Deliverable D2.1

State of the art on modernization methodologies, methods and tools

Work Package 2

Leading partner: SINTEF

Author(s): SINTEF, SOFTEAM, FRAUNHOFER, TECNALIA, NETFECTIVE

Dissemination level: Public

Delivery Date: 10 March 2011

Final Version: 1.0
## Versioning and contribution history

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Document initialized</td>
<td>P Mohagheghi (SINTEF)</td>
</tr>
<tr>
<td>0.2</td>
<td>Added comments on structure, Reorganized</td>
<td>G Benguria (TECNALIA), P Mohagheghi (SINTEF)</td>
</tr>
<tr>
<td>0.3</td>
<td>Contributions added</td>
<td>F Barbier (Netfective), B Morin (SINTEF), D Roman (SINTEF), J Oldevik (SINTEF), C Carrez (SINTEF), C Hein (FRAUNHOFER)</td>
</tr>
<tr>
<td>0.4</td>
<td>Updates and additions after comments</td>
<td>F Barbier (Netfective), B Morin (SINTEF), C Hein (FRAUNHOFER)</td>
</tr>
<tr>
<td>0.5</td>
<td>Added contributions Sent to reviewers</td>
<td>A Sadovykh (SOFTTEAM), A Abherve (SOFTTEAM), G Benguria (TECNALIA)</td>
</tr>
<tr>
<td>0.6</td>
<td>Corrected according to comments of internal reviewers Final deliverable</td>
<td>F Barbier (Netfective), C Hein (FRAUNHOFER)</td>
</tr>
<tr>
<td>1.0</td>
<td>Final deliverable</td>
<td></td>
</tr>
</tbody>
</table>
Executive Summary

This document “D2.1 – State of the art on modernization methodologies, methods and tools” is a public deliverable of the Project “REuse and Migration of legacy applications to Interoperable Cloud Services (REMICS)” in “Small or medium-scale focused research project (STREP)” within the European seventh framework program for the ICT Call 5 (FP7-ICT-2009-5) challenge 1 Pervasive and Trusted Network and Service Infrastructures.

In this deliverable, we describe the state of the art on the following technologies areas that complete the vision of the REMICS project (see Figure 1). The covered areas are:

- Software development methodologies with focus on model-driven, service-oriented, migration and agile methodologies;
- Model-driven modernization of legacy systems;
- Cloud platforms and SOA; technologies as well as design patterns;
- Model execution and Models@runtime;
- Model-driven interoperability;
- Model-driven verification and testing

The analysis showed that while there are several methodologies for developing service-oriented systems, they are based on developing systems from scratch and not using a legacy system as basis for identifying and implementing services. On the other side, we have migration tools and methods that need integration with model-based development methods. For REMICS research on model-based verification and testing, related verification methods and techniques are identified. However, it is unclear which methods fit best for the SOA and cloud paradigms which will be addressed by our research.

The migration to interoperable cloud infrastructure introduces some challenges that are not addressed in the current methodologies in a complete way. We have analysed existing cloud computing platforms and cloud deployment models while there is no methodology that guides developers through the process of selecting technology and migration to cloud. Additionally, while state of the art in SOA is quite established and covered in literature, the cloud design patterns are more an ad-hoc knowledge that still has to be studied. Cloud infrastructures provide features for measuring and controlling services which shall be studied in relation to our research on models@runtime which is an exploratory research to do this at model level or evolve systems based on this knowledge.
Model-Driven Interoperability is a rather new domain which builds on top of long history on data and service interoperability. We foresee the need for semi-automated methods that assist users to handle interoperability issues between services, which are going to be addressed in REMICS.

In all the above technological areas we have identified and analysed standardization activities, products and relevant research projects that provide input to REMICS research. The state of the art analysis has therefore fulfilled its goals of identifying gaps for REMICS research. The results will be used in the technical work packages and also provide a baseline for evaluating REMICS innovations and the role of REMICS in advancing the state of the art.
Table of contents

EXECUTIVE SUMMARY ................................................................................................................................. 3
TABLE OF CONTENTS .................................................................................................................................... 5
1  INTRODUCTION ..................................................................................................................................... 7
2  ABBREVIATIONS .................................................................................................................................... 7
3  SOFTWARE DEVELOPMENT METHODOLOGIES ......................................................................... 9
   3.1  INTRODUCTION ...................................................................................................................................... 10
   3.2  PROJECTS .............................................................................................................................................. 11
   3.3  STANDARDS ........................................................................................................................................... 17
   3.4  PRODUCTS FOR METHODOLOGY DESCRIPTION ....................................................................................... 24
   3.5  SERVICE-ORIENTED METHODOLOGIES .......................................................................................... 27
   3.6  AGILE METHODOLOGIES ........................................................................................................................ 34
   3.7  CONCLUSIONS ....................................................................................................................................... 36
4  MODEL-DRIVEN MODERNIZATION OF LEGACY SYSTEMS ................................................... 37
   4.1  CONCEPTS AND LANDSCAPE, RELEVANCE TO REMICS .............................................................. 37
   4.2  EXISTING TECHNOLOGIES, PRODUCTS AND PROJECTS ............................................................................ 39
   4.3  STANDARDS ........................................................................................................................................... 40
   4.4  CONCLUSIONS ....................................................................................................................................... 41
   4.5  REFERENCES .......................................................................................................................................... 41
5  CLOUD COMPUTING TECHNOLOGIES ......................................................................................... 43
   5.1  OVERVIEW OF CLOUD COMPUTING ........................................................................................................ 43
   5.2  CLOUD COMPUTING ARCHITECTURE ...................................................................................................... 44
   5.3  CLOUD TECHNOLOGY STANDARDIZATION ........................................................................................ 48
   5.4  COMPARING CLOUD COMPUTING PLATFORMS ........................................................................................ 52
5.5  CONCLUSIONS ....................................................................................................................................... 62
6  MIGRATION: DESIGN PATTERNS IN SOA AND CLOUD ........................................................... 63
   6.1  SOA DESIGN PATTERNS ........................................................................................................................ 63
   6.2  CLOUD DESIGN PATTERNS ..................................................................................................................... 64
   6.3  CONCLUSIONS ....................................................................................................................................... 66
7  MODEL-DRIVEN INTEROPERABILITY .......................................................................................... 67
   7.1  GENERAL INTRODUCTION ..................................................................................................................... 67
   7.2  PROJECTS .................................................................................................................................................. 67
   7.3  STANDARDS ............................................................................................................................................ 69
   7.4  PRODUCTS AND APPROACHES .............................................................................................................. 72
   7.5  CONCLUSIONS ....................................................................................................................................... 73
8  MODEL EXECUTION AND MODELS@RUNTIME ........................................................................ 74
   8.1  CONCEPTS AND LANDSCAPE, RELEVANCE TO REMICS ............................................................... 74
| 8.2 | NATIONAL, EUROPEAN AND WORLDWIDE R&D INITIATIVES, CLUSTERS, WORK PROGRAMMES AND STRATEGIC ROADMAPS | 75 |
| 8.3 | STANDARDS | 78 |
| 8.4 | CONCLUSIONS | 78 |
| 8.5 | REFERENCES | 78 |
| 9 | MODEL-BASED VERIFICATION | 80 |
| 9.1 | GENERAL INTRODUCTION | 80 |
| 9.2 | METRICS | 81 |
| 9.3 | PROJECTS | 81 |
| 9.4 | STANDARDS | 82 |
| 9.5 | TOOLS AND TECHNOLOGIES | 83 |
| 9.6 | CONCLUSIONS | 84 |
| 9.7 | REFERENCES | 84 |
| 10 | MODEL-BASED TESTING | 85 |
| 10.1 | INTRODUCTION | 85 |
| 10.2 | PROJECTS | 86 |
| 10.3 | STANDARDS | 88 |
| 10.4 | TOOLS AND TECHNOLOGIES | 89 |
| 10.5 | CONCLUSIONS | 89 |
| 10.6 | REFERENCES | 90 |
| 11 | SUMMARY AND CONCLUSIONS | 91 |
1 Introduction

This report provides the current state-of-the-art analysis of methodologies, methods and tools for application modernization and Service Clouds paradigms. The goal is to provide a basis for research in work packages.

The remainder of this document is organized as follows. Sections 3-10 provide state on the art on various technologies and research challenges with focus on REMICS needs while Section 11 is summarizing and conclusions.

Each section has the following structure:

1. a subsection with general introduction focusing on:
   a. defining the main concepts and the landscape of the technological area, this of course will change among the different technology areas;
   b. The relevance to REMICS approach and needs;
   c. List of technologies that will be analyzed, with a short summary about the applicable solution area.

2. Subsections for related projects, standards, products, and other information that do not fit in the other areas.
   a. The section on related projects describe relevant achievements with references to website and documents;
   b. The section on standards describe on-going or planned work, the state of the standards and participants in the standardization process, and the needs of REMICS;
   c. The section on technologies and products provides a general introduction, accessibility (open source, commercial), languages and platforms it is based on, summary about the applicable solution area (this could be move to a conclusion), and possible examples. Following this will be the description of the technology.

3. A final conclusion on the technology area, summarizing the conclusions of the different subsections

We have provided links and references (either as footnotes or at the end of each section) to indicate the sources of the information. Figures are often taken from the references while comparison and analysis and done by the authors of this deliverable.

2 Abbreviations

ADM  Architecture-Driven Modernization
ASTM  Abstract Syntax Tree Metamodel
DSL  Domain-Specific Language
EMF  Eclipse Modeling Framework
EPF  Eclipse Process Framework
IaaS  Infrastructure as a Service
KDM  Knowledge Discovery Metamodel
MDI  Model-Driven Interoperability
MDM  Model-Driven Migration
MDRE  Model-Driven Reverse Engineering
M2M  Model to Model
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2T</td>
<td>Model to Text</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PDM</td>
<td>Platform Description Model</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SMM</td>
<td>Software Metrics Metamodel</td>
</tr>
<tr>
<td>SWOT</td>
<td>Strengths, Weaknesses, Opportunities, and Threats[^1]</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual machine</td>
</tr>
</tbody>
</table>

3 Software development methodologies

This section presents the state of the art on software development methodologies and supporting technologies. This analysis is focused in those methodologies and technologies that may be applicable in the migration of existing applications into interoperable cloud services.

Along the analysis we find diverse technologies that deal with multiple concepts in the software development domain, some of these concepts are: the methodology, the methodology framework, life cycle, the representation, the execution support, the service orientation, the model orientation, agile development, component-based software development, service management and service withdrawal.

- A Methodology provides guidance on the activities that must performed in order to develop a software product.
- A Methodology Framework is a mechanism to create methodologies that better fit to the needs of the projects.
- Life cycle represent the way in which the activities of the methodology are sequenced to produce the software product
- The representation makes reference to the way in which the methodologies are more or less formally represented.
- The service orientation is a software product architectural approach focused on the remote invocation of features rather than the integration of software pieces.
- The model orientation is a software development approach that makes an intensive use of models as a way to deal with the software complexity.
- Agile development is an alternative development that prioritizes the fast delivery of tangible results in contrast to other methodological approaches such as those based on the waterfall life cycle.
- Component-Based Software Development is also a software development approach that fosters the reuse of previous software components in contrast to the development from scratch.
- Service Management makes reference to the activities that are performed from the software product finalization until the software product withdrawal.
- Withdrawal includes the activities that are carried out to finalize the support of a software product.

The projects expected to be supported by the REMICS results are facing new challenges that will require innovative technologies and approaches:

- Most of the requirements come from the previous application, but not all;
- The product is provided to the final user as a service;
- The service will require new maintenance procedures;
- The service will require withdrawal procedures;
- The software must be open to changes in the supporting cloud.

Some of these challenges have been partially or almost totally addressed in other initiatives. These initiatives will be taken into account in the development of the integrated REMICS methodology. The initiatives have been classified in four main categories:

- Projects: several projects have taken place in the past that may contribute to the methodological approaches of the REMICS project.
- Standards
• Products
• Other Approaches

In this analysis we focus on previous initiatives. The monitoring of the ongoing and new initiatives in projects, standards, products, etc will be done as a part of continuous collaboration activities of the REMICS project.

3.1 Introduction

Along the short history of the computer science, several methodologies have been proposed to address the development of software projects and some are depicted in Figure 2. There are multiple reasons for this evolution on the way in which the software products are developed:

• Increasing complexity: the evolution from batch->backup->function->user interface->multiuser->security->monitoring->...
• New coding languages: binary->assembler->cobol->c->java->...
• New reuse approaches: subroutines->functions->objects->components->services->...
• New infrastructures: file systems->databases->net->internet->clouds->...
• New business needs: quality->traceability->flexibility->availability->reliability->monitoring->certification->...

There are many in which we could catalogue all these methodologies. Some are related to the design or implementation aspects:

• Structured: They separate the system in a subset of functional blocks (Separation of concerns). A functional block has a know entry and exit point. In order to implement a high level functional block it could be necessary to define lower level functional blocks (layers of abstraction).
• Object-oriented: They decompose the system in objects. Objects encapsulate both data and functionalities. Objects can include private functions and information (obfuscation). Several languages support the use of objects such as Java, C++, C#, and Visual basic.
• Component-Based: They decompose the system in components. Components are accessed through their interfaces. Components could contain several objects and functions to implement those interfaces. Several technologies have been developed to support
component development: Java RMI, COM/DCOM, CORBA, etc., and also methodologies such as the Catalysis approach\(^2\) and KobrA\(^3\).

- **Service-Oriented**: Methodologies that make use of existing services to build new systems or services. They help in the development of EAI (Enterprise application integration) and B2B (Business to business) systems. They rely on service provision technologies such as Web Service technologies, and service composition technologies as BPEL.

In combination to all these methodologies we should also consider the multiple lifecycle that could be applied to them. There most well know life cycle models are the following ones:

- Waterfall or classic life cycle model;
- V-shaped model;
- Prototyping model;
- Incremental model;
- Iterative model;
- Spiral model\(^4\);
- **Agile**\(^5\) model:

In the REMICS context we intend to apply an Agile and a service-oriented approach for the methodology development. This is motivated mainly for two reasons:

- Some of the companies implementing the scenarios already have some experience with agile methodologies
- The knowledge in the domain is not mature

With respect to the life cycle model, in the REMICS context it is imposed by the methodology approach that we intend to develop in the project. Agile approaches highly recommend applying iterative lifecycle on the development process.

### 3.2 Projects

#### 3.2.1 SHAPE

The SHAPE project\(^6\), “Semantically-enabled heterogeneous service architecture and platforms”, is a finalised collaborative project developed with the support of the EC (European Commission) in the context of the FP7 objective 1.2.

\(^1\) Objects, Components, and Frameworks With UML: The Catalysis Approach (Addison-Wesley Object Technology Series), by Desmond Francis D’Souza, Alan Cameron Wills, Addison-Wesley Professional; (October 19, 1998)


\(^3\) A Spiral Model of Development and Enhancement, by B. W. Boehm. Software Engineering Notes, Vol. 11, No. 4, August, 1986

\(^4\) It embraces methodologies that are focused on the customer, develop the software in an incremental way, are oriented to the change on the requirements, push the team work, provide executable results as soon as possible, automate the testing activities whenever possible, or accept refactoring as a necessary activity. These approaches are meant to be light-weight and with less up-front analysis than approaches such as the Rational Unified Process (RUP).

\(^5\) http://www.shape-project.eu/
The objective of the project was to support the development and realization of enterprise systems based on a Semantically-enabled Heterogeneous Service Architecture (SHA). SHA extends service-oriented architecture (SOA) with semantics and heterogeneous infrastructures (Web services, agents, Semantic Web Services, P2P and grid) under a unified service-oriented approach.

From the methodological point of view this project provides model-driven engineering (MDE) tool-supported methodology. The SHAPE methodology approach provides some interesting inputs to the REMICS project that will be taken into account during the definition of the REMICS methodology.

- Methodological Framework
- Reference Matrix
- Model based approach
- Transformation based approach

SHAPE approach for the definition of proper methodologies was to propose a methodological framework as a basis for the derivation of custom methodologies to cover specific project needs and constrains.

In REMICS we pretend to provide a methodology rather than a methodology framework. The reason for this is to be closer to the application of the methodology, even if some adaptation may be required based on the specific scenarios.

The SHAPE framework contains a bunch of method components covering different software engineering aspects at different abstraction levels. In order to classify these components the SHAPE project defined a Reference Matrix. Figure 3 presents the reference matrix of the SHAPE methodology framework.

<table>
<thead>
<tr>
<th>ASPECT LEVEL</th>
<th>Information</th>
<th>Service</th>
<th>Process</th>
<th>Rules</th>
<th>Events</th>
<th>Organization</th>
<th>Goals</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIM2PIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIM2PSM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3 - Initial structure of the SHAPE Reference Matrix

We can use the SHAPE reference matrix to evaluate the coverage of the different aspects by the REMICS methodology. It could be also interesting to place the different components of the REMICS methodology into the SHAPE reference matrix.

The SHAPE project methodology was based on a model based approach and also relies on model to model and model to text transformation to speed up the development process, to ensure the traceability and to reduce the number of defects. This will be covered in the analysis of the MDE methodologies in the next section.

3.2.2 MOMOCS

The MOMOCS project, “MOdel driven MOdernisation of Complex Systems”, is a finished collaborative project developed with the support of the EC (European Commission) in the context of the FP6.

The objective of the project was to study a methodology and related tools for fast reengineering of complex systems. The project is studying how to solve the dilemma between rigorous methodologies
and agile and unstructured ones, allowing the modernisation engineer to concentrate on what to do and not how to do it.

From the methodological point of view, this project focuses on the reengineering of complex systems. Thus MOMOCS project provides interesting aspects to the REMICS project regarding

- Migration Survey
- Methodology

The MOMOCS project performed two surveys during the execution of the project to identify the tool support needs for the migration projects. Figure 4 presents the results from the surveys. The numbers represent the number of votes from the participating companies.

![Figure 4 - Requested modernization steps to be better supported by tools](image)

These surveys provide information on the kind of task to be included in the REMICS methodology and their relevance. It provides an idea of the relevance of the tool support to be achieved in that phases by other technical work packages of the REMICS project.

The MOMOCS methodology is a migration approach that relies on the usage of models and models transformation and verification. Figure 5 describes the top level activities covered by the methodology. The MOMOCS methodology provides details on the following activities: existing system architecture recovery, architecture modernization, code generation, transformations and model evaluation.

![Figure 5 - XIRUP process model for modernization](image)
The MOMOCS methodology is another input for the REMICS methodology. Even though, there will be some differences.

- The REMICS methodology will be service and cloud based.
- There will be more support for the exploitation and maintenance as we are aiming cloud infrastructures.

3.2.3 WeBMoV

The WeBMoV project, “Web Service Modelling and Validation”, is a finished project developed with the support of French National Agency of Research.

The objective of the project was to contribute to the design, composition and validation of Web Services through a high level of abstraction view and a SOA-based logical architecture vision. The project was focussed on the construction of new services by the composition of existing ones. They were interested in the design and composition mechanisms for web services as well as their validation using different types of testing techniques.

From the methodological point of view, this project focuses on the modelling, validation and testing of web service composition. The WeBMoV project provides two interesting inputs to the project.

- Composition based methodology
- Tool chain

WebMov is focused on the composition of existing services while REMICS is more focused on the recovery of the knowledge of existing system in order to create those services in a cloud environment. But this does not mean that REMICS will not require service composition techniques since there may be cases in which service composition could be a viable technique to migrate certain parts of the legacy system.

WebMoV provides, as intended in REMICS as well, a set of tools to support the methodology. All WeBMoV tools are quite focused on the composition of services and they may be a good input for the REMICS project regarding composition.

3.2.4 COMBINE

The COMBINE project, “COMponent-Based INteroperable Enterprise System Development”, is a finished collaborative project developed with the support of the EC (European Commission) in the context of the 6FP.

The objective of the project was to support model-driven development of enterprise systems - using Components. The project was focussed on the development of methods, infrastructures and tools as well as business solutions for modelling, designing, deploying, testing and running components successfully in an enterprise-wide scale.

From the methodological point of view, this project focuses on component based solutions with no special coverage of the legacy modernization. The COMBINE project provides the following potential inputs to the REMICS project:

- Multiple views
- Component centre

The COMBINE project relies on multiple models for the development of the component based solution: business, requirement, architecture and design (see Figure 6). It may be interesting to evaluate the possibility of using some of these models in REMICS depending on the specific
scenario. The REMICS methodology plans to be agile whenever possible; therefore the introduction of multiple models should be done without affecting the agility of the methodology.

![Diagram of Business Model, Requirements Model, Design & Implementation Model, Architecture Model]

Another aspect from the COMBINE project that is interesting is the Component centre; “a Component Centre (see Figure 7) is a socio-technical system comprising people, IT systems and lightweight procedures that delivers quality, low-cost IT business solutions to a business, using modelling and a component-based approach”. Even if this not affects directly the REMICS migration methodology it can be included as a possible organisational approach.

![Diagram of Architecture and Product Development]

### 3.2.5 ModelWare

The ModelWare project, “Modeling solution for softWARE systems”, is a finished collaborative project\(^9\) developed with the support of the EC (European Commission) in the context of the 6FP.

---

\(^9\) SINTEF and ESI participated in Modelware and have information on the project artefacts although the website does not work.
The objective of the project was to support the large-scale deployment of Model-Driven Development. ModelWare was a large project with multiple results such as an open platform, maturity models and TCM, critical modelling technologies integrated inside the platform, and the launch of three core communities.

From the methodological point of view, this project focused on model driven methodologies. The ModelWare project provides two major inputs to the REMICS project:

- Process Framework
- MDD Maturity model

The process framework represent a repository of processes that may be used together to support the development of a specific project. The set of processes from the framework to be used in a specific project depends on the nature of the project and the process from ModelWare is depicted in Figure 8. It is not the purpose of the REMICS project to create a process framework, but that does not prevent to create the REMICS methodology in a way that can be integrated in an existing methodology.

Figure 8 - Detailed operational environment for constructing a system development process

The maturity model is a way to measure the degree of development and adoption of a process framework in a given organisation. The usage of this maturity model in the context of REMICS could be difficult as we are not fully centred in the model driven domain and neither in the creation of a process framework. Anyway it could be the basis for a possible way to measure the adoption of the methodology defined in the REMICS project.

Table 1 - ModelWare maturity levels
<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>Engineering Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2: Basic MDD</td>
<td>ENG 1 Decide upon modelling conventions</td>
</tr>
<tr>
<td></td>
<td>ENG 2 Develop a technical model</td>
</tr>
<tr>
<td></td>
<td>ENG 3 Generate code from the technical model</td>
</tr>
<tr>
<td></td>
<td>ENG 4 Generate documentation from the technical model</td>
</tr>
<tr>
<td>Level 3: Initial MDD</td>
<td>ENG 5 Develop platform independent model</td>
</tr>
<tr>
<td></td>
<td>ENG 6 Develop technical model-to-text transformations</td>
</tr>
<tr>
<td></td>
<td>ENG 7 Verify models</td>
</tr>
<tr>
<td>Level 4: Integrated MDD</td>
<td>ENG 8 Develop architecture-centric metamodel</td>
</tr>
<tr>
<td></td>
<td>ENG 9 Develop platform independent metamodel</td>
</tr>
<tr>
<td></td>
<td>ENG 10 Develop business model</td>
</tr>
<tr>
<td></td>
<td>ENG 11 Develop model-to-model transformations</td>
</tr>
<tr>
<td></td>
<td>ENG 12 Maintain traceability between models</td>
</tr>
<tr>
<td></td>
<td>ENG 13 Integrate product and product family infrastructure development</td>
</tr>
<tr>
<td></td>
<td>ENG 14 Perform advanced QA practices</td>
</tr>
<tr>
<td>Level 5: Ultimate MDD</td>
<td>ENG 15 Develop domain-specific languages</td>
</tr>
<tr>
<td></td>
<td>ENG 16 Verify and Validate products based on models</td>
</tr>
</tbody>
</table>

### 3.3 Standards

#### 3.3.1 ISO/IEC TR 24774:2007


- Title
- purpose statement
- outcomes
- activities
- tasks

The objective of this standard is to promote consistency in the definition of these kinds of standards.
In the development of the REMICS methodology we will need to define a suitable structure for the definition of the methodology. This standard could be an starting point, we should take into account that we are intending to define a methodology which is focused in how to do work while the process models supported by this standard are focused in what should be done rather than how should be done.

### 3.3.2 ISO 24744

This International Standard “Software Engineering Metamodel for Development Methodologies” establishes a formal framework for the definition and extension of development methodologies for information-based domains (IBD), such as software, business or systems. This international standard was approved in 2007\(^\text{10}\). This formal framework cover the following main constructs (see Figure 9):

- work units, the work that has to be done to obtain a system to be delivered
- work products artefacts to be used or created
- Producers, these cover the roles
- Stages, cover the behaviour of the methodologies
- Model units, models that developers can use to develop the workproducts

![Figure 9 - ISO 24744 Partial Inheritance Diagram](image-url)

In the development of REMICS we will require to represent the software engineering process in some way. In the state of the art there are several ways to represent processes (IDEF0\(^\text{11}\), BPMN\(^\text{12}\), OPF\(^\text{25}\), POP\(^\text{13}\), etc), and also some even more specific ways to represent software processes such as this standard. This standard and other similar standards will be evaluated for the representation of the methodology.

---


\(^{11}\) http://en.wikipedia.org/wiki/IDEF0


\(^{13}\) ATHENA-IP: POP* Revised framework. Version 1.0. ATHENA-IP FP6-507849 public deliverable WD A1.8, April 14, 2006
3.3.3 SPEM – Software process engineering metamodel

The OMG standard for software process modelling, SPEM, defines a metamodel containing the typical concepts in software process engineering (process, phase, role, model, etc) that can be used to construct models that describe software engineering process. This metamodel was initially approved in 2002\(^{14}\) and a new revision of the standard was issued in 2008\(^{15}\).

The specification is organised in several packages see Figure 10. It provides packages to model not only the static structure of the methodology such as the “methodContent”, but also the behavioural nature of the software engineering processes such as the “processBehavior”.

![Figure 10 - Structure of the SPEM 2.0 Meta-Model](image)

The specification is quite large in order to support all the possible modelling needs and styles for software methodologies. It covers basic methodology elements such as:

- Roles
- Workproducts
- Tools
- Tasks

But it also covers less common elements such as “qualification”.

\(^{14}\) “Software Process Engineering Metamodel Specification”, Version 1.0, formal/02-11-14, November 2002

The SPEM specification also provides the process engineers, project leads, project and program managers a notation to support them in the graphical representation of relevant parts of the process engineering model. The following table presents the graphical representation of some of the main concepts managed by the SPEM specification (see Table 2).

### Table 2 - SPEM 2.0 concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td><img src="image" alt="Activity" /></td>
<td>Activity is a work breakdown element and work definition. It describes a piece of work performed by one ProcessPerformer: the tasks, operations, and actions that are performed by a role or with which the role may assist. An Activity may consist of atomic elements called Steps.</td>
</tr>
<tr>
<td>Artifact</td>
<td><img src="image" alt="Artifact" /></td>
<td>It is a work product definition. It is used to define tangible work products. One artefact could be composed by other artefacts. This feature is very useful in order to identify parts and the whole.</td>
</tr>
<tr>
<td>CapabilityProcess</td>
<td><img src="image" alt="CapabilityProcess" /></td>
<td>CapabilityProcess represents a process component. This element is defined only once and it can be reused in several situations and processes.</td>
</tr>
<tr>
<td>ContentPackage</td>
<td><img src="image" alt="ContentPackage" /></td>
<td>This element is used to structure the method content and the library.</td>
</tr>
<tr>
<td>DeliveryProcess</td>
<td><img src="image" alt="DeliveryProcess" /></td>
<td>DeliveryProcess represents a final process that will be published.</td>
</tr>
<tr>
<td>Guidance</td>
<td><img src="image" alt="Guidance" /></td>
<td>Guidance elements may be associated with ModelElements, to provide more detailed information to practitioners about the associated ModelElement. Possible types of Guidance depend on the process family and can be for example: Guidelines, Techniques, Metrics, Examples, UML Profiles, Tool mentors, Checklist, Templates.</td>
</tr>
<tr>
<td>Concept</td>
<td>Notation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Phase</td>
<td><img src="image" alt="Phase" /></td>
<td>A Phase is a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal (often called a &quot;milestone&quot;) defines the phase exit criteria. Phases are defined with the additional constraint of sequentiality; that is, their enactments are executed with a series of milestone dates spread over time and often assume minimal (or no) overlap of their activities in time.</td>
</tr>
<tr>
<td>Process</td>
<td><img src="image" alt="Process" /></td>
<td>A Process is a special Activity that describes a structure for particular types of developments.</td>
</tr>
<tr>
<td>Team Profile</td>
<td><img src="image" alt="Team Profile" /></td>
<td>TeamProfile represent a set of roles. Usually it is used to gather roles with particular mission.</td>
</tr>
<tr>
<td>Role</td>
<td><img src="image" alt="Role" /></td>
<td>A Role is a ProcessPerformer and it defines a performer for a set of WorkDefinitions in a process. ProcessPerformer has a sub-Class, ProcessRole.</td>
</tr>
<tr>
<td>RoleUse</td>
<td><img src="image" alt="Role Use" /></td>
<td>RoleUse is related to a specific task descriptor. Defines responsibilities over specific WorkProducts, and defines the roles that perform and assist in specific activities. (called RoleDescriptor in EPF)</td>
</tr>
<tr>
<td>Task</td>
<td><img src="image" alt="Task" /></td>
<td>Usually one of the primary tasks during the construction of a methodology it is to define the main tasks within method content.</td>
</tr>
<tr>
<td>TaskDescriptor</td>
<td><img src="image" alt="TaskDescriptor" /></td>
<td>This element is the relationship with a task defined in method content.</td>
</tr>
<tr>
<td>Template</td>
<td><img src="image" alt="Template" /></td>
<td>SPEM2.0 and EPF help users to define their templates and thus they are used univocally.</td>
</tr>
</tbody>
</table>

In the development of REMICS, as we wrote before, we will require to represent the software engineering process in some way. This standard and other similar standards will be evaluated for the representation of the methodology.

### 3.3.4 CMMI for services

CMMI® (Capability Maturity Model® Integration)\(^\text{16}\) is known as a common way to measure the capability of an organisation to develop their software or system within the expected time, with the assigned budget and with the required quality. The CMMI® contains a list of good practices that are expected to be found in organisations a high maturity level.

The CMMI® measures organisations in five levels under its staged model:

1. Initial: no processes in place
2. Managed: some planning and management is in place
3. Defined: processes are executed in the same way in the organisation
4. Quantitatively Managed: processes are measured
5. Optimizing: there are practices to improve the processes

This CMMI® for services is a specialization of the CMMI® that focuses on the service industry. It has the following process areas (Process area title (acronym) – area - level):

- Capacity and Availability Management (CAM) - Project Management - 3
- Causal Analysis and Resolution (CAR) - Support - 5
- Configuration Management (CM) - Support - 2
- Decision Analysis and Resolution (DAR) - Support - 3
- Integrated Project Management (IPM) - Project Management - 3
- Incident Resolution and Prevention (IRP) - Service Establishment and Delivery - 3
- Measurement and Analysis (MA) Support 2
- Organizational Innovation and Deployment (OID) - Process Management - 5
- Organizational Process Definition (OPD) - Process Management - 3
- Organizational Process Focus (OPF) - Process Management - 3
- Organizational Process Performance (OPP) - Process Management - 4
- Organizational Training (OT) - Process Management - 3
- Project Monitoring and Control (PMC) - Project Management - 2
- Project Planning (PP) - Project Management - 2
- Process and Product Quality Assurance (PPQA) - Support - 2
- Quantitative Project Management (QPM) - Project Management - 4
- Requirements Management (REQM) - Project Management - 2
- Risk Management (RSKM) - Project Management - 3
- Supplier Agreement Management (SAM) - Project Management - 2
- Service Continuity (SCON) - Project Management - 3
- Service Delivery (SD) - Service Establishment and Delivery - 2
- Service System Development (SSD) - Service Establishment and Delivery - 3
- Service System Transition (SST) - Service Establishment and Delivery - 3
- Strategic Service Management (STSM) - Service Establishment and Delivery - 3

The REMICS methodology is expected to cover not only the migration of the legacy applications but also to the establishment of the basis to provide the resulting service. These maturity models are a good basis to identify the areas that should be latterly supported.

### 3.3.5 ITIL

ITIL is a set of concepts and practices for managing Information Technology (IT), services (ITSM), IT development and IT operations.

ITIL gives detailed descriptions of a number of important IT practices and provides comprehensive checklists, tasks and procedures that any IT organization can tailor to its needs. ITIL is published in a series of books, each of which covers an IT management topic.
### Service Strategy
- Service Portfolio Management
- Demand Management
- IT Financial Management

### Service Transition
- Service Asset and Configuration Management
- Service Validation and Testing
- Evaluation
- Release and Deployment Management
- Change Management
- Knowledge Management

### Service Design
- Service Catalogue Management
- Service Level Management
- Risk Management
- Capacity Management
- Availability Management
- IT Service Continuity Management
- Information Security Management
- Compliance Management
- IT Architecture Management
- Supplier Management

### Service Operation
- Event Management
- Incident Management
- Problem Management
- Request Fulfillment
- Access Management

---

**Figure 12 - ITIL Processes**

These maturity models are a good basis to identify the areas that should be latterly supported.

### 3.3.6 DSL4SE

The Domain-Specific Language and a kernel of essentials for Software Engineering, DSL4SE\(^\text{17}\), is an ongoing standard in the OMG for the creation of a language and a kernel that allows people to describe the essentials of their current and future practices and methods in a light way; so that they can be composed, simulated, applied, compared, enacted, evaluated and measured by practitioners as well as taught and researched by academic and research communities. The initiative is supported by the committers of the SEMAT\(^\text{18}\) initiative.

Some of the main requirements from the Request For Proposal (RFP)\(^\text{19}\), or objectives of this future standard are:

- **Description.** The Language shall support the description of Practices and Methods in terms of the essential elements of the Kernel.
- **Composition.** The Language shall support the composition of Practices to describe existing and new Methods.
- **Work Progress.** The Language shall allow the representation of work progress. For example, describing a Practice that involves iterative development requires describing the starting and ending states of each iteration.
- **Enactment.** The Language shall support the enactment of Methods, both as used to help plan endeavors and as applied (or executed) as part of the day to day activities in real projects.

---


\(^\text{18}\) http://www.semat.org/bin/view

• Extensibility. The Language shall provide the ability to add practices, levels of detail and lifecycle models. It shall support tailoring to specific domains of application and to projects involving more than a software.

• Assessment/Measurability. The Language shall support quantitative assessment of all the relevant artifacts and provide mechanisms to assess quantitatively all the relevant artifacts of software processes and products.

• Comparison. The Language shall support the comparison of Methods and Practices to see which are suitable for a given situation.

• Measurement. The Language shall support the measurement of Practices and Methods, both to enable performance evaluation and to guide evaluation and validation in research.

The SEMAT initiative is currently supported by 1438 people around the world, among them some known people in the field of the software engineering such as Scott Ambler, Barry Boehm, Erich Gamma, Watts Humphrey, Ken Schwaber20 etc. The initiative is also supported by corporations such as IBM, Microsoft, Ericsson, ABB and Samsung, while criticized by others, see for example responses21.

In REMICS we are involved in the RFP process and the following work as an alternative way to represent a methodology as it promises to solve some of the problems of existing software engineering process modelling approaches.

3.4 Products for methodology description

3.4.1 EPF

The EPF, Eclipse Process Framework22, is an open source software process engineering tool developed over eclipse framework. The tool can be downloaded for free and it will run in any platform capable to run the eclipse framework. The licence of the EPF allows to use or to extend the tool for commercial purposes.

The EPF implements SPEM standard, and is able to create intranets with the description of the processes for the support of the developers in charge of the realisation of the software or system projects.

Two main roles are expected to be supported by the EPF, the process author and the process user. The tool supports the process author in the definition of two kinds of elements:

• Method content
• Process content

Method content represents the methodology elements that are later reused to define the process content. The elements that can be defined here are: roles, tasks, workproducts, guidance, categories and relationship between them. Guidance allows providing support for roles, task and workproducts. Categories allows to group roles, tasks and workproducts. There are five initial predefined categories: disciplines, domains, work product kinds, role sets and tools.

Process content creates end to end processes by reusing the previously defined method content. There are two kinds of process content:

• Capability patterns

20 Schwaber, Ken The Scrum Development Process (OOPSLA’95 Workshop on Business Object Design and Implementation
22 http://www.eclipse.org/epf/
• Delivery processes

Capability patterns are reusable processes, and instance of a capability pattern is called activity. Delivery processes are the process instances that will be directly followed by the process users. A process author will use the EPF to (see Figure 13):

• Create/change method content
• Create new processes
• Configure existing process
• Publish a process in an intranet

A process user will look at the process in the intranet published by the process author. The user would then look up the appropriate role (or roles) and start working according to the given instructions.

Figure 13 - EPF screenshot from the viewpoint of a process author

3.4.2 IPA

The IPA, IRIS Process Author\textsuperscript{23}, is a commercial process authoring tool by the Canadian company Osellus. IPA is a web based tool and in principle is usable from any web browser. IPA is also SPEM

\textsuperscript{23} \url{http://www.osellus.com/}
compliant. IPA is focused on the process author while the process user will use the published version of the process. IPA allows exporting the process in different formats, HTML, Microsoft word, pdf, wiki and Microsoft Project.

IPA is based on the definition of two kinds of contents:

- Method content
- Process content

Method content includes roles, work products, tasks, guidance, and other items and is organized into packages. Once the method content is created, the process author creates the process packages that import the method content from their packages and reuse them in the processes.

There is also the possibility to allow process users to provide feedback in the processes. To implement this, IPA provides a wiki version that allows providing that feedback.

### 3.4.3 essWork

essWork\(^{24}\) is a recent software process management tool. It is related with the SEMAT initiative and it tries to realise the SEMAT vision. EssWork provides features for developing, browsing, composing, and executing practices.

The process authoring is only supported in windows and it is based on a set of Microsoft office add-ons. Therefore it is also necessary to have Microsoft office. The process browsing and composing is web based and can be run in windows and also in Linux. The interface seems quite simple and intuitive (see Figure 15).

![Figure 14 - View of essWork](http://www.ivarjacobson.com/process_improvement_technology/esswork/)

It also supports the users of the processes through a web site.

---

The execution support for the defined practices is limited to the temporal planning of the practices and some export facilities to common project management tools.

In REMICS in principle we do not plan to make a direct use of this tool as it is not based on a real standard, and also because it is a commercial tool with no open source alternative. However, we follow the implementation of DSL4SE which may appear here.

3.4.4 Open Process Framework

The OPEN Process Framework (OPF) is a public-domain object-oriented framework of free, open source, reusable method components (i.e., classes of process components)\(^2\). OPF is maintained and extended by the OPF repository organization (OPFRO).

The OPF recognizes that one size does NOT fit all. No matter how tailorable it might be, no single development method is appropriate for all situations given their great differences in terms of application size, complexity, and criticality as well as the great differences between their development and operations organizations in terms of size, experience, maturity, resources, etc.

Therefore, this on-line repository (class library) of over 1,100 state-of-the-art reusable method components is intended to be used to engineer endeavor-specific methods for developing and operating software-intensive systems. OPF’s reusable method components are designed to be easily tailorable by the user, and the entire repository of method components is continually being iterated and extended with new classes by its existing user base.

OPF is not a tool but a set of practices that REMICS will consider when developing the methodology if applicable to REMICS activities in work packages.

3.5 Service-oriented methodologies

In the SOA, Service Oriented Architecture, domain in the last years several approaches have appeared to help the organisations in the development of SOA based solutions. Some of the most relevant ones are:

- SMART
- SAE developed by CBDI, which provides a process framework
- SOAD by IBM, which provides a methodology

\(^2\) [http://www.opfro.org/](http://www.opfro.org/); OPF metamodel retrieved June 1, 2010
SODA by IBM

SOMA

ISE

3.5.1 SMART

The SMART, Service-Oriented Migration and Reuse Technique\textsuperscript{26}, was developed by Software Engineering Institute at Carnegie Mellon University to assist enterprises to analyze their legacy systems to evaluate their feasibility to integrate a Service-Oriented Architecture (SOA).

SMART provides a service migration strategy as its primary product. The methodology allows gathering a vast range of information about existing legacy systems, the SOA to deploy, and potential services to produce. The implementation of SMART involves five major composite activities:

- Establishing Stakeholder Context: this is a set of tasks in which the stakeholders involved in the migration process are consulted. As a result, several pieces of information are obtained about legacy elements: the distribution of responsibilities and knowledge, and about existing concerns regarding the migration process.
- Describe Existing Capability: this activity intends to obtain descriptive data about the components of the legacy system.
- Describe the SOA State: in this group of tasks, the target SOA state is described. At the same time, SMART gathers evidence about the potential for turning existing capabilities into services, these decisions are made considering what the intended SOA state is.
- Analyze the Gap: an activity that aims to balance the breach existing between legacy capabilities and the target SOA state.
- Develop Migration Strategy: the final set of tasks in which one or several goals are determined out of the gap analysis made; and a strategy to achieve them is developed.

The first activity seeks to identify the people that have more knowledge about existing legacy systems. It seeks also to identify the objective of each system and what it should do as a service. The task of the second activity is to find a clear description of legacy systems. Examples of information that may be collected include the name, function, programming language, operating systems, and age of legacy systems. The main objective of the third activity, describe the SOA state, identifies the set of potential services that can be created from legacy applications. The fourth activity identifies the gap between the current state of the IT infrastructure and the desired SOA. It determines the effort and cost associated with the conversion of legacy systems into services. The last activity provides a strategy and a set of recommendations to achieve the initial organizational goal. Figure 16 shows a relatively simple technique to establish an initial analysis for the fourth activity.


Copyright © REMICS Consortium 2010-2013
Figure 16 - Analyses of converting the selected components to services

SEI arguments that the approach allows to adapt legacy systems to services without affecting the involved systems while exposing functionality to a large number of client applications using standard-based interfaces. The SMART approach was applied with success in the U.S. Department of Defence (DoD). The methodology has helped the DoD in converting their existing systems into services. It has been established that specialized methodologies and approaches are indispensable for large organizations since the number of people and the heterogeneity of software involved may introduce a level of complexity that leaves the organizations in a worse position than before introducing an SOA.

In the context of REMICS, the SMART techniques can be used in the first stages of the methodology to help the companies in the analysis of existing legacy components and the selection of the most appropriate strategy for their modernisation.

3.5.2 SAE

SAE, Service Architecture Engineering, extends the reference model by OASIS for SOA, developed by CBDI. SAE framework provides a more detailed and pragmatic conceptual view than the base OASIS model for SOA27.

This framework provides the methodology elements that are intended to be tailored to a certain organisation’s need in order to build the processes to drive the SOA adoption cycle. Consequently, it includes a pool of artefacts (methodology elements) for supporting the migration to a service oriented enterprise. These artefacts, tools and techniques are grouped in three main perspectives: organization, reference architectures and processes.

---

**Figure 17 - CBDI SAE SOA Reference Framework**

**Process**: This perspective covers four key disciplines areas for SOA processes:

- **Manage**: the processes required for defining the organization’s SOA capabilities, the current ones and the ones desired for the future. As well as, the processes motivated by the transition and the required SOA governance.
- **Consume**: Assembling software solutions specially focused on the consumption of services.
- **Provide**: Provisioning and implementation of services.
- **Enable**: establishing the platform required to run services and ensuring its performance during the execution of services.

**Organization**: This group of elements is oriented to describe the organization: its roles and responsibilities, project profiles and funding models. The recommendations in order to successfully support the service lifecycle are exposed.

**Architecture**: The architecture involves two orthogonal mechanisms:

- **SOA views**: they comprise a consistent level of abstraction for deliverable artefacts related to distinct set of stakeholders. These are: business, specification, implementation, deployment and technology.
- **SOA practices**: different SOA practices in the organisation are defined and clustered in each of the SOA views.

Under that structure the following processes are located:

- **SOA Adoption and Excellence**: It compares current SOA capabilities against the desired outcomes according to certain maturity levels. The result is a suitable adoption plan.
- **SOA Governance**: ensuring that the SOA adoption process complies with the SOA Reference Framework.
- **SO Business Requirements**: This process consists of identifying business requirements specific to service orientation that otherwise would not be addressed.
• SO Business Improvement: transforming existing processes, products and capabilities into services; and creating new service-enabled ones that in turn improve the business.
• Solution Assembly: using several techniques based on services to design, code, and test the solution.
• Service Oriented Architecture and Design: this process adapts the SOA reference framework to a particular organization; evolving the service portfolio plan and the SO Security Architecture.
• Legacy Transition Planning: creates a plan heading towards an SOA vision but taking into consideration the existing IT architecture.
• Service Implementation: enabling services by means of the automation units, these will be either subcontracted or implemented in the organization. Equally, legacy systems have to be considered during this process.
• Service Provisioning: service brokering, quality assurance and certification.
• Service Platform Design and Installation: the development of the underlying service platform, potentially consisting of an ESB or a set of ESBs.
• Service Operations and Management: deploying, running, monitoring and controlling run time services.

In the context of REMICS the SAE framework as many of the previously introduced frameworks will be used to check the completeness of the methodology.

3.5.3 SOAD

SOAD, Service-Oriented Analysis and Design\textsuperscript{28} developed by IBM, investigated how Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA) frameworks, and Business Process Modeling (BPM) could be used to support successful SOA deployments. The authors termed their approach as Service-Oriented Analysis and Design (SOAD). It is argued that the methodologies employed fall short when applied independently of each other and that there is the need for a hybrid approach that combines concepts of all of the disciplines with a number of new aggregating elements.

![Figure 18 - The SOAD approach](image-url)

The methodology has been developed having in mind that OOAD, EA, and BPM cannot give a satisfactory solution when used in isolation from each other. The OOAD cannot be used in isolation since it does not capture the full scope of an enterprise-wide SOA system. EA approaches, such as

Zachman framework\(^\text{29}\) do not capture how “enterprise-wide abstractions of quality facilitating reuse and longevity can be found”. The Zachman framework has some limitations since it shows interconnection at a macro-level which does not fit very well since services need a low-level of detail to represent technical devices. Also, the framework is rather generic and does not reach down to the design and implementation domain which may impair architects and developers due to a lack of tactical information. As a result, some researchers believe that it does bridge the gap between enterprise and implementations. Nonetheless, they provide a basis for architectural consistency between individual solutions across multiple lines of business and organizational units. Finally, BPM does not adequately represent the architecture and implementation domain, and in many cases, process modelling and implementation are carried out separately. Nonetheless, existing BPM approaches can be a good starting point for SOAD, but they have to be extended with new elements to describe services and operations at a correct level. SOAD recommends a meet-in-the-middle approach, rather than pure, top-down or bottom-up process.

At a global view, SOAD provides the “glue” to put together OOAD, BPM, and EA. Its main goals are the following:

- Interconnect and interrelate classes/objects from OOAD to events of processes.
- Replace “use case” descriptions with business events and processes.
- Business events and processes are first class citizens.
- Consider syntax and semantics for ad hoc composition, semantic brokering, and runtime discovery.
- Definition of quality factors and best practices.
- Roles management (CEO, Business analysts, Developer, Architect, etc).
- Characterize functionalities that are good candidates to become services (level of reuse and importance).
- Facilitate end-to-end modelling and supply a comprehensive tool support.
- Develop a method spawning from the business to the architecture and application design domains.

SOAD puts emphasis on quality factors to deploy services. The following elements are considered:

- Reduce dependencies between services.
- Make dependencies between services explicitly.
- Design services based on a CRUDS (Create, Read, Update, Delete, and Search) metaphor.
- Keep service naming understandable.
- Service granularity.

Other important issues include:

- Service categorization (core service, composite services, operational services, security services, etc).
- Policies (business traceability - Sarbanes-Oxley (SOX) act).
- Meet-in-the-middle processes.
- Semantic brokering.
- Service harvesting and knowledge brokering.

The SOAD methodology defines the following processes:

• Service Identification: This process consists of a combination of top-down, bottom-up, and middle-out techniques of domain decomposition, existing asset analysis, and goal-service modelling.

• Service classification or categorization: This activity consists of a service classification into a service hierarchy, reflecting the composite or fractal nature of services: services can and should be composed of finer-grained components and services.

•Subsystem analysis: This activity takes the subsystems found during domain decomposition and specifies the interdependencies and flow between the subsystems.

• Component specification: The details of the component that implement the services are specified: data, rules, services, configurable profile and variations.

• Service allocation: It consists of assigning services to the subsystems that have been identified.

• Service realization: In this step a decision is made as to which legacy system module will be used to realize a given service and which services will be built from the “ground-up”. Other realization decisions taken include: security, management and monitoring of services.

In the context of REMICS, the SOAD methodology is difficult to reuse as it is focussed in the development of new services. Therefore it may be considered as an input for the development of those parts of the migration that are developed from scratch.

3.5.4 SODA

SODA methodology, Service Oriented Development of Applications, developed by Gartner\textsuperscript{30} was conceived as a result of a set of activities or processes for SOA development. Those processes that have been considered to be critic for SOA development are tackled by SODA. SODA is supposed to enable an organization for better, cheaper, faster software development. SODA is based on artefact reuse (from patterns and models to test plans and test beds). It will only be beneficial in those projects in which reuse are one of the main principles.

The SODA methodology includes the following processes:

• Design: Solution requirements are established at this stage. SODA requires that the design is mainly focused on process oriented design elements, rather than on component based systems. Process flow and process integration should be initially integrated in the application design instead of being considered at a later stage.

• Modelling: the structure of the solution is built by means of modelled services (with UML or other modelling mechanisms) Modelling affects three areas: business modelling, application modelling and technical modelling. Modelling is one of the key elements that cater to the required agility and reuse.

• Fabrication: that is creating the actual, functional components for the services in the application design. This includes writing and/or generating the code. The components created represent the functionalities that have been designed.

• Integration: in this activity, components are integrated into services. Assemblers or visual editors may help at this task.

• Governance: information flows and work flows among services are defined. Information flows require being defined; this can be achieved by means of work flow management products. At the same time, the mechanisms that are necessary to transform service data flows and to manage the data structure are determined.

• Automatization: this activity refers to automatically generating code, reducing the necessity for actual code writing. Several tools are to be used for this purpose. Less experienced developers are helped by this facility and experienced developers may rapidly manipulate code when required.

• Maintenance (Rapid Application Maintenance): this facility means undertaking several small changes rather than big changes: favouring service variability. SOA demands that services are able to adapt to new evolving objectives. This adaptation becomes a key element to make the solution achieved agile.

In the context of REMICS the SODA methodology is again focused in the development of a new system. However it contains interesting processes such as the governance and the automatization that may fit in the REMICS methodology.

3.6 Agile methodologies

Agile is a generic term that groups those methodologies that rather than having a fixed plan from the beginning they define the project schedule as the project advance. Some examples of agile methodologies are Extreme Programming (XP)\(^{31}\) and SCRUM\(^ {32}\), Feature Driven Development (FDD)\(^ {33}\), and Dynamic Systems Development Method (DSDM)\(^ {34}\).

Agile methods are characterised by:

- Adaptive Planning
- Open to requirements change at any point in the development
- Continuous iteration with the customer
- Working software after each iteration, working software is the progress measure unit
- Collocation
- Continuous attention to technical excellence and good design, refactoring when needed
- Test automation
- Development environment optimisation
- Small teams

We describe some best-known agile approaches in the following sections.

3.6.1 SCRUM

SCRUM is an iterative methodology based that organise the development of the software product in a set of sprints. The methodology makes an intensive use of meetings with different purposes: for the definition of the objectives of each sprint, for the tracking of the sprint, for the sprint review and for the sprint analysis.

At the start of the SCRUM project the team is defined:

- Product Owner: Represents the voice of the customer and is accountable for ensuring that the Team delivers value to the business. He is the one that prioritizes the backlog items.

---

\(^{31}\) http://xprogramming.com/index.php

\(^{32}\) http://www.scrumalliance.org/

\(^{33}\) http://www.featuredrivendevelopment.com/

\(^{34}\) www.freetutes.com/systemanalysis/sa2-dynamic-system-development-method.html
• ScrumMaster: he is accountable for removing impediments to the ability of the team to deliver the sprint goal. Besides, he ensures that the Scrum process is used as intended, protects the team and keeps them focused on the tasks in hand.

• Team: build with people with cross-functional skills who do the actual work (analyse, design, develop, test, technical communication, document, etc.). The team is self organised.

Then the product backlog is initiated, the product backlog is the list of things to be implemented during the project. It has not to be complete; it should be enough to start some sprints. A backlog item describes a feature with a value index and an effort index.

Once the roles are defined and a product backlog exists, the SCRUM sprints can start. The sprint starts with the Sprint Planning Meeting where the backlog items to be implemented are selected; the main output is the Sprint backlog. After that meeting each day, a Daily Scrum meeting is carried out where recent progress, next actions and problems are shortly discussed. At the end of the Sprint Review Meeting is performed where the progress is shown in demonstration based approach. Optionally at the end of the sprint a Sprint Retrospective meeting can be held to improve the next ones.

In REMICS one of the user companies is familiarised with this methodology, therefore we will evaluate it in order to adopt possible approaches.

3.6.2 XP

XP\textsuperscript{35} is also an iterative methodology based on a sequence of small releases. The methodology relies on continuous integration and feedback. Each release starts with a planning game and it is completed through the execution of four main activities:

• Listening: programmers must understand the customer requirements

• Designing: to organise the different parts of the system, evaluate if refactoring is needed

• Coding: this is the most important product for XP, is the way to demonstrate the solutions

• Testing: each piece of code (function) must be accompanied with the corresponding test.

XP has been described as having 12 practices, grouped into four areas:

• Fine scale feedback:
  o Pair programming
  o Planning game
  o Iteration planning
  o Test-driven development
  o Whole team

• Continuous process
  o Continuous integration
  o Design improvement or refactoring
  o Small releases

\textsuperscript{35} Kent Beck, "Extreme programming Explained: Embrace change" (The XP Series), Addison Wesley, 1999
• Shared understanding
  o Coding standards
  o Collective code ownership
  o Simple design
  o System metaphor
• Programmer welfare
  • Sustainable pace

In REMICS we are dealing with a project scenario where the knowledge on the technology is not very detailed and the implementation requirements could not be very stable. In this case agile methodologies could be applicable. Therefore we will take this methodology as well as SCRUM as potential sources of practices for the methodology.

3.7 Conclusions

From the analysis of the state of the art and taking into account the characteristics of the REMICS projects, we may extract some conclusions with respect to some of the concepts on the software development domain.

• A Methodology: For the service nature of the cloud we will take into account the service oriented methodologies, such as SOAD, SAE or SODA. Also migration methodologies should be considered for the migration part. All of the methodologies should be model-driven or be adapted to a model-driven engineering approach.

• A Methodology Framework: A methodological framework is focused in the development of methodology, while we are more interested in providing direct support for the implementation of REMICS projects. Therefore in REMICS our approach with respect to methodology frameworks such as the ones in SHAPE, ModelWare, or SAE will be to use them as a source of potential method fragments and as a way to ensure that the coverage of the REMICS methodology is the right one.

• Life cycle: We will apply an iterative approach such as the one implemented in XP or SCRUM.

• The representation: we will use EPF for the formal description of the process, which is SPEM compliant. Besides we will keep track of the evolution of the SEMAT initiative and are involved in the standardization effort.

• Component Based Software Development: For the identification of the better strategies for the migration of the different components in the first stages of the REMICS project we will introduce some practices form the SMART method.

• Cloud computing: this aspect requires focus on deployment and service management and we should focus on these activities for SOA methods and introduce perhaps other activities as well.

The migration to interoperable cloud infrastructure introduces some challenges that are not addressed in the current methodologies in a complete way. In this analysis we have identified the potential sources of software engineering practices from the agile and service oriented methodologies that will later be used to create the REMICS methodology.

Besides, this analysis has been also used to establish the representation approach for the methodology in a way that fosters their latter usage on future migration projects.
4 Model-driven modernization of legacy systems

On the Model-Driven Modernization Web site\(^{36}\), one can read: “Imagine a set of automated tools that can disassemble a legacy software system, (separating services, business rules, and data), transform the components in high-level models, reconfigure these models using the best-practices from Model Driven Architecture (MDA), and finally regenerate a modern system. Is this even possible? Or do the laws of software entropy preclude such a goal from ever being achieved?”

The last sentence implies that model-driven modernization is at its earliest stage and earliest utilization/assessment for conducting modernization. In this respect, this section characterizes the current state of the art on Model Driven Modernization (MDM), how and why REMICS tackles such a challenge, and how and why REMICS will bring out innovation in this R&D area. Please note that references are listed at the end of this section.

4.1 Concepts and landscape, relevance to REMICS

Lehman’s law\(^{37}\) about software evolution states that any actively used software application must continually change to satisfy stakeholders. The complexity induced by changes generates uncontrolled higher and higher costs, often leading to reengineering.

“Reverse engineering is the process of analyzing a subject system to: identify the system’s components and their interrelationships; and create representations of the system at a higher level of abstraction” [Chikofsky and Cross, 1990].

Modernization refers to a general-purpose, more global, self-contained process to move old-fashion software to renewed applications/information systems running on the top of the most up-to-date technologies. Modernization encompasses reverse engineering activities but also migration as highlighted and promoted in REMICS. The coupling between recovering and migrating stresses the need of an appropriate seamless methodology to process the legacy material with agility, namely, to reason about it, to make it intelligible and interpretable, to be able to separate concerns (e.g., business assets versus technology-oriented features) and to have the possibility to easily deferring and customizing design, development & deployment choices when envisaging and generating the new system (choosing between computing paradigms like component, service, cloud, target technology, platform, library, product, standard compliance).

In REMICS, models are the first-class artefacts to express all outputs of the modernization process and of its discrete steps. PIMs (Platform-Independent Models) especially facilitate the expression of business facets of a system while PDMs (Platform Description Models) may serve as supports for describing a given technology and its implementation in an execution platform. Modernization covers:

- The extraction of the legacy material (code, data, configuration parameters and these parameters’ values…) including more logical assets like business services/functionalities and business rules;
- The analysis, interpretation and comprehension of the rough extractions to make very explicit business rules, architectures, services/functionalities, links to operating systems and user interfaces;
- The representation of the legacy material with the possible distinction of business requirements, technology-oriented features like technological attributes and how QoS constraints and requirements are currently satisfied and fulfilled;

---

\(^{36}\) www.modeldrivenmodernization.com

\(^{37}\) en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution
• The reformatting, reformulation even augmentation of the legacy material to prepare migration;
• The easier and straightforward generation of new systems with a variety of choices and adaptability to target (any) possible new technology;
• The formal tracing of the global process allowing explanations, returns, even commits and rollbacks.

Although the idea of model-driven reverse engineering (MDRE) is not new [Rugaber and Stirewalt, 2004], the more ambitious idea of model-driven modernization has not been deeply explored until now. The key technological barrier is to cope with dinosaur information systems/applications from the ’70s and ’80s with massive code bases, the absence of structuring, the lack of a general synthetic view of the system, often replaced by a shared knowledge between stakeholders even the absence of knowledge at all (due to retired people for instance) on certain portions of the legacy system. As a result, in [Van Geet and Demeyer, 2010] for example, software visualization and dynamic feature location are the prerequisites for monitoring the legacy system when functioning to capture “the lacking knowledge” before any (reverse engineering) intervention on it is made possible.

There are few contributions at this time to MDM except a short list of projects/products (see next section) along with initiatives, out of which, the “famous” ADM standardization initiative at the OMG (see next section). In the same time, in [Favre, 2010], one may observe that, unfortunately, fundamental research on reverse engineering rather emphasizes not-so-old technologies like object-orientation, XML or Java, missing the point of industry in which, the bigger systems are in COBOL or in C (in the telecom area for example). Fundamental research also omits economical issues: the scalability of MDM methodologies is indeed critical taking account of the huge size of the legacy material to deal with. MDM costs are accordingly uncontrollable without automation especially.

In this spirit, within the MOMOCs Web site38, a list of modelling products is proposed along with their reverse engineering capabilities. A great number of them only stress modern language parsing (Java, C++, XML…) to produce model maps with poor downstream semantic interpretation. There are no full model-driven methodologies that both concentrate on dinosaur information systems/applications and seamless integration of both the reverse & migration phases in order to move applications/information systems to a broad variety of open topical platforms, from .NET, Java EE to cloud platforms.

If one looks at other academic research works about MDM, contributions are often very specific with weak links to, and slight reuse potentiality for REMICS. For example, in [Harman and Mansouri, 2010], search-based software engineering (SBSE) encompasses metaheuristics, optimization techniques (e.g., genetic algorithms) to analyze and differentiate solutions in software engineering activities. Applied to testing for instance, one may imagine the discovering of the most stressing execution paths of an existing application. Note that in REMICS, Task 3.4 aims at capturing material to regenerate and thus replay test cases with adequate (probably reformatted but semantically equivalent) data. Other upstream research like that exposed in [Lin & al., 2010] may be an inspiration source for REMICS as well. In this technique, one is able to precisely reverse input syntactic structures from execution when data formats are unpublished even unknown. These formats are key elements to create test data and replay tests for guarantying that the new system possesses, at least, equivalent functionalities compared to the old system. Progresses in SBSE address architecture and prediction issues. Results are however at their very early premises in the sense that, again, few are concerned with large-scale legacy systems.

38 www.momocs.org; from the Reverse Engineering button in the Tools folder (general menu on the right hand side)
4.2 Existing technologies, products and projects

Based on the above observations, there are few technologies that address model-driven modernization issues along with seamless recovering/migration, actual scalability and treatment of non homogenous dinosaur applications (i.e., any COBOL dialect but also any 4th generation language like PL/SQL, Nat System’s Command Language in Nat Star, etc.).

General-purpose technologies like Abstract Syntax Trees (AST) have been used for a long time to model code but MDM calls for the merging of such proven techniques with metamodeling, giving birth to new supports like the non-mature ASTM from the OMG ADM (see below). Close supports (technologies, products, projects) to REMICS currently are:

- **MOMOCS**: The MOMOCS EU project (end in August 2008) aimed at studying a methodology and related tools for fast reengineering of complex systems. A complex system is characterized by an interconnection of hardware, software, user interfaces, firmware, business and production processes. MOMOCS studied how a complex system can be modernized with a focus on the software portion of it, with the goal of keeping up with a very fast changing business and technical environment taking human beings as the centre of the interaction. MOMOCS results and tools will be injected in REMICS thanks to SOFTEAM that participated in MOMOCS and produced most of the tools.

- **MoDisco** (from the MODELPLEX project which ended in 2010): MoDisco provides an extensible framework to develop model-driven tools to support use-cases of existing software modernization. MoDisco has a great focus on Java/XML legacy systems with “demonstrative” case studies based on these two technologies. MoDisco has the benefit to be highly customizable with adaptive components, a model discoverer and an “intelligent” model browser especially, to establish a tailored model processing chain. Support for KDM seems an intention of this Eclipse project but no tangible results are available on this.

- **REMICS**: REMICS is a toolkit for reverse engineering tools. The goal of REMICS is enabling developers (or researchers) to quickly write their own tools for reverse engineering, on their own demands of investigating software products. REMICS is carefully designed and implemented as a collection of small scripts with independency (enough for unit testing) and interoperability with the other libraries.

- **MOOSE**: Moos is an extensive platform for software and data analysis. It offers multiple services ranging from importing and parsing data, to modelling, to measuring, querying, mining, and to building interactive and visual analysis tools. Like MoDisco, for historical reasons, MOOSE rather focuses on “modern” technologies (e.g., Smalltalk) with a weak interest on old-style systems (green screens, flat files….). Like MoDisco too, these tools have the advantage to be open, highly customizable and, potentially, adaptable/testable for dinosaur applications/information systems. However, nowadays, no proof of concept exists.

- **SOAMOG**: Standing for Migration von Legacy-Software in serviceorientierte Architekturen is a MDM German project that focuses on Java and COBOL but it does not use KDM and ASTM. This project also emphasizes typed graph theory as the underlying theory for analyzing and querying legacy material represented by models.

---

38 www.eclipse.org/MoDisco
40 Be careful here, this is “another” REMICS! www.remics.org
41 www.moosetechnology.org
42 www.soamig.de
• JANUS Studio® from the software revolution inc.\(^{43}\) provides a certain degree of automation for software assessment, documentation, transformation, re-factoring and web-enablement. This product stresses code-to-code transformation even if refactoring is supported. UML documentation is generated for both the “As Is” as well as the “To Be” systems but neither KDM nor UML are the formalism of this tool that prefers its own MDM DSL.

4.3 Standards

Architecture-Driven Modernization initiative or ADM is the chapter of the OMG in charge of promoting and constructing concepts, techniques, methods and tools for MDM. Out of the great challenges of MDM (such as agility of IT systems and renewed business value), tough challenges are:

• Controlling reverse engineering activities, effort and costs (incrementally) to anticipate the fact that the construction of the new system might be impossible in space and time;

• Software assurance, security by construction, end-to-end QoS (Quality of Service) transportation in general from old to new systems; New systems must be "better" under all facets that old systems;

• Interoperability including reverse engineering methodology independence: each component (or particular group of components) is reverse engineered by different stakeholders and exchanged with the rest of interested people to assemble KDM-based system model. This issue is also part of WP5 in REMICS.

Conforming to ADM standards goes beyond the simple use of these standards. A complex MDM scenario is, for instance, as follows: After assembled KDM-based system model, vendors may perform particular modernization/assurance scenario on top of assembled model and produce outcome that have to feed into one report that would be presented to imaginary client. For example: system A is client-server based that consists of COBOL back end with mix of Java, C++ and C# as a front end. Server is a Microsoft platform while client is deployed on Microsoft, Linux and Apple. Examples of KDM-based MDM concerns are: set of quality, complexity and security metrics for such a system; security vulnerability findings; architecture findings and conversion into UML diagrams of particular components, cross-validation through verifying the experts' knowledge; incremental and iterative analysis (reverse engineering techniques must support raid incremental and iterative analysis to quickly verify or reject hypotheses), seamless migration (see above) from the outputs of the recovering phase.

Knowledge Discovery Metamodel (KDM): KDM is a specification from OMG. KDM is a common intermediate representation for existing software systems and their operating environments that defines common metadata required for deep semantic integration of Application Lifecycle Management tools. KDM is designed as the OMG's foundation for software modernization, IT portfolio management and software assurance. KDM is a metamodel for knowledge discovery in software. It defines a common vocabulary of knowledge related to software engineering artefacts, regardless of the implementation programming language and runtime platform - a checklist of items that a software mining tool should discover and a software analysis tool can use. KDM is designed to enable knowledge-based integration between tools. More specifically, KDM uses Meta-Object Facility (MOF) to define an interchange format between tools that work with existing software as well as an abstract interface in Eclipse for the next-generation modernization tools.

The most well-known implementation of KDM is that of the KDM analytics company\(^{44}\). KDM is made up of a set sub-packages and complementary packages. For instance, KDM includes Micro-KDM for low-level modelling (code) while other KDM sub-(or related) packages emphasize the modelling of

\(^{43}\) www.tsri.com

\(^{44}\) www.kdmanalytics.com/kdm
architectures, business rules (Semantic of Business Vocabulary and Business Rules or SBVR). Some packages are under development even design. This is the case of Abstract Syntax Tree Metamodel (ASTM) or Software Metrics Metamodel (SMM). The former is in beta version without significant utilizations and thus feedbacks. REMICS aims intensively enhancing and thus influencing these standards through competitive implementations offered by some REMICS partners.

A specific focus on service/cloud platform migration in REMICS invites us to study the SOA paradigm more intensively. The early identification of services (called functionalities in the legacy system) is indeed crucial. Moreover, there is no one-to-one mapping between discovered services and the services that aims at being designed and generated in the new technology (web services or others).

### 4.4 Conclusions

The power of models, metamodelling and model transformation creates an opportunistic convergence between MDD and software reengineering/migration: "Migration projects can benefit from the vision of MDD by abstracting legacy systems (reverse engineering), transform them and concretize the migrated system (forward engineering). However, both fields of research are not yet entirely understood. Neither is the combination of both fields examined very well. The MDSM workshop will bring together latest research in the field of model-driven software migration approaches." This observation comes from the First International Workshop on Model-Driven Software Migration Web site\(^45\) demonstrating the neat absence of both scientific results and industrial solutions.

REMICS tackles the problem with a high degree of automation, the handling of large-scale information systems based on old but widespread technologies/languages, COBOL especially. The key issue is the production of a well-established methodology (genericity especially is expected to avoid any inappropriate adherence to the two REMICS case studies) in terms of discrete well-delimited steps. Each step output is a model conforming to a metamodel and each step is a model transformation with a more or less degree of intervention from software reengineers. All metamodels including intermediate metamodels must conform to KDM/ASTM to increase the methodology's credibility and potential adoption.

### 4.5 References


\(^45\) userpages.uni-koblenz.de/ist/mdsm/2011/


5 Cloud computing technologies

5.1 Overview of cloud computing

5.1.1 Cloud computing paradigm

Cloud computing is an architectural model enabling on demand access to a common pool of configurable IT resources, such as networks, servers, storage, platforms, applications or services. These resources can be accessed dynamically, and their reservation does not imply any intervention of the part of the service provider.

The term "Cloud Computing" comes from the fact that the user cannot specify the physical position and organization of the equipment used to host the resources he may need to use.

From the client's point of view, it is no longer necessary to own the hardware (and sometimes software) infrastructure on which an information system is based. Instead, the client bases his system on services made available to him by a third-party provider. He consumes IT resources as a service, and pays only for those resources that he truly uses at a given moment in time.

From the service provider’s point of view, the fact that infrastructures are shared between several clients means that service providers are able to improve server usage rates, by limiting the phases during which they are inactive, thereby realizing significant scale economies.

The notion of Cloud Computing groups together several heterogeneous forms, both in terms of the services provided (Service, Platform, Infrastructure,...) and the types of implementation (Private, Public, Hybrid,...). Despite this, some fundamental points remain central to all facets of Cloud Computing, such as resource virtualization, scalability, interoperability and the notion of service quality. This document will describe these different forms of Cloud Computing.

5.1.2 Cloud computing usage

Cloud computing can be seen as a change in paradigm with regard to leading client-server approaches that have dominated since the beginning of the 1980s. The main differences between the cloud and earlier approaches can be summed up as follows:

- **Limited initial investment**: Cloud computing uses a pricing model based on the payment of resources that are actually used. Clients do not need to invest heavily before they can begin to take advantage of an infrastructure based on Cloud Computing. They can simply hire cloud resources to meet their own individual needs and pay for the resources they really use.

- ** Scalability**: Cloud computing provides a solution whereby individual user needs and the constraints linked to infrastructures are separated. This means that it can easily adapt to rapid increases or decreases in the demand for resources.

- **Invoicing resources according to needs**: At any given time, the only resources that are reserved are those that are actually being used. The client, therefore, only pays for the resources that he actually uses.

- **Quality of service (QoS)**: QoS is the ability to efficiently transport a given type of traffic, in terms of availability, transfer, transmission time, and loss rate. Cloud computing provides the possibility of dynamically re-evaluating the "Service Level Agreement (SLA)" associated with users or user groups, meaning that it can rapidly react to changes in conditions, so as to guarantee a constant quality of service.

- **Self-service resources**: Cloud computing enables users to deploy their own IT resource sets (machines, network, storage devices, ...) where necessary, without the delays and
complications usually associated with the deployment of this type of infrastructure using standard means.

- **Reliability and breakdown resistance**: Cloud computing concentrates on improving crucial infrastructure elements, in order to attain predetermined levels of reliability. These breakdown tolerance policies can be modified easily and with no impact on users.
- **Optimization**: Cloud computing maximizes the use and increases the efficiency of existing infrastructure resources. Infrastructure lifecycles are prolonged, and the need for investment consequently reduced.

### 5.1.3 Cloud computing and virtualization

The implementation of an infrastructure based on cloud computing is mainly based on virtualization techniques.

Virtualization is characterized by the possibility of running "virtual machines" using a hypervisor. A virtual machine (VM) is the software implementation of a machine that runs programmes in the same way as a physical machine. Every virtual machine includes its own core, operating system, with supporting libraries and applications.

A hypervisor provides a uniform abstraction of the underlying physical machine. Several virtual machines can run simultaneously on a single hypervisor. The separation of the virtual machine and the underlying physical hardware means that a single virtual machine can be run on different physical machines.

In this way, virtualization is perceived as being a catalyst to cloud computing. It provides the cloud service provider with the necessary flexibility regarding the allocation of the resources requested by the user on the available physical resources. According to the level of service used, this virtualization aspect can be visible or invisible to the end-user.

### 5.1 Cloud computing architecture

#### 5.1.1 A layered model of cloud computing

In general, the architecture of a cloud computing environment can be broken down into five distinct layers: the **Hardware** layer, the **Infrastructure** layer, the **Platform** layer, the **Application** layer and the **Service** layer. The following paragraphs provide a detailed description of each of these layers:

---

**Figure 19 - Layers in the cloud computing paradigm**
The **hardware layer** is in charge of managing the cloud’s physical resources, including physical servers, routers, switches, power supply and cooling systems. Typical problems managed by the hardware layer include hardware configuration, breakdown tolerance, traffic management, electricity consumption management and data storage system management.

The **infrastructure layer** or virtualization layer creates a storage pool for IT resources, based on physical resources and using virtualization technologies such as Xen, KVM and VMware. The infrastructure layer manages the essential services provided by cloud computing, such as the dynamic allocation of resources, as these services are not managed by virtualization technologies.

The **platform layer** is built above the infrastructure layer. This layer is made up of operating systems and application frameworks. Its objective is to minimize unwarranted workload by deploying and managing low level applications, such as frameworks and operating systems.

The **application layer** is made up of real applications implemented in the cloud. Unlike traditional applications, these applications use services provided by the platform layer, such as data storage or transactional management systems.

The **service layer** hosts the services provided by an application. This optional layer appears in the context of the SaaS (Software as a Service) approach, described in the following chapter.

### 5.1.2 Cloud computing service level

Among cloud computing architectures, three approaches can be distinguished, corresponding to three levels of services rendered: the **Infrastructure as a Service (IaaS)** approach, the **Platform as a Service (PaaS)** approach and the **Software as a Service (SaaS)** approach. Each of these approaches covers a different aspect of the layered architecture model described in the previous chapter.

![Figure 20 - Cloud computing service level](image)

**Infrastructure as a Service**

The Infrastructure as a Service approach covers the **Hardware** and **Infrastructure** layers of the cloud computing layered model as shown below.

The service provider makes a resource virtualization platform available as a service. Application developers can access calculation, data storage and network management infrastructures on demand.

This model provides great flexibility to the detriment of the level of services rendered, which remains limited to the management of hardware aspects. The only constraints encountered concern the
necessity that it be possible to deploy the application in a virtual machine managed by the cloud service provider.

![Figure 21 - Layers covered in IaaS](image)

Among the "Infrastructure as a Service" infrastructure providers in the marketplace, we find Amazon EC2, GoGrid or Eucalyptus and OpenNebula.

**Platform as a Service**

The Platform as a Service approach covers the Hardware, Infrastructure and Platform layers of the cloud computing layered model.

In this approach, the service provider makes available a development environment that includes the operating system, as well as a set of services dedicated to the development, testing, and deployment and hosting of sophisticated web applications.

Clients use the services provided by the framework to build an application without thinking about the underlying hardware and software aspects.

Issues related to breakdown tolerance, application scalability or resource reservation are managed by this type of platform and are therefore transparent for the user.

![Figure 22 - Layers covered in PaaS](image)

Among the "Platform as a Service" infrastructure providers in the marketplace, we find Google App Engine or Microsoft Azure.

**Software as a Service**

The Software as a Service approach covers the Hardware, Infrastructure, Platform, Application and Service layers of the cloud computing layered model.
Using this approach, the service provider makes available software in the form of an internet service. Users access this application, generally paying each time they use it. The Software as a Service approach can therefore be seen as making available to companies the means, services and expertise to externalize an aspect of their information system, with the cost being assimilated as an operational cost rather than an investment.

Among the "Software as a Service" infrastructure providers in the marketplace, we find Google Apps, SalesForce or Microsoft Online Services.

5.1.3 Cloud delivery models

There are several ways of apprehending cloud computing. The issues encountered are not the same if you are part of an organization wishing to deliver a service or wishing to reduce operational costs. As a consequence, there are different types of cloud, each meeting specific needs:

Public cloud computing: We talk of a public cloud in the case of a service provider who provides services to public users over the Internet. The public cloud computing service provider allows his clients to reserve access to resources that he delivers via a Web service interface.

Public cloud computing services generally provide access to vast reservoirs of developing resources. At any given time, the client reserves and pays for the use of the tiny part of these resources that he has actually used.

Private cloud computing: We talk of a private cloud in the case of an organization that installs its own farm of servers and deploys a cloud computing infrastructure for its exclusive use.

The implementation of a private cloud service provides local users with a flexible and agile infrastructure destined to run virtualized services that require a significant workload.

Hybrid cloud computing: We talk of hybrid clouds in the case of an organization that uses a private cloud infrastructure, while retaining the possibility of entrusting part of its workload to a public cloud service provider.

A hybrid cloud combines, therefore, IT resources (machines, network, storage,…) provided by one or several public clouds and one or several private clouds according to user demand.

This approach requires that the APIs managing both private and public clouds be compatible, in order to enable an application to be deployed on both supports.
5.2 Cloud technology standardization

5.2.1 Standardization activities

Cloud computing is currently experiencing a phase of rapid expansion, giving businesses the opportunity to access resources and services through new channels. However, cloud computing is also a victim of its own success, with new ranges of services quickly appearing. These services are not necessarily interoperable between themselves, and this poses the risk of creating subsequent blockages and incompatibilities that could harm the overall market.

Faced with this situation, several initiatives have emerged over the past year, with the objective of putting new cloud computing standards in place. Each of these initiatives aims to create a standard relating to one or two aspects of cloud computing, without trying to provide an overall view of what standardized cloud computing could be.

These normalization initiatives have appeared in a disordered manner, and no reference consortium that would play for the cloud a role similar to W3C for the Web has yet emerged.

Table 3 - Cloud computing standardization activities

<table>
<thead>
<tr>
<th>Standards Organisations</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Standards Coordination (OMG)</td>
<td>Cloud fora, consortia, DMTF, CSA, OASIS, OGF</td>
</tr>
<tr>
<td>Open Grid Forum</td>
<td>Microsoft, Sun, Intel, HP, AT&amp;T, eBay</td>
</tr>
<tr>
<td>Distributed Management Task Force</td>
<td>IBM, Microsoft, Novell, Oracle, Sun, Vmware, EMC</td>
</tr>
<tr>
<td>Open Cloud Consortium</td>
<td>Cisco, MIT Lincoln Labs, Yahoo, various colleges,…</td>
</tr>
<tr>
<td>Cloud Security Alliance</td>
<td>eBay, ING, Qualys, PGP, zScaler</td>
</tr>
<tr>
<td>Storage Networking Industry Association</td>
<td>Dell, EMC, Oracle, Juniper Networks, Qlogic, HP, Vmware, Hitachi, NetApp</td>
</tr>
</tbody>
</table>

In this context, an organization, the Cloud Standards Coordination, set up and managed by the OMG (Object Management Group) attempts to put in place dialog between the different standardization initiatives. For the moment, its influence remains relatively limited.

The situation is made more complicated by the appearance of de facto standards resulting from the centre-stage position of the unavoidable leaders in the world of cloud computing. These entities, which dominate their market sectors, impose their API on other players in the marketplace. This is notably the case of Amazon EC2 in the domain of IaaS platforms.

The diagram below aims to replace these different standardization initiatives within an overall view of cloud computing.
5.2.2 Standardization efforts

**OMG - Cloud Standards Coordination**
Participants: Cloud fora, consortia, DMTF, CSA, OASIS, OGF,…

This group, set up and managed by the OMG (Object Management Group), is involved in setting up a coordination process between the different standardization initiatives currently present in the world of cloud computing.

Different normalization organizations have come together within the coordination, to share their expertise on subjects as diverse as storage, execution models, deployment models, service definition, security, and authentication and confidentiality issues.

Standardization work specific to the OMG has begun. This work focuses on modelling the deployment of applications in clouds, and on interoperability between cloud computing services. The work is in part based on the work carried out by, among others, the Design Task Force (ADTF) and SOA Special Interest Group (SOA SIG) work groups.

**Open Grid Forum (OGF)**
Participants: Microsoft, Sun, Intel, HP, AT&T, eBay,…
Website: [http://www.ogf.org/](http://www.ogf.org/)

The OGF group is a community of developers, service providers and users belonging to the Grid Computing sector.

Formed in 2006 following the merger of Global Grid Forum and Enterprise Grid Alliance, the OGF has two main objectives: the development of standards related to the Grid Computing sector, and the creation of a community destined to promote Grid Computing.

The following list outlines the main standards proposed by the Open Grid Forum:
• GLUE (Grid Laboratory Uniform Environment): GLUE is a technology-agnostic information model for a uniform representation of Grid resources.

• SAGA: The Simple API for Grid Applications describes an interface for high-level Grid application programming.

• Open Grid Services Architecture (OGSA): The OGSA describes an architecture for a service-oriented grid computing environment for business and scientific use.

• DRMAA: Distributed Resource Management Application API is a high-level API specification for the submission and control of jobs to one or more Distributed Resource Management Systems (DRMS) within a Grid architecture.

• Job Submission Description Language: An extensible XML specification for the description of simple tasks to non-interactive computer execution systems. The specification focuses on the description of computational task submissions to traditional high-performance computer systems like batch schedulers.

• CDDLM: Configuration Description, Deployment, and Lifecycle Management Specification is a standard for the management, deployment and configuration of Grid Service lifecycles or inter-organization resources.

*Study Group on Cloud Computing (ISO/IEC JTC 1 SC38)*

Participants: 17 countries


This group belongs to the ISO organization and is working on several standards in the field of cloud computing.

*Distributed Management Task Force (DMTF)*

Participants: IBM, Microsoft, Novell, Oracle, Sun, Vmware, EMC,…

Website: [http://www.dmtf.org/](http://www.dmtf.org/)

The DMTF group, also called "Open Cloud Standards Incubator", is interested in the standardization of interactions between cloud computing platforms. It concentrates on defining management protocol, communication formats and security mechanisms, in the aim of facilitating interoperability between these platforms.

The following list outlines the main standards proposed by the Distributed Management Task Force:

• Common Information Model (CIM): CIM is a common data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment.

• Web-Based Enterprise Management (WBEM): WBEM is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments.

• Alert Standard Format (ASF): This specification defines remote control and alerting interfaces that best serve clients’ OS-absent environments.

• System Management BIOS (SMBIOS): The SMBIOS Specification addresses how motherboard and system vendors present management information about their products in a standard format by extending the BIOS interface on x86 architecture systems.

• Desktop Management Interface (DMI): DMI generates a standard framework for managing and tracking components in a desktop PC, notebook or server. Due to the rapid
advancement of DMTF technologies, such as CIM, the DMTF defined an “End of Life” process for DMI, which ended March 31, 2005.

Open Cloud Consortium (OCC)
Participants: Cisco, MIT Lincoln Labs, Yahoo, various colleges including University of Illinois,…
Website: http://opencloudconsortium.org/
The OCC group is an organization destined to develop and promote standards for cloud computing. It is particularly interested in the issue of interoperability between cloud computing platforms, and in the definition of standardized benchmarks.
Among other things, OCC produces a benchmark tool named MalStone and designed to analyse the performances of data storage platforms on the cloud.

Cloud Security Alliance
Participants: eBay, ING, Qualys, PGP, zScaler,…
Website: http://www.cloudsecurityalliance.org/
The Cloud Security Alliance is an organization formed in the aim of promoting best security practices for cloud computing. This group also participates in educational projects concerning security in the world of cloud computing, and has published several guides which deal with this subject, among them “Security Guidance for Critical Areas of Focus in Cloud Computing V2.1”, http://www.cloudsecurityalliance.org/csaguide.pdf

Storage Networking Industry Association
Participants: Dell, EMC, Oracle, Juniper Networks, Qlogic, HP, Vmware, Hitachi, NetApp
Website: http://www.snia.org/
The Storage Networking Industry Association (SNIA) is a not-for-profit organization that brings together 400 companies active in the field of distributed data storage.
Its activities focus on promoting distributed storage solutions and producing standards dedicated to the issues of data storage and management, interoperability between different storage solutions and normalization of access protocols to these tools.
The following list outlines the main standards promoted by the Storage Networking Industry Association:

- Common RAID Disk Data Format (DDF): The Common RAID Disk Data Format specification defines a standard data structure describing how data is formatted across the disks in a RAID group. The Disk Data Format (DDF) structure allows a basic level of interoperability between different suppliers of RAID technology. The Common RAID DDF structure benefits storage users by enabling data-in-place migration among systems from different vendors.

- Cloud Data Management Interface (CDMI): The Cloud Data Management Interface defines the functional interface that applications will use to create, retrieve, update and delete data elements from the Cloud. As part of this interface the client will be able to discover the capabilities of the cloud storage solutions available and use this interface to manage containers and the data that is placed in them. In addition, metadata can be set on containers and their contained data elements through this interface.

5.3 Comparing cloud computing platforms

5.3.1 Taxonomy of cloud computing in REMICS

In light of the fundamental differences that exist between the different types of cloud computing platform (Software as a Service, Platform as a Service and Infrastructure as a Service), comparing two platforms that do not belong to the same category would make no sense. In this study, we will look at platforms after first grouping them into categories.

Cloud computing platforms are compared from three different angles: the study of the services provided, the study of the architecture of the platforms and the study of development methods on these platforms.

Criteria related to services provided by platforms:
- **Service level**: Level of service delivered by the platform in question. Possible values are as follows: SaaS, PaaS, and IaaS.

- **Delivery models**: Types of service delivered by the platform in question. Some platforms can work according to several models. Possible values are as follows: public cloud, private cloud, hybrid cloud.

- **License**: Type of usage license associated with the platform. Possible values are as follows: Open Source or Commercial.

- **Business model**: Invoicing model used by the platform.

- **Interoperability**: Interoperability and convergence of the platform with other cloud computing platforms on the market.

Criteria related to the architecture of platforms:
- **Computing architecture**: Design and architecture of the cloud computing platform.

- **Virtualization management**: Technology used to separate software aspects from the services provided by the hardware platform on which these services are run. This issue includes several aspects such as server virtualization, storage space virtualization or network virtualization.

- **Load balancing**: Load-balancing mechanisms and systems used to monitor the services provided by the platform.

- **Fault tolerance**: Error restart mechanisms in the case of software or hardware problems implemented by the platform.

- **Storage**: Data storage system used by the platform.

- **Security policy**: Policy used to secure the data stored in the cloud.

Criteria related to development constraints on these platforms:
- **Supported API**: Type of API implemented by the platform and enabling access to the services the API provides.
• Programming Framework: Types of framework supported by the platform.

5.3.2 Platform as a Service providers

Google App Engine

Made available by Google in April 2008, the Google App Engine platform is destined for the design, development and deployment of web applications hosted on Google servers. This computing platform belongs to the PaaS category.

Originally built around Python development tools, the platform has significantly evolved over the past two years. It now supports Java applications and enables the integration of a certain number of common Java frameworks.

An application based on Google App Engine runs in a secure environment that provides limited access to the underlying operating system. This environment isolates the application in its own secure environment, which is totally separate from the hardware infrastructure (server, operating systems, geographical location) necessary for the running of the application. This application communicates with the outside world through web queries (http and https).

The architecture of an application based on the SDK Java Google App Engine depends on the developers’ choice of framework. For example, it is possible to use JSPs (Java Server Pages) or the GWT API when implementing the presentation layer. There are a number of limitations concerning the use of Java frameworks.

Data storage is handled through a database based on the BigTable database system. Data access is based on the implementation of Java Data Objects (JDO) Java Persistence API (JPA) interfaces.

The platform provides several services designed to facilitate application development, such as a non-persistent key/value type storage service (Memcache), a storage service dedicated to the storage of large files (Blobstore), an API enabling the recurring tasks to be planned and run (Cron) and an electronic messaging (email) management service.

46 http://code.google.com/intl/no-NO/appengine/
In October 2008, Microsoft made available Windows Azure, a cloud computing platform designed for its .Net environment. This platform belongs to the PaaS or Platform as a Service platform category.

The Windows Azure platform is based around two proposals: a proposal for hosting applications and data, and a proposal of various services including workflow, data storage and data synchronization services, as well as message buses. The platform is first and foremost presented as being a public cloud, although some advertisements have concentrated on the possibility of deploying it as a private cloud.

The architecture of a Windows Azure application is based on the assembling of components called roles, provided by the platform. Each role represents an aspect of the application and allows access to a set of predefined services. Two main types of role can be distinguished - WebRoles and WorkerRoles:

- **WebRole**: A Web Application that can be accessed through an http or HTTPS terminal point (presentation layer). Essentially made up of ASP.NET presentation mechanisms, possibility of using PHP.
- **WorkerRole**: Background processing application. Exposes internal (for service composition) and external communication points.

The platform provides several methods of storing data:

- **Blobs**: Storage of all types of file: images, video, text…
- **Tables**: Key/value-type property table. These are not relational tables.

---

47 http://www.microsoft.com/windowsazure/
• **Queues**: FIFO stack designed for the asynchronous processing of messages. Can be used to set up asynchronous communication between roles.

• **SQL Azure**: Service for storing data in SQL databases. The implementation of the database is close to SQL Server 2008. Access to the database is based on standard Microsoft object/relational mapping technologies (ADO.NET, ODBC, OLEDB ...).

Windows Azure also implements a data service bus: **AppFabric Service Bus**. This is a set of services enabling services to be composed, integrated and made available, while providing solutions to interoperability and integration issues. This service bus enables applications to be interconnected independently of their location or the identity of the service provider.

Figure 26 - Windows Azure

The two platforms are compared below.

**Table 4 - Comparison of Google App Engine with Windows Azure**

<table>
<thead>
<tr>
<th></th>
<th>Google App Engine</th>
<th>Windows Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMERCIAL FEATURE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVICE LEVEL</td>
<td>PaaS</td>
<td>PaaS</td>
</tr>
<tr>
<td>DELIVERY MODEL</td>
<td>Public cloud</td>
<td>Public / private</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Commercial</td>
<td>Commercial</td>
</tr>
<tr>
<td>BUSINESS MODEL</td>
<td>Payment based on the resources consumed</td>
<td>Payment based on the resources consumed</td>
</tr>
<tr>
<td>INTEROPERABILITY</td>
<td>Compatible with Google Apps Services</td>
<td>.Net applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TECHNICAL FEATURES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTING ARCHITECTURE</td>
<td>Google geo-distributed architecture</td>
<td>Internet-scale cloud service which provides an OS and a set of independent...</td>
</tr>
<tr>
<td>Google App Engine</td>
<td>Windows Azure</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>LOAD BALANCING</strong></td>
<td>Automatic scaling which is transparent for users</td>
<td>Automatic scaling based on application roles and configuration specified by users.</td>
</tr>
<tr>
<td><strong>FAULT TOLERANCE</strong></td>
<td>Automatically pushed to a number of fault tolerant servers</td>
<td>Containers are used for load balancing and availability: in case of failure, services will automatically begin using another replica.</td>
</tr>
<tr>
<td><strong>STORAGE</strong></td>
<td>BigTable storage service (No SQL proprietary database)</td>
<td>SQL Azure storage service and SQL Data Services.</td>
</tr>
<tr>
<td><strong>SECURITY POLICY</strong></td>
<td>Google Secure Data connector</td>
<td>Token service (STS).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVELOPMENT FEATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SUPPORTED API</strong></td>
</tr>
<tr>
<td><strong>PROGRAMMING FRAMEWORK</strong></td>
</tr>
</tbody>
</table>

### 5.3.3 Infrastructure as a Service technologies

**Amazon Elastic Compute Cloud**[^48]

Amazon Elastic Compute Cloud or Amazon EC2 is a cloud computing platform from Amazon, enabling third parties to hire servers on which they will run their own applications. Amazon EC2 belongs, therefore, to the IaaS category.

Amazon EC2 enables the client to dynamically create virtual machines, in other words, server instances that can be used by the server as physical machines. A client can create, run and stop server instances according to its needs, and pays for the time during which he uses the servers. These instances are managed through a set of services that are made available in the form of Web Services.

The architecture of the Amazon EC2 platform is based on Xen virtualization technology. Each virtual machine, called "instance", functions as a private virtual server. The client deploys on these instances "disk images" which include information on the infrastructure (an operating system), the platform (framework, web servers, application servers) and the cloud computing environment application.

[^48]: http://aws.amazon.com/ec2/

[^48]: http://aws.amazon.com/ec2/
Amazon EC2 is only one of several web services sold by Amazon under the Amazon Web Services (AWS) brand. This brand groups together a collection of various services, notably containing database management services, messaging services, network management services, payment systems or workflow services.

Some of these services are compatible with Amazon EC2, such as Amazon Elastic Block Store (EBS) or Amazon Simple Storage Service (Amazon S3), two persistent data storage systems, used as virtual hard disks by virtual machine instances managed by Amazon EC2.

The web-service-based control interface provided by EC2 is in the process of becoming a standard for IaaS platforms. As a consequence, Amazon EC2 is compatible with a certain number of private cloud computing platforms such as Eucalyptus or OpenNebula, which use the same control web services format.

Since Amazon EC2 falls into the public cloud computing service category, it is often used in conjunction with one of the private cloud platforms to build these hybrid cloud systems.

**Eucalyptus** 49

An open source solution resulting from a research project conducted at the University of California, Eucalyptus is a software application that enables a private cloud computing platform to be deployed and used. Available since the end of 2007, this software provides companies with the possibility of implementing their own cloud computing environment.

Eucalyptus belongs to the IaaS category. It enables the deployment of private or hybrid cloud computing platforms (if it is used in conjunction with a compatible public cloud computing service).

From the point of view of the administrator of the platform, Eucalyptus has to be deployed on a set of physical servers in order to create an expected private cloud computing infrastructure.

The architecture of the Eucalyptus cloud computing platform is based on five types of component: the Cloud Controller, the Walrus, Clusters Controllers, Storage Controllers and Node Controllers.

The Cloud Controller and the Walrus are high-level components deployed on the server that is used as the front end on the cloud computing infrastructure. The Cloud Controller is a Java program that implements the platform’s communication interfaces (Amazon Service Web API, Query API and Web interface) from the outside. Walrus manages the virtual machine instances deployed on the platform.

A Cluster Controller, a Storage Controller and a Node Controller are deployed in each of the servers used as a cloud computing infrastructure cluster. These components are used to control a hypervisor (virtual machine system) on each of the clusters.

---

49 http://www.eucalyptus.com/
Figure 27 - The architecture of the Eucalyptus cloud computing platform

From the point of view of the user of a private cloud computing platform, it is possible to connect to the front-end of Eucalyptus, to create instances of virtual machines and to deploy disk images of the applications that are to be run on these instances.

The Eucalyptus platform currently implements a management API that is compatible with the Amazon Service Web (AWS) web service. The support of this API guarantees the interoperability of the Eucalyptus platform with the cloud computing services provided by Amazon. This also enables a hybrid cloud computing platform to be put in place, allowing users to benefit from the advantages linked to a private internal cloud, as well as those of an external public cloud.

OpenNebula\textsuperscript{50}

Available from 2008, OpenNebula is an open source application designed to handle the issues of the dynamic deployment and implementation of virtual services hosted by data centres or clusters of physical servers.

OpenNebula is used as a virtualization tool, enabling the centralized management of a pool of resources coming from a set of servers. This application is generally considered as a private cloud computing platform belonging to the IaaS or Infrastructure as Service platform category.

The platform is presented as a virtual infrastructure management tool used to orchestrate a set of virtual storage services, virtual network management services and virtualization technologies. Open and flexible, the platform adapts to different environments and enables the deployment of all types of cloud computing infrastructures. Used in conjunction with an external public cloud service, OpenNebula enables the deployment of a hybrid cloud computing infrastructure.

The architecture of the private cloud computing platform OpenNebula is built around a front-end that hosts cluster, instance and disk image management services. The clusters host the hypervisors on which instances are run. Communication between the front end and the clusters takes place in SSH.

\textsuperscript{50} http://opennebula.org/
Available from July 2010, OpenStack is a collection of open source technologies delivering a scalable cloud operating system. Designed for institutions and service providers with physique hardware that they like to use for large-scale cloud deployments, this software is backed by Rackspace, NASA, Dell, Citrix, Cisco, Canonical and over 50 other organizations.

OpenStack allows creating and offering cloud computing capabilities using open source software running on standard hardware and is currently developing two interrelated projects: OpenStack Compute and OpenStack Object Storage.

- **OpenStack Compute**: In category of infrastructure as a service cloud platform, OpenStack Compute is a software to provision and manage large groups of virtual private servers. He manages all resources, networking, authorization, and scalability needs for the cloud.
- **OpenStack Object Storage**: software for creating redundant and scalable object storage using clusters of commodity servers to store terabytes of data. It is a long term storage system for a permanent type of static data that can be retrieved, leveraged, and then updated if necessary.

---

IaaS platform comparison

The IaaS platforms are compared below.

Table 5 - Comparison of Amazon EC2 and Eucalyptus

<table>
<thead>
<tr>
<th>COMMERCIAL FEATURE</th>
<th>Amazon EC2</th>
<th>Eucalyptus</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE LEVEL</td>
<td>IaaS</td>
<td>IaaS</td>
</tr>
<tr>
<td>DELIVERY MODEL</td>
<td>Public</td>
<td>Private</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Commercial</td>
<td>Open source</td>
</tr>
<tr>
<td>BUSINESS MODEL</td>
<td>Payment based on the resources consumed</td>
<td>Sales of licenses for an Enterprise Edition</td>
</tr>
<tr>
<td>INTEROPERABILITY</td>
<td>Compatible with Amazon Service Web</td>
<td>Compatible with Amazon EC2 and related platforms</td>
</tr>
</tbody>
</table>

<p>| TECHNICAL FEATURES                  |                                                |                                                |
|-------------------------------------|                                                |                                                |
| COMPUTING ARCHITECTURE              | Allow uploading Xen virtual machine image to the infrastructure and give client API to instantiate and manages them | Multiple clusters with private internal network address |
| VIRTUALIZATION MANAGEMENT          | OS level running on a Xen hypervisor           | Xen hypervisor                                 |
| LOAD BALANCING                      | Service allow users to balance incoming requests across | Simple load balancing                          |</p>
<table>
<thead>
<tr>
<th>Amazon EC2</th>
<th>Eucalyptus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FAULT TOLERANCE</strong></td>
<td>Automatic resynch to last validated state</td>
</tr>
<tr>
<td><strong>STORAGE</strong></td>
<td>Amazon Elastic Block Store (EBS), Amazon Simple Storage Service (Amazon S3)</td>
</tr>
<tr>
<td><strong>SECURITY POLICY</strong></td>
<td>SSL-protected API, SAS70 Type 2 certification</td>
</tr>
</tbody>
</table>

**DEVELOPMENT FEATURE**

| SUPPORTED API | Amazon Service Web API (AWS) | Amazon Service Web API (AWS), Query API |
| PROGRAMMING FRAMEWORK | Amazon machine Image, Amazon MapReduce framework | Java, Ruby |

**Table 6 - Comparison of OpenNebula and OpenStack**

<table>
<thead>
<tr>
<th><strong>COMMERCIAL FEATURE</strong></th>
<th>OpenNebula</th>
<th>OpenStack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SERVICE LEVEL</strong></td>
<td>IaaS</td>
<td>IaaS</td>
</tr>
<tr>
<td><strong>DELIVERY MODEL</strong></td>
<td>Private and Public</td>
<td>Private</td>
</tr>
<tr>
<td><strong>LICENSE</strong></td>
<td>Open source</td>
<td>Open source - Apache 2.0 license</td>
</tr>
<tr>
<td><strong>BUSINESS MODEL</strong></td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td><strong>INTEROPERABILITY</strong></td>
<td>Compatible with Amazon EC2 and related platforms</td>
<td>Compatible with Amazon EC2 and related platforms</td>
</tr>
</tbody>
</table>

**DEVELOPMENT FEATURE**

| **COMPUTING ARCHITECTURE** | Clusters into an IaaS cloud, based on Haizea scheduling | Allow uploading Xen virtual machine image to the infrastructure and give client API to instantiate and manages them |
| **VIRTUALIZATION MANAGEMENT** | Xen, KVM or VMware | KVM hypervisor and XenAPI hypervisors |
OpenNebula | OpenStack
---|---
**LOAD BALANCING** | Round-robin or weighted selection mechanism, Nginx server used as load balancer | Simple load balancing
**FAULT TOLERANCE** | Persistent database to store host and VM information | Isolated processes avoid cascading failures
**STORAGE** | SQLite3, persistent storage for ONE data structure | OpenStack Object Storage
**SECURITY POLICY** | Virtual Private Network tunnel | Certificate management

| SUPPORTED API | Command line, XML-RPC API, Libvirt virtualization API | Amazon Service Web API (AWS)
| PROGRAMMING FRAMEWORK | Hibernate, Axis2, Java | Pyton, Eventlet and Twisted frameworks |

### 5.4 Conclusions

The term 'cloud computing' includes several realities which are reflected by the existence of deployment platforms providing services and imposing different constraints from each other. The concepts of architecture model layer, level and type of service delivery models are essential for the understanding of cloud computing.

Our experiments with “Platforms as a Service” platforms identified several major constraints. These platforms are heterogeneous, and the services and API provided by platforms are not standardized. These platforms even tend to impose a specific architecture of deployed applications. However client applications are built on top of the platform services. Accordingly, there is a significant dependence between the services provided by the platform and client applications. Migrating an existing application to one of its platforms is a challenging task as the abandonment of the platform to return to another form of deployment.

Regarding platforms of “Infrastructure as a Service”, our experiments have shown that this type of platform offers more flexibility than PaaS level platforms. They can handle deployment issues and their impact on application architecture remains limited. However these platforms deliver a service level below that of PaaS platforms. For example, issues of scalability and load balancing are not managed by the platforms and must be managed by client architectures. Since the paradigm of cloud computing at IaaS recommends the use of a large number of low-power virtual machines instead of powerful unique servers, and since migrating an application requires significant resources, it becomes necessary to re-factor the application so as to break it down into several scalable subsystems.

With regard to standardization activities, several initiatives have emerged over the past year, with the objective of putting new cloud computing standards in place. Each of these initiatives aims to create a standard relating to one or two aspects of cloud computing, without trying to provide an overall view of what standardized cloud computing could be. These normalization initiatives have appeared in a disordered manner, and no reference consortium which would play for the cloud a role similar to W3C for the Web has yet emerged.
6 Migration: Design patterns in SOA and cloud

The migration process defined in REMICS includes the task on refactoring the architecture of existing legacy applications using design patterns for SOA and Cloud Computing.

The SOA and Cloud are enabling technologies for each other. Indeed, SOA paradigm implies decomposition of the architecture in coarse-grain components providing business services. Componentization of the architecture enables the scalability and thus a possibility to multiply number of instances of the same component. This fits naturally to the Cloud Computing deployment paradigm. In this way, SOA enables Cloud Computing. On the other hand, when creating a cloud application, it is necessary to ensure that the application can run on a set of low performance resources, to cope with network latencies in a loosely coupled environment. This requires decomposition and decoupling of the application architecture, which basically very similar to SOA paradigm. In this way SOA and Cloud Computing are in the perfectly complementary. SOA provides the software architecture guidelines enabling scalability, while the Cloud Computing addresses the deployment architecture that easily scale.

In the coming sections, we provide a brief summary of our findings on design patterns for SOA and Cloud Computing.

6.1 SOA design patterns

The Service-Oriented Architecture (SOA) design has been a hot topic for at least last 8 years. It is currently in an established state and well covered in literature. SOA relies on the following design principles:

- Standardized Service Contract - the interface to access the service has to be standardized;
- Service Loose Coupling - the dependency between service and the outside world has to be loose;
- Service Abstraction - the service is an abstraction that hides the complexity of business logic behind a standardized interface;
- Service Reusability - reuse is the key aspect of SOA promoting the business benefits for applying this paradigm;
- Service Autonomy - this principle enables consistency and reliability of capabilities provided by a service component;
- Service Statelessness - the services have to be stateless as much as possible to enable the scalability;
- Service Discoverability - to enable service reuse, the services have to be easy to discover;
- Service Composability - complex SOA applications are often built by composition of existing services.

Indeed, these principles enable better scalability and reuse of application components, while concentrating on providing business services.

The SOA design patterns are specified by the community of practitioners and collected in a website. All the patterns have to be taken into account

---

53 http://www.soapatterns.org
when migrating the architecture to SOA, since they provide practical guidelines on different design issues. However, the decomposition and legacy encapsulation patterns are particularly identified as relevant in the context of REMICS. The detailed description and discussion can be found in Erl et al.\textsuperscript{52}

### 6.2 Cloud design patterns

In the contrast to the SOA patterns, the cloud design patterns are quite less covered due to the novelty of the domain. In the REMICS context this is more a research area than a firm state of the art to rely on. In the section below we intend to summarise our first findings which will be enriched later during the REMICS project. The major patterns and conclusions were derived during the analysis by Marks\textsuperscript{56}.

#### 6.2.1 Deployment design patterns

For many, the cloud design patterns are strictly linked with the deployment models presented in Section 5. In the section below we review the design implications imposed by different deployment models.

- **Public cloud**

  The major benefit of the public cloud from the application designer stand point is the ease of deployment, scalability and elasticity of the resources. However, the cloud comes with a cost, the application has to be capable to run on a set of relatively low performance servers. The disk space is accessed by network that implicitly means much higher latencies for data access, though some a limited in-memory caching is possible. This means that drastic refactoring of the monolithic applications may often be required if a migration to a public cloud is envisioned. This may not suite for all applications and business domains. Even if the above mentioned are particularly relevant for public IaaS platforms (Amazon EC2), they are also to be considered for PaaS platforms since the underlying infrastructure is quite similar to public IaaS. Thus, a careful analysis of platforms and migration implications are required before switching to the public cloud.

- **Private cloud**

  In the private or internal cloud, there are fewer limitations on server farm resources, the economical advantages of migration to cloud, i.e. renting the cloud resources and dynamically managing the resource costs are quite lower. In the meantime, the enterprises may benefit from more flexibility in application design, higher resource utilisation rates and possibilities to simplify migration to public clouds. The implications for the design are quite the same as for the public cloud except that the control on network latencies are more manageable by the company. Thus the instance performance issue may be alleviated.

  The private cloud also imposes a limitation on cloud technologies usage. For example, currently Google Apps Engine or Microsoft Azure cannot be freely deployed in the private cloud\textsuperscript{57}. Thus the whole set of PaaS technologies is not available for the private clouds.

- **Hybrid cloud**

  The hybrid cloud option allows application to use public computation resources to cope with peak service demand, to experiment with new business services that can be deployed rapidly at relatively low cost. This is referred to as “cloud bursting”.

  In addition, the applications may consider running part of the system internally and part in a public cloud. For example, to simplify the migration process a monolithic part of the system can be wrapped into an SOA container, while some other parts are refactored and migrated to the public cloud and access the legacy block over the Internet. In this way the system, may be progressively moved to the


\textsuperscript{57} Microsoft Azure has a option for private clouds, but this is affordable for big enterprises only.
public cloud. Another scenario may be when data and processed locally, while the GUI and part of the business logic is moved to the public cloud. This may allow having higher performance servers with high speed data access and controlled reliability, while combining them with elastic public computation resources for business logic and GUI in order to enable on demand availability.

From the design perspective, when considering the hybrid clouds, one should take into account both public cloud and private cloud design methods. In addition, for the cloud bursting the cloud interoperability issues has to be covered.

6.2.2 PaaS and IaaS design patterns

From the application design stand point, methods for creating PaaS and IaaS applications are quite different.

At PaaS level, it is relatively easier to create and deploy the applications, since the platform is already set and given, the scalability and elasticity of the resources is out of developer control. The scalability, elasticity and resources are managed by the PaaS provider. In this way, developing a PaaS application is no different from developing a conventional Application Server hosted system. A PaaS provider provides some more scalability oriented features than a regular Application Server, but the design principles are roughly the same. This is more valid for Microsoft Azure which mimics the .Net environment, while Google Apps Engine is more revolutionary and oriented on the python. Although Google PaaS provides a Java and JDO APIs, this application server is quite restrictive. The communication between services is quite limited, since no bus implementation is available. All data is stored in big trees with no built-in cross-referencing. A service request over HTTP cannot be longer than 15 seconds. These all to provide means to ease the scalability, though this impose severe limitations on application designers. Our conclusion is that these limitations are very platform depend and should be considered provider by provider. Thus, deriving some generic design patterns that fits all PaaS is not possible. It is suggested to rely on design patterns guidelines by PaaS providers.

At IaaS level, it is the application designer responsibility to specify and deploy the right platform. The application development methods are quite conventional; one may follow SOA or Enterprise Application design patterns. The scalability is managed by the application designer, which chooses suitable design principles and frameworks.

The difference between traditional design and IaaS is that designer has an additional degree of freedom - resource elasticity, i.e. a possibility to easily pop-up new instances of computation resources.

It should be also emphasized that the designer has to decompose the application in a set of components capable to run on relatively low performance computation resources. Although, this is alleviated for the private clouds, it is currently a fundamental limitation for public and hybrid clouds.

6.2.3 Data structures design patterns

One of the particular aspects of the application design is the data structures and storage. Currently the mainstream technology for data storage is relational databases following ACID principles. RDBMS scalability has been a subject of huge number of debates, since the data layer is too often a bottleneck in distributed applications. In particular, the data structure design suffers from the "false dependency" issue imposed by relational database paradigm and design decisions. The transactional mechanism imposes additional delays in database replication, while procedure storage mechanisms are very costly for computation resources.

With appearance of data intensive, massive multiuser cloud service like Google search, Amazon and Facebook, the relational databases could not meet the performance requirements for data access. The so-called "NoSQL" databases became prominent technology for the cloud community. The big tables, trees, technologies like MapReduce implemented in Google Megastore and Apache Hadoop became references for cloud applications and platforms. The "partial inconsistency" during replication is now considered as acceptable in many applications in order to address rigidity of transactional mechanism.
For the migration to cloud the following design patterns may be derived:

- **Data partitioning.** In order to cope with "false dependencies" issues and enable scalability, the data have to be partitioned in independent structures. This can be achieved by designers by analysing the data semantics and deriving semantically independent subsets.

- **Avoid stored procedures.** The stored procedures have to be migrated to proper business logic components.

- **Partial inconsistency.** Consider partial inconsistency, which can reduce the pressure on the database layer.

### 6.3 Conclusions

In this section a summary of relevant SOA and Cloud design patterns was provided. While state of the arte in SOA is quite established and covered in literature, the cloud design patterns are more an ad-hoc knowledge that still has to be studied.

In conclusion to this section, we would like to provide simplified guidelines for the legacy system re-design for migration to cloud.

1. Decompose the application in a set of components that can run on low resources.
2. Choose or build a unified platform for your application that can be easily replicated/instantiated
7 Model-driven interoperability

7.1 General introduction

Interoperability is defined as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”\(^{58}\). Interoperability issues arise when to systems should collaborate. It is possible to define interoperability at different levels: process, data, etc.

In the context of the REMICS project (WP5), we are particularly interested in applying Model-Driven Interoperability techniques to support cloud services interoperability after the legacy system has been migrated. This way, the migrated system could interoperate with existing and future cloud services that offer almost the same services with different properties (e.g. QoS).

More precisely, and according to the Description of Work, the Model Driven Interoperability (MDI) aims at adapting existing services to services required to complete the Target Architecture using for example automatically generated mediators. This could be achieved by extending SoaML\(^{59}\) models with support for interoperability, such as a proper modelling of the data manipulated (e.g., class diagrams) by the services and their behaviour. In this section, we thus survey projects, standards and products that focus on data interoperability to achieve service interoperability.

7.2 Projects

7.2.1 SWING and ENVISION

Semantic Web Services Interoperability for Geospatial Decision Making (SWING\(^{60}\)) and Environmental Services Infrastructure with Ontologies (ENVISION\(^{61}\)) are two relevant interoperability projects in the geospatial decision making and environmental modelling and monitoring domains, respectively. SWING developed a set of tools and technologies for integration of geospatial services and content, based on semantic Web and Web services technologies. The techniques for modelling, discovery, composition and execution of geospatial services developed in SWING are currently being extended in the ENVISION project, with a focus on discovery, composition, and mediation of environmental services.

7.2.2 EMPOWER

A Semantic Service-Oriented Private Adaptation Layer Enabling the Next Generation, Interoperable and Easy-to-Integrate Software Products of European Software SMEs (EMPOWER\(^{62}\)) is another relevant project in the area of data and service interoperability, with a particular focus on integration of ERP systems. The project develops two interoperability layers: one that exposes data and service using standardized data and service models, and one that focuses on the semantic interoperability of the exposed data and services.

7.2.3 INTEROP NoE

The INTEROP network of excellence is composed of several partners organized in geographical poles: the **UK Pole** represented by TANET Ltd located at the University of Coventry; the **German Pole** represented by DFI Association, located at IPK Fraunhofer, Berlin; the **PGSO (Pole Grand...**


\(^{59}\) Service Oriented Architecture Modeling language; http://www.omg.org/spec/SoaML/

\(^{60}\) http://138.232.65.156/swing/

\(^{61}\) http://www.envision-project.eu/

\(^{62}\) http://empower-project.eu/
Sud- Ouest, France) a SIG (Scientific Interest Group) represented by the University Bordeaux 1; the China Pole a SIG represented by Harbin Institute of Technology; the Spanish Pole, represented by INTERVAL Association located at the Universidad Politecnica de Valencia; the Portuguese Pole, a SIG represented by Uninova, New University of Lisbon; the North Pole (Sweden, Norway, Finland) a SIG represented by University of Bergen; the Italian Pole, represented by IASI-CNR located in Roma; and the Pole Grande-Region (Lorraine, Belgium) a SIG* represented by University of Nancy Henri Poincaré.

Relevant to REMICS, a MDI Method has been proposed by INTEROP. This MDI method aims at making two enterprises to interoperate not only at the code level but also at Enterprise Modelling level. This method relies on model transformations:

- Vertical transformations following a MDA approach. However, the CIM level is split into top and bottom CIM in order to reduce the gap with the PIM level and make the transformation CIM2PIM more straightforward.

- Horizontal transformations to ensure interoperability at different levels of abstraction. The horizontal transformations rely on a Common Interoperability Ontology to provide semantic support for solving interoperability issues.

7.2.4 ATHENA

The aim of ATHENA IP project was to develop solutions to overcome interoperability problems in ICT which affect networked businesses including SMEs. The project produced a set of innovative solutions in the ICT areas: Enterprise Models Interoperability, Cross-organizational Business Processes, Semantic Business Document Reconciliation, IT Service selection and composition and Model-driven Architecture for SOA.

The ATHENA Interoperability Framework (AIF) is based on MDI (Model-Driven Interoperability) principles where model-driven development is integrated to adaptable interoperability architectures. The ATHENA interoperability reference model has four horizontal axes (i.e. Information, Service, Process, Organisation interoperability) and two vertical cross-cutting dimensions (i.e. Model-driven and Semantic-enabled interoperability) focusing on model transformations and interoperability models. The approach resulted in a set of tools and services to address interoperability issues. Six classes of EI services were identified in the reference model: services dynamic composition and orchestration, enterprise modelling, cross-enterprise business processes, data interoperability, model-driven interoperability and semantic mediation of business documents.

The AIF also provides an associated methodological framework, the ATHENA Interoperability Methodology (AIM), which describes the approach towards interoperability from the decision to evaluate collaboration until solution maintenance, and the reference guidelines for the adoption of the reference architecture. One of the main components of this methodology was the EIMM (Enterprise Interoperability Maturity Model) which provides assistance in capturing the collaborative processes of the company with the support of one of several adequate modelling approaches.

7.2.5 COIN

The aim of COIN IP project is to study, design, develop and prototype an open, self-adaptive, generic ICT integrated solution to support enterprise collaboration and enterprise interoperability services. In particular, a COIN business-pervasive open-source service platform is able to expose, integrate, compose and mash-up in a secure and adaptive way existing and innovative to-be-developed Enterprise Interoperability (EI) and Enterprise Collaboration (EC) services, by applying intelligent maturity models, business rules and self-adaptive decision-support guidelines to guarantee the best combination of the needed services in dependence of the business context, as industrial sector and domain, size of the companies involved, openness and dynamics of collaboration. The COIN project is also developing an original business model based on the SaaS-U (Software as a Service-Utility) paradigm where the open-source COIN service platform will be able to integrate both free-of-charge and chargeable, open and proprietary services depending on the case and business policies.
The COIN approach to Enterprise Interoperability is to distinguish Information Interoperability, Knowledge Interoperability and Business Interoperability, and to provide the required enterprise interoperability services. Information interoperability explores new services communication and coordination spaces (see below); knowledge interoperability is concerned with exploiting at best the knowledge of each enterprises collaborating together, focusing on their complementarities; business interoperability in COIN goes beyond the cross-enterprise business process coordination, and provides new formalisms for interactive and collaborative Business Process Management.

Of interest for REMICS are the Information Interoperability Services. The project introduces the notion of Interoperability Spaces, with three services:

- **Innovative Services for Semantic Reconciliation**, which focus on the translation between documents and ontologies;
- **Data Payload Interoperability Services**, which perform communication, coordination and exchange of business documents (1:1, 1:n, n:m communications);
- **Innovative Services for Federated Interoperability**, which concentrate on information interoperability in a federated space, where no common reference models are available.

### 7.2.6 CONNECT

The CONNECT Integrated Project aims at enabling continuous composition of networked systems to respond to the evolution of functionalities provided to and required from the networked environment. Current interoperable middlewares cannot cover the ever growing heterogeneity dimensions of the networked environment. CONNECT aims at providing new interoperability mechanisms consisting in synthesizing on the fly the connectors via which networked systems communicate.

These connectors are synthesized according to the behavioural semantics of application- down to middleware-layer protocols run by the interacting parties. The synthesis process is based on a formal foundation for connectors, which allows learning, reasoning about and adapting the interaction behaviour of networked systems at run-time. Synthesized connectors are concrete emergent system entities that are dependable, unobtrusive, and evolvable, while not compromising the quality of software applications.

### 7.3 Standards

#### 7.3.1 W3C

Semantic Web standards/recommendations, mainly developed by the World Wide Web consortium (W3C), provide a common framework for integration and combination of data drawn from diverse sources. Core W3C semantic Web recommendations include the Resource Description Framework (RDF) - a standard model for data interchange on the Web; SPARQL - a query language for RDF; Web Ontology Language (OWL) - an ontology language; and Rule Interchange Format (RIF) - an XML language for expressing rules which computers can execute.

#### 7.3.2 OMG

The Semantics of Business Vocabulary and Business Rules (SBVR) is a standard developed by the Object Management Group (OMG) and aims to provide the means for a declarative and structured natural language description of complex entities, such as a business. SBVR's focus is on

---


64. [http://www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)

65. [http://www.omg.org/spec/SBVR/1.0/](http://www.omg.org/spec/SBVR/1.0/)
capturing business vocabularies and rules in a more structured way that can improve communication and eventually enable better interoperability.

The Model-Driven Architecture (MDA) is an OMG standard for the development of software systems. MDA promote the use of different models: CIM, PIM and PSM, which describes the software system at different levels of abstraction. The use of model transformation or code generation makes it possible to automatically bridge the gap between these levels. The MDI method proposed by the INTEROP NoE for relies on MDA.

Model transformations play a key role in MDI. QVT (Query/View/Transformation\(^{66}\)) is an OMG standard compatible with MDA to define model transformation. It is a super-set of the OCL 2.0 (Object Constraint Language\(^{67}\)) standard that allows designer to manipulate and transforms models conforming to a metamodel, which itself conforms to MOF (Meta Object Facility\(^{68}\)).

### 7.3.3 Web Services Interoperability (WS-I)

The Web Services Interoperability Organization (WS-I\(^{69}\)) is an open industry organization chartered to establish Best Practices for Web services interoperability, for selected groups of Web services standards, across platforms, operating systems and programming languages.

They provide resources for Web services developers to create interoperable Web services and verify that their results are compliant with WS-I guidelines. They define profiles that specifies implementation guidelines for how related web services specifications should be used together to achieve interoperability. Currently, WS-I has defined a basic profile that address usage (guidance and clarification of usage) of standard, and basic web service technologies to promote interoperability. They also provide a profile for attachments, SOAP bindings, and security.

To further guide the development of interoperable web service, they also provide sample applications and testing tools to validate interoperability issues.

### 7.3.4 OAGIS

The Open Applications Group Integration Specification (OAGIS) by the Open Applications Group is an effort to provide a canonical business language for information integration\(^{70}\). It uses XML as the common alphabet for defining business messages, and for identifying business processes (scenarios) that allow businesses and business applications to communicate.

The core of the OAGIS architecture is the Business Object Documents that are exchanged between collaborating business partners. OAGIS reuses definitions from the UN/EDIFACT standard. UN/EDIFACT is the international standard for electronic data interchange (EDI) for Administration, Commerce, and Transport, developed by United Nations. UN/EDIFACT defines an exchange protocol and a set of standard messages. EDIFACT for XML is standardised within ISO as ISO/TS 20625.

---

\(^{66}\) [http://www.omg.org/spec/QVT/1.0/](http://www.omg.org/spec/QVT/1.0/)

\(^{67}\) [http://www.omg.org/spec/OCL/2.0/](http://www.omg.org/spec/OCL/2.0/)

\(^{68}\) [http://www.omg.org/spec/MOF/2.0/](http://www.omg.org/spec/MOF/2.0/)

\(^{69}\) [http://www.ws-i.org](http://www.ws-i.org)

\(^{70}\) [http://www.oagi.org and](http://www.oagi.org)


[http://www.unece.org/trade/undid/welcome.htm](http://www.unece.org/trade/undid/welcome.htm)
7.3.5 OASIS

OASIS\textsuperscript{71} is a non-profit organisation that works on open standards. They have developed a wide range of standards in the web services domain and e-commerce, such as \textit{Web Service Security, Trust, Transactions} for general web services, and \textit{Universal Business Language (UBL)} and \textit{ebXML} in the e-Commerce domain.

The \textit{ebXML}\textsuperscript{72} - Electronic Business using eXtensible Markup Language – defines a standard for exchange of business messages, conducting trading relationships, communicate data, and defining business processes.

7.3.6 Open Group

The Open Group has within its \textit{Semantic Interoperability WorkGroup} has defined \textit{Universal Data Element Framework (UDEF)}\textsuperscript{73}, which is a framework for semantic interoperability. It provides a standard way of indexing enterprise information, through conceptual naming conventions with associated structured unique identifiers, which allow for indexing and aligning semantically equivalent concepts.

The Open Group has also defined TOGAF\textsuperscript{74} (The Open Group Architecture Framework\textsuperscript{74}), which is a method for enterprise architecture development. One part of the framework is guidelines for defining and establishing \textit{interoperability requirements}.

7.3.7 ISO/IEC 19763 - Metamodel Framework for Interoperability

The ISO/IEC 19763\textsuperscript{75} defines a set of standards for a Metamodel Framework For Interoperability (MFI). Much of this work is still ongoing with only preliminary specifications available. The standard consists of multiple parts: a reference model (part 1), a core model (part 2), a metamodel for ontology registration (part 3), a metamodel for model mapping (part 4), a metamodel for process model registration (part 5), a metamodel for service registration (part 7), a metamodel for role and goal registration (part 8), and a specification for on demand service selection (part 9).

The framework (part 2) provides a common basis for consistent registration of metamodels that should help to map between metamodels that are developed independently. The metamodel for ontology registration (part 3) aims to provide a unified framework for registering administrative information of ontology. The metamodel for model mapping (part 4/ part 10) provides a reference for model mappings, or transformations. This part is tightly connected with OMG standards such as QVT. The metamodel for service registration aims at standardising semantic description of services that can be used for description and discovery, like a UDDI registry, but with additional semantics like quality characteristics.

7.3.8 ISO STEP

The ISO STEP standard (ISO 10303) defines a standard for data exchange for product model data, and has been widely applied for computer aided design, manufacturing, and product data management in various manufacturing industries, such as automotive, ship building, construction, aerospace, and oil-and-gas. It defines application protocols (domain-specific information models) for the various domains, and provides a standard description format (EXPRESS), as well as standard formats for data exchange and API interoperability through the STEP/STEP-XML formats and \textit{Standard Data Access Interface (SDAI)}, respectively. The main idea for supporting interoperability

\textsuperscript{71} http://www.oasis-open.org

\textsuperscript{72} http://www.ebxml.org/

\textsuperscript{73} http://www.opengroup.org/udef/

\textsuperscript{74} http://www.opengroup.org/architecture/togaf9-doc/arch/index.htm

\textsuperscript{75} http://metadata-stds.org/19763/index.html
and interchange of data is conformance to the well-defined application protocols, or common domain-models.

A related standard, specifically targeting integration of lifecycle data for process plants is defined by POSC/Caesar in ISO 1592676).

Derivatives of the STEP standards have been standardised within OMG as Product Lifecycle Management Services77.

7.4 Products and Approaches

This sub-section presents representative academic tool-supported approaches from the Model-Driven Interoperability community. FloraMap is concerned about the migration of XML-based data while ModMap is more concerned about the alignment of “legacy” (Java-based) API.

7.4.1 FloraMap

FloraMap78 is a technique and tool for XML data mediation/exchange developed in the context of the EMPOWER project. FloraMap is based on logical rules for specifying mappings at the schema level and executing those mappings at the instance level. It implements a consistency checking technique that checks for semantic inconsistencies in schema documents before the actual XML data exchange takes place. The current implementation of FloraMap uses F-logic as the language for formalizing schemas, mapping rules, and instances, and its Flora2 system as the underlying reasoning engine for mapping execution.

7.4.2 ModMap

Clavreul et al. present ModMap79, a tool-supported academic approach to integrate legacy systems. They combine reverse-engineering, model-to-model transformation, code generation and aspect weaving to achieve this result. The APIs of the two legacy systems (preferably implemented in Java) are transformed into models. The designer then uses an EMF-based graphical DSL to express mapping between the APIs. They propose different cardinalities for the mapping (1 to 1, 1 to many, etc) and several strategies with different degrees of automation:

- Automatic Alignment using a clone strategy
- Named-based Alignment using a renaming strategy
- Constrained Alignment
- Free Alignment using a strategy coded by the user

From these specifications, all the glue code is generated. This generated code is expressed in AspectJ and extends the legacy systems (with no modification to the existing code) to make them to interoperate.

76 http://ng.tc184-sc4.org/, https://www.posccaesar.org

77 http://www.omg.org/spec/PLM/2.0/, OMG document no formal/2009-08-01

78 Yongxin Liao, Dumitru Roman and Arne J. Berre. – Model-Driven Rule-based Mediation in XML data Exchange. – In 1st Model-Driven Interoperability workshop at MODELS’10

79 Mickael Clavreul, Olivier Barais, and Jean-Marc Jézéquel. -- Integrating legacy systems with MDE0. -- In ICSE’10: 32nd ACM/IEEE International Conference on Software Engineering, Cape Town, South Africa, May 2010.
7.4.3 Model Transformation

There exist several model transformation languages such as ATL\textsuperscript{80} or Kermeta\textsuperscript{81}, which are based on the EMF implementation of OMG MOF, and which provides advanced languages (compared to directly using Java and EMF APIs) to manipulate models. The tools can serve as a basis to implement interoperability mechanisms at a platform-independent level. The MOF QVT standard\textsuperscript{82} (Query/View/Transformation) is also for model transformations and there are several products (commercial or open source) that claim compliance to the QVT standard.

7.5 Conclusions

Model-Driven Interoperability is a rather new domain (see for example MDI workshop 2010, the first workshop on Model-Driven Interoperability), which builds on top of long history on data and service interoperability. REMICS will actively contribute to this emerging domain. In particular, REMICS will leverage platform-independent models to define mappings independent from any specific technologies and will leverage M2M (model to model) and M2T (model to text) transformations to generate mediators, to propose mappings to the designers based on heuristic, etc.

One important challenge in a dynamic context such a Service-Oriented Architecture or Cloud computing is the ability to map services (e.g., the ones recovered and migrated in WP3-4) to a range of evolving external services. A fully automated process seems difficult and even unrealistic, due to complex semantic mismatches/misalignments between these services. Rather, REMICS will investigate semi-automatic techniques that involve human-in-the-loop by proposing some mappings to the user, who will be able to validate/discard, or modify/update these mappings according to his knowledge.

\textsuperscript{80} www.eclipse.org/atl
\textsuperscript{81} http://www.kermeta.org
\textsuperscript{82} http://www.omg.org/technology/documents/modeling_spec_catalog.htm
8 Model execution and Models@runtime

8.1 Concepts and landscape, relevance to REMICS

A recent issue of the visionary IEEE computer magazine has drawn the forthcoming perspectives about the utilization of models@runtime. 5 papers show how “models” in the spirit of MDD may instrument software adaptation. A decade ago, in the same magazine, J.O. Kephart and D.M. Chess stated their "Vision of Autonomic Computing". They explain why there is an increasing need of application management tools covering supervision, maintenance and autonomy.

The exponential presence of software in everyday objects and computer systems (including supercomputers) requires self-management. In a self-managing software system, the human operator takes on a new role: she/he no longer controls the system directly. Instead, she/he defines general policies and rules that serve as an input for the self-management process that is, as far as possible, carried out automatically. This includes:

- Self-Configuration: automatic configuration of components;
- Self-Healing: automatic discovery, and correction of faults;
- Self-Optimization: automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
- Self-Protection: proactive identification and protection from arbitrary attacks.

Of course, cloud computing refers to a family of autonomic systems in the sense that users expect greater adaptability of their computing systems and environments, e.g., adaptation of storage capacities, adaptation of performance, self-assurance in general about applications behaviours.

Expected advances generate new requirements like the observation of software through appropriate visualization: in this sense, models are first-class candidates since MDSD essentially fosters graphical modelling languages. At this time, researchers from the MDSD field have gained significant results in the sub-field of executable modelling languages. However, the notion of models@runtime is not fully stabilized and thus still subject to (healthy!) controversies. REMICS WP6 will develop a more agreed notion and its software support for models@runtime and its link to dynamic verification on one side and SoaML (its cloud extensions as well) on another side. Software components that embody and power services at runtime may benefit from having embedded models to monitor (trace, understand, etc.) and control (adjust, change etc.) their behaviours when environments/users require any kind of adaptation.

The idea of models@runtime is at the confluence of MDSD and autonomic computing to solve software adaptation problems in contemporary software applications and software-intensive systems. The idea to make these models persistent at execution time is a way to instrument adaptation. Software applications are growing rapidly in size and in complexity while the dependencies of the global economy and societies against software are rapidly spreading. Mastering software evolution moves from a craft scale to an industrial strategic scale. The lasting quality of software systems becomes a hot topical question that was not solved by legacy applications. Reconstructing these systems on the top of new technologies impose not to make again the same mistake.

Sustainability more generally requires in the forthcoming decades to manage software in an energy-efficient way, to maximize reuse, to drastically decrease uncritical maintenance tasks... In short, a technology is required to implement adequate abilities in software to really become autonomous. To give an accurate overview of the rising problem, 5 billions of Java Virtual Machines (JVMs) are currently installed in computing systems, from smart/professional devices to high-performance computers and, of course, personal computers. Java is the springboard to create a digital chain that...

83 www.computer.org/portal/web/csdl/doi/10.1109/MC.2009.326
84 en.wikipedia.org/wiki/Autonomic_Computing
is inexorably growing. How to master billions of billions of computing (logical) machines in the future with high componentization (components, services…) in the cloud especially?

8.2 National, European and Worldwide R&D initiatives, clusters, work programmes and strategic roadmaps

8.2.1 Europe

Several completed or ongoing projects stress self-* software systems. Three important and significant projects are DIVA\(^85\), MUSIC\(^86\) and PLASTIC\(^87\). On another side, the FET (Future Emerging Technologies) Proactive Initiative: Self-Awareness in Autonomic Systems\(^88\) emphasizes self-optimization only in the logic of resource burden decreasing, energy savings. This is also concerned with networks rather than software systems on their own.

The ERC has chosen two projects closely related to the topics of REMICS WP6 for its very selective 'Advanced Investigators grant'. The SMScom project\(^89\) led by Carlo Ghezzi aims at revisiting software engineering in order to discover innovative approaches for engineering self-adaptive systems. Veriware\(^90\) led by Marta Kwiatkowska investigates stochastic model checking for verifying ubiquitous software systems. The research topics addressed REMICS WP6 match some of the key concerns announced by the EU for ICT in 2009-2010, Objective ICT-2009.1.2: Internet of Services, Software and Virtualisation encompasses:

“b) Highly Innovative Service / Software Engineering

- Service / Software engineering methods and tools covering automatic support at run-time for decisions and changes that are currently adopted at design time. Focus is on innovative approaches to very large, dynamic open service networks, user development of services/software, systems evolvability and acquisition, reasoning and incorporation of domain knowledge in all phases of the service/software life cycle. High-level description and executable languages for services/software with support for adaptation and technologies for improving system response time, performance and throughput are in the scope of the research,
- Verification and validation methods, tools and techniques assuring the quality of open, large-scale, dynamic service systems without fixed system boundaries, addressing the complete service and software life cycle.”

8.2.2 USA

The position of REMICS WP6 with regard to USA strategic roadmaps is as follows:

**NITRD:** The position of REMICS WP6 with regard to the USA National Coordination Office (NCO) for Networking and Information Technology Research and Development (NITRD) is in the Software Design and Productivity (SDP) R&D area. In this area, the “President’s 2010 Request, Strategic Priorities Underlying This Request” SDP R&D priorities are:

- “Research to rethink software design: From the basic concepts of design, evolution, and adaptation to advanced systems that seamlessly integrate human and computational capabilities;"
• Foundations: New computational models and logics, techniques, languages, tools, metrics, and processes for developing and analyzing software for complex software-intensive systems (e.g., a principled approach to software engineering that can provide assurances, (e.g., accountability, real-time, security, and affordability);

• Predictable, timely, cost-effective development of software-intensive systems: Disciplined methods, technologies, and tools for systems and software engineering, rapidly evaluating alternative solutions to address evolving needs; measuring, predicting, and controlling software properties and tradeoffs; virtualized and model-based development environments; scalable analysis, test generation, and optimization with traceability to requirements."

The position of REMICS WP6 with regard to the NCO/NITRD is also in the High Confidence Software and Systems (HCSS) R&D area. In this area, the “President's 2010 Request, Strategic Priorities Underlying This Request” SDP R&D priorities are:

• “High-confidence systems and foundations of assured computing: Methods and tools for modelling, measuring, analyzing, evaluating, and predicting performance, correctness, efficiency, dependability, scalability, and usability of complex, real-time, distributed, and mobile systems; high-confidence platforms for sensing and control; virtualization, architectures, components, composition, and configuration; systems-of-systems governance, engineering, analysis and testing of software and hardware;

• Specification and synthesis, programming language semantics, and computational models; advance tools design, development, V&V, and analysis for dependable computing; techniques for assuming developed applications are free from malware and vulnerabilities”

NSF: The position of REMICS WP6 with regard to the USA National Science Foundation (NSF) is in the Computer & Information Science & Engineering (CISE) cluster, and more precisely in the 2009 Computer and Network Systems sub-cluster91 of the CISE.

The Computer Systems Research (CSR) program especially addresses the following issues: “The need to operate in heterogeneous, unpredictable and challenging environments requires ground-breaking approaches and methodologies to advance our understanding of how computation is performed and how resources are managed, at varying levels of granularity and scale. The proliferation of Internet-scale applications and services poses new challenges and require radical thinking of how distributed computation is carried out (…). This is further compounded by the need for energy-efficient and self-managing systems (…)”

The Network Science and Engineering (NetSE) program of the Cross-Cutting Programs: FY 2010 sub-cluster of the CISE, highlights the following challenges: “(…) the NetSE program encourages research on Internet-scale, (…), context-awareness, and architectures that enable the discovery, invocation and composition of globally distributed, highly evolving services and information systems. These new kinds of models, capabilities, and architectures in turn enable the exploration of new applications (…)”

Now, in terms of model execution supports, the two main worldwide initiatives about model (statecharts) execution are SCXML and MXF.

8.2.3 Worldwide R&D initiatives

State Chart XML92 is a normative XML dialect from the W3C for expressing and executing Harel's Statecharts. It differs from UML XMI (XML Metadata Interchange). Indeed, XMI includes a subset dedicated to recording State Machine Diagrams and Sequence Diagrams (scenarios). SCXML offers its own document organization to record statecharts. As a result, it also provides a quite formal accurate algorithm to interpret statecharts. This algorithm is divided into coherent blocks and sequentially described in the SCXML documentation while UML only lists declarative rules of

91 www.nsf.gov/div/index.jsp?div=CNS
92 SCXML, www.w3.org/TR/scxml/
execution whose union is supposed to compose an execution semantics. The SCXML interpretation algorithm is reliable in the sense that it precisely relies on the SCXML tags and their attributes. Implementors of this standard are invited to provide execution engines by respecting the offered algorithm.

The SCXML semantics is precise and closely related to that of UML. For example, in case of conflicts, inner transitions outgoing from nested states have priority on outer transitions outgoing from their wrapper states. The reverse rule is used in the semantics of the original Harel's Statecharts. This semantic choice is thus representative of the difference between UML/SCXML on one side and Harel's initial language on the other side. The difference between UML and SCXML is that UML tolerates semantic variation points. This is both an inconvenient and an opportunity.

Commons SCXML from the Apache foundation\(^93\) is the most well-known implementation (in Java) of SCXML. It does not support all of the SCXML notation. It does not also support semantic variation points. SCXML indeed considers that these points may break portability but, concomitantly, SCXML authors acknowledge that this creates flexibility. PauWare in contrast is able to address semantic variation issues by means of configuration values assigned to state machines.

Commons SCXML puts forward models@runtime because other Java technologies include it as a component. Indeed, SCXML has been mainly designed for VoiceXML-based applications and more generally computer-telephony integration (call centre applications for instance) and application management in general. No search for integration within EMF is currently carried out. Issues about model-based self-* software systems are not currently addressed in SCXML. In contrast, PauWare is dedicated to the design of self-* systems (see Evaluation above).

**Model Execution Framework (MXF)**\(^94\) is a recent initiative. The main aim of MXF is to provide a framework for development, execution and debugging of models with operational semantics. In contrast with SCXML, MXF is EMF-compliant and benefits from all of the surrounding tools of the EMF technology, model transformation facilities especially. The M3Actions framework is currently the available implementation. This implementation strongly focuses on design issues; development of a models@runtime support is a forthcoming goal of this EMF sub-technology.

Others academic and/or industrial (even commercial) tools exist. They all address execution/simulation issues. However, many of them reveal strong limitations, even bugs, with respect to the challenges and goals of this project. These are:

- The state machine compiler\(^95\)
- Statechart Virtual Machine\(^96\)
- C++ Harel's Statecharts execution engine\(^97\)
- Statecharts tools\(^98\)
- CHSM\(^99\) standing for “Concurrent Hierarchical State Machine” is a specification (homemade textual) language from which C++ or Java programs may be generated. These programs respect of course the specified behaviour with Statecharts
- PauWare (www.PauWare.com/PauWare_software) is a rich Java library to implement Statecharts as done with CHSM. This library supports Java ME, SE and EE including Java

---

\(^93\) commons.apache.org/scxml
\(^94\) www.eclipse.org/proposals/mxf/
\(^95\) smc.sourceforge.net
\(^96\) msdl.cs.mcgill.ca/people/tfeng/
\(^97\) www.codeproject.com/KB/cpp/Statechart.aspx?display=Print
\(^98\) patf-biokyb.lf1.cuni.cz/wiki/statecharts
\(^99\) chsm.sourceforge.net/
EE 7 that is more dedicated to cloud computing. SimUML\textsuperscript{100} is a PauWare-inspired component built for the Topcased\textsuperscript{101} large MDD IDE/platform. Otherwise, many commercial products (Stateflow\textsuperscript{102}, Smartstudio\textsuperscript{103}, etc.) or open source large-scope tool suites exist. Many of them offer model execution facilities and a support for extensibility like Fujaba\textsuperscript{104}. The implementation of this project’s research results in these products is possible.

8.3 Standards

FUML (Semantics of a Foundational Subset for Executable UML Models) from the OMG has been just released in June 2010 (version 1.0, beta 3). The scope of this specification is the selection of a subset of the UML 2 metamodel that provides a shared foundation for higher-level UML modelling concepts, as well as the precise definition of the execution semantics of that subset. To support a variety of different execution paradigms and environments — including a number of widely used commercial and research variants of executable UML— the specification of the execution model incorporates a degree of genericity. In this scope, FUML may serve as backbone for REMICS subject to an alignment with SoaML and a concrete implementation is available and/or provided.

Concrete Syntax for UML Action Language (a.k.a. Action Language for Foundational UML or ALF) complements FUML. It is a specification is a textual surface representation for UML modelling elements. The execution semantics for ALF are given by mapping the ALF concrete syntax to the abstract syntax of FUML. This specification is not finalized yet.

8.4 Conclusions

As shown above, the maturity of executable modelling language is real but the compatibility between a variety of products/libraries/tools and the very recent FUML standard is poor. Beyond, the potential usages and beneficial impacts of models@runtime remain unclear even if this computing paradigm is promising. This is especially true for applications running in the cloud. The probable loss of runtime observation/control means for designers/developers on these applications impose, \textit{a minima}, the introduction of specific material to measure the progresses of any renewed application compared to its old-fashion shape when in its legacy state. The research carried out in REMICS on this point is almost exploratory. It looks for open cloud solutions that let in software engineers’ hands means for observing and controlling information systems even if cloud infrastructures aim at addressing this issue (\textit{i.e.}, self-adaptation in general) with greater attention and care.

8.5 References


\textsuperscript{100} sourceforge.net/projects/simultm/
\textsuperscript{101} www.topcased.org
\textsuperscript{102} www.mathworks.com/products/stateflow
\textsuperscript{103} www.smartstatestudio.com
\textsuperscript{104} wwwcs.uni-paderborn.de/cs/finite/projects/realtime/main.html


9 Model-based verification

9.1 General introduction

Verification is the activity of proving that a system satisfies a certain set of properties. Different kinds of properties can be verified and also different techniques and methods can be applied. In general, a specific property cannot be proven by every technique, but rather the observation of a certain property implies the usages of corresponding techniques to verify this property. For instance, in order to prove safety aspects of a system it is not valuable to use consistency checking methods. Thus, in the context of complex systems, and in particular in the Cloud domain which are distributed and heterogeneous, several different verification techniques have to be used.

In the REMICS project the whole software development methodology will be based on models. Therefore we are focussing on model verification or model-based verification approaches. In general, model-based verification mainly aims to verify properties on model level. For instance model checker\(^\text{105}\) dealing with the verification of deadlock freedom in behaviour models as well as consistency checker\(^\text{106}\) that are addressing the correctness and fulfilment of specific modelling rules such as naming conventions. A lot of available tools and techniques are also relevant for the REMICS approach. However, properties of a system that should be proven imply the selection of tools and techniques that fit best to actually commit the proof. The verification may address correctness, soundness and completeness of different aspects of a system, including static properties (e.g. system architecture and structure) and dynamic properties (e.g. behaviour model). But also safety aspects as well as the quality of the model could be relevant properties. Therefore, in general, properties are categorised in functional (correctness or deadlock freedom) and non-functional (safety or quality) properties. In REMICS, model-based verification of functional and non-functional properties is relevant in different areas and will be exploited at different points in the overall methodology. The following list outlines a selection of relevant verification goals where the model verification is obviously relevant. This list will be adjusted and possibly extended during the definition of the REMICS migration methodology:

- Checking the reverse engineered models of the legacy system
  The reverse engineered models, which present the legacy system, need to meet certain requirements for example regarding structure and level of completeness. This can ensure that the further processing following the migration process can be conducted properly. In particular missing parts needs to be identified and maybe added in manual steps

- Checking reverse engineered specification and documentation
  The specification and documentation of the legacy system might be taken into account in the migration process. In order to do so a certain level of formalism and completeness needs to be achieved when processing the specification and documentation.

- Checking the migrated system models
  During the migration process a number of different models residing on different abstraction levels will be created. For every abstraction level certain formalisms and constraints do apply. Verification of such properties on these various system models can in particular help to identify any mistakes, derivations, or unintended decisions during the migration process.

- Checking consistency of manual rework
  Manual work may occur at any process step in the REMICS methodology. The quality of this manual work needs particular consideration and slightly different verification methods. So there might be the need to verify simple points like naming schemes as well as more difficult ones like dependency checks.

\(^{105}\) Model checker

\(^{106}\) Consistency checker
• Test Case specification
  Checking the test cases generated for the migrated system is not addressing the system model but the test models. This is important to ensure certain properties of the test cases (e.g. consistency and coverage) which will influence the quality of the migrated system.

• Checking the final system (model) before it gets actually deployed
  This is the verification point which is in particular important in order to verify certain properties for the final result of the migration process.

• Checking the running system
  The verification of system properties at run-time can be used complementary to testing. The usage of the models@runtime paradigm will facilitate these verification steps.

A specific verification goal dictates a set of verification methods which should be applied. Therefore, different technologies will be used within the REMICS approach in order to formulate and to verify certain properties. Furthermore, the formalization of properties differs from the used method. Properties can be expressed by native language as guidelines for engineers, by programming language like Java or C, by formal language such as OCL\textsuperscript{67} or Z\textsuperscript{107}, but also by proprietary languages for a specific verification tools (e.g. SPIN\textsuperscript{108}). Thus, a goal of the REMICS model-based verification approach will be to encapsulate different activities by services as well as to provide a common interface for defining and specifying properties for different verification goals and targets. A possible candidate for a common understanding is the Software Metrics Metamodel (SMM) provided by the OMG. The SMM is described later.

9.2 Metrics

Metrics are an often used means to determine software quality. Applied to several artefacts of the software during its whole life cycle, they can produce comparable evaluations of these artefacts as a basis for later assessments of several quality properties. Numerous metrics defined on code level can be found in literature. In terms of model driven development, a number of approaches to define metrics, which are useful to determine the quality of models, have been proposed in the meantime. A number of them are referenced in [Engelhardt et al.].

In many cases verification is linked with a Boolean statement. Such a verification result indicates whether or not a model satisfies certain property. But, it is sometimes also important to cover the range of possible values and to quantify to what extend a certain property is fulfilled. This can help to assess the difference of the current situation compared to the optimal one. In order to allow for this kind of assessment particular metrics needs to be defined. The quantification of properties needs to be based on measurement. While working in a model-driven migration process the used measurement techniques are related and partially overlapping with the techniques used for model based verification. Metrics can also being used for comparing properties of the legacy system and the migrated one.

9.3 Projects

9.3.1 MODELPLEX

MODELPLEX project\textsuperscript{109} targeted Model-Driven Engineering of complex software systems. Various aspects of the industrialization of Model-Driven Engineering techniques have been investigated in MODELPLEX.

\textsuperscript{67} Z
\textsuperscript{107} SPIN; http://spinroot.com/spin/whatispin.html
\textsuperscript{108} http://www.modelplex.org/
MODELPLEX worked on a combined set of different verification tools and techniques. The focus was on the development of an integrated verification workbench. Also different verification languages such as Epsilon and OCL were analyzed with respect to limitations for specific verification goals.

9.3.2 CESAR

The CESAR (Cost-efficient methods and processes for safety relevant embedded systems\textsuperscript{110}) project targets the embedded systems domain. CESAR's main objective is the reduction of cost for the development of safety-critical systems by improving the processes and methods for design decisions, analysis and V&V (Verification and Validation), reuse and change management. The CESAR Reference Technology Platform (RTP) aims to facilitate the creation of integrated development environments and tool chains for various application domains.

With respect to REMICS approach a lot of methods and tools can be reused in order to verify certain properties of the created and recovered models.

9.4 Standards

9.4.1 OCL

The Object Constraint Language (OCL\textsuperscript{67}) is a specification provided by the OMG. Primary, it was defined align with the UML specification. However, OCL can also be used for any other kind of MOF\textsuperscript{68} models.

OCL is often used to express modelling rules, guidelines and metrics for models due to its balance between formality and understandability. In contrast to other formal languages like Z or Object Z\textsuperscript{111}, OCL is widely accepted in academic and in particular also in industrial world.

A lot of implementations of the OCL specification are available. Currently in the modelling world, the most used implementation is the OCL component of the Eclipse Modeling Framework (EMF)\textsuperscript{112}. As a lessons learned from the widespread use of OCL some weak points appears. For instance, a major problem is the absence of a more workable library for specific standard functions on data types. A more structural problem is that no temporal logic can be expressed, but this wasn't an original objective of the language specification.

9.4.2 SMM

The Software Metrics Metamodel (SMM) specification\textsuperscript{113} provided also by the OMG distinguishes between measures as the evaluation process of particular quality aspects of software artefacts and measurements that can be interpreted as the results of those processes. The specification contains an extensible meta-model that primarily establishes an interchange of measurements over existing software artefacts. Those artefacts could be source code or - more interesting in the context of REMICS – models, as well. SMM contains meta-model classes for numerous types of measures and their measurements; including a set of contextual information.

The meta-model specifies several types for different outcome values of the evaluation processes (measures). The latter could assign either numeric values of a domain with a pre-defined ordering relation or numeric values representing ratios (e.g. percentages). In addition, the SMM provides appropriate classes for the mapping of values of a particular interval to a related symbol. In terms of SMM, these types of measures are called Ranking. Symbolic values like “good”, “satisfying” and “bad” can be considered as such a Ranking.

\textsuperscript{110} http://www.cesarproject.eu/
\textsuperscript{111} Object Z; http://itee.uq.edu.au/~smith/objectz.html
\textsuperscript{112} Eclipse OCL, http://www.eclipse.org/modeling/mdt/downloads/?project=ocl
\textsuperscript{113} http://www.omg.org/spec/SMM/1.0/Beta1/
REMICS can benefit from the usage of the SMM in order to specify and collect model metrics as part of the model-based verification approach. Furthermore, REMICS can enhance and contribute to the Cloud aspects to the SMM specification.

9.5 Tools and technologies

9.5.1 Static analysis

Static analysis as such is not a new topic in system engineering at all, but rather it is already commonly accepted and applied in academia as well as in the industrial world. There exists a wide range of approaches and tools on the market in order to analyze and evaluate source code or other artefacts of a system development process for instance. However, with the change from traditional code-centric development to model-driven engineering, new methods are needed. For instance the analysis of distributed models or impact analysis of a change within the development is currently a painful (cost intensive activity) situation in industrial context.

With respect to model based verification static analysis is a more generic approach and limited to static aspects of models. The dynamic aspects of models (behavioural models) are not covered by this category of verification methods. The most popular available static analysis tools for models are available within the Eclipse world. In particular these tools are working on EMF based models, for instance Epsilon\(^{114}\), ATL\(^{80}\) as well as Metrino\(^{115}\). The latter is described in one of the following sections in more detail.

9.5.2 Model checking

Model Checking is an automatic formal verification technique. The input for a formal verification tool (Model Checker) is a model (description) and a property (specification) in which the description of the property is done with a formal language. Often, these formal languages support temporal logic. A typically use case for model checking is the verification of deadlock freedom in a model.

For many years, model checking is a well-funded technique within the theoretical computer science discipline. Thus, a lot of implementations and tools are available. The important ones are Spin\(^{108}\), UPPAAL\(^{116}\) or SMV\(^{117}\).

9.5.3 Fault Tree Analysis

In order to analyse safety aspects of a critical system fault tree analysis (FTA) is an often applied verification method. An FTA is built top-down. This means that it is necessary to identify possible events that could lead to failures of the system. The description of faults is done by Boolean algebra.

In the context of Model-Driven engineering, fault description is realized as a model. The model contains a specific set of elements node types (events and gates). An FTA Event could be a failure but also a normal event, for instance. Gates are logical operators of the Boolean algebra which combine the FTA nodes to fault trees.

In terms of REMICS the FTA approach could be relevant for the Models@runtime aspects as well as for the analysis of the recovery models of the legacy system. Some significant tools are OpenFTA or FT+

\(^{114}\) Epsilon; http://www.eclipse.org/gmt/epsilon/
\(^{115}\) Metrino; http://www.modelbus.org/modelbus/index.php/metrino
\(^{116}\) UPPAL; http://www.uppaal.org/
\(^{117}\) SMV; http://www.kenmcnil.com/smv.html
9.5.4 Metrino

Metrino\textsuperscript{115} is an integrated set of tools to support the analysis, verification and quality assurance of models based on OCL and SMM. The tool additionally provides the functionality to manage and compute generated or user-defined measures and to visualize their computational results in appropriate charts. The application of the presented rule set could result in a huge number of metrics (measures). Not all of them are of the same importance and therefore an adequate and tool-supported metrics management is inevitable.

This tool is a basis for REMICS to perform the verification of all engineered models (recovery, SOA or Cloud) by using the SMM as a common interface for all verification methods and techniques.

9.6 Conclusions

In a model-driven engineering approach the quality of system depends highly on the quality of the models. In particular early error detection at design phase is essential in order to deal with the complexity of the developed systems and the verification of several system properties. In terms of the Cloud paradigm with focus on recovery and migration of legacy systems a lot of model verification methods and techniques can be applied. However, it is unclear which methods fit best for a specific property. The selection is a creative process driven by a human engineer. It might be possible that some parts of the verification steps can be automated. Therefore, the REMICS project should focus on providing templates, guidelines, best practices and sets of standard verification techniques and goals with respect to the migration process for legacy to the Cloud.

9.7 References

- R. Reißing: “Towards a Model for Object-Oriented Design Measurement.” In: Quantitative Approaches in Object-Oriented Software Engineering, (QAOOSE’01), 2001
- J.A. McQuillan, J.F. Power: “Towards the re-usability of software metric definitions at the meta level.” In: European Conference on Object-Oriented Programming, 2006
- MODELPLEX Deliverable D4.3a: “Verification by Composition”
10 Model-based testing

10.1 Introduction

In recent years Model-Based Testing (MBT) has been promoted in order to achieve high quality and high performance in the work products of the testing process steps. Figure 30 outlines the general idea of the model-based testing approach.

![Figure 30 - Model-Based Testing](image)

Similar to model-based system development, the key concept of model-based testing is to generate the test cases from a test model. Although there are a number of issues to be addressed during the test case generation (e.g., what is the test execution environment) the key question of MBT is: How to create the test model.

The test model may be derived from the requirements (similar to the system model but with a different purpose) or it can be partially derived from the system model. Both ways do have pros and cons.

In REMICS, model-based testing is relevant in different areas and will be exploited at different points in the overall methodology. The following list outlines a selection of relevant places where model-based testing is obviously relevant. This list will be adjusted and possibly extended during the definition of the REMICS migration methodology:

- **Test Migration from legacy system to the Cloud**
  The migration of tests which was originally defined for the legacy system into the model-driven engineering domain as well as with respect to the Cloud paradigm.

- **Test Recovery of implicit tests from the legacy system**
  The reverse engineering of legacy test specification as well as the recovery of implicit tests and test data is relevant for the REMICS approach.

- **Test Generation**
  Based on recovery and migrated models, a large number of tests could be generated based on the Cloud directives and the specific used Cloud platform such as Microsoft Azure or Amazon EC2.

- **Test Data Generation**
In order to enhance the quality of the Cloud-based systems it is useful to generate specific test data from the underlying models which are recovered and migrated from the legacy systems.

- **Test Execution in the Cloud**
  The system under test (SUT) is rather the cloud under test (CUT) within REMICS. Therefore, it is relevant to execute the test in a Cloud environment. This reflects also the derivation of adequate test suites for the test execution.

- **Test Reporting**
  To summarize the results of the test execution the generation of test reports is relevant within REMICS. The coverage of tests and requirements could also be a good input for the test reports.

### 10.2 Projects

#### 10.2.1 TT-Medal

The objective of the TT-Medal project[^TT-Medal] was to develop the methodologies, tools and industrial experience to enable European industry to test more effectively and more efficiently. The approach of this project was to apply TTCN3 into a wide range of application domains. The methodology and tools developed were expected to fit many domains not just telecommunications. The goal was to develop testing methods and tools that cover the whole life cycle of a product starting from the initial simulations and ending up with regression testing during the maintenance phase. The methods should enable reuse of test data both between products and between different phases of the development process. The methods enable automatic generation of tests as well as automated deployment of tests. The methods and tools were evaluated in several real industrial cases in different domains.

The methods and algorithms established in TT-Medal, in particular for test-automation and model-based testing, are of special importance for REMICS.

#### 10.2.2 D-Mint

The ITEA project D-Mint[^D-Mint] (Deployment of Model-Based Technologies to Industrial Testing) aimed on the development, enhancement, and deployment of high performance methods and tools for quality assurance of large and software-intensive systems. So the project was focused on the testing part of a development process.

The results coming from D-Mint as well as results coming from other sources are summarized in the D-Mint Common Approach, which constitutes an overall D-Mint Methodology and builds a kind of Asset Box at the same time. This Asset Box can be used in order to approach the domain of model-based testing on a practical level. The Asset Box offers methods and tools that can be chosen in order to realize the different test related process steps. Figure 31 outlines the D-Mint Asset Box.

Further details on the D-Mint Common Approach are contained in the D-Mint White Paper[^D-Mint-White-Paper]. However, there are a couple of innovations that can be listed here explicitly:

- **Architecture-Driven Testing**: A particular test derivation strategy which takes the different architectural viewpoints into account and focuses on integration specific faults
- **Pattern-oriented model-driven test engineering**: The exploitation of a pattern approach in order to facilitate the engineering of test models

[^TT-Medal]: http://www.tt-medal.org/
[^D-Mint]: http://www.d-mint.org/
• Test Management: Integration of MBT tools and methods in state of the art test management tools
• Test Quality: The assessment of the test model in the context of specific goals, guidelines and regulations
• Test Process Evaluation: The assessment and metricalation of the test process as such

![Figure 31 - D-Mint Common Approach (Asset Box)](image)

In the context of REMICS, architecture-driven testing and the pattern-oriented testing might be of special interest. It is expected that during the migration of systems into the cloud certain architectural aspects needs to be considered which at the same time apply to the general layout of the test environment and the test cases to be executed. In addition it is planned to identify and apply patterns during the migration process. This information about the patterns in the system model may apply also to the test models. Furthermore, the experiences gained in application of test methodologies, tools and technologies (e.g. UTP, TTCN-3) in various domains are of particular help in the context of REMICS.

10.2.3 RTE-Space

The RTE-Space project did some special investigation into the area of MBT in a round-trip-engineering process. The RTE-Space project in particular investigated the round-trip-engineering for a custom made distributed application.

The methods and techniques for the derivation of test specification based on system models used in RTE-Space are relevant in the REMICS context. It is important to adjust these methods for the cloud environment and for the use cases. Furthermore, the considerations of the automatic execution as initially shown in RTE-Space are very relevant and need further investigation in REMICS.

---

121 RTE-Space;
10.2.4 MODELPLEX

The MODELPLEX project worked on a test metamodel for the harmonisation of different testing tools. In addition, MODELPLEX created MBT solutions for the MODELPLEX use cases, while integrating different test tools (test case derivation and test execution).

The metamodel for testing as one of the major outcomes of MODELPLEX is in particular interesting for REMICS. It has been integrated into the UML Testing Profile. This metamodel will serve as a means for creating test models and for deriving abstract test cases and for incarnating them in the real test and test execution environment in REMICS.

10.3 Standards

10.3.1 UTP

The UML Testing Profile is a language for designing, visualizing, specifying, analyzing, constructing, and documenting the artefacts of test systems. It is a test modelling language that can be used with all major object and component technologies and be applied to testing systems in various application domains. The UML Testing Profile can be used stand alone for the handling of test artefacts or in an integrated manner with UML for a handling of system and test artefacts together."

The UML Testing Profile has been released in April 2002 in response to the OMG request for proposal ad/2001-06-08 [UTP RFP]. The specific objective of this RFP is to define a UML profile that allows capturing all needed information for black-box testing approaches to evaluate the correctness of system implementations. The profile is intended to support an effective, efficient and automated testing of system implementations according to their UML models. As solicited by the OMG’s RFP, UTP intends to fulfil three main objectives:

- It is based upon the UML meta-model,
- It enables the specification of tests for structural (static) and behavioural (dynamic) aspects of UML models, and
- It is capable of interoperation with existing test technologies for black-box testing.

UTP is based on the UML 2.0 specification and extends the UML with test-specific concepts like test component, verdicts, etc. by using the UML Profiles mechanism defined in the UML 2 specification. Profiling is not the only way to extend UML. The UML metamodel is defined via MOF; therefore, it is possible to extend it via MOF. UML extensions that use the full power of MOF are sometimes called heavyweight extensions, unlike the profile mechanism, which has been specifically defined as a lightweight extension mechanism to the UML standard.

The UTP specification provides two possibilities to extend UML: a MOF-based metamodel for testing concepts as well as test-specific stereotypes and tagged values defined in the profile. The metamodel can be used for any MOF-based testing tool. In addition to the profile definition and the metamodel the specification provides two mappings: between UTP concepts and JUnit language concepts, and between UTP concepts and TTCN-3 concepts.

In REMICS, UTP as a UML-based notation for describing test models is of high relevance since the migration process will be essentially based on UML. Furthermore, the corresponding metamodel could be of interest.

---

122 UTP; UML Testing Profile http://www.omg.org/technology/documents/profile_catalog.htm
123 UML2.0; http://www.uml.org/
10.3.2 TTCN-3

Testing has gained also much momentum with the development of the test technology TTCN-3\textsuperscript{124}. It is the Testing and Test Control Notation developed by ETSI and also adopted by ITU, which addresses testing needs of modern communication, software and embedded systems technologies. TTCN-3 is a modern and powerful test specification and test implementation language. Typical areas of application are protocol and service testing, component and system testing, testing of embedded systems, testing of communication-based distributed systems, etc. The standardized test language has a similar look and feel to a typical programming language. However, in addition to the typical programming constructs, it contains all the important features necessary to specify test procedures and campaigns for functional, conformance, interoperability, load and scalability tests. So its testing specific features makes it distinct to classical scripting or programming languages, while being technology independent. Its application in industrial-scale environments in different domains includes mobile communication, railway control systems, financial applications and telematics systems has been proven by the ITEA TT-Medal project only recently. TTCN-3 got also an entry into UML-based development processes via the definition of the UML 2.0 Testing Profile (U2TP). This enables test specifications with UML concepts, which can e.g. automatically being executed on TTCN-3 platforms.

10.4 Tools and technologies

10.4.1 TPTP

The Eclipse Test and Performance Tools Platform (TPTP)\textsuperscript{125} Project is an open source Top-Level Project of the Eclipse Foundation. TPTP is divided into four projects: TPTP Platform, Monitoring Tools, Testing Tools, and Tracing and Profiling Tools. The TPTP Platform project covers the common infrastructure in the areas of user interface; EMF-based data models, data collection and communications control, as well as remote execution environments. In addition, the platform provides the extension points for leveraging or extending the common infrastructure in solution specific tooling or test execution runtime. This includes Eclipse workbench plug-ins as well as runtime plug-ins on a target and optionally remote system.

10.4.2 TTWorkbench

The TTWorkbench\textsuperscript{126} is a commercial tool from Testing Technologies. It is an integrated test development and execution environment based on the Eclipse framework. It supports the definition, management and execution of test suites. It comes with a graphical editor and supports the TTCN-3 ETSI specification.

In the context of REMICS this tool can be used to define and run the tests expressed with the TTCN-3 language.

10.5 Conclusions

The validation of a system is crucial, also in a Cloud infrastructure. However, model-based testing for the Cloud is a new research topic. REMICS has to find solutions for how to design a specific test model for the Cloud domain. Also the recovery of test specification, in particular the recovery of implicit test specification, could be quite challenging within REMICS. Nevertheless, the model-based testing approach is well founded and a lot of established methods could be applied within the REMICS project with respect to recovery and migration of a legacy system into the Cloud.

\textsuperscript{124} http://www.ttcn-3.org/
\textsuperscript{125} http://www.eclipse.org/tptp/
\textsuperscript{126} http://www.testingtech.com/products/ttworkbench.php
10.6 References


- MODELPLEX Deliverable D4.4.a: Testing Metamodel
11 Summary and conclusions

In this document we have analysed state of the art in the technological areas that are relevant to the REMICS research. Each section provided an analysis of related projects, products, standards and approaches in addition to conclusions and pointing out shortcomings and ideas for REMICS research. The research of REMICS is exploratory and iterative and will take advantage of current work as far as possible, with improvements, additions or new innovations where there are gaps. REMICS work packages will use the findings of this deliverable in their work in the initial phase of the project to further analyse technologies and choose baseline for REMICS R&D and standardization efforts.