on the Geronimo application server from Apache<sup>5</sup> and IBM provide you with a complete documentation in several languages as well with professional support for 30 days.

For this project the decision felt to the other open source application server next to Geronimo, to the JBoss application server from RedHat. Mainly because there was many experience from previous projects and this server has simply established. Since an application server is a middleware for the Java EE technology, the implementation approach should be easy adoptable to any other Java Enterprise Edition and Enterprise JavaBean 3.0 compatible application server.

### 3. Implementation Model

In Figure 4.2, “Implementation Model: Web Service” you see the platform-specific model (PSM) for the Java Enterprise Edition 5 middleware platform.

The interface `r2ml.WebserviceInterface` is called a *business interface* in terms of the Enterprise JavaBeans 3.0 concept and is necessary to describe the application in an abstract way. The purpose and benefit of a business interface is described later in Section 3.2, “Enterprise JavaBean Container”. In the business interface of the application appear only the three of four operations we had modelled before in the platform-independent model (PIM) you can see in Figure 3.3, “Design Model: Rule Translator System”. The operation `getDescriptionCatalogue` is missing. Since the Enterprise JavaBeans 3.0 web service implementation of the JBoss application server provide an automatically generated web service description in form of an WSDL<sup>3</sup> catalogue, there is no need for such an operation. Previously in the design model, this operation in combination with the attribute `endpointURL` was used to markup the web service description. The classes `r2ml.WebserviceInterface` and `r2ml.Webservice` are derived from the previously modelled class `TranslatorWebService` of the platform-independent design model. The class `TranslatorWebService` does not need a variable to hold a instance<sup>5</sup> of the class `Translator`. As mentioned before, a web service is stateless and so the `r2ml.WebService` class creates for every client call always a

---

<sup>5</sup> <http://geronimo.apache.org/>

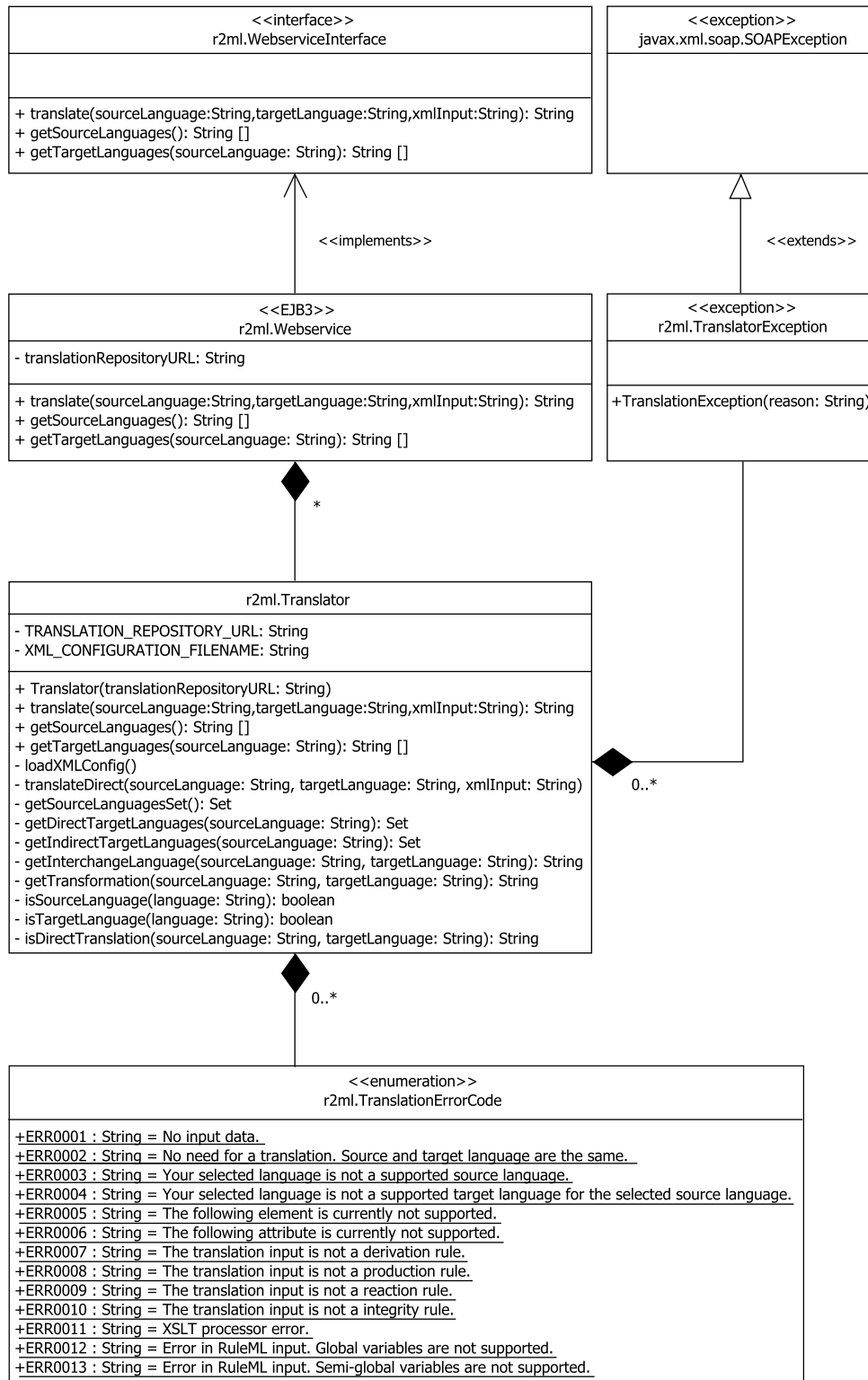
new instance of the `r2ml.Translator` class. All attributes of the enumeration class `r2ml.TranslationErrorCode` are declared static and do not need be instantiated separately. In order to use the error codes in the Java source code you simply need to write e.g. `r2ml.TranslationErrorCode.ERR0001` to receive the message "*No input data.*".

In the this concrete implementation of the rule translator system only source languages in XML<sup>2</sup> syntax are supported. The XML transformation language XSLT is used to translate from one XML rule language to another. This is the reason why the parameter `inputData` has changed to `xmlInput`.

The stereotype `EJB3` in top of the class name `Webservice` markup that the class is an Enterprise JavaBean 3.0. The property `translationRepositoryURL` hold the URL to the repository of the translations. The concept that lay behind is described in Section 3.5.3, "URL Injection during Deployment Time".



Figure 4.2. Implementation Model: Web Service



The `Translator` class, as mentioned before, is designed as POJO<sup>6</sup> in a way to that it could be re-used. This is the reason why this class is no Enterprise JavaBean and provide the same operation names as the `Webservice` class. The class implements the translator functionality of the system. It has two properties which are private constants and therefore written in capital letters. In the construction phase of the translator object the constant property `TRANSLATION_REPOSITORY_URL` receive her value from the value of the `translatorRepositoryURL` property of the web service. Afterwards this value is not changeable by the instantiated translator object. The value of the constant property `XML_CONFIGURATION_FILE` is the name of the translation descriptor file. This is explained more precisely in Section 3.4, “Management of Translations”. In Figure 4.2, “Implementation Model: Web Service” the class `Translator` became one additional operation called `getSourceLanguagesSet`. The purpose of adding this method is the following. The translator implementation use intern a data structure of sets and extern an array of strings. The name `getSourceLanguages` was already used to retrieve an array of source languages, which is simply forwarded by the web service to the client, as we already know from Section 6.2, “Reception of all Source Languages”. Since operation names in Java need to be non-ambiguous, we need to choose the name `getSourceLanguagesSet` to retrieve the source languages as a set.

A new class in the implementation diagram is `r2ml.TranslationException`. This class inherit the complete functionality of the class `java.xml.soap.SOAPException`. It has only different name. This is looks maybe a little bit strange, but after reading the Section 3.5.5, “Custom Exceptions” you will see that this make perfect sense. In Section 3.5.6, “Packaging of the Application” is explained why the name `Webservice` and not `TranslatorWebService` as in the design model was chosen in the implementation model.

Before the source code can be explained in detail, some basic about the Enterprise JavaBeans 3.0 and the their container need to take into account. After the section about the importance of the right choice of the interchange language, the section management of translations follows.

---

<sup>6</sup>Plain Old Java Object

## 3.1. Enterprise JavaBeans 3.0 Technology

“The primary goal of the EJB 3.0 and Java Persistence specifications was to make it as easy as possible to write and deploy an EJB-based application. Creating an application is as easy as compiling your code, jarring up your classes, and running your application server.” [BM 06]

There are three different type of Enterprise JavaBeans in version 3.0. *Entity Beans*, *Message-Driven Beans* and *Session Beans*. Since we are only interested into creating a web service we concentrate only on stateless Session Beans.

To deploy a web service the only thing you need to do is to add some meta information to you existent POJO's<sup>6</sup>. Since Java version 1.5 (also known as Java 5) these additional meta informations are a new construct called Java Annotations. An explanation of this technology you find in Section 3.5.1, “Java Annotations”.

## 3.2. Enterprise JavaBean Container

An Enterprise JavaBean container is the environment for an Enterprise JavaBean application. When an applications outside of the Enterprise JavaBean container want to interact with applications inside ...

“... it is not working directly with instances of the bean class; it is working through the beans remote or local interface. ” [BM 06] Both interfaces are also known as business interface and correspond to the class `WebserviceInterface` which could be seen in Figure 4.2, “Implementation Model: Web Service”.

When clients invoke methods on the business interface,...

“... the object instance you are using is something called a *proxy stub*. This proxy stub implements the remote or local interface of the session bean and is responsible for sending your session bean method invocation over the network to your remote EJB container or routing the request to an EJB container that is local in the JVM<sup>7</sup>. It is the EJB container's job to manage bean class instances as well as security and transaction demarcation. The EJB container has knowledge of the metadata defined as annotations on the bean class or as elements in the XML deployment descriptor. Bases on this metadata,

---

<sup>7</sup>Java Virtual Machine

it will start a transaction and perform authentication and authorization tasks. It is also responsible for managing the life cycle of the bean instance and routing the request from the proxy to the real bean class instance. After the EJB container has managed the life cycle of the bean instance, started any transaction, and performed its security, it routes the invocation to an actual bean instance.” [BM 06]

Since every enterprise application run in her own environment, the Enterprise JavaBean container makes it impossible to affect other applications or even the server itself. When a web application written as Java Server Page (JSP) connected to a JavaBean on a JavaServer Pages server like the Apache Tomcat crash due a programming mistake, the complete server crash as well. All other running applications are affected and not available as long as the server is restarted manually.

At this point it is necessary to mention that the file creation and access inside of the EJB container is prohibited. This make perfect sense for security and portability reasons. However this makes it sometimes also difficult to migrate already existent applications. Everybody should be aware that this a could create delays during the implementation process, if this was not scheduled before. In Section 3.4, “Management of Translations” a solution is introduced how files could be used nevertheless by a EJB container. The concept was of course scheduled during designed process before, but just not mentioned since the implementation is the topic of the current chapter.

### 3.3. Choice of the Interchange Language

The quality of the translation depend directly on the choice of the interchange language. She need to capture every information of the source language and need to produce the same content after a reverse translation. A lost of semantic is not acceptable for a interchange language.

#### **Example 4.1. Lack Markup in RuleML**

```
<Atom>
  <Rel>spending</Rel>
  <Ind>Peter Miller</Ind>
  <Ind>min 5000 euro</Ind>
  <Ind>previous year</Ind>
</Atom>
```

The Example 4.1, “Lack Markup in RuleML” is taken from the *RuleML Tutorial* [BGT 05]. Verbalised it means *Peter Miller was spending a minimum of 5000 euro last year*. The markup is not strict. It is written in a human understandable way. We can understand the meaning. But what about machines? There is no semantic after *min 5000 euro* or *previous year*. It is not declared that *min* means minimum or *previous year* means the last year before the current. We have many problems to translate such a lack markup into other rule languages. With SWRL<sup>8</sup> built-ins and XPath functions<sup>9</sup> the same context could be marked up in more strict and clear way. The previous year is always the year before the current year. To markup we need to know in which year we are currently and to subtract the value 1. A minimum is always the lowest bound. Everything greater than the lowest bound is sufficient. We compare the spending of the customer in the previous year with the value 5000 and did mark up everything in a strict and clear way. Our interchange language need to support this or a similar kind of strict markup.

#### **Example 4.2. Strict Markup with SWRL Built-Ins and XPath Functions**

```
swrlb:subtract(previous_year ,  
               fn:year-from-dateTime(  
                 fn:current-dateTime() ) ,  
               1 )  
  
swrlb:greaterThan(spending(customer,previous_year) , 5000 )
```

The rule language R2ML supports such a strict markup with the help of SWRL and XPath functions. You can see in the following two examples how to markup the content without losing his semantic in R2ML. At first it need to be defined in a separate atom what `previous_year` means.

---

<sup>8</sup> Semantic Web Rule Language [<http://www.w3.org/Submission/SWRL/#8>]

<sup>9</sup> <http://www.w3.org/TR/xpath-functions/>

**Example 4.3. Strict Markup in R2ML: "previous year"**

```
<r2ml:DatatypePredicateAtom
  r2ml:datatypePredicateID="swrlb:subtract">
  <r2ml:dataArguments>
    <r2ml:DataVariable r2ml:name="previous_year"
      r2ml:datatypeID="xs:gYear"/>
    <r2ml:DatatypeFunctionTerm
      r2ml:datatypeFunctionID="fn:year-from-dateTime">
      <r2ml:dataArguments>
        <r2ml:DatatypeFunctionTerm
          r2ml:datatypeFunctionID="fn:current-dateTime">
            <r2ml:dataArguments/>
          </r2ml:DatatypeFunctionTerm>
        </r2ml:dataArguments>
      </r2ml:DatatypeFunctionTerm>
    </r2ml:dataArguments>
  </r2ml:DatatypeFunctionTerm>
  <r2ml:TypedLiteral r2ml:datatypeID="xs:integer"
    r2ml:lexicalValue="1"/>
  </r2ml:dataArguments>
</r2ml:DatatypePredicateAtom>
```

Afterwards `previous_year` can be used inside of the operation `spending` which is used by the operation `greaterThan` along with the positive integer value 5000.

---

**Example 4.4. Strict Markup in R2ML: "customer spending a minimum of 5000 previous year"**

```
<r2ml:DatatypePredicateAtom
  r2ml:datatypePredicateID="swrlb:greaterThan">
  <r2ml:dataArguments>
    <r2ml:DataOperationTerm
      r2ml:operationID="spending">
      <r2ml:contextArgument>
        <r2ml:ObjectVariable r2ml:name="customer"
          r2ml:classID="Customer"/>
      </r2ml:contextArgument>
      <r2ml:arguments>
        <r2ml:DataVariable r2ml:name="previous_year"
          r2ml:datatypeID="xs:gYear"/>
      </r2ml:arguments>
    </r2ml:DataOperationTerm>
    <r2ml:TypedLiteral r2ml:datatypeID="xs:positiveInteger"
      r2ml:lexicalValue="5000"/>
  </r2ml:dataArguments>
</r2ml:DatatypePredicateAtom>
```

### 3.4. Management of Translations

In the implementation of the rule translator system we support the translation of XML based rule languages. Translations are XML files as well. XSLT is a dialect that describes the transformation of XML data. To transform a XML file an XSLT transformation is needed. This transformation can be applied to a XML file and can be processed by an XSLT processor.

The set of all translation files lay in a file system. As we know from Chapter 3, *Design* the file system is distributed on another server and this has two reasons. Files can be modified over a source control system. With an integration of the SVN source control system via WEB-DAV, files can be provided at a URL with the help of a web server. Therefore it is good to have one single file that describes all translations and where to find them. The translation descriptor can now be read from his URL by the web service

application in order to find all available translations. The access from the Enterprise JavaBean container to the underlying file system is prohibited. This is a solution to access files from a Enterprise JavaBean Container. Translations can be modified outside and are read when the web service need them.

The content of the translation descriptor `translations.xml` file you can see in Example 4.5, “XML: Translation Descriptor” . In this case there are two direct translations and one indirect translation available. From F-LogicXML to R2ML and from R2ML to RuleML. The XSLT files for the translation lay both in the sub folder `translations` and can be addressed relative to the folder where the descriptor file lays.

#### **Example 4.5. XML: Translation Descriptor**

```
<?xml version="1.0" encoding="UTF-8"?>
<translator
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="translations.xsd">

  <translation
    xsltFile="translations/F-LogicXML_to_R2ML.xslt"
    source="F-LogicXML" target="R2ML"
    defaultInput="rules/F-LogicXML_01.xml"/>

  <translation
    xsltFile="translations/R2ML_to_RuleML.xslt"
    source="R2ML" target="RuleML"
    defaultInput="rules/R2ML_01.xml"/>

</translator>
```

It is also possible to use another interchange language than R2ML for the translation. Interchange language is the language, which is source and target language inside of a indirect translation. In the case of this translation descriptor, the indirect translation is a translation from F-LogicXML over R2ML to RuleML. Here R2ML is the interchange language. Imagine there would be another indirect translation from RuleML over R2ML\_version\_2 to Jess. We would have two indirect translations and two different



interchange languages. This is a important advantage if we want to switch to another version or to another complete new interchange language. In this case we could write a translation from the old to the new interchange language and do not need to drop all our existing translations. You see it is possible to keep more than one interchange language in the same translation descriptor file.

There are several benefits to use a XML syntax for the translation descriptor. Files in XML syntax could be read from any programming language, since they are encoded like every ordinary text file. There are also many parsers for XML data available and which simplifies the data retrieve process. But the main benefit of XML is that a XML structure can be validated against her XML Schema definition to check if she is well-formed. In Example 4.6, “XML Schema: Translator Descriptor” you see a XML Schema for the translation descriptor file of the Example 4.5, “XML: Translation Descriptor”.

#### **Example 4.6. XML Schema: Translator Descriptor**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xml:lang="en">
  <xs:element name="translator" type="translatorType"/>
  <xs:complexType name="translatorType">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="translation"
                 type="translationType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="translationType">
    <xs:attribute name="source" type="xs:string"
                 use="required"/>
    <xs:attribute name="target" type="xs:string"
                 use="required"/>
    <xs:attribute name="xsltFile" type="xs:string"
                 use="required"/>
    <xs:attribute name="defaultInput" type="xs:string"
                 use="required"/>
  </xs:complexType>
</xs:schema>
```

There is one root element `translator` defined that can have a sequence of elements with the name `translation`. An element of `translationType` is required to have the attributes `source`, `target`, `xsltFile` and `defaultInput` of type `xs:string`.

## 3.5. Explanation of the Source Codes

In this section the most important parts of the Enterprise JavaBeans 3.0 source codes, which are essential to build the web service, are explained in detail.

### 3.5.1. Java Annotations

Java annotations are used in the Enterprise JavaBeans 3.0 to specify necessary and additional informations for the application server. He need to know when he have to create a web service with a specific binding. Which operations should be used for the web service and which not. How the operation names and her parameters should be mapped to the names that later appear in the web service description catalogue.

#### **Example 4.7. Java Annotations: Remote Interface**

```
package r2ml;

import javax.ejb.Remote;

@Remote
public interface TranslatorWebserviceInterface {
    ...
}
```

In Example 4.7, “Java Annotations: Remote Interface” you see the business interface for the Enterprise JavaBean 3.0 web service. Inside of the business interface are only two different annotations possible. The annotation with the name `Remote` or `Local` of the `javax.ejb` package need first to be imported. Afterwards they can be used by writing the `@` sign in front of the name of the annotation, in this case `Remote`. These two annotation need to appear before the ordinary Java interface declaration.

The remote interface defines all operations that can be used from outside of the Enterprise JavaBean container. When we want to create a web service we have to choose this

annotation. The local interface defines the operations that can be used from other classes and beans inside of the same Enterprise JavaBean container.

It is possible to annotate either one interface with one or both annotations. But it is also possible to have each annotation in a separate interface declaration. This makes sense when a web service, using the `Remote` annotation, should only have a subset or different operations than the `Local` interface.

**Example 4.8. Stateless Bean Web Service**

```
package r2ml;

import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.soap.SOAPBinding;
import javax.annotation.Resource;

@Stateless
@WebService(serviceName="R2MLTranslatorWebService")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
             use=SOAPBinding.Use.LITERAL)
public class Webservice
implements WebserviceInterface {
    @Resource(mappedName=
        "java:comp/env/translationRepositoryURL")
    private String translationRepositoryURL;

    @WebMethod
    public String translate(
        @WebParam(name="sourceLanguage")
        String sourceLanguage,
        @WebParam(name="targetLanguage")
        String targetLanguage,
        @WebParam(name="xmlInput")
        String xmlInput )
        throws TranslationException { ... }
    ...
} //class: Webservice
```

In Example 4.8, “Stateless Bean Web Service” you see several annotations are used. To specify a web service, at least the `Stateless` and `WebService` annotation are necessary. Web services do not keep any state. Therefore we have to use the

Stateless annotation. The `WebService` annotation specifies that this class is an implementation of a web service and accept several parameters. An complete overview gives us a look into the implementation, which you see in Example 4.9, “Java Annotation: `@WebService`”.

#### **Example 4.9. Java Annotation: `@WebService`**

```
package javax.jws;

@Target({TYPE})
@Retention(value=RetentionPolicy.RUNTIME)
public @interface WebService {
    String name() default "";
    String targetNamespace() default "";
    String serviceName() default "";
    String wsdlLocation() default "";
    String portName() default "";
    String endpointInterface() default "";
}
```

To specify for instance the target namespace of the web service, we need to change our existing annotation to `@WebService(targetNamespace="http://www.example.com/mynamespace")`. If you do so, be sure that the web service is available at this URL. The client introduced in Chapter 5, *Web Interface* for instance, will not work when the web service is not reachable at URL of the target namespace declared in the web service description catalogue. This need to be considered in order to do not exclude clients of your web service.

According to the Java Specification Request (JSR) 181, the parameter name specifies the name of the web service. This name is used as attribute name of the `wsdl:portType` element when the web service description catalogue (in version 1.1) is generated. The default value is the name of the Java class or interface.

“The `targetNamespace()` attribute specifies the XML namespace used for the WSDL and XML elements that are generated from this annotation. The default value is generated from the package name of the annotated type. The `wsdlLocation()` attribute defines the URL of the WSDL document that represents this web service. You

need this attribute only if you are mapping your service to a preexisting WSDL document. [...] In most cases, you can use the default values for each of these attributes.” [BM 06]

The `serviceName` appears as value of the `name` attribute of the `service` and `definitions` element in the WSDL catalogue. It defines the name of the web service. If no `serviceName` is given the name of the Java class plus the string `Service` appears in the WSDL catalogue.

When the class `Webservice` use the keyword `implements` to specify that this class implement the business interface `WebserviceInterface` then is no need to use the `Remote` annotation. In this case knows the application server which class is the remote interface. It is the implemented interface `WebserviceInterface`.

### **Example 4.10. Java Annotations: Alternative Business Interface Declaration**

```
package r2ml;

import javax.ejb.Stateless;
import javax.ejb.Remote;
import javax.jws.WebService;
...

@Stateless
@WebService
@Remote(WebServiceInterface.class)
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
              use=SOAPBinding.Use.LITERAL)
public class Webservice {
    ...
} //class: Webservice
```

If we do not want to declare that the class `Webservice` implements the remote interface, we need to use the annotation `Remote` with the class name of the remote interface. This could be seen in Example 4.10, “Java Annotations: Alternative Business Interface Declaration”. This approach is not recommended, but it is a possible and working solution.

When no method of a class, annotated with `WebService`, is annotated with `WebMethod` then all methods are considered to be available for the web service. If only one method is annotated with `WebMethod` and another method not, then only the annotated method is used by the web service. In order to have an easy understandable source code I suggest to annotate all methods even if they are all used for the web service.

#### **Example 4.11. Java Annotation: @WebMethod**

```
package javax.jws;

@Target({ElementType.METHOD})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface WebMethod {
    String operationName() default "";
    String action() default "";
}
```

Again we take a look into the implementation. The `operationName()` parameter specifies the name of the operation inside the web service description catalogue. The default operation name is always the same as the Java operation name. “The `action()` attribute is used to set the `SOAPAction` hint that corresponds with this operation. This hint allows a service endpoint to determine the destination by simply looking at the `SOAPAction` HTTP header instead of analyzing the contents of the SOAP message body.” [BM 06]

#### **Example 4.12. Java Annotation: @WebParam**

```
package javax.jws;

@Target({ElementType.METHOD})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface WebParam {
    public enum Mode(IN, OUT, INOUT);
    String name() default "";
    String targetNamespace() default "";
    Mode mode() default Mode.IN;
    boolean header() default false;
}
```

If the SOAP binding style of the web service is Remote Procedure Call (RPC) the attribute name specifies the `wsdl:part` name in the web service description catalogue (in version 1.1). The default is `type_N`, where *N* represent the index of the parameter in the method parameter declaration and *type* the Java type e.g. `String`. The JBoss application server generates a starting index of 1 and not of 0 as written in the Java Specification Request (JSR) 181. If we would not annotate the parameter of the method `translate` as we did in Example 4.8, “Stateless Bean Web Service” the parameter `sourceLanguage` would be mapped to the attribute name `String_1` in the web service description catalogue on the JBoss application server. Due to the differences in specification and implementation of the application server, I suggest to used the `WebParam` annotation in order to avoid problems with web service clients. Since the purpose of the web service description catalogue is to describe, the `WebParam` annotation should be used to provide a proper name for each method parameter. A default parameter like `String_1` could be mean almost everything.

“The `targetNamespace` attribute, used only if the style is `Document/Literal`, sets the `targetNamespace` of the schema definition that contains the element. The behaviour of `targetNamespace()` on `Document/Literal` wrapped is not fully explained in the Java Specification Request (JSR) 181. [...] The `header()` attribute is used to indicate that the parameter should be put in a SOAP header rather that in the SOAP body. The `mode()` attribute is used to indicate whether the parameter is to be used for input, output, or both. Due to Java semantics, if a parameter is used for output, it must be wrapped using a special holder type.” [BM 06]

The missing annotations `SOAPBinding` and `Resource` annotations are explained in Section 3.5.2, “WSDL Generation” and Section 3.5.3, “URL Injection during Deployment Time”.

### 3.5.2. WSDL Generation

The JBoss application server automatically generates the web service description catalogue every time a web service application inside of a Java archive (JAR) file is deployed. The `SOAPBinding` annotation plays an important role and affects the structure of the WSDL catalogue. The use of the annotation is not mandatory. If the web service is not annotated with it, the application server use the default SOAP binding. As in the implementation of the `SOAPBinding` annotation could be seen, the default style is `DOCUMENT` style, the default value for use is `LITERAL` and the default parameter style is `WRAPPED`.



**Example 4.13. Java Annotation: @SOAPBinding**

```
package javax.jws.soap;

@Target({ElementType.METHOD})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface SOAPBinding {
    public enum Style {DOCUMENT, RPC};
    public enum Use {LITERAL, ENCODED};
    public enum ParameterStyle {BARE, WRAPPED}

    Style style() default Style.DOCUMENT;
    Use use() default Use.LITERAL;
    ParameterStyle parameterStyle()
        default ParameterStyle.WRAPPED;
}
```

What are the differences between RPC and DOCUMENT SOAP binding style?

“The choice corresponds to how the SOAP payload - i.e., how the contents of the <Soap body> element - can be structured. Here are some details of how each style affects the contents of <Soap body>. [...] Document: the content of <soap:Body> is specified by XML Schema defined in the <wsdl:type> section. It does not need to follow specific SOAP conventions. In short, the SOAP message is sent as one "document" in the <soap:Body> element without additional formatting rules having to be considered. Document style is the default choice. [...] RPC: The structure of an RPC style <soap:Body> element needs to comply with the rules specified in detail in Section 7 of the SOAP 1.1 specification. According to these rules, <soap:Body> may contain only one element that is named after the operation, and all parameters must be represented as sub-elements of this wrapper element. As a consequence of the freedom of choice that the document style offers, the SOAP messages conforming to a document style WSDL may look exactly the same as the RPC equivalent. The decisive question now is: What are the consequences of choosing one option or another? Why choose RPC over document, or document over RPC? In many cases, the SOAP messages generated from either RPC or document style WSDLs look exactly the same - so why offer the choice at all? The reason may be found in the history of the SOAP standard. SOAP has its roots in synchronous remote procedure calls over HTTP and the appearance of the

document accordingly followed these conventions. Later, it was seen as a simplification to use arbitrary XML in the SOAP body without adhering to conventions. This preference is reflected in the document style WSDL documents. So far, both options are represented in the WSDL specification and the choice of one or the other is mainly a question of personal taste since most SOAP clients today accept both versions.” [ROTHAUG 04]

In the implementation of the rule translator Web Service the DOCUMENT style SOAP binding was chosen simply because the implementation of the PHP Client was more convenient and straightforward. There were also nothing that spoke against choosing this binding style.

### 3.5.3. URL Injection during Deployment Time

In section Section 3.5.1, “Java Annotations” the explanation of the `Resource` annotation was postponed to this section. In Chapter 3, *Design* was explained that the web service use a distributed repository to receive the translation files. The URL of this repository can depend on the underlying network and differs in each environment. When you develop such a web service you have maybe one machine and several servers are running on this machine. This is your development environment. But you have also a productivity environment. A real network with several distributed machines each with a different URL. It would be a bad solution to re-compile the web service, every time it should be used in another environment. Imagine after some time of developing your web service has reached a final state and is running stable. Now you want to move the repository to another server. Does it make sense to compile the web service again? No, but it makes sense to modify a user changeable property of the web service. The URL of the translation repository. This is possible with the `Resource` annotation. It gives the opportunity to declare a object which refers to an external resource.

**Example 4.14. Java Annotation: @Resource**

```
package javax.annotation;

@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface Resource {
    public enum AuthenticationType {CONTAINER, APPLICATION}
    String name() default "";
    Class type() default Object.class;
    AuthenticationType authenticationType()
        default AuthenticationType.CONTAINER;
    boolean shareable() default true;
    String description() default "";
    String mappedName() default "";
}
```

In Example 4.8, “Stateless Bean Web Service” you see the usage of the annotation. You have to use it at with the parameter `mappedName`, which specifies a unique name in the JNDI<sup>10</sup>. Therefore you need to add your unique name to name of the standard JNDI environment `java:comp/env/`. The `Resource` annotation need to appear in front of a type declaration. Over the object of the declaration you can access afterwards the external resource in your source code and use it like any other ordinary Java object.

The `Resource` annotation is highly overloaded and could also be used to set methods, member fields or on the class itself. For further usage patterns I suggest to read the specification [JSR220] or the book *Enterprise JavaBeans 3.0* [BM 06].

The initialisation of the injected value has to be made from outside of the Java source code by the help of a deployment descriptor. The explanation of these descriptors is topic of the next section.

### 3.5.4. Deployment Descriptors

Deployment descriptors are XML files and the alternative to Java annotations. Everything that can be specified with annotation, could made as well with a deployment

---

<sup>10</sup> Java Naming and Directory Interface [<http://java.sun.com/products/jndi/tutorial/>]

descriptor. The `ejb-jar.xml` file helps us to specify the value of the object `translationRepositoryURL` which already was declared inside the Java source code. During the packaging process of the application the file need to have the name `ejb-jar.xml` and must appear inside of the `META-INF` folder of the Java archive.

**Example 4.15. Deployment Descriptor: ejb-jar.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
  version="3.0">
  <enterprise-beans>
    <session>
      <ejb-name>Webservice</ejb-name>
      <ejb-class>r2ml.Webservice</ejb-class>
      <env-entry>
        <env-entry-name>
          translationRepositoryURL
        </env-entry-name>
        <env-entry-type>
          java.lang.String
        </env-entry-type>
        <env-entry-value>
          http://hydrogen/trans/
        </env-entry-value>
        <injection-target>
          <injection-target-class>
            r2ml.Webservice
          </injection-target-class>
          <injection-target-name>
            translationRepositoryURL
          </injection-target-name>
        </injection-target>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

In Example 4.15, “Deployment Descriptor: ejb-jar.xml” you see the content of the file. The `ejb-name` is the class name, `ejb-class` is the package and class name of your web service. The name we specified as `mappedName` parameter in the `Resource` annotation in the Java source code before, has to match the value of the `env-entry-name` element. The `Resource` annotation was in Example 4.8, “Stateless Bean Web Service” previously written in front of `Java String` declaration. Since the type `String` belongs to the `java.lang` package, we have to specify this as well with the name of the type as value of the `env-entry-type` element to match the same type as in the Java source code. The `env-entry-value` is the value that should be injected into the Enterprise JavaBean. The `injected-target-class` is the target class of the injection, again written with her package name. The `injection-target-name` is the name of the object inside the Java source code where the value should be injected.

### 3.5.5. Custom Exceptions

The clients need to receive an error messages when they or the web service itself produce an error. If during the life cycle any kind of exception happens, then the EJB container creates a `Java SOAPException` and sends a SOAP fault message, instead of the operation result, back to the client.

**Example 4.16. throwing of a SOAPException**

```
@WebMethod
public String getGreeting(
    @WebParam(name="firstName") String name)
throws SOAPException {
    if(name == null) {
        throw new SOAPException("empty parameter");
    } else {
        ...
    }
}
```

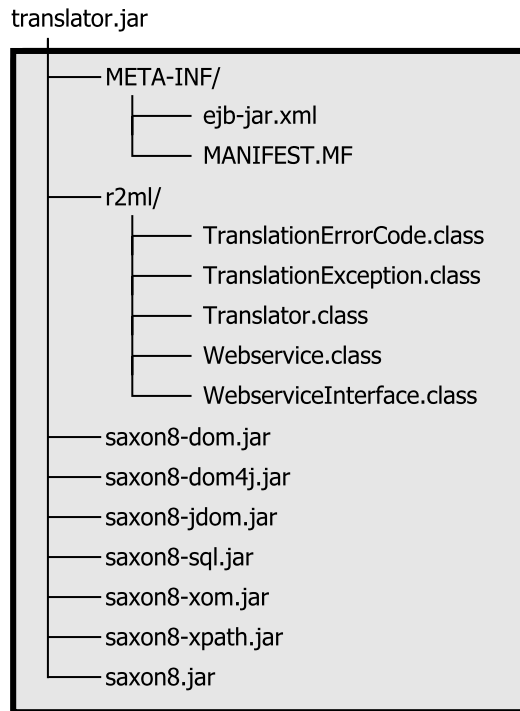
In Example 4.16, “throwing of a SOAPException” you see how a SOAPException can be thrown by the web service. It does not matter which type of exception is be thrown, because all exceptions are send as SOAP fault messages back to the client. But the name of the type of exception that is specified after the keyword throws in the Java method declaration is the type that is mapped to the WSDL catalogue. That means SOAPException appears in the WSDL catalogue. To change this name we need to extend the class SOAPException. In our case therefore the class name TranslationException appears in the WSDL catalogue, which has a clearer meaning in contradiction to the name SOAPException.

**3.5.6. Packaging of the Application**

The packaging of the enterprise application is an important step during and after the development. As we already know from the previous sections the ejb-jar.xml descriptor file is responsible to set up the right URL to the translation descriptor file translations.xml. But this is maybe a different path in the development stage where we maybe all server instances running on the same machine. Therefore it is a good idea to create another deployment descriptor file only for our development environment. The file name does not really matter, here we name it ejb-jar-local.xml. In Example 4.15, “Deployment Descriptor: ejb-jar.xml” you see the content of the ejb-jar.xml file. The content of the ejb-jar-local.xml differs only in one line. Inside the ejb-jar.xml descriptor file the element env-entry-value has the value http://hydrogen/trans/ which points to the folder trans on server hydrogen inside of the productivity environment of the

web service. The value inside `ejb-jar-local.xml` is `http://localhost/R2MLTranslatorWS_Translations/` and point to the local running web server and the folder `R2MLTranslator_Translations` on the same machine.

**Figure 4.3. Packaging of the Translator Web Service**



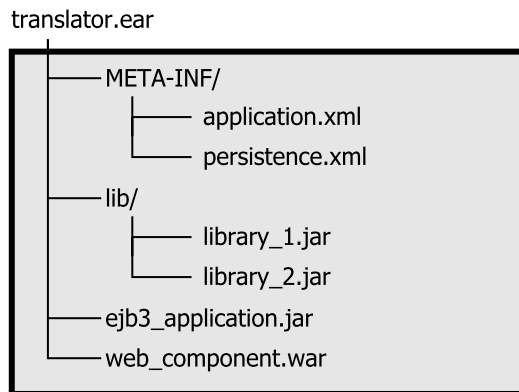
Maybe you have already wondered why the class name `Webservice` instead of `TranslatorWebservice` was chosen in the implementations model. First, the name `WebService` with the capital letter `S` was already used by the `@WebService` annotation. But the name `Webservice` with the small letter `s` was available. A class name like `TranslatorWebservice` would of course been a better and more clearer choice, but here we need knowledge about how the application server create the endpoint URL's for the web services. The endpoint URL is created from the name of the server in addition with his port, both separated by a colon e.g. `localhost:8080`. The name of the enterprise application archive, here `translator` (without the `.jar` extension), and the name of the class, here `Webservice`, that implements the remote (business) interface of the Enterprise JavaBean web service is used as well. Every part is separated by a slash. The complete endpoint URL for Figure 4.3, “Packaging of the Translator Web Service” deployed on your local machine on port 8080 is therefore `http://localhost:8080/translator/Webservice`. In order to receive the WSDL



catalogue, the parameter WSDL need to be added to the endpoint URL of the web service. The URL for the WSDL catalogue is therefore `http://localhost:8080/translator/Webservice?WSDL`.

Enterprise JavaBeans applications are packaged to Java archive (JAR). When the application has also a web component then this component is packaged to a web component archive (WAR). Both the JAR and the WAR archive are packaged afterwards to an enterprise application archive (EAR). The structure of such an example enterprise application archive could be seen in Figure 4.4, “Packaging of an Enterprise Archive”.

**Figure 4.4. Packaging of an Enterprise Archive**



The deployment descriptor files `application.xml` and `persistence.xml` are not required for the EAR archive and can also be substituted by Java Annotations. The libraries in the `lib` folder are shared libraries used by both, the Enterprise JavaBeans 3.0 and the web component.

For the automation of the packaging process we can use the Apache ANT<sup>11</sup> tool. ANT can be used inside of an integrated development environment like Eclipse<sup>12</sup> or on the command line. ANT read XML files and executes the script code inside. The default file name of an ANT script file is `build.xml`.

---

<sup>11</sup> <http://ant.apache.org/>

<sup>12</sup> <http://www.eclipse.org/>

**Example 4.17. ANT build.xml File**

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="R2ML Translator Web Service"
    default="build (chair net)" basedir=".">

    <property name="ejb3.filename" value="translator.jar"/>
    <property name="JBoss_local.path"
        value="c:/Programme/jboss-4.0.5.GA/server/
            default/deploy"/>

    <description>
    This ANT file builds the JAR archive for the JBoss
    application server.
    </description>

    <target name="drop"
        description="drop the EJB3 JAR archive">
        <delete>
            <fileset dir="." includes="{ejb3.filename}"/>
        </delete>
    </target>

    <target name="create bin"
        description="build the application for the JBoss server
        of the chair network">
        <mkdir dir="bin"/>
        <mkdir dir="bin/META-INF"/>
        <copy todir="bin">
            <fileset dir="WEB-INF/classes"/>
        </copy>
        <copy todir="bin">
            <fileset dir="WEB-INF/lib" excludes="javaee.jar"/>
        </copy>
    </target>

    <target name="build (chair net)" depends="create bin"
```

```
description="build the application for the
productivity environement" >

<!-- use the ejb-jar.xml file with the path to the
translation descriptor of the productivity env. -->
<copy file="META-INF/ejb-jar.xml"
      todir="bin/META-INF"/>
<jar destfile="${ejb3.filename}">
  <fileset dir="bin"/>
</jar>
<delete dir="bin"/>
</target>

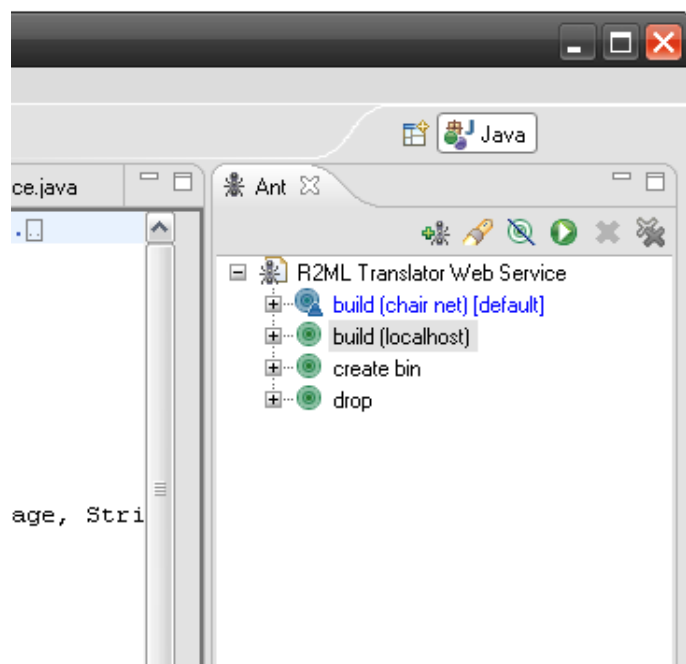
<target name="build (localhost)" depends="create bin"
description="build the application for the
development environment and deploy it" >
<!-- use the ejb-jar.xml file with the path to the
local web server -->
<copy file="META-INF/ejb-jar-local.xml"
      tofile="bin/META-INF/ejb-jar.xml"/>
<jar destfile="${ejb3.filename}">
  <fileset dir="bin"/>
</jar>
<delete dir="bin"/>
<copy file="translator.jar"
      toDir="${JBoss_local.path}"/>
</target>
</project>
```

Every ANT script has a `project` tag where the name of the project, the default task and the `basedir` the base directory is specified. Nested inside of the `project` are `property` tags used to define variables that can be used everywhere in the rest of the script. The value of a property can be accessed by writing `${name of the property}`. Tasks are specified with `target` tags. They have at least a name and a `description` attribute. The attribute `depends` specifies what other tasks are performed by ANT before it start this very task. The values of the attribute `depends` are comma separated

values of the name attributes of other tasks in the `project`. The `jar` tag packages a folder structure as Java archive file. In this case the previously created folder `bin` is packaged to the file name `translator.jar`. To check the content of a Java archive you can use any ZIP extraction program, because the same compression algorithm than in the ZIP format is used for the creation of Java archives. On a Windows XP machine make first a copy of your JAR file, rename the file extension to `.zip` and double-click on the ZIP file in order to open and view it. But the other way around is not possible. A ZIP archive is not a Java archive. Inside of a Java archive you find always a `MANIFEST.MF` file as could be seen in Figure 4.3, “Packaging of the Translator Web Service”. This is another benefit of using ANT for the packaging of the application. We have not to care much about these things and can package more complex application than with the console JAR tool of the Java Standard Developer Kit. All other ANT tags used in the `build.xml` are self explaining. A complete documentation can be found at ANT project website.

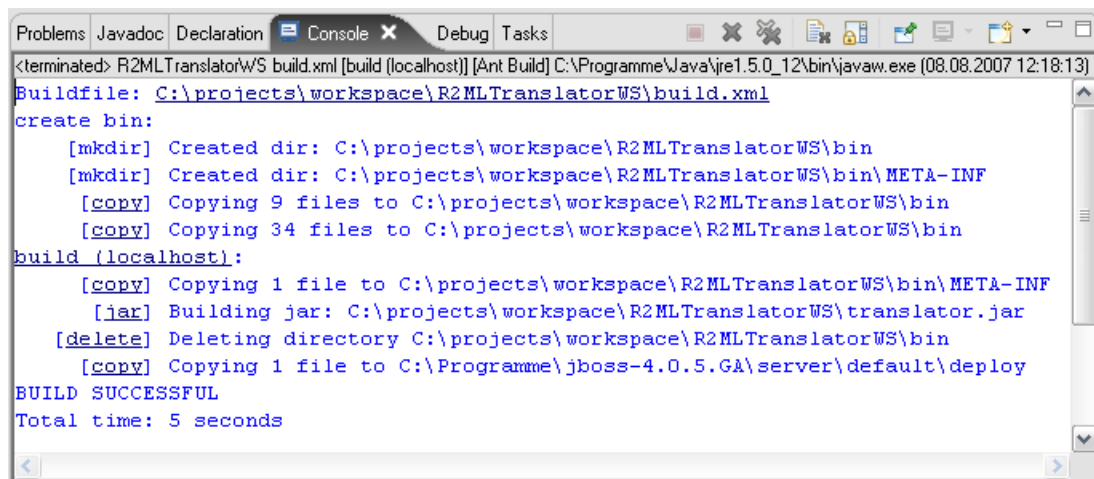
In Figure 4.5, “ANT Integration in the Eclipse IDE” could be seen how ANT is integrated into the integrated development environment Eclipse. Every ANT target of the `build.xml` appears there as a button.

**Figure 4.5. ANT Integration in the Eclipse IDE**



When the user double-clicks on a button, the selected task is started and the output written to the integrated console. This makes the packaging and deploying of the application as easy as clicking a button.

**Figure 4.6. ANT Console Output in the Eclipse IDE**

The image shows a screenshot of the Eclipse IDE's console window. The console displays the output of an ANT build process. The output starts with the buildfile path: C:\projects\workspace\R2MLTranslatorWS\build.xml. The build process includes creating a 'bin' directory, copying 9 files to the bin directory, and copying 34 files to the bin directory. The build process then copies 1 file to the META-INF directory, builds a jar file named translator.jar, deletes the bin directory, and copies 1 file to the JBoss server's default deploy directory. The build process ends with the message 'BUILD SUCCESSFUL' and a total time of 5 seconds.

```
<terminated> R2MLTranslatorWS build.xml [build (localhost)] [Ant Build] C:\Programme\Java\jre1.5.0_12\bin\javaw.exe (08.08.2007 12:18:13)
Buildfile: C:\projects\workspace\R2MLTranslatorWS\build.xml
create bin:
  [mkdir] Created dir: C:\projects\workspace\R2MLTranslatorWS\bin
  [mkdir] Created dir: C:\projects\workspace\R2MLTranslatorWS\bin\META-INF
  [copy] Copying 9 files to C:\projects\workspace\R2MLTranslatorWS\bin
  [copy] Copying 34 files to C:\projects\workspace\R2MLTranslatorWS\bin
build (localhost):
  [copy] Copying 1 file to C:\projects\workspace\R2MLTranslatorWS\bin\META-INF
  [jar] Building jar: C:\projects\workspace\R2MLTranslatorWS\translator.jar
  [delete] Deleting directory C:\projects\workspace\R2MLTranslatorWS\bin
  [copy] Copying 1 file to C:\Programme\jboss-4.0.5.GA\server\default\deploy
BUILD SUCCESSFUL
Total time: 5 seconds
```

### 3.5.7. Translation of Rule Languages

The translation of rule languages inside of the translator system is made with the help of so called XSLT files and an XSLT 2.0 processor. XSL is the abbreviation for eXtensible Stylesheet Language. The letter *T* in the abbreviation XSLT stand for transformation. An XSLT processor transforms XML data with the help of an XSLT in order to create a new representation of the same data. In the case of the rule translator we transform with the help of an XSLT 2.0 processor rules from one representation into another representation. The meaning of the rule is in both representation formats the same. The representation format we called already before a rule languages. Concrete languages names are e.g. F-Logic, Jena, Jess, JBoss Rules, RuleML, R2ML or Oracle Business Rules.

In Section 1, “Rule Languages” we already transformed a rule from F-Logic step by step to R2ML. Now we take a look in the XSLT file that describes the R2ML to F-Logic translation. Since the complete XSLT translation file has more than 850 lines of source code, only the mapping of R2ML's `ReferencePropertyAtom` is explained here. This was the only kind of atom used for the markup of the ancestor relation in R2ML.

**Example 4.18. R2ML ReferencePropertyAtom**

```

<?xml version="1.0" encoding="UTF-8"?>
<r2ml:RuleBase
  xmlns:r2ml="http://www.reverse.net/I1/2006/R2ML">
  ...
<r2ml:ReferencePropertyAtom
  r2ml:referencePropertyID="father">
  <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="X"/>
  </r2ml:subject>
  <r2ml:object>
    <r2ml:ObjectVariable r2ml:name="Z"/>
  </r2ml:object>
</r2ml:ReferencePropertyAtom>
  ...
</r2ml:RuleBase>

```

The Example 4.18, “R2ML ReferencePropertyAtom” is not valid against the XML schema of R2ML. A complete R2ML rule can be validated against the XML schema of R2ML in order to be sure that the structure is valid. The valid R2ML markup, where this snippet is taken from, you can find in the Appendix A, . Since the right structure of the data and not the data itself is relevant for the translation, we need to look only at an abstract ReferencePropertyAtom in order to understand how the translation of all reference property atoms are transformed.

**Example 4.19. XSLT Snippet of the R2ML to F-Logic Translation**

```

00| <?xml version="1.0" encoding="UTF-8"?>
01| <xsl:stylesheet version="1.0"
02|     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
03|     xmlns:r2ml="http://www.reverse.net/I1/2006/R2ML"
04|     xmlns:dc="http://purl.org/dc/elements/1.1/">
05| <xsl:output method="text"/>

```

The <xsl:stylesheet> tag is the root element in our XSLT stylesheet. Therefore it is the first element after the XML prolog. With the version attribute in

`<xsl:stylesheet>` the XSLT version need to be indicated. The `xmlns` attribute declares XML namespaces. With `xsl` the namespace of XSLT, with `r2ml` for R2ML and with `dc` the namespace of Dublin Core<sup>13</sup> which is been used by R2ML. With the `<xsl:output/>` tag the output parameters can be specified. The `method` attribute defines that the output format of this transformation is `text`.

```
06 | <xsl:template match="/">
07 |   <xsl:apply-templates/>
08 | </xsl:template>
```

The `<xsl:template>` tag has an attribute `match`. The value of the attribute is a slash, which symbolise the root node of an XML structure. The empty element tag `<xsl:apply-templates/>` means that templates should be applied to nested elements of the matched node. This are in our case the child elements of the node `r2ml:RuleBase`.

```
09 | <xsl:template match="r2ml:ReferencePropertyAtom">
10 |   <xsl:if test="string(./@r2ml:isNegated)='true'">
11 |     <xsl:text>NOT </xsl:text>
12 |   </xsl:if>
```

The template above matches all `r2ml:ReferecncePropertyAtom` elements in an XML representation. The `<xsl:if>` element has always an attribute `test`, here with the value `string(./@r2ml:isNegated)='true'` . This statement is written as XPath function<sup>14</sup> and means: *If the string representation of the attribute value of `r2ml:isNegated` of the currently applied node (here `r2ml:ReferencePropertyAtom`) is the same than the character string `true`, then the value of the attribute `test` is true.* If the attribute `test` is true, then the child elements are processed. The `<xsl:text>` elements simply print the nested text, here it is the word `NOT` followed by a space character.

```
13 |   <xsl:apply-templates select="r2ml:subject"/>
14 |   <xsl:text>[</xsl:text>
```

---

<sup>13</sup>“The Dublin Core Metadata Initiative is an open organization engaged in the development of interoperable online metadata standards that support a broad range of purposes and business models.” ( <http://dublincore.org/>, August 15, 2007)

<sup>14</sup> <http://www.w3.org/TR/xpath-functions/>

```
15| <xsl:value-of
16|   select="translate(string(@r2ml:referencePropertyID),
17|     ':','\#')"/>
18| <xsl:text>-&gt;</xsl:text>
19| <xsl:apply-templates select="r2ml:object"/>
20| <xsl:text>]</xsl:text>
21| </xsl:template>
```

The `<apply-templates>` tag in line 13 applies the right template directly to all matching child nodes `r2ml:subject`. The `xsl:text` tag prints here an opening bracket. The `xsl:value-of` tag prints out the value of the selected statement. The statement `translate(string(@r2ml:referencePropertyID), ':','\#')` means in natural language: *the value of the attribute `r2ml:referencePropertyID` as character string, where all `:` in the string are exchanged with (or translated to) the `#` sign.*

```
22| <xsl:template match="r2ml:subject | r2ml:object">
23|   <xsl:apply-templates select="child::node()"/>
24| </xsl:template>
```

When the template for `r2ml:subject` or `r2ml:object` is applied, it does nothing more than simply apply again a matching template to the child node of the current node.

```
25| <xsl:template match="r2ml:ObjectVariable">
26|   <xsl:text>?</xsl:text>
27|   <xsl:value-of select="@r2ml:name"/>
28| </xsl:template>
29| </xsl:stylesheet>
```

The template for `r2ml:ObjectVariable` prints simply out the question mark sign followed by the value of the attribute `r2ml:name` of the current node, which is the node `r2ml:ObjectVariable`.

With `<xsl:message>` tags, error messages or informations can be created inside of XSLT transformations. You remember in Section 4.2.4, “The Translator Error Codes” an error code document was introduced. These error codes are for the developer of translations and the translator. The usage of these error codes in the translator part is explained in Section 3.5.8, “Using the Saxon 8 XSLT 2.0 processor”. In addition to the



last XSLT transformation you can see in Example 4.20, “Usage of `xsl:message`” how easy such a message can be created inside of XSLT transformations.

#### **Example 4.20. Usage of `xsl:message`**

```
<xsl:template match="r2ml:ProductionRuleSet
| r2ml:IntegrityRuleSet | r2ml:ReactionRuleSet">
  <xsl:message terminate="yes">
    <xsl:text>[ERR0007] The translation input is not a
    derivation rule.</xsl:text>
  </xsl:message>
</xsl:template>
```

F-Logic can only express derivation rules. When for instance the transformed R2ML rule is marked up as production rule, the XSLT translation need to create an error message. The attribute `terminate="yes"` specifies that the XSLT transformation need to be aborted. When the value is `no` then the message is written to the output along with the translation result. This behaviour can be used to create additional informations. But this is not useful for a translation web service.

As you can see, there is no magic behind a translation of rules. When rules are marked up right and the constructs of the source language can be distinguished exactly, it is simple and straightforward.

### **3.5.8. Using the Saxon 8 XSLT 2.0 processor**

In the previous section we saw how a XSLT transformation or translation of XML based languages could look like. The translation was really simple and did not use any element of XSLT 2.0 . But to give the translation developers the opportunity to use the latest version of XSLT the decision felt to use the Saxon 8 XSLT 2.0 processor. In this section is explained how Saxon 8 instead of the default Xalan XSLT 1.0 processor can be used in the Java programming language.

**Example 4.21. Implementation of the translateDirect() Operation**

```
00|private String translateDirect(String sourceLanguage,  
01|                               String targetLanguage, String xmlInput)  
02|throws TranslationException {  
03| String result = "";  
04| URL url = getTransformation(sourceLanguage,  
05|   targetLanguage);  
06|  
07| StringWriter messageStringWriter = new StringWriter();  
08| MessageEmitter messageEmitter = new MessageEmitter();
```

The operation `translateDirect` we already know from the Chapter 3, *Design* especially from Figure 3.8, “Sequence Diagram: Translation of a Rule”. In line 2 you see that the operation can throw translation exceptions, which need to be handled by the operation that want to use this operation. An empty string object `result` is created. This string is used later to return the result of the direct translation after everything is finished. With the source and target language the URL<sup>15</sup> to the matching XSLT translation is received. In order to support the `<xsl:message>` tag, a `StringWriter` and a `MessageEmitter` object need to be instantiated.

```
09| try {  
10|   System.setProperty(  
11|     "javax.xml.transform.TransformerFactory",  
12|     "net.sf.saxon.TransformerFactoryImpl");  
13|  
14|   TransformerFactory transformerFactory  
15|     = TransformerFactory.newInstance();  
16|  
17|   Transformer transformer  
18|     = transformerFactory.newTransformer(  
19|       new StreamSource(url.openStream()));  
20|  
21|   messageEmitter.setWriter(messageStringWriter);  
22|   Controller controller = (Controller) transformer;
```

---

<sup>15</sup>Unified Resource Locator

```
23| controller.setMessageEmitter(messageEmitter);
```

In line 10 you see we set the implementation class that should be used when we instantiate a new object of the class `TransformerFactory`. When we set the property `javax.xml.transform.TransformerFactory` to `net.sf.saxon.TransformerFactoryImpl` then we want to use the implementation of Saxon 8 instead the default one (Xalan). Of course all libraries of Saxon need to be accessible in the Java classpath.

The transformer object is necessary to transform later the rule in XML structure to another rule representation. A new object of the class `Transformer` is created with the help of the previously instantiated object `transformerFactory` and the operation `newTransformer`. This operation need as parameter a `StreamSource` object. This object is created with the streamed content of the XSLT translation by calling the operation `openStream` of the `url` object which points to the XSLT translation file.

In line 21 you see that the previously declared object `messageStringWriter` of the class `StringWriter` is set as writer of the object `messageEmitter`. In the next line is a new object `controller` created by type casting the object `transformer` to an object of the class `Controller`. This need to done in order to use the operation `setMessageEmitter` which is only available for objects of the class `Controller`. By calling this operation we set the object `messageEmitter` as message emitter of the object `controller`. When inside the XSLT transformation an `<xsl:message>` is being created, the emitter writes the error message to the `messageStringWriter` object.

```
24| StringReader inputXML=new StringReader(xmlInput.trim());
25| StringWriter stringWriter = new StringWriter();
26| transformer.setOutputProperty(
27|     OutputKeys.OMIT_XML_DECLARATION, "no");
28| transformer.setOutputProperty(
29|     OutputKeys.METHOD, "xml");
30| transformer.setOutputProperty(
31|     "{http://xml.apache.org/xslt}indent-amount", "4");
32| transformer.setOutputProperty(
33|     OutputKeys.INDENT, "yes");
34| transformer.transform(new StreamSource(inputXML),
35|     new StreamResult(stringWriter));
```

```
36|  
37| result += stringWriter.toString();
```

The `trim()` operation removes all whitespaces from the beginning and the end of a string and it trims all ASCII control characters as well. With the `xmlInput`, which holds a rule in XML structure as a string, a new object `inputXML` of the class `StringReader` is created in line 24. Also a instance of the class `StringWriter` for the result of the XSLT transformation is required. From line 26 to 33 the default output properties for translations are overridden. The transformer should not omit the XML declaration, which is also known as XML prolog, when he is creating the output. As well should XML be the output method and tags should be indented with the amount of 4 characters.

In line 34 the `transform` operation of the transformer is called with two parameters. The first parameter is the XML rule represented by the object `inputXML` as new instance of the class `StreamSource`. The second is the object `stringWriter` as a new instance of the class `StreamResult`. That means in this line we set the XML input and write the result to the `stringWriter`.

The operation `+=` appends the string representation of the content of the object `stringWriter` to the `result` string.

```
38| } catch (javax.xml.transform.TransformerException te){  
39|   if( (messageStringWriter.toString()).equals("") ) {  
40|     throw new TranslationException(  
41|       TranslationErrorCode.ERR0011  
42|       + " (" + te.getMessageAndLocation() + ")");  
43|   } else { //if a xsl:message appears,  
44|     throw new TranslationException(  
45|       messageStringWriter.toString());  
46|   } //if
```

The part above describes what should be done if during the transformation an exception happens. In line 39 you see that if the string representation of the object `messageStringWriter` is empty, a new `TranslatorException` should be thrown. This means the operation that called this `translatedirect` operation need to handle this exception. The exception reason is a concatenation of the error message of

the translation error code 11 and the message plus location of the error in the structure of the XML rule.

The `else` branch defines the behaviour which should happen when at least one `<xsl:message>` was created during the transformation. In this case all messages are used to create a translation exception as well.

The difference between both cases is the first case is an exception created by the XSLT processor. The second in contradiction, is a exception created by the XSLT translation.

```
47| } catch (IOException ioe) {
48|   System.err.println(THIS_APPLICATION
49|   + "created an input/output exception "
50|   + "during translation.");
51| }//catch
52| return result;
53|} //translateDirect()
```

In line 19 we use the operation `openStream` on a object of the class `URL`. In case the file at the location could not be found an input output exception need to be caught. As you see above no translator exception is thrown. If this case happens, there is something wrong with the translations files and therefore with the complete translator system. No client need to know about this. A good solution here would be to inform the administrator per email. The explanation of such an implementation is not purpose of this document.

---

## Chapter 5. Web Interface

In this chapter a web service client is introduced. The client act as an web interface for the previously build rule translator web service. He shows how web services can be accessed. Since the client is pretty easy to implement, we do not need to use the Model Driven Architecture (MDA).

A web interface for a web service differs not much from other dynamical created web appearances. The user of the web interface does not even know that a web service is used in the background. The connection to a web service can be made by the programmer itself by parsing the web service description catalogue and creating the corresponding SOAP messages. Or the programmer can do that with a SOAP client library that handles and hides all the complicated things. For the dynamical website that could be seen in Figure 5.1, “R2ML Translator Web Interface” the latter solution was chosen because this solution is more convenient and error proof. There are at least two different types of available SOAP libraries. Both are created from the web service description catalogue. The first kind of SOAP library is created on the fly. This solution is used by script languages like PHP, JavaScript<sup>1</sup> or ECMAScript. Script languages are always interpreted and not compiled. For compiled languages like Java or C#, the source codes of the library is generated and need to be compiled afterwards. Advanced programming interfaces (API) for any kind of Java applications can be generated with Apache Axis. Such a generated API allows us to connect to this specific web service from the JavaBean of the web presentation layers like JavaServer Pages or JavaServer Faces.

---

<sup>1</sup>**JavaScript SOAP Client:** <http://www.codeproject.com/Ajax/JavaScriptSOAPClient.asp> (13/08/2007)

**Figure 5.1. R2ML Translator Web Interface**

The Hypertext Preprocessor language (PHP) is an open source script language for web servers. Today almost every web server supports it. PHP is easy understandable and provided by many web space hosting vendors. Thus makes it easy for almost every web developer to use web services. Choosing the PHP language for the web interface has only one reason. It is a completely different language than Java, which is used for the Enterprise JavaBeans 3.0 web service, and this shows the language independence of the translator web service. Another free opportunity would have been a web interface made in JavaServer Pages. But for JavaServer Pages running on the same application server we do not need really a web service. In this case we would have packaged the JavaServer Page in one enterprise archive and connected them internally via the local business interface of the Enterprise JavaBean.

## 1. Design Hints

Before a dynamical website is made it is good to create first a static version. This gives us the opportunity to test and design it. A website is marked up with the help of the Hypertext Markup Language (HTML). The design of a website is defined with Cascading Style Sheets (CSS). It is possible to style HTML elements, classes or elements with a special identity. In Example 5.1, “CSS for HTML textarea element” you see a snippet CSS code that styles all textarea elements.

**Example 5.1. CSS for HTML textarea element**

```
textarea
{
  position: relative;
  float: left;
  left: 1em;
  width: 98%;
  height: 40.5%;
}
```

The width and height of the `textarea` element has a relative value. That means the width of the `textarea` element is 98 percent of the available width and the height is 40.5 percent of the available height of the current browser window. Such CSS code make your website independent from a fixed browser resolution. This is an important issue. People today are using very high display resolutions and their browser maybe full screen. When people want to use the web interface, of course they want to have a benefit of their huge display and of course they want to have a huge text area where they have a better overview over the rule markup code.

Actually, more than 5 different display resolutions are common. An overview can be seen is Table 5.1, “Displays and Resolutions”.

**Table 5.1. Displays and Resolutions**

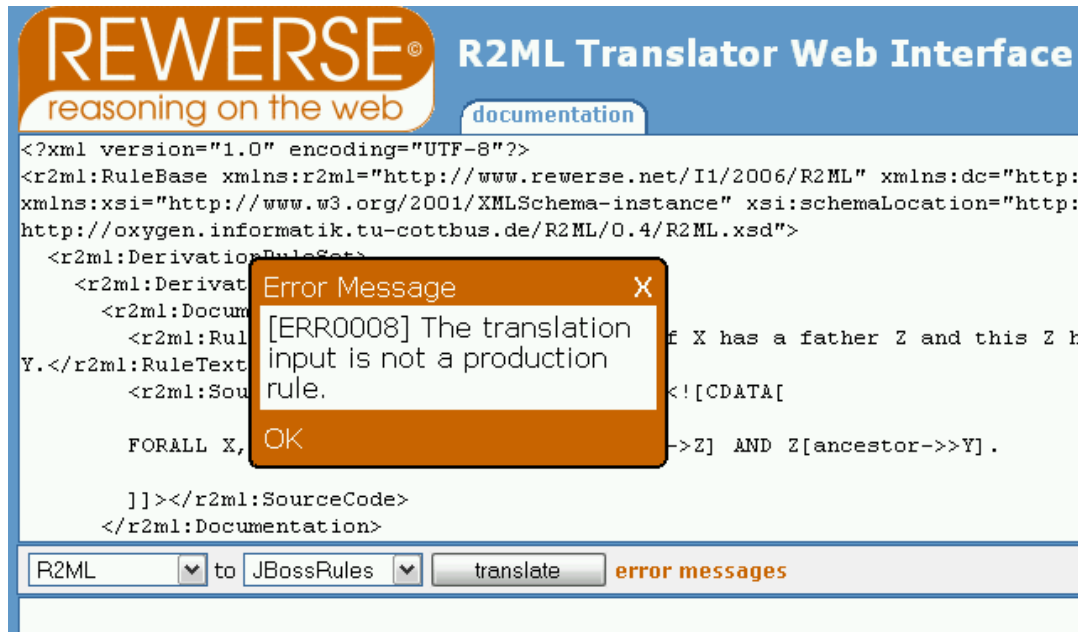
Size [inch]	Resolution [pixel]	Format
12", 14", 15"	1024 x 786	XVGA
17"	1280 x 1024	SXGA
19"	1400 x 1080	SXGA+
22"	1600 x 1200	UXGA+
22" (wide screen)	1680 x 1050	WSXGA+

Another issue are JavaScript pop up windows. These pop up windows are often used to display advertising when people are surfing around in the Internet. The Firefox web browser for instance has a option to block pop up windows. Thus makes it hard to show



warning or error messages in form of a new window. When we use JavaScript's pop up windows there is a huge chance that these warning or error messages are never be seen by the user. The question is: *How can we display warning or error messages when the browser would block pop up windows?*

**Figure 5.2. Pop-up Blocker Resistant Message Window**



Pop-up blocker resistant windows can be created very simple. As you can see in Example 5.2, "Pop-up Blocker Resistant Message Window", an area is created with the help of the `<div>` element. The style attribute defines that this area has a width of 250 pixel and appears 150 pixel away from the left and top side in the browser. In line 3 can be seen that the style attribute definition which starts in line 1 is not closed with a second double quote. In line 4 the `<?php` characters defines that PHP script code are following until the tag is closed with the `?>` characters.

**Example 5.2. Pop-up Blocker Resistant Message Window**

```
00| <div id="ErrorWindow"
01|     style="position:absolute;
02|         width:250px; left:150px; top:150px;
03|         border: 1px solid black;
04| <?php
05|     if($showErrorWindow) {
06|         print 'visibility: visible;';
07|     } else {
08|         print 'visibility: hidden;';
09|     }
10| ?>
11| ">
12| <p><?php print $exception->detail->
        TranslationException->message; ?></p>
13| <a href="javascript:hideErrorWindow();
        return false">OK</a>
14| </div>
```

The PHP script code from line 5 to 9 means that *if the variable `showErrorWindow` is true then print the characters string `visibility: visible;` otherwise print `visibility: hidden;`*. PHP is a script language that is being executed on the server. If a error appears the variable `showErrorWindow` is set to true and the HTML code is generated on the web server. Afterwards it is being send to the web browser that interprets the HTML markup. The browser never see anything of the PHP code. In the error case, the browser see the in the style attribute `visibility: visible` and this tells him to show the `<div>` area and all nested elements. In the example above in line 12, a nested paragraph element is used to show the error message. Again PHP is used to insert the content. It is the message of the `TranslationException`, which is a detail of a super-imposed PHP exception. The operation call `hideErrorWindow()` is used to close the error window. The implementation of this JavaScript operation need to be executed by the web browser. He need to search for the element with the `id` value `ErrorWindow` (see line 1) and changes the `visibility` attribute inside of the style attribute to the value `hidden` in order to hide the error window.

## 2. Using Web Services with PHP

When you want to create a web interface for a web service like in Figure 5.1, “R2ML Translator Web Interface”, you need to put several technologies together. The HTML markup styled with cascading style sheets (CSS), the pop up blocker resisted way of showing warnings and error messages and the dynamic content creation controlled by PHP. In this section the parts where PHP is used to communicate with the rule translator Web Service, introduced in this document, are explained.

### Example 5.3. PHP Web Service Call: `getSourceLanguages()`

```
01|<html>
02|<body>
03|<h2>R2ML Translator Web Service</h2>
04|<?php
05| $WebServiceEndpointURL = 'http://hydrogen.informatik.
    tu-cottbus.de:8080/translator/Webservice' ;
06| $WSDL = "$WebServiceEndpointURL?wsdl" ;
07| $client = new SoapClient($WSDL);
08|
```

The endpoint URL of the web service is specified in line 5 with the assignment of a character string. The URL of the web service description catalogue is a concatenation of the endpoint URL and the characters `?wsdl`, as you can see in line 6. In order to access a web service we need to create a new object of the class `SoapClient`. The only mandatory parameter is the URL that points to the web service description catalogue. The class `SoapClient` provides afterwards the same operations than the web service and can be used in the same way as all other PHP operations.

```
09| try {
10|   $resultObject = $client->getSourceLanguages();
11|   $sourceLanguages = $resultObject->result;
12|   if(gettype($sourceLanguages) != 'array') {
13|     print "<b>Source Language:</b>";
14|     print $sourceLanguages;
15|   } else { //if we get an array of results
```

```
16| print "<b>Source Languages:</b>";
17| print "<ul>";
18|   foreach($sourceLanguages as $availableSourceLanguage){
19|     print "<li>$availableSourceLanguage</li>";
20|   }//foreach
21|   print "</ul>";
22| }//if-else
23|} catch (SoapFault $exception) {
24|print "ERROR in getSourceLanguages()";
25|} //try-catch
26|
27|?>
28|</body>
28|</html>
```

In order to catch exceptions we write the script code, that want to use an operation of the web service, inside of a try-catch statement. In line 10 you see how the operation `getSourceLanguages()` of the web service is used. The object `responseObject` that holds the result of the operation has an attribute `result`. The name of the attribute `result` corresponds to the value of the `name` attribute of the element which is defined in the `complexType` with the name `getSourceLanguagesResponse` in the XML Schema part of the web service description catalogue. The corresponding part you see in Example 5.4, “WSDL: Response Type Declaration”.

**Example 5.4. WSDL: Response Type Declaration**

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
  <types ...>
    <schema ...>

      <complexType name="getSourceLanguagesResponse">
        <sequence>
          <element name="result"
            minOccurs="0" maxOccurs="unbounded"
            nillable="true" type="string"/>
        </sequence>
      </complexType>

    </schema>
  </types>
</definitions>
```

In line 12 of Example 5.3, “PHP Web Service Call: `getSourceLanguages()`” you see how in PHP could be distinguished between the two kinds of result values. The first case is when the result attribute itself holds the result value. This is the case when only one result element is returned. The second case is that the type of the result is an array of values. Then the result values can be accessed like every PHP array. An example you see in line 18.

When the lines 9 to 25 of the PHP script code of Example 5.3, “PHP Web Service Call: `getSourceLanguages()`” are exchanged with the following script code, you have a second working example of the usage of an web service operation in PHP. The big difference to the previous example is that the operation `getTargetLanguages()` need a parameter that specify the source language.

**Example 5.5. PHP Web Service Call: `getTargetLanguages()`**

```
$sourceLanguage = 'R2ML';
$parameter = array('sourceLanguage' => $sourceLanguage);
try {
    print "<h3>Target Languages of $sourceLanguage :</h3>";
    $responseObject = $client->getTargetLanguages($parameter);
    $targetLanguages = $responseObject->result;
    if(gettype($targetLanguages) != 'array') {
        print "<b>Target Language of $sourceLanguage :</b>";
        print $targetLanguages;
    } else { //if we get an array of results
        print "<b>Target Languages of $sourceLanguage :</b>";
        print "<ul>";
        foreach($targetLanguages
                as $availableTargetLanguage) {
            print "<li>$availableTargetLanguage</li>";
        } //foreach
        print "</ul>";
    } //if-else
} catch (SoapFault $exception2) {
    print "ERROR in getTargetLanguages(...)";
} //try-catch
```

The usage of the `translate()` operation of the web service is almost the same than in Example 5.5, “PHP Web Service Call: `getTargetLanguages()`”. The parameters inside of a PHP array are separated with a comma.

## 2.1. SOAP Exception Handling

The good exception handling is an important part during the development of applications. In PHP it is not necessary in order to run the script, but to handle exceptions does also not make an application instable. Quite contrary does is make an application absolute more stable and let the developer think about error cases that could might happen.

**Example 5.6. PHP: SOAP Exception Handling**

```
$parameter = array('sourceLanguage' => $sourceLanguage,
                  'targetLanguage' => $targetLanguage,
                  'xmlInput' => $input);

try {
    $translation = $client->translate($parameter);
} catch (SoapFault $exception) {
    print $exception->detail->TranslationException->message;
} //try-catch
```

In the example PHP script code you see the call of the web service operation `translate` is wrapped in a try-catch clause. If a `SoapFault` exception happens this exception is printed. The attribute name `message` can be found again in the node element in the XML Schema section of the `complexType` declaration in the web service description catalogue. The attribute `TranslationException` is the name of the custom exception class we created in the implementation phase of the translator web service application. According to the SOAP 1.1 specification “is the detail element [...] intended for carrying application specific error information related to the Body element.” [DEK+ ]

Another much smarter idea is the following. Instead of simply printing out the exception message, this message could be stored in a PHP variable and showed inside of the error message window which was introduced in Section 1, “Design Hints”.

## 2.2. SOAPClient Bugs

PHP before version 5.1.6 had problems with the mapping of parameter names of operations. The parameters of the operation `translate(sourceLanguage, targetLanguage, xmlInput)` for instance are mapped to a complex type `translate` in the schema definition of types in the web service description catalogue (in document style). A snippet of the parameter mapping can be seen in Example 5.7, “WSDL: Parameter Mapping of an Operation”.

**Example 5.7. WSDL: Parameter Mapping of an Operation**

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
  <types ...>
    <schema ...>

      <complexType name="translate">
        <sequence>
          <element name="sourceLanguage"
            nillable="true" type="string"/>
          <element name="targetLanguage"
            nillable="true" type="string"/>
          <element name="xmlInput"
            nillable="true" type="string"/>
        </sequence>
      </complexType>

    </types>
  </schema>
</definitions>
```

The library SOAPClient that came along with PHP before version 5.1.6 did not parse the parameter names right. The library call every time the first parameter `String_1`, the second `String_2` and the third `String_3`. She works pretty well when we avoid to use the `@WebParam` annotation in our Enterprise JavaBean 3.0 web service to specify the name of the parameter in operations. But as we already know a parameter name like `sourceLanguages` says more than `String_1`. This bug in the PHP implementation was not so easy to find. The PHP web interface worked fine in the development, but not the productivity environment. After analysing the log files of the web server I had a clue. It was necessary to update and recompile the PHP implementation of the productivity server.

As could be seen it is quiet easy to develop dynamical websites with PHP, but you can never expect that your PHP script code is portable and runs on every PHP implementation. Finding bugs in the PHP implementation can be very tricky. This is the reason why always good to check another opportunities.



---

# Chapter 6. Conclusion

With basic knowledge about technologies like the Java programming language and Enterprise JavaBeans 3.0 it is possible to develop easy and fast, web services on Java Enterprise Edition 5 application servers. In order to build translation web services for the rule interchange, it is absolute necessary to understand the markup of rules in order to choose the right interchange language. Sophisticated knowledge about XML as superior language of dialects like SOAP and WSDL, is essential to build a working system. For the translation system, introduced in this document, is knowledge about XSLT inevitable as well. Understanding about different server and client side programming languages like PHP or JavaScript is useful for testing the whole system and helpful for the development of web interfaces or basic examples for other developer or researchers.

## 1. Extensions of the System

The implementation of the rule translator system, introduced in this document, use only XSLT transformations to translate rules. Since XSLT describes only transformations of XML data, this language is not useful for rule languages written in a different syntax than XML. One opportunity to support the translation of rule languages, that are not based on XML, is to use another kind of translation language. In order to support other kinds of translation alternatives, the definition of a translation interface, that describes translations independent from a real implementation, is necessary. Only this would open the system for other translation methods. In order to use the existing translation method with XSLT 2.0, the next logical step would be to use a separate XSLT 2.0 processor component. This could be a separate Enterprise JavaBeans 3.0 application deployed on the same application server. Every application on the application server could then access the XSLT processor Enterprise JavaBean application over their remote (business) interface.

Another translation alternative next to XSLT could be the Atlas Transformation Language<sup>1</sup>. Milan Milanovic already proved that is possible to translate with the Atlas Transformation Language (ATL) from SWRL to R2ML<sup>2</sup> or OCL to R2ML<sup>3</sup> and back

---

<sup>1</sup>“ATL (Atlas Transformation Language) has been defined to perform general transformations within the MDA framework (Model Driven Architecture) recently proposed by the OMG.” <http://www.sciences.univ-nantes.fr/lina/atl/> (23.08.2007)

<sup>2</sup> <http://oxygen.informatik.tu-cottbus.de/translator/SWRLtoR2ML/>

<sup>3</sup> <http://oxygen.informatik.tu-cottbus.de/translator/OCLtoR2ML/>

again. This can be tested with R2ML Translator web application. In order to use already existent ATL transformations of Milan Milanovic, they need to be migrated to Enterprise JavaBean 3.0 platform. The current implementation of Milan Milanovic use the underlying file system to find in Java archives the meta models for the translations. Due to the direct file access, the current implementation of the ATL translations can not be used inside of a Enterprise JavaBean container. But this is a only implementation specific problem that can be solved by redesigning the source codes.

Another and more simpler way to extent the current implementation of the rule translator Web Service is to allow external XSLT translations. Thus would make it possible for others researchers to use the web service to translate their XSLT transformations. But when such a behaviour is allowed, we need to test every XML input if it is a rule, in order to check that not ordinary XML data in contradiction is used to became transformed by web service.

Nice features are often statistics. The translator application could be extended to create statistics about the usage of the application and the most used rule translations. These statistics could be stored in a database and were accessible afterwards maybe by a web interface for selected user groups.

---

# Appendix A.

## Example A.1. R2ML Rule: Ancestor Relation

```
<?xml version="1.0" encoding="UTF-8"?>
<r2ml:RuleBase
  xmlns:r2ml="http://www.rewerse.net/I1/2006/R2ML"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.rewerse.net/I1/2006/R2ML
  http://oxygen.informatik.tu-cottbus.de/R2ML/0.4/R2ML.xsd">
  <r2ml:DerivationRuleSet>
    <r2ml:DerivationRule r2ml:ruleID="DR005">
      <r2ml:Documentation>
        <r2ml:RuleText r2ml:textFormat="plain">
          If X has a father Z and this Z has ancestors Y then X
          has ancestors Y.</r2ml:RuleText>
        <r2ml:SourceCode r2ml:language="FLogic">
          <![CDATA[
            FORALL ?X,?Y,?Z ?X[ancestor->?Y] <- ?X[father->?Z]
                                                    AND ?Z[ancestor->?Y].
          ]]></r2ml:SourceCode>
      </r2ml:Documentation>
      <r2ml:conditions>
        <r2ml:ReferencePropertyAtom
          r2ml:referencePropertyID="father">
          <r2ml:subject>
            <r2ml:ObjectVariable r2ml:name="X"/>
          </r2ml:subject>
          <r2ml:object>
            <r2ml:ObjectVariable r2ml:name="Z"/>
          </r2ml:object>
        </r2ml:ReferencePropertyAtom>
        <r2ml:ReferencePropertyAtom
          r2ml:referencePropertyID="ancestor">
```

---

```
<r2ml:subject>
  <r2ml:ObjectVariable r2ml:name="Z"/>
</r2ml:subject>
<r2ml:object>
  <r2ml:ObjectVariable r2ml:name="Y"/>
</r2ml:object>
</r2ml:ReferencePropertyAtom>
</r2ml:conditions>
<r2ml:conclusion>
  <r2ml:ReferencePropertyAtom
    r2ml:referencePropertyID="ancestor">
    <r2ml:subject>
      <r2ml:ObjectVariable r2ml:name="X"/>
    </r2ml:subject>
    <r2ml:object>
      <r2ml:ObjectVariable r2ml:name="Y"/>
    </r2ml:object>
  </r2ml:ReferencePropertyAtom>
</r2ml:conclusion>
</r2ml:DerivationRule>
</r2ml:DerivationRuleSet>
</r2ml:RuleBase>
```

---

# Appendix B.

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig und ohne unerlaubte Hilfe angefertigt habe, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Cottbus, den

Unterschrift:

---

# Bibliography

- [BRF 05] 9th International Business Rules Forum. Copyright © Business Rules Forum. 2005. *UServ Product Derby Case Study* [[http://www.businessrulesforum.com/2005\\_Product\\_Derby.pdf](http://www.businessrulesforum.com/2005_Product_Derby.pdf)] .
- [ONTOPRISE 04] Ontoprise. November 2004. *How to Write F-Logic Programs* [[http://www.ontoprise.de/documents/tutorial\\_flogic.pdf](http://www.ontoprise.de/documents/tutorial_flogic.pdf)] .
- [WEINDEL 06] Martin Weindel. Ontoprise. 28-10-2006. *F-Logic Forum: Results and Open Issues left* [<http://www.informatik.uni-freiburg.de/~dbis/workshop/slides/weindel%20-%20f-logic%20forum.ppt>] .
- [BHM+ 04] World Wide Web Consortium (W3C). 11-02-2004. *Web Services Architecture* [<http://www.w3.org/TR/ws-arch>] .
- [HENDERSON 06] Cal Henderson. May 2006. O'Reilly. *Building Scalable Web Sites - The Flickr Way*. 0-596-10235-6.
- [OMG] Object Management Group Inc. (OMG). *Model Driven Architecture (MDA) FAQ* [[http://www.omg.org/mda/faq\\_mda.htm](http://www.omg.org/mda/faq_mda.htm)] .
- [SIEGEL 02] Dr. Jon Siegel. 15-10-2002. Object Management Group Inc. (OMG). *Making the Case: OMG's Model Driven Architecture* [<http://www.sdtimes.com/article/special-20021015-01.html>] .
- [BALZERT 01] Helmut Balzert. Spektrum Akademischer Verlag. 16-02-2004. *Lehrbuch der Software-Technik, 2. Auflage*. 3-8274-0480-0.
- [BELL 04] Donald Bell. IBM developerWorks. 16-02-2004. *UML's Sequence Diagram* [<http://www.ibm.com/developerworks/rational/library/3101.html>] .
- [HOGG 03] John Hogg. IBM Software Group. 13-06-2003. *Brass Bubbles: An Overview of UML 2.0 (and MDA)* [<http://www.omg.org/news/meetings/workshops/UML%202003%20Manual/Tutorial7-Hogg.pdf>] .
- [JACOBSON 99] Dr. Ivar Hjalmar Jacobson. 1999. *Interview with Ivar Jacobson (by Adriano Comai)* [<http://www.analisi-disegno.com/uml/JacobsonInterview.html>] .
- [JSR181] Copyright © BEA Systems. 27-02-2005. *JSR 181: Web Services Metadata for the Java Platform, Version 2.0* [<http://jcp.org/aboutJava/communityprocess/final/jsr220/>] .

- [JSR220] Sun Microsystems. 08-05-2006. *JSR 220: Enterprise JavaBeans™, Version 3.0* [<http://jcp.org/aboutJava/communityprocess/final/jsr220/>] .
- [BM 06] Bill Burke & Richard Monson-Haefel. O'Reilly. *Enterprise JavaBeans 3.0, Fifth Edition*. 0-596-00978-X.
- [CRENSHAW 99] Chris Crenshaw. Nova Laboratories. 1999. *Developer's Guide to Understanding Enterprise JavaBeans* [<http://java.sun.com/products/ejb/developers-guide.pdf>] .
- [BGT 05] Harold Boley. Benjamin Grosf. Said Tabet. Copyright © The RuleML Initiative [<http://www.ruleml.org/>] . 13-05-2005. *RuleML tutorial* [<http://www.ruleml.org/papers/tutorial-ruleml.html>] .
- [ROTHAUG 04] Susanne Rothaug. 20-11-2004. SAP. *The Difference Between RPC and Document Style WSDL* [<https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/c018da90-0201-0010-ed85-d714ff7b7019>] .
- [JSR244] Bill Shannon . Sun Microsystems, Inc.. 11-05-2006. *Java™ Platform, Enterprise Edition 5 (Java EE 5) Specification* [<http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html>] .
- [BCE+] Jennifer Ball. Debbie Carson. Ian Evans. Scott Fordin. Kim Haase. Eric Jendrock. Sun Microsystems, Inc.. 16-06-2006. *The Java™ EE 5 Tutorial* [<http://java.sun.com/javaee/5/docs/tutorial/doc/>] .
- [YFC] Sun Microsystems, Inc.. 04-10-2006. *Your First Cup: An Introduction to the Java EE Platform* [<http://java.sun.com/javaee/5/docs/firstcup/doc/firstcup.pdf>] .
- [DEK+ 00] Don Box. David Ehnebuske. Gopal Kakivaya. Andrew Layman. Noah Mendelsohn. Henrik Nielson. Satish Thatte. Dave Winer. World Wide Web Consortium. Copyright © 2000 DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software. 08-05-2000. *Simple Object Access Protocol (SOAP) 1.1* [<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>] .