



<http://mature-ip.eu>

D5.2

Specification of the System Architecture (DRAFT)

Date	12. April 2009
Dissemination Level	Public
Responsible Partner	BOC
Editors	Andrea Leutgeb
Authors	Andrea Leutgeb, Robert Woitsch, Hannes Eichner, Daniela Feldkamp, Roman Brun, Simone Braun, Nicolas Weber, Alexander Sandow, Tobias Nelkner, Knut Hinkelmann, Barbara Thönssen, Bo Hu
Work Package	WP5 (System Architecture, Integration and Deployment)

MATURE

Continuous Social Learning in Knowledge Networks

<http://mature-ip.eu>

Grant No. 216356

*MATURE is supported by the European Commission within the 7th Framework Programme, Unit for Technology-Enhanced Learning
Project Officer: Martin Májek*

DOCUMENT HISTORY

Version	Date	Contributor	Comments
V0.1	09.12.2009	Andrea Leutgeb	Initial Document
V0.2	23.01.2009	Andrea Leutgeb	First Draft
V0.3	02.03.2009	Andrea Leutgeb	Restructuring of Documents, Draft for Internal Review
V0.4	09.03.2009	Andrea Leutgeb, Wilfrid Utz, Robert Woitsch Daniela Feldkamp, Roman Brun , Simone Braun, Nicolas Weber, Alexander Sandow, Tobias Nelkner	Incorporated results of the Integration Meeting
V0.5	13.03.2009	Andrea Leutgeb Knut Hinkelmann, Barbara Thönssen, Roman Brun, Bo Hu	Incorporated Internal Review Results
V0.6	02.04.2009	Andrea Leutgeb, Robert Woitsch, Hannes Eichner	Restructuring and Improvement of Deliverable
V0.7	10.04.2009	Andrea Leutgeb	Finalisation of the Deliverable
V1.0	12.04.2009	Andreas Schmidt	Final editorial work
	15.04.2009	Andreas Schmidt Pablo Franzolini	Submission to the EC

Table of Contents

1 EXECUTIVE SUMMARY	11
2 INTRODUCTION	12
2.1 Introduction to this Deliverable.....	12
2.2 Integration Philosophy	13
2.3 Structure of this Deliverable	14
3 CONCEPTUAL BACKGROUND OF THE SYSTEM ARCHITECTURE	16
3.1 MATURE Specific Concepts.....	16
3.1.1 <i>Process Orientation</i>	16
3.1.2 <i>Model Orientation</i>	17
3.1.3 <i>Knowledge Management vs. Knowledge Work</i>	17
3.2 State of the Art Concepts	19
3.2.1 <i>Service Orientation and Virtualisation</i>	19
3.2.2 <i>Semantics</i>	23
3.2.3 <i>Web 2.0</i>	25
3.2.4 <i>Security and Trust</i>	27
4 MATURE ARCHITECTURE OVERVIEW	29
4.1 Bottom-Up View on the MATURE System.....	29
4.1.1 <i>Introduction to the Bottom-Up Approach – The Rapid Prototyping Approach</i>	30
4.1.2 <i>Service Collection</i>	31
4.1.3 <i>Integration Scenario</i>	34
4.2 Top-Down View on the MATURE System	40
4.2.1 <i>Introduction to the Top-Down Approach – The Architecture View Model Approach</i>	40
4.2.2 <i>Logical View on the MATURE Architecture</i>	41
4.2.3 <i>Process View on the MATURE Architecture</i>	43
4.2.4 <i>Development View on the MATURE Architecture</i>	48
4.2.5 <i>Physical View on the MATURE Architecture</i>	48
4.2.6 <i>Scenario View on the MATURE Architecture</i>	50
5 KNOWLEDGE BUS AS INTEGRATION TOOL	58
5.1 Knowledge Bus Integration Layer	59
5.1.1 <i>Conceptual View on the Knowledge Bus Integration Layer</i>	59
5.1.2 <i>Implementation View on the Knowledge Bus Integration Layer</i>	63
5.2 Knowledge Bus Infrastructure Layer	64
5.2.1 <i>Conceptual View on the Knowledge Bus Infrastructure Layer</i>	64
5.2.2 <i>Implementation View on the Knowledge Bus Infrastructure Layer</i>	66

6 SUMMARY AND OUTLOOK	75
6.1 Outlook to the Further Procedure in WP5.....	75
7 REFERENCES	77
ANNEX A DESIGN STUDIES - INTEGRATION RELEVANT ASPECTS	81
Annex A.1 Design Study “DS1: OLMEWiki” Service Collection.....	82
Annex A.2 Design Study “DS2: Dialogue Games for Ontology Maturing” Service Collection.....	84
Annex A.3 Design Study “DS3: Interacting Widgets” Service Collection	85
Annex A.4 Design Study “DS5: OLMEntor” Service Collection	87
Annex A.5 Design Study “DS6: APOSDLE” Service Collection.....	91
Annex A.6 Design Study “DS7: Kasimir” Service Collection.....	92
Annex A.7 Design Study “DS8: SOBOLEO” Service Collection	93
ANNEX B SERVICE FACT SHEET	98
ANNEX C MATURE MESSAGE MODEL	103
ANNEX D KNOWLEDGE ITEM META DATA	107

List of Figures

Figure 1: WP5 – Objectives and Dependencies	12
Figure 2: The MATURE Integration Philosophy	14
Figure 3: Overview of the Structure of D5.2 “Specification of the System Architecture”	15
Figure 4: Knowledge Work as Industrial Process	18
Figure 5: Interaction of Service-Oriented Architecture (SOA) Principles	20
Figure 6: Web-Services Architectural Model	21
Figure 7: ESB Architecture	22
Figure 8: The Virtual Organisation Lifecycle.....	23
Figure 9: Illustration of the WSDL-S Approach (after (Moran et al, 2005))	25
Figure 10: Mashup - Example “Housingmaps.com”	26
Figure 11: Hybrid Approach to the MATURE System Architecture Design	29
Figure 12: Bottom-Up View on the System Architecture	30
Figure 13: Overview of the Rapid Prototyping Approach (Bijay, 2006)	31
Figure 14: Bottom-Up Approach on the System Architecture – Integration Scenario.....	35
Figure 15: Implementation of the Integration Scenario at the 1 st Technical Partner Meeting	39
Figure 16: Top-Down View on the System Architecture	40
Figure 17: The MATURE System Architecture Design Method	41
Figure 18: Logical View on the Knowledge Bus Architecture	42
Figure 19: Process View on Knowledge Modelling.....	43
Figure 20: Process View on Service Registering	44
Figure 21: Process View on Workflow Design	45
Figure 22: Process View on Meta Data Management	45
Figure 23: Process View on Ontology Management	46
Figure 24: Process View on Service Provisioning	47
Figure 25: Process View on Monitoring and Administration.....	47
Figure 26: Development View on the Knowledge Bus Architecture	48
Figure 27: Physical View on the Knowledge Bus Architecture - Possible Integration at an Application Partner	49
Figure 28: Deployment of the MATURE System in a Trusted Environment	50
Figure 29: Knowledge Modelling Scenario.....	52
Figure 30: Semantic Service Registry Scenario	53
Figure 31: Workflow Modelling Scenario.....	54
Figure 32: Ontology Management Scenario.....	55
Figure 33: Meta Data Management Scenario	56
Figure 34: Service Provisioning Scenario	56
Figure 35: Administration and Monitoring Scenario.....	57
Figure 36: Layers of the Knowledge Bus	58

Figure 37: The Integration Layer of the Knowledge Bus.....	59
Figure 38: Knowledge Dimensions of PROMOTE®	61
Figure 38: From the Conceptual to the Technical Level	62
Figure 39: Integration Patterns	63
Figure 40: Model-Based Knowledge Management Design Framework	64
Figure 41: The Infrastructure Layer of the Knowledge Bus.....	65
Figure 42: jBoss ESB Architecture (jBoss ESB, 2009).....	67
Figure 43: ESB – Direct Transformations between Service Consumers and Providers.....	68
Figure 44: ESB – Integration of Applications and Services in MATURE.....	69
Figure 45: Implementation View on the Semantic Service Registry and Discovery.....	71
Figure 46: BPEL Meta-Model (ebPML BPEL, 2009).....	72
Figure 47: Schematic Representation of the Hierarchy of Elements in the LOM Data Model (LOM, 2009).....	73
Figure 48: Further Procedure in WP5 – From the Initial Prototype to the MATURE System.....	76

List of Tables

Table 1: Comparison Traditional vs. Semantic Web (adapted from (Solazzo et al, 2002))	24
Table 2: Service Integration Template for the Design Studies	33
Table 3: Soboleo Service - Overview	36
Table 4: Rule Engine Service - Overview	36
Table 5: Maturing Service - Overview	37
Table 6: Data Persistence Service - Overview.....	38
Table 7: UML Use Case Diagram Notation	51
Table 8: Knowledge Item Meta-Data - Knowledge Item Tag.....	73
Table 9: Service Integration Template for the Design Studies	81
Table 10: Design Study “DS1: OLMEWiki” Service Collection.....	82
Table 11: Design Study “DS2: Dialogue Games for Ontology Maturing” Service Collection.....	84
Table 12: Design Study “DS3: Interacting Widgets” Service Collection.....	85
Table 13: Design Study “DS5: OLMEntor” Service Collection	87
Table 14: Design Study “DS6: APOSDLE” Service Collection	91
Table 15: Design Study “DS7: Kasimir” Service Collection	92
Table 16: Design Study “DS8: Soboleo” Service Collection	93
Table 17: WSDL implementing the MATURE Message Model (First Version).....	103
Table 18: Knowledge Item Meta-Data - General Tag	107
Table 19: Knowledge Item Meta-Data - Lifecycle Tag.....	108
Table 20: Knowledge Item Meta-Data – Meta-Metadadata Tag.....	110
Table 21: Knowledge Item Meta-Data - Technical Tag	111
Table 22: Knowledge Item Meta-Data - Rights Tag	112
Table 23: Knowledge Item Meta-Data - Relation Tag	112
Table 24: Knowledge Item Meta-Data - Annotation Tag.....	113
Table 25: Knowledge Item Meta-Data - Classification Tag.....	114

List of Abbreviations

AAI	Authorization and Authentication Infrastructures
API	Application Programming Interface
BPM	Business Process Management
BPMS	Business Process Management System
DAML-S	DARPA Agent Markup Language for Services
D	Deliverable
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
IdP	Identity Provider
IRS	Internet Reasoning Service
KM	Knowledge Management
KMP	Knowledge Management Process
KMS	Knowledge Management System
LMI	Labour Market Information
MMM	Mature Message Model
MOF	Meta-Object Facility
OASIS	Organization for the Advancement of Structured Information Standards
OCML	Operational Conceptual Modelling Language
OLME	Organisational Learning and Maturing Environment
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Web-Services
P.A.	Personal Advisor
PLME	Personal Learning and Maturing Environment
PM	Project Month
REST	Representational state transfer
RPC	Remote Procedure Call
SAWSDL	Semantic Annotations for WSDL
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOTA	State of the Art
SP	Service Provider
UDDI	Universal Description, Discovery and Integration
WS	Web-Service
WSDL	Web-Service Description Language
WSDL-S	Web-Service Semantics
WSMO	Web-Service Modeling Ontology



WP	Work Package
WP1	Work Package 1: State of the Art, Empirical Analysis & Conceptual Model
WP2	Work Package 2: Personal-to-Community
WP3	Work Package 3: Community-to-Organisation
WP4	Work Package 4: Maturing Services
WP5	Work Package 5: System Architecture, Integration and Deployment
WP6	Work Package 6: Evaluation
XML	eXtensible Markup Language
XSD	XML Schema Definition Language

1 Executive Summary

The present document Deliverable D5.2 (D5.2: Specification of the System Architecture) of the MATURE project (Continuous Social Learning in Knowledge Networks, Grant Agreement no.: 216356) has been prepared under WP5 “System Architecture, Integration and Deployment” and is one result of task 5.1 (“System Architecture Design”). According to the MATURE Description of Work at this point in time it is in a DRAFT status. In project month 18 the final version of this deliverable will be submitted.

This task aims to design the overall MATURE system architecture, which has to integrate various components: wrapped services of already existing functionalities in WP2 and WP3, the personal learning and maturing environment (PLME) in WP2, the organisational learning and maturing environment (OLME) in WP3, the Maturing Services in WP4, as well as co-existing knowledge sources at the MATURE application partners. Central component for this integration is the Knowledge Bus which acts as a middle tier between the various services and knowledge sources.

Based on the integration philosophy covering MATURE specific concepts Process Orientation, Model Orientation, Knowledge Management vs. Knowledge Work and on current SOTA (Service Orientation and Virtualisation, Semantics, Web 2.0, Security and Trust), the system will be analysed following a bottom-up and top-down approach.

Rapid Prototyping is applied to analyse existing services and tools for their applicability and to build a first prototype to identify requirements for the MATURE system architecture. From top-down the architecture view model will be applied to analyse the Knowledge Bus (as the integration layer) from the view points of different stakeholders (end user, programmer, integrator, and system engineer). The Knowledge Bus is analysed from a high level to identify the necessary components for registry, discovery and invocation of services as well as the registration of knowledge sources.

The Knowledge Bus as Integration Tool is further specified by describing its layers both from a conceptual and from an implementation view. Amongst others this includes the specification of the service description, so that services can be discovered, and the specification of knowledge items, so that knowledge items as the smallest data elements, can be exchanged between sources and services. The Knowledge Bus described in this deliverable aims to act as an integration layer of a service-oriented architecture providing an uniform interface for accessing various sources and for registry, discovering and invoking of services as well as messaging between them.

This deliverable concludes with an outlook on the further procedure within this work package.

2 Introduction

2.1 Introduction to this Deliverable

The present deliverable D5.2 (“Specification of the System Architecture”) has been prepared under WP5 (“System Architecture, Integration & Deployment”) and is result of task 5.1 (“System Architecture Design”). The objective of this task is to design the overall architecture of the MATURE system. Central component for the integration is the Knowledge Bus which acts as a middle tier between the various knowledge sources, the wrapped services, the two learning and maturing environments developed in WP 2 (PLME) and WP 3 (OLME) and the maturing services developed in WP 4.

Figure 1 highlights WP5’s dependencies with other work packages. As the figure depicts WP1 provides input (indicated by arrow 1) in the form of empirical studies, the conceptual maturing model as well as the analysis of the state of the art.

WP5 focuses on the integration, whereas the actual specification and implementation of the services is considered in WP2 for PLME services (indicated by arrow 2), WP3 for OLME services (indicated by arrow 3) and WP4 for Maturing services (indicated by arrow 4). These work packages provide input in the form of service descriptions that indicate support needed from the MATURE system. Later the MATURE system will be integrated at the application partners’ sites to show the applicability and success of MATURE in a real world environment. Then also existing enterprise systems which are used at the MATURE application partners will be integrated through WP5 (indicated by arrow 5).

The developed infrastructure test bed as well as the deployed system will be evaluated in WP6 (indicated by arrow 6) focusing on the effectiveness of the system in promoting learning tailored to the needs of the user.

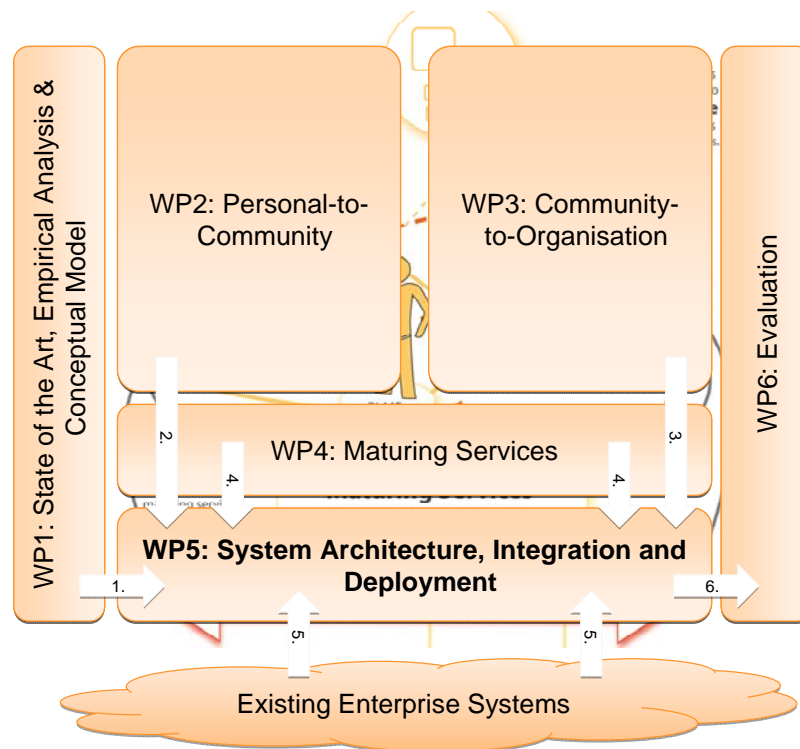


Figure 1: WP5 – Objectives and Dependencies

The deliverable at hand is seen as a DRAFT for the specification of the system architecture. According to the DoW (MATURE DoW, 2007) the final version of this deliverable will be provided in PM18, taking into account the results and the final service specifications of WP2, WP3 and WP4 and the preliminary results of the infrastructure test bed (T5.2).



2.2 Integration Philosophy

This section introduces the integration philosophy, which is used to demonstrate the underlying principles for the MATURE system. Figure 2 provides an overview of the integration principles that will be introduced in the following.

In order to gather requirements empirical studies were realized in WP1 (see D.1.1 (MATURE D1.1, 2009) for a detailed description) including ethnographic studies at all application partners, i.e. Careers Scotland, Connexions Kent and Structuralia. Studying the situation at the application partners is considered very important for driving the requirements process. Within these studies processes and people were observed. The focus was on persons who had to be primarily engaged in knowledge work, i.e. an ideal type of work, an abstraction comprising key characteristics of a wide array of activities in organizations across occupations that creates, translates or applies new knowledge. One of the findings was that maturing is made up of a complex pattern of individual steps, the so-called knowledge maturing process. This points out that a *Process Oriented Approach* has to be followed to define integration sequences and therefore enable a flexible approach for service integration.

To support Knowledge Managers or subject matter experts an OLME will be developed during the project enabling them e.g., to analyze the current state of organisational learning in terms of contents, semantics and processes, to take up and reuse results of community activities, and to apply breeding strategies to topics or communities identified as relevant. This environment will help them to guide maturing activities towards organizational goals. Individual Knowledge Workers use their PLME, which is embedded into the working environment, to engage in maturing activities within communities and beyond. Therefore we should clearly distinguish between *Knowledge Management vs. Knowledge Work*.

The MATURE system is very complex as it involves various actors and has to integrate various technologies. Integration and maturity requires a common understanding. Modelling and *Model Orientation* became commodity in system architecture and are seen as an enabler for a common understanding. Models are representations of a selected portion of the perceived reality of an individual or a group of observers. They have many purposes, e.g. to facilitate human understanding, communication, organisational learning and transfer of know-how. This is achieved because models are understandable by humans. This model oriented approach was already followed in WP1 to gather the results of the ethnographic studies, see D1.1 (MATURE D1.1, 2009) for further details on the results.

Following a rapid prototyping approach (as pointed out in the DoW (MATURE DoW, 2007)) the tools available at the technical partners have to be analysed for their applicability to support the MATURE end user. Furthermore new services will be developed for the PLME and OLME and to support maturing and have to be integrated into the system. Therefore a *Service Oriented Approach* will be followed for the system architecture design. Following a Service Oriented approach available tools have to be provided as services, thus *Virtualisation* will be applied to provide tool functionality as a service. Also human services can be integrated following the virtualisation approach.

Current research challenge is to provide a proper conceptual framework in order to semantically describe services on different levels of granularity. MATURE contributes to this research by investigating in the semantic description of knowledge services. *Semantic* technology introduces intelligent mechanisms into service oriented systems. The vision of the Semantic Web is to make content machine interpretable, hence it is not only the human that generates and interprets content but also machines. MATURE considers such techniques when realising mechanisms to configure and orchestrate the system for instance when applying semantic service discovery.

The MATURE infrastructure should facilitate openness and easy adoption of the system for both user and service provider. The tools that will be integrated make use of *Web 2.0* technologies. Therefore some *Web 2.0* technologies (e.g. Ajax, Mashups or Widgets) should be taken into account when building the MATURE system.

During the project the MATURE system will be integrated at the application partners' sites in order to prove the applicability of the implemented system in real-world environments. Then, at the latest, *Security and Trust* will play a major role as access to sensitive or important information needs to be protected.

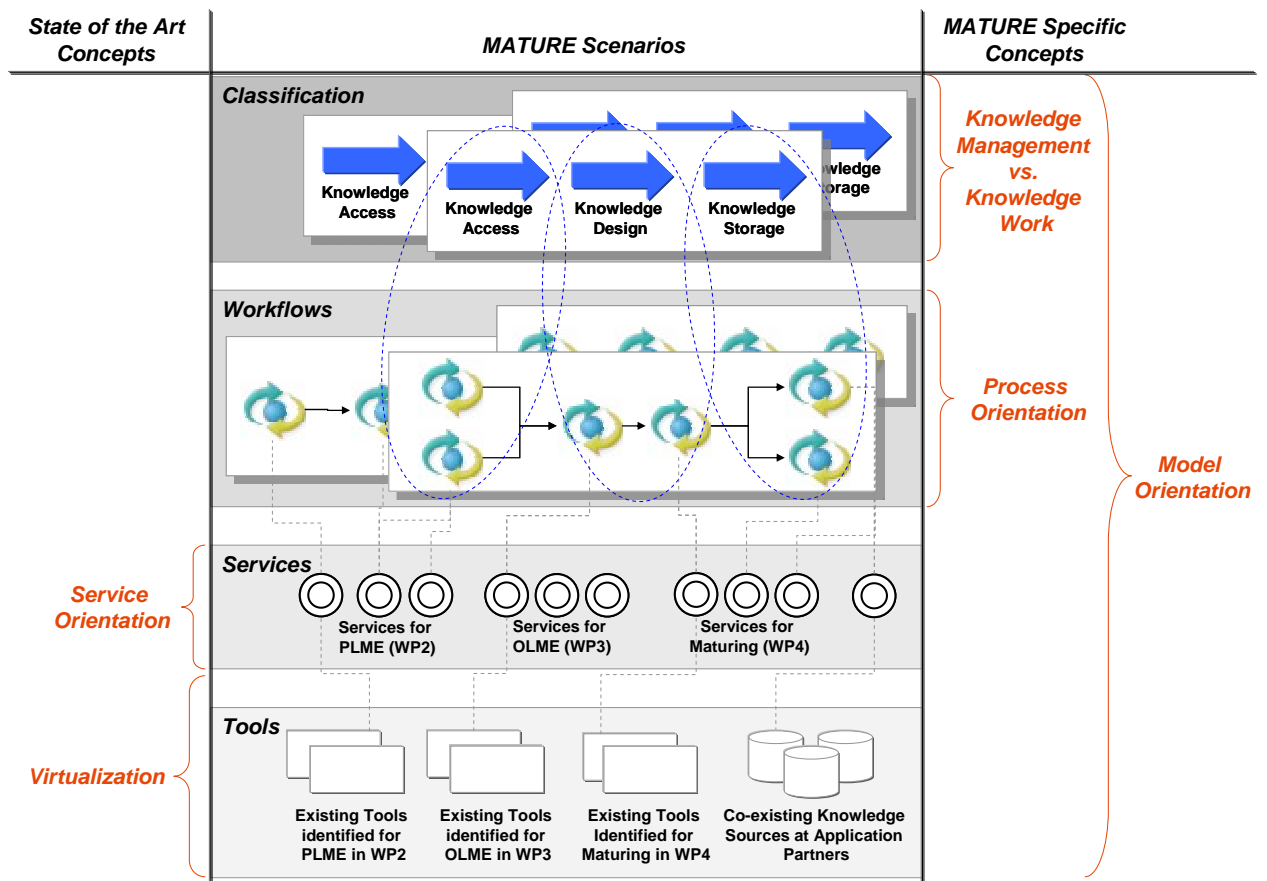


Figure 2: The MATURE Integration Philosophy

This section provided a first overview of the underlying principles for the MATURE system, namely Process Orientation, Model Orientation, Knowledge Management vs. Knowledge Work, Service Orientation and Virtualisation, Semantics, Web 2.0 and Security and Trust. As the above figure depicts it can be differentiated between MATURE specific and SOTA concepts. MATURE specific concepts are concepts that have to be taken into account when designing the MATURE system architecture and are derived from the empirical studies in WP1. SOTA concepts are current state of the art in system design and result form the state of the art analysis. All introduced principles will be discussed in more detail in Chapter 3.

2.3 Structure of this Deliverable

This section introduces the structure of the present deliverable. Figure 3 depicts the main chapters and their relationships.

After this chapter presented the role of WP5 within the project and the underlying integration philosophy for the MATURE system architecture, Chapter 3 lays down the conceptual background. It introduces the relevant MATURE specific concepts and the current state of the art concepts and technologies. The concepts process orientation, model orientation, service orientation, semantics, Web 2.0 and security and trust, which are seen as the foundation for the MATURE architecture, will be described.

Chapter 4 introduces the overall MATURE System Architecture. To specify the MATURE system architecture a bottom-up and a top-down approach was followed. From bottom-up a rapid prototyping approach is followed to analyse existing service within the consortium for their applicability within MATURE. From the top-down view the system is described from multiple, concurrent views. These views will be used to describe the system from the viewpoint of different stakeholders, such as end-users, developers or project managers.

Chapter 5 is dedicated to the Knowledge Bus, which will act as the middle tier of the architecture in order to integrate the various knowledge sources, services, the two learning and maturing environments (WP2 and WP3) and the Maturing services (WP4). This chapter provides details on the different layers of the Knowledge Bus from a conceptual and from an implementation point of view.

Finally this deliverable concludes in Chapter 6 with a summary and outlook on the further procedure within this work package.

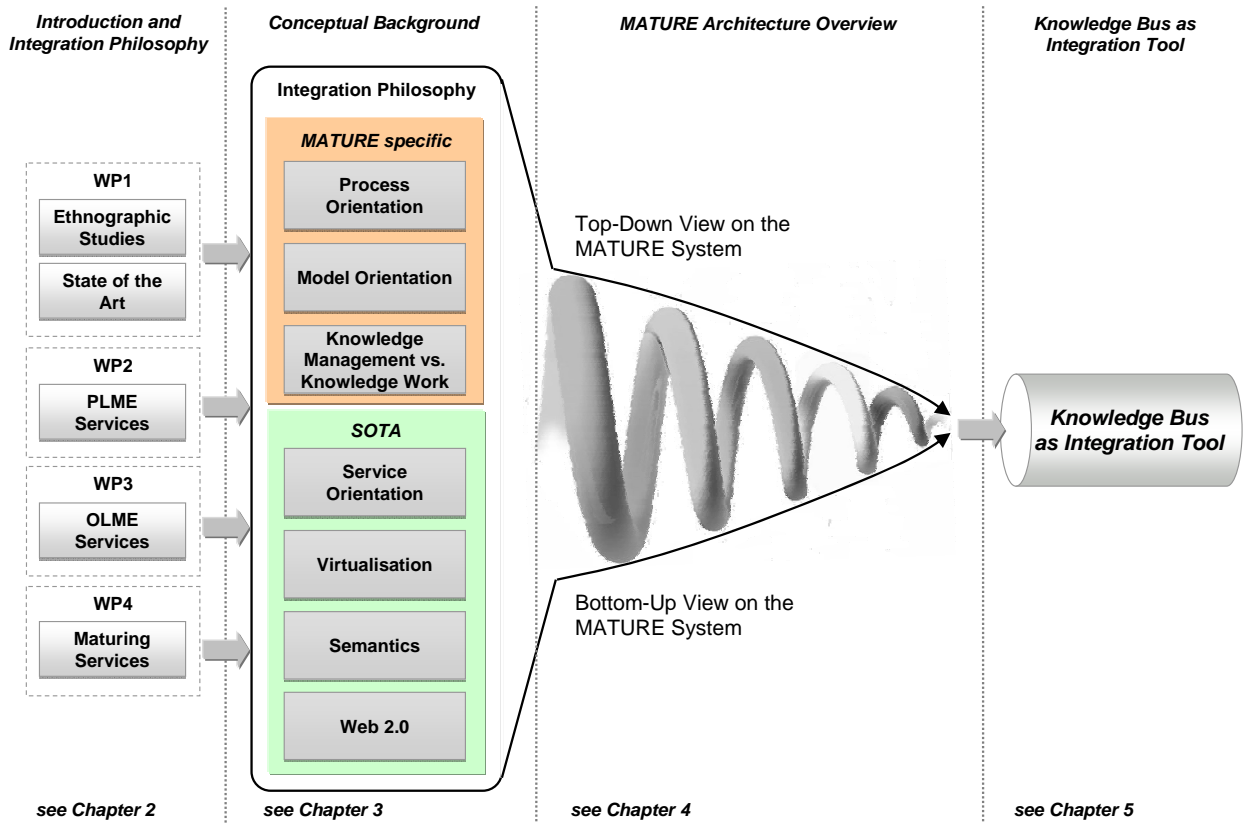


Figure 3: Overview of the Structure of D5.2 "Specification of the System Architecture"

3 Conceptual Background of the System Architecture

This chapter aims to lay down the conceptual background that needs to be considered when designing the architecture for the MATURE system. This chapter is based on the integration philosophy as introduced in section 2.2, where a first overview of the underlying principles for the MATURE system, namely Process Orientation, Model Orientation, Knowledge Management vs. Knowledge Work, Service Orientation and Virtualisation, Semantics, Web 2.0 and Security and Trust, was provided. These principles will be discussed in more detail in the following.

These concepts will be separated in MATURE specific concepts and State of Art concepts influencing the system architecture. MATURE specific concepts are Process Orientation, Model Orientation and Knowledge Management vs. Knowledge Work, whereas the relevant State of the Art concepts are Service Orientation and Virtualisation, Semantics, Web 2.0 and Security and Trust.

3.1 MATURE Specific Concepts

This section introduces the MATURE specific concepts Process Orientation, Model Orientation and Knowledge Management vs. Knowledge Work. These concepts were identified looking at the empirical studies conducted in WP1 (see D1.1 (MATURE D1.1, 2009) for details).

3.1.1 Process Orientation

Nowadays the significance of process orientation is widely acknowledged. Almost every organisation deals with process oriented concepts e.g. business process reengineering, business process modelling, business process optimisation, total quality management or implementation of process oriented software. The integration of KM activities into the organization's business processes is an important factor as an effective and efficient handling of knowledge requires it to be part of the organization's daily routine. Also maturing is made up of a complex pattern of individual steps, the so called knowledge maturing process. Following a process oriented approach, knowledge can be offered to the Knowledge Workers in a well-targeted way.

Process Oriented Knowledge Management (POKM) is built upon facts that (Hinkelmann, et al, 2002) knowledge has to be embedded in the business process and knowledge processes can be modelled. The POKM approach is specified by the following views on processes:

1. The first level in the POKM is covered by simple BPM. Here the business process is seen as content, and the graphical representation in combination with a textual description is seen as making implicit organisational knowledge about working procedures explicit. Usually this content is stored in Web-Based repositories like providing a Web-documentation of the business processes, but can also be stored in content management systems or in organisational handbooks.
2. The second level sees the aforementioned business process as a starting point and integration platform for the KM system. In this case the graphical representation of the business process is used to analyse knowledge intensive activities and to gain a common understanding, where valuable knowledge is created and where it is required. Similar to the Model Driven Architecture (MDA), the business process is seen as the starting point for requirements that need to be fulfilled by the knowledge management system. The "integration platform" view sees the process as the as a mediator between knowledge services, ontologies and information repositories to finally fulfil the needs of the business process
3. The third level interprets the process as a management approach, thus defining the sequence of performed knowledge management activities as a knowledge management process (KMP). The key difference as opposed to business processes is that KMP's are mainly domain-independent and deal with knowledge identification, knowledge accessing, knowledge usage, knowledge storage and knowledge distribution.

After this phase has introduced the first MATURE specific concept “Process Orientation”, the following section will focus on Model Orientation.

3.1.2 Model Orientation

Often the reality is too complex and not reproducible for third parties. This is also true for the MATURE system, which involves various actors and has to integrate various technologies. Carefully crafted models seem to be able to overcome this drawback. Models are representations of a selected portion of the perceived reality of an individual or a group of observers. They have many purposes, e.g. to facilitate human understanding, communication, organisational learning and transfer of know-how. This is achieved because models are understandable by humans. In the year 1973 Stachowiak (Stachowiak, 1973) introduced a comprehensive concept of "model" that can be used by all disciplines.

According to the author a model has the following characteristics:

- Mapping feature: A model is based on an original.
- Reduction feature: A model only reflects a (relevant) selection of the original's properties.
- Pragmatic feature: A model needs to be useable in place of the original with respect to some purposes.

More information on modelling theory can be found in (Stachowiak, 1973) and (Kühne, 2005).

Modelling is one of the key tasks that helps on the one hand to understand, analyze and improve business processes, organizational structures in general and structures and processes of KM initiatives in particular while on the other hand, modelling supports the design, implementation and management of information systems, in our case of the MATURE system.

In MATURE modelling will be applied on several levels to get a grip on the complexity of the MATURE system:

- Knowledge Modelling: To gather the requirements for the business oriented end user (the MATURE application partners)
- Ontology Modelling: For the alignment of the business oriented requirements (Knowledge Modelling) and the technical realisation (Service Modelling) an ontology will be designed. There are various ontology representation languages, whereas the most prominent one is the Web Ontology Language (OWL) (W3C OWL, 2009).
- Service Modelling: The services that will be integrated into the MATURE system will be described and registered in order to be found by an end user.

3.1.3 Knowledge Management vs. Knowledge Work

As highlighted in section 2.2 the MATURE system has to support the Knowledge Manager and the Knowledge Worker in their daily work. This section aims to differentiate between the two concepts Knowledge Management and Knowledge Work.

The transformation of society and economy into a knowledge society and a knowledge economy has substantially changed the work places of the majority of employees. To describe such a phenomenon, the term knowledge work was coined by Drucker (Drucker, 1973). Knowledge work can be characterized as follows (Maier, 2007):

- *Target:* Knowledge work solves ill-structured problems in complex domains with a high degree of variety and exceptions.
- *Content:* Knowledge work is creative work that requires creation, acquisition, application and distribution of knowledge and bases inputs and outputs primarily on data and information.

- *Mode of Work:* Knowledge work consists of a number of specific practices, for example creating new knowledge, interpreting, integrating, representing, retaining and securing it, producing and reproducing knowledge, expressing or extracting experiences or networking with other people.
- *Personal Skills and Abilities:* Knowledge work requires intellectual abilities and specialized knowledge rather than physical abilities. This requires a high level of education, training and experiences resulting in skill and expertise.
- *Organization:* Knowledge work is often organized in a decentralized way. It has strong coordination, communication and cooperation needs and is highly mobile, flexible and distributed.
- *ICT:* Knowledge work requires a strong but flexible support by information and communication technologies.

By contrast Knowledge Management is defined as “*the management function responsible for the regular selection, implementation and evaluation of goal-oriented knowledge strategies that aim at improving an organization’s way of handling knowledge internal and external to the organization in order to improve organizational performance. The implementation of knowledge strategies comprises all person-oriented, organizational and technological instruments suitable to dynamically optimize the organization-wide level of competencies, education and ability to learn of the members of the organization as well as to develop collective intelligence*” (Maier, 2007).

The Application of the Knowledge Management to steer the Knowledge Work conducted in the enterprise has become a commodity today. It is used across many different industries and applied in many different scenarios and use cases. Similar to a conveyer belt the knowledge work carried out daily in a company should be transformed into an industrial process. In such a scenario the Knowledge Worker utilizes the conveyer belt to tackle the assigned tasks and based on the configuration of the conveyer belt, defined by the Knowledge Manager, receives support by being provided with appropriate knowledge services and knowledge sources for this specific task. The Knowledge manager is responsible for the configuration of the conveyer belt- Based on the experience of the Knowledge Worker appropriate knowledge services and knowledge sources are selected at the execution time based on the parameters set by Knowledge Worker and/or Knowledge Manager.

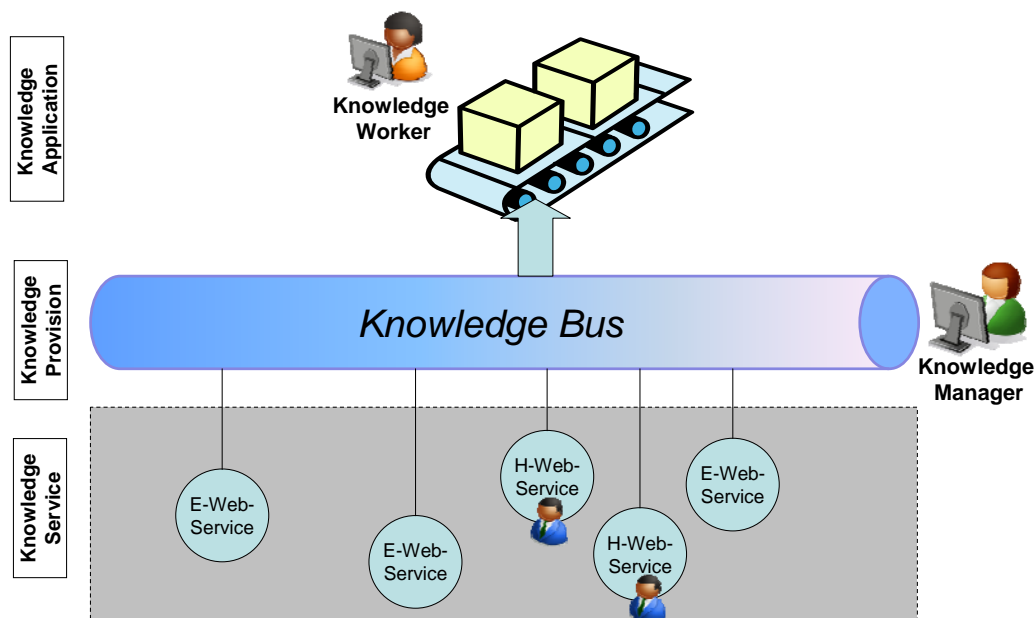


Figure 4: Knowledge Work as Industrial Process

Figure 1 provides an overview of a conveyer belt implementation using a three layer architecture: The conveyer belt – providing Knowledge Workers with required knowledge services (KM tools and knowledge sources), the Knowledge Bus – allowing the configuration of the conveyer belt, and



orchestration of the available knowledge services (human and electronic services) based on the KM services, and Virtual Knowledge Organisation, which repository containing all available knowledge services. The Knowledge Bus will be the central component of the MATURE system architecture and will be described in detail in Chapter 5.

3.2 State of the Art Concepts

After the previous section introduced concepts derived from the empirical studies in WP1 that are specific for the MATURE system, this section will introduce the current State of the Art (SOTA) that lays down the conceptual and technological background which has to be taken into account for the system specification.

3.2.1 Service Orientation and Virtualisation

3.2.1.1 Service Oriented Architecture

The OASIS article “Reference Model for Service-Oriented Architecture” defines Service Oriented Architecture (SOA) as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” (OASIS, 2006).

SOA is considered to be a perspective of the software architecture which is used to support needs and requirements of the software users by defining the usage of the loose coupling software. According to (Erl, 2005) loose coupling “is a condition wherein a service acquires knowledge of another service while still remaining independent of that service”. Although SOA and Web-Services are built on similar principles, it is important to realise that they are not the same: SOA is considered to be more than just a set of technologies and is in fact able to run independently without any specific technology, meaning that it can be implemented/described using any of the interoperability standards (e.g. WDSL).

The evolution of SOA can be described as follows:

1. Traditional point-to-point architecture: Different services and components know everything about the API below and above and can directly access them.
2. Service-oriented architecture: All services on different levels and layers are available via the Internet/Intranet. One service takes over controlling responsibility for retrieval and coordination.
3. BPEL and/or WS-Coordination: All services on different levels and layers are available via the Internet/Intranet. A well-defined process/workflow takes over controlling responsibilities.

The main SOA principles include (Erl, 2005): Service reusability, Service contract, Service loose coupling, Service abstraction, Service composability, Service autonomy, Service statelessness and Service discoverability. The interaction of the SOA principles is shown in Figure 5.

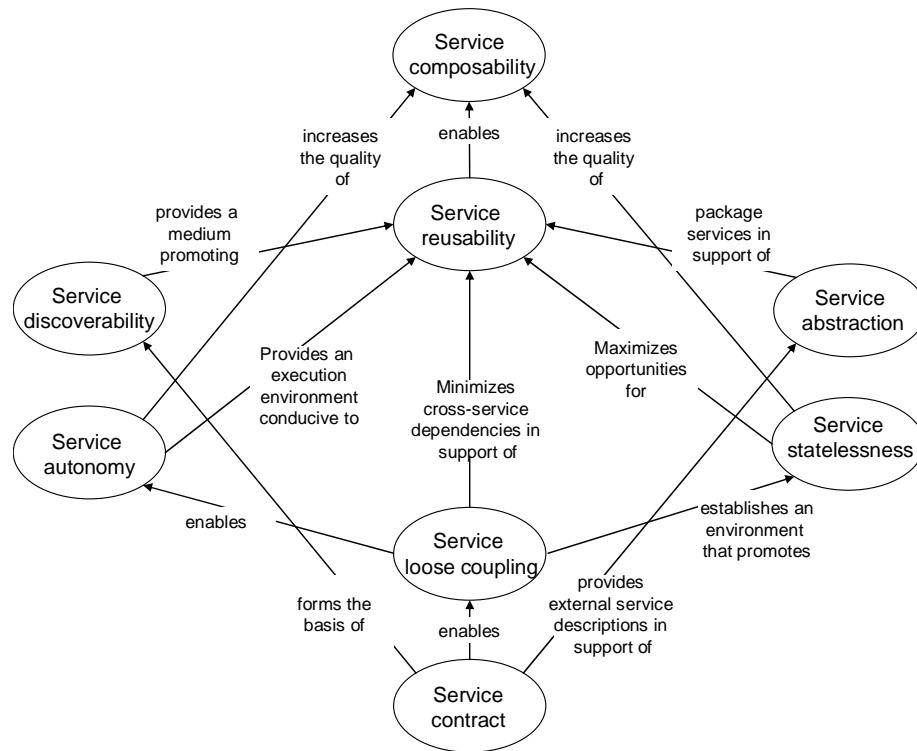


Figure 5: Interaction of Service-Oriented Architecture (SOA) Principles

Various benefits can be identified for applying the aforementioned SOA principles. The major ones are (Stevens, 2009): Better return on investment due to reuse of previously developed components that are used as services; Code mobility since the client doesn't have to be concerned where the service is located; Additional security due to multiple authentication, at both the client and at the service end; Support for multiple client types and enhanced scalability as the middleware (load balancer) can forward the requests to multiple instances of the service, or in case that one server goes offline re-route the requests to others (improved availability).

In contrast to traditional architecture, where the components of a system know about existing interfaces and how to access them, in a SOA environment all services exist as loosely coupled, highly interoperable application services and are independent of the underlying platform and programming language. SOA is a conceptual and technology-independent concept on how to design a system in a heterogeneous environment. Nevertheless the relation of SOA to Web-Service technology is obvious. Web-Services as small functional units accessible through the standardized Internet protocols fulfil the requirements of SOA: different components of the system can reside within different domains, are programmed in different programming languages and are accessible by offering interfaces in a standard compliant manner via the Internet.

In MATURE a SOA approach will be followed to integrate the different services developed in WP2 – WP4.

3.2.1.2 Web-Services

Web-Services are services that can communicate with other services over a network, using a set of standard technologies. Web-Services, and more general the previously outlined SOAs emerged as the technologies and architectures of choice for implementing distributed systems and performing application integration and interoperability within and across companies' boundaries (Alonso, 2004). Figure 6 presents the basic Web-Services architectural model, which explains how Web-Services are advertised, discovered, selected, and used. This model is seen as a basis for a SOA.

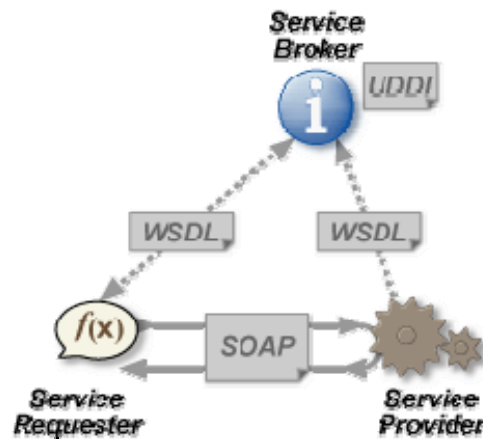


Figure 6: Web-Services Architectural Model

A compilation of the most influential standards for Web-Services has been defined by the Web-Services Interoperability organisation (WS-I, 2009). The WS-I Basic Profile in the last final version 1.1 (see (WS-I, 2006) for the specification) contains five XML based and one network level standard. On the bottom level the Hypertext Transfer Protocol (HTTP) (W3C HTTP, 2009) ensures the interoperability on the level of network protocols. Followed by the XML standard and the corresponding XML schema for defining the structure and constraints for XML dialects the syntactic level of the profile are defined. To enable the actual service interaction the Simple Object Access Protocol (SOAP) (W3C SOAP, 2009) is included that specifies the exchange of information between service providers and service requesters. SOAP is a method for exchanging XML based messages over the Internet for providing and consuming Web-Services. SOAP messages are transferred forming the SOAP-Envelope. There are also other ways of providing and consuming Web-Services: XML-RPC and REST. XML-RPC (remote procedure call) (XML-RPC, 2009) uses XML to encode and decode the remote procedure call along with its parameter. Representational State Transfer (REST) (Fielding, 2000) is a comparatively simpler method for providing and consuming Web-Services. In contrast to the above two methods, it is not necessary to use XML as a data interchange format in REST.

To express the properties of the published services (e.g. the concrete endpoints, operations, etc.) a service provider uses the Web-Service Description Language (WSDL) (W3C WSDL, 2009). REST services don't need WSDL service-API definitions. Finally the Universal Description, Discovery and Integration (UDDI) (OASIS UDDI, 2009) language is a specification for the realisation of the discovery level of services. UDDI acts as a kind of yellow page directory where service descriptions can be published by service providers and discovered by service requesters.

In MATURE different existing Web-Services will be integrated. These Web-Services are of different kinds. The MATURE architecture has to enable the integration of SOAP, XML-RPC and REST services.

3.2.1.3 Enterprise Service Bus

When adopting a SOA it is now common to use an Enterprise Service Bus (ESB) infrastructure. The core task of the ESB is to provide a set of services, which allow the creation of systems of higher complexity, by offering the functionality for the exchange of messages between different parts of the system. An ESB is a good foundation for building a SOA, as it enables the communication of different software components over a common bus and is also applicable for integrating legacy applications into a common system. The basic principle allowing this is the decoupling of the service called from the transport medium, which is done using transformation mechanisms to translate messages, thereby mediating the differences between message formats of different software components participating in the system.

The core principles of the ESB are virtualisation and aspect-oriented connectivity. The virtualisation is threefold in that it virtualises the communication protocols and patterns, which means the ESB provides conversion capabilities between different protocols and communication patterns; it virtualises the interface, which means that the requester and the provider services do not have to agree on a common

interface while the ESB provides the functionality to reconcile the difference; and it virtualises the identity, which means that the requester does not need to know about the service provider as the ESB provides the functionality necessary to hide the identity of the provider. Aspect-oriented connectivity means that the ESB can implement or enforce cross-cutting aspects like security, management, logging, and auditing on behalf of service requesters and providers, removing such aspects from the concern of the requesters and providers (Flurry, 2007).

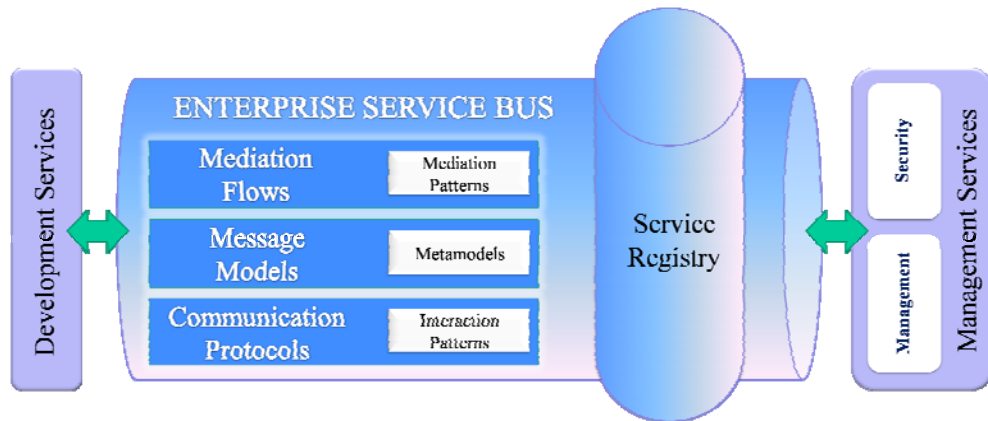


Figure 7: ESB Architecture

Figure 7 shows the architectural core elements of the ESB. The service registry stores the services metadata like interface descriptions and endpoints. It is accessed by the ESB at runtime to configure the behaviour of the system (the ESB looks up services metadata in order to gather the necessary data for mediation). The management services comprise functionality like security, logging and monitoring. The development services are used for the development of business- and integration logic, as well as for administration purposes. The ESB itself offers adapters allowing connecting software components. In order to achieve the communication between the different parts of the system, the ESB supports

- different interaction patterns in order to allow communication between different protocols,
- different message meta-models in order to mediate between message formats, and
- different mediation patterns in order to allow for more complex mediation flows, which include several transformation steps.

All main vendors like Microsoft, IBM, BEA and Oracle offer ESB products, either in form of ESB implementations or add-ons for existing application servers. In addition also many open source ESBs are available. For MATURE an open source ESB will be selected, which will be in charge of transporting the messages from requesters to providers and vice versa.

3.2.1.4 Virtualisation and Virtual Organisations

The virtualisation is an interesting trend originally introduced by the service-orientation in the Grid and now, in a different context massively pushed in the Internet of Things. The original concept was to represent resources in a so-called virtual environment so that the resources were able to communicate as virtual entities. In the Internet of Things the Internet is used as the virtual environment and physical resources are represented as services. MATURE considers this trend, as resources are represented as virtualised services. Here electronic services and human services are virtualised and represented as knowledge services within the system.

An interesting trend that should be observed for its applicability within the MATURE project are virtual organizations. Virtual organizations (VOs) are groups of people who share a data-intensive goal. To achieve their common goal, people within a VO choose to share their resources, creating a computer grid. This grid can give VO members direct access to each other's computers, programs, files, data, sensors and networks. This sharing must be controlled, secure, flexible, and usually time-limited. Organisations face ongoing pressures to become more flexible and responsive to change, looking increasingly to virtual

organisation to reduce organisational slack, facilitate continuous learning, and capitalise on core competencies (Hemingway and Breu, 2003).

The VO life cycle contains all stages of the VO from creation or identification to dissolution. Figure 8 depicts the VO lifecycle consisting of the following phases (BREIN D4.1.2, 2008):

- Preparation: during which service providers specify how and what kind of products they can offer.
- Identification: during which the service providers and resources are identified that are required to fulfil the respective collaboration's goals.
- Formation: during which the participants are configured to enable cross-enterprise transactions according to the collaboration plan.
- Operation and Evolution: during which the tasks according to the collaboration plan are executed, i.e., the transactions between participants take place.
- Dissolution: during which the collaboration is dissolved and each participant is freed from the collaboration requirements.

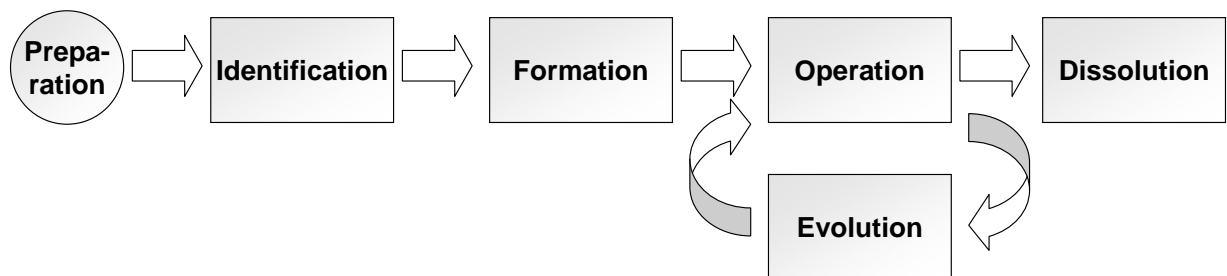


Figure 8: The Virtual Organisation Lifecycle

See (Foster, 2001), (Foster, 2004) and (Milke et al, 2006) for further information on virtual organizations.

3.2.2 Semantics

Semantic technology aims to introduce intelligent mechanisms into service oriented systems. In MATURE semantic technologies will be applied to realise mechanisms to configure and orchestrate the system. The SOTA in semantic service description and discovery will be presented in the following.

3.2.2.1 Semantic Service Description and Discovery

With the introduction of semantics in a Service-Oriented Architecture (SOA) the automated exchange of semantic information between machines shall be enabled (Domingue and Fensel, 2008). For this purpose the interfaces of services and their operations are described by semantic schemata so that a reasoning engine may understand the properties and operations of a service. In Table 1 seven dimensions of Web-Service features are compared in the context of a traditional Web environment against a Semantic-Web environment. Traditionally, services can only be invoked as single services whereas in the context of the Semantic-Web they may also take over the composition of other services to achieve a distinct goal. The role of the requester is not executed by a human anymore but by machines while the registration of services is not required due to their universally understandable semantic description that can be understood by Web crawlers. The role of a broker that has been central to traditional environments becomes that of a facilitator discovering appropriate services on the Web. The semantic description of services corresponds to a formally defined ontology instead of a taxonomy that only contains a classification of categories without inter linkages between the concepts and other semantic constraints. The interaction between services is not fully explicitly described (closed world assumption) but only partially, i.e. it would be possible that agents exercise additional behaviour that is not part of the service description (open world assumption). Finally, the exchange of data is not only based on a common syntax but also on common semantics.

Dimension	Traditional	Semantic Web
Service	<i>Simple</i>	<i>Composed</i>
Requestor	<i>Human</i>	<i>Machine</i>
Provider	<i>Registration</i>	<i>No-registration</i>
Broker	<i>Key Player</i>	<i>Facilitator</i>
Service description	<i>Taxonomy</i>	<i>Ontology</i>
Descriptive elements	<i>Closed world</i>	<i>Open world</i>
Data exchange	<i>Syntactic-based</i>	<i>Semantics-based</i>

Table 1: Comparison Traditional vs. Semantic Web (adapted from (Solazzo et al, 2002))

To achieve these goals of semantically interoperating services a number of approaches have been discussed in the scientific literature and have been partly proposed as standards. See (Martin, 2007) (Polleres et al, 2006), (Cabral et al, 2004) and (Stollberg et al, 2007) for an overview on semantic descriptions of Web-Services. One of the first initiatives in this regard that still attracts a lot of interest has been DAML-S that is now known as OWL-S (DAML, 2009). It is still under development and has not yet been officially accepted by one of the major standards institutions. OWL-S is an ontology that contains concepts directed towards the automatic discovery, composition, and invocation of Web-Services. The current specification of OWL-S can be found at (OWL-S, 2009). Despite the number of citations related to DAML-S and OWL-S major drawbacks of OWL-S have been identified due to sufferings in conceptual ambiguity, certain lacks of a concise axiomatisation, and its narrow view on Web-Services (Mika et al, 2004). Another approach for the standardisation of integrating semantics into Web-Services descriptions, called SAWSDL (Semantic Annotations for WSDL) can be found in the Working Group for semantic annotations in Web-Service descriptions (W3C SAWSDL, 2009).

Further related proposals for standards include WSDL-S (Akkiraju et al, 2005), WSML and the related ontology WSMO (Roman et al, 2005) and IRS-III (Cabral et al, 2006). In WSDL-S semantics are included directly in the abstract definition of the WSDL documents. This approach allows for the annotation of the input and output messages with domain concepts, the annotation of operations with preconditions and effects and the annotation of interface definitions of services with category information. The advantage of WSDL-S in comparison to OWL-S is seen in the availability of a language that is already partly familiar to Web-Service developers and whose semantic description is not specific to an ontology representation language.

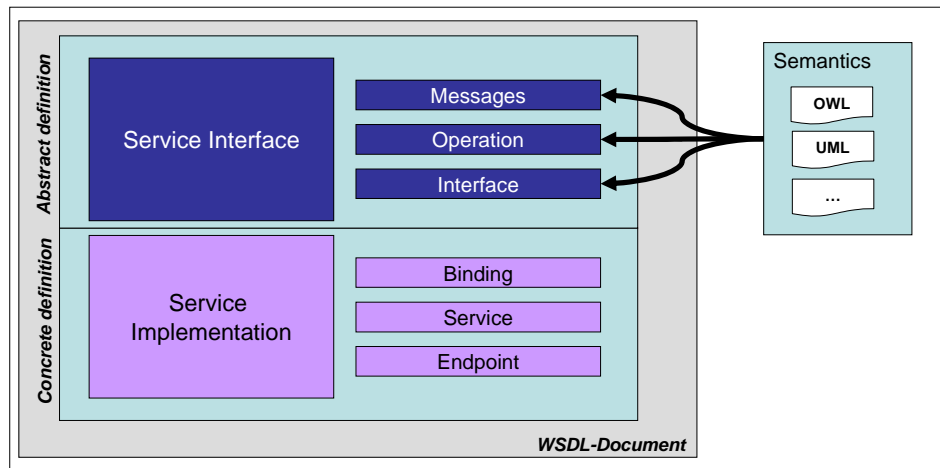


Figure 9: Illustration of the WSDL-S Approach (after (Moran et al, 2005))

The Web-Service Modelling Ontology (WSMO) is a formal ontology that contains concepts for describing general services that are accessible through a Web-Service interface. WSMO builds upon a meta model approach based on the Meta Object Facility (MOF) (OMG MOF, 2009) with the four top level elements: Ontologies, Goals, Web-Services, and Mediators. Its goal is the enabling of the partial or total automation of tasks in terms of service discovery, selection, composition, mediation, execution, monitoring, etc. The Internet Reasoning Service (IRS) (KMI, 2009) builds upon WSMO and provides the representational and reasoning mechanisms for implementing the WSMO meta-model to describe Web-Services. For this purpose IRS uses the Operational Conceptual Modelling Language (OCML) ontology representation language and describes an execution environment including application-programming interfaces.

3.2.3 Web 2.0

The term "Web 2.0" refers to a perceived second generation of web development and design, that aims to facilitate communication, secure information sharing, interoperability, and collaboration on the World Wide Web (O'Reilly, 2005). Web 2.0 concepts have led to the development and evolution of web-based communities, hosted services, and applications; such as social-networking sites, video-sharing sites, wikis, blogs, and folksonomies. The most relevant Web 2.0 concepts and technologies for the project will be described in the following.

3.2.3.1 AJAX

Ajax is not a technology on its own but a new concept of combining existing technologies to create and port applications to the web. The goal is to create interactive and performing web-applications sharing the look-and-feel of desktop applications with the same usability (Cardwell, 2005).

Traditional web-applications are based on the request-response paradigm – the user is sending a request to the server, the request is processed and as a response the user interface is updated. In case of long-processing time on the server or loss of connectivity, the flow of the application is interrupted.

In contrast, using AJAX each request that the user sends is a call of a JavaScript function that is delegated to the AJAX-engine which means that the request is resolved on the client. Small updates which do not involve server side processing are dealt with on the client. This concept increases the reaction time of the user interface and reduces the data traffic between the server and client.

Historically the first step of this concept was taken in the late 90's when the Outlook Web Access was developed by the Microsoft Exchange Server developers. The concept got well known through web-applications developed and provided by Google like Google Groups, Google Maps, Google Suggest and Google's emailing system Gmail.

Some of the existing applications that are evaluated for their applicability within MATURE are AJAX-based.

3.2.3.2 Widgets

A web widget is a portable chunk of code that can be installed and executed within any HTML-based web page by an end user without requiring additional compilation. Widgets are now commonplace and are used by bloggers, social network users, auction sites and owners of personal web sites.

Widgets are a way for a site or service to creatively offer products, services or news without the need of the user to visit the actual site. Similar to feeds and syndication, widgets can save a user time by making everything they care about on the web easily accessible in one place.

There are a number of widget marketplaces. One example is Google Gadgets¹ where various widgets from different categories (e.g. News, Tools, Communication, Sports, Finance) are offered and can be integrated and used on any web page.

The applicability widgets for MATURE has been evaluated in the technological design study Interacting Widget (see D2.1 for details).

3.2.3.3 Mashups

Mashup is an upcoming concept that can be defined as a web application that combines data from more than one source into a single integrated tool (Merrill, 2006). The term Mashup implies easy, fast integration, frequently done by access to open APIs and data sources. An example for a Mashup is Housingmaps², which uses cartographic data from Google Maps³ to add location information to real-estate data, thereby creating a new and distinct Web-Service that was not originally provided by either source. Figure 10 depicts a screenshot of housingmaps.com. The left-hand side depicts the integrated location information from Google Maps, whereas the right-hand side provides real estate information.

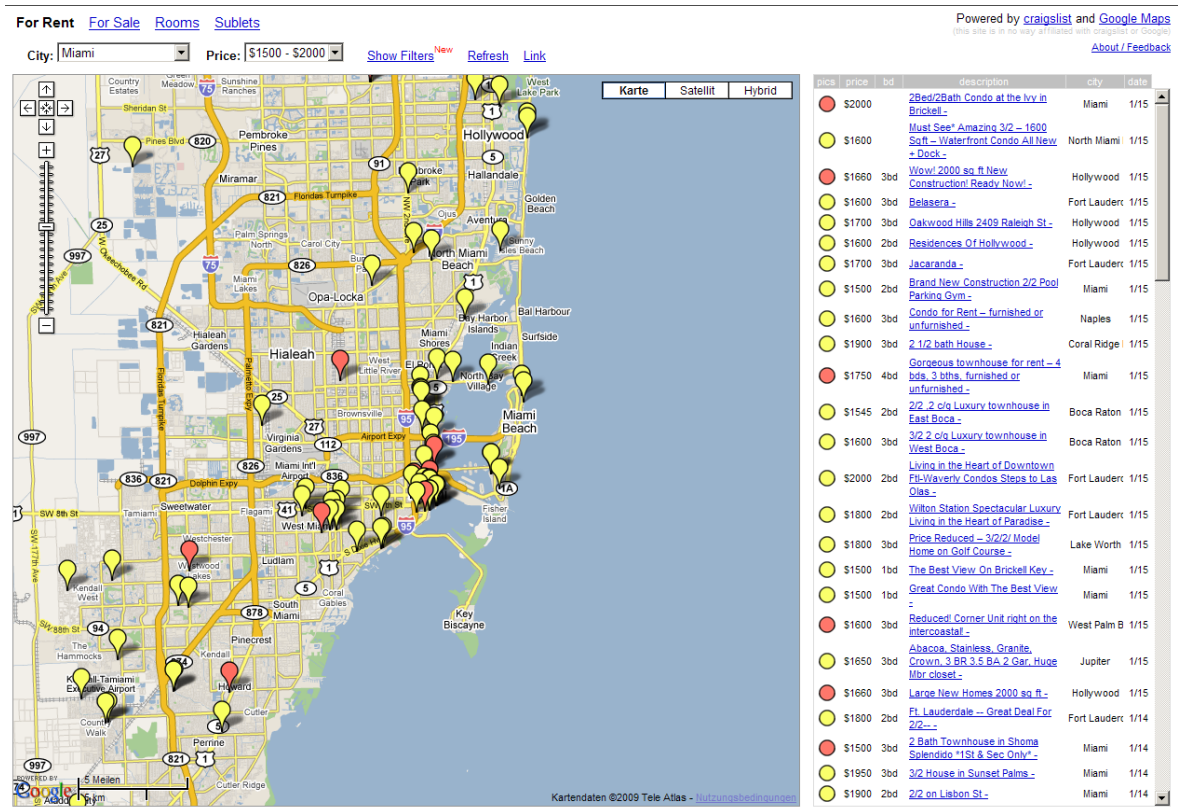


Figure 10: Mashup - Example "Housingmaps.com"

¹ Google Gadgets. Access: <http://www.google.com/ig/directory?hl=en&synd=open> [13.03.2009]

² Housingmaps. Access: <http://www.housingmaps.com/> [13.03.2009]

³ Google Maps. Access: <http://maps.google.com/> [13.03.2009]

Enterprises are now investigating into using mashups in the business environment. Gartner Group (Gartner Group, 2006) identifies enterprise mashups as one of the 10 strategic technologies. Through 2010, the enterprise mashup product environment will experience significant flux and consolidation, and application architects and IT leaders should investigate this growing space for the significant and transformational potential that it may offer their enterprises.

In a mashup world, SOA can provide the services that supply the raw materials to a community of mashup users (Warner et al, 2008). Mashups enable the integration of business and data services, as mashup technologies provide the ability to develop new integrated services quickly, to combine internal services with external or personalized information, and to make these services tangible to the business user through user interfaces.

Mashups are an interesting trend (see e.g. (Hoyer and Stanoevska-Slabeva, 2009)) and should be evaluated for the MATURE system for UI integration in later phases of this WP.

3.2.4 Security and Trust

As knowledge management has become a more central part of organizational activities and dependent upon technologies, securing organizational knowledge has become one of the most important issues in the knowledge management area (Lee et al, 2005). As identified in section 2.2 Security and Trust issues will also affect the MATURE system when it will be integrated at the application partners' sites into a real world environment. Then access to sensitive or important information needs to be protected.

The term trust management, introduced in (Blaze et al, 1996) as “a unified approach to specifying and interpreting security policies, credentials, and relationships which allow direct authorization of security-critical actions”. Later this definition has been broadened, not limited to authorizations, but also covering the activity of collecting, encoding, analyzing and presenting evidence related to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships. Two main approaches are currently available for managing trust: policy-based and reputation-based trust management.

- Policy-based Trust Management: This approach has been proposed in the context of open and distributed services architectures (Bonatti et al, 2005) as a solution to the problem of authorization and access control in open systems. The focus here is on trust management mechanisms employing different policy languages and engines for specifying and reasoning on rules for trust establishment. The goal is to determine whether or not an unknown user can be trusted, based on a set of credentials and a set of policies.
- Reputation-based trust management: This approach has emerged in the context of electronic commerce systems, e.g. eBay. In distributed settings, reputation-based approaches have been proposed for managing trust. The focus here is on trust computation models capable of estimating the degree of trust that can be invested in a certain party based on the history of its past behaviour. The main issues characterizing the reputation-based systems are the trust metric and the management of reputation data (Aberer et al, 2001).

The Security Assertion Markup Language (SAML) (OASIS SAML, 2009) is an XML-based standard issued by OASIS (OASIS, 2009), the Organization for the Advancement of Structured Information Standards. It serves as a framework for exchanging authentication, entitlement and attribute information between networked entities.

Based on the foundation of SAML, Shibboleth is an architecture and open-source implementation for a federated identity-based Authentication and Authorization Infrastructure (AAI). It supports features such as Single Sign on (SSO) across organizational boundaries and removes the need for service providers to maintain user names and passwords. Identity providers (IdPs) supply user information, while service providers (SPs) consume this information and get access to secure content.

The Shibboleth architecture defines a way of exchanging information between an organisation and a provider of digital resources (such as data, video, documents, and so on). By using Shibboleth, the information is exchanged in a secure manner, protecting both the security of the data and the privacy of

the individual. In the Shibboleth model, the IdP is responsible for authenticating the user - that is, for checking that the credentials the user presents are correct (typically with a username/password combination). The IdP is also responsible for providing information about the user. This information is called attribute information. The decision to authorise access to information is the responsibility of the owner of the resource (the Service Provider), and is based on the user's attribute information.

Many organizations are using Shibboleth today to solve multi-organizational Web access problems. A list of Shibboleth enabled applications and services can be found at (Shibboleth, 2009). Shibboleth will also be evaluated for its applicability in MATURE.

This section provided an overview of the conceptual background and the state of the art relevant to meet the requirements of the MATURE system architecture. In the following chapter the MATURE system architecture will be presented from a high level taking into account the introduced conceptual background.

4 MATURE Architecture Overview

This section will provide an overview of the MATURE system architecture. To design the MATURE system architecture a hybrid approach was followed since the beginning of WP5 in project month 6. Figure 11 illustrates this approach. Following a bottom-up approach services were collected and a rapid prototyping approach was followed to integrate existing services into the MATURE system as already introduced in the DoW (MATURE DoW, 2007). From top-down the architecture view model as proposed by Kruchten (Kruchten, 1995) was used to analyse and describe the architecture from different view points. The figure also depicts that the alignment of the bottom up and the top down view will take place in several iterations. Therefore in the deliverable at hand (which is submitted as DRAFT) we provide an overview of the current status of the MATURE system from both view points but would like to point out that the aligned version of the system architecture will be presented in the final version of this deliverable in project month 18.

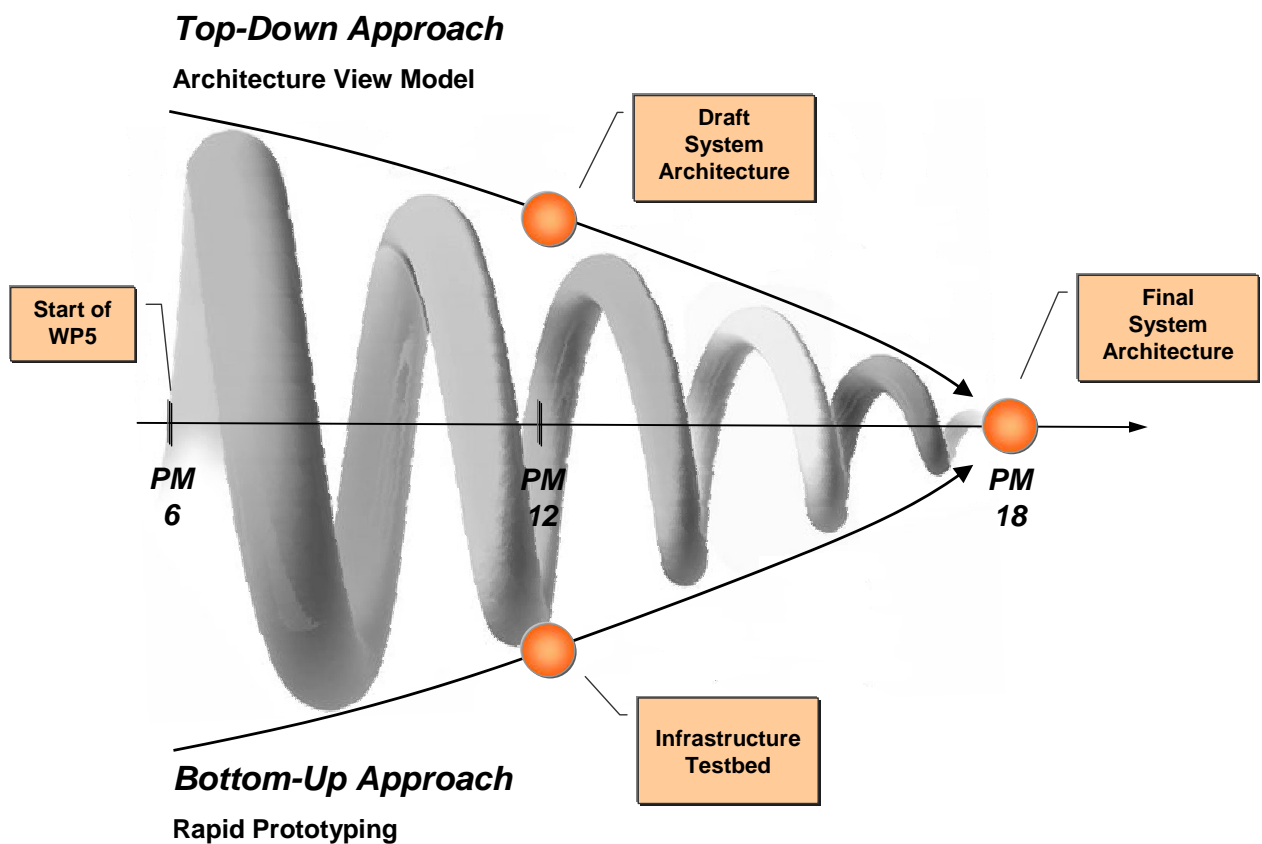


Figure 11: Hybrid Approach to the MATURE System Architecture Design

In the following the system will be described first, from the bottom-up view and second, from the top-down view.

4.1 Bottom-Up View on the MATURE System

This section will introduce the bottom-up view on the MATURE system as depicted in Figure 12. A rapid prototyping approach was followed in order to develop the MATURE system architecture from bottom up, as already introduced in the DoW (MATURE DoW, 2007). This approach will be introduced in the following section.

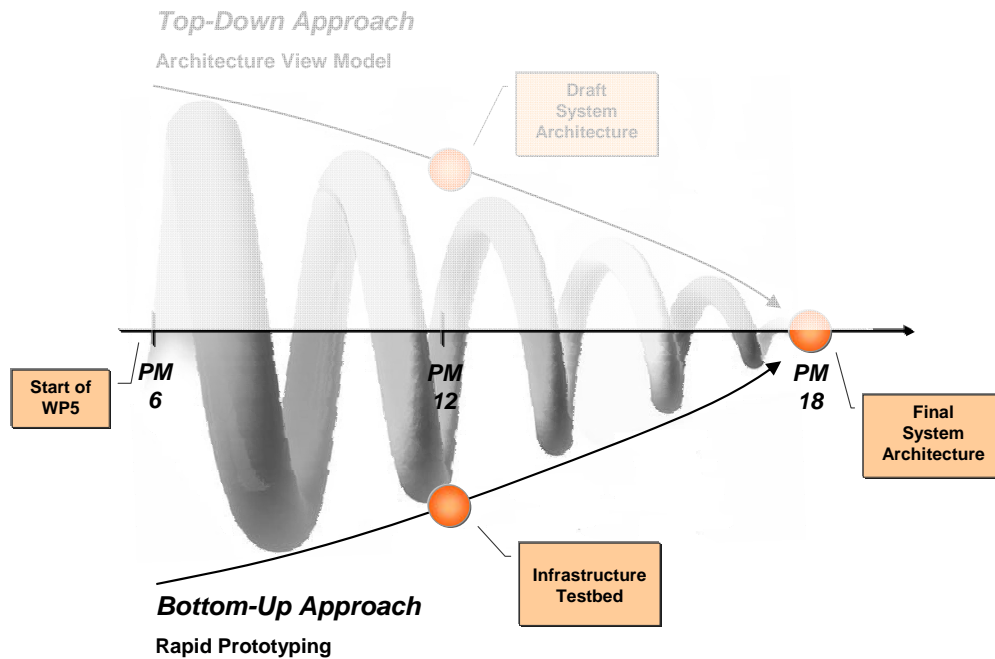


Figure 12: Bottom-Up View on the System Architecture

4.1.1 Introduction to the Bottom-Up Approach - The Rapid Prototyping Approach

Research and complex software development is a realm with high degrees of novelty, creativity and change. Therefore it is rarely possible to create up-front unchanging and detailed specifications. Therefore for software engineering a Rapid Prototyping approach is followed, as it supports agile, iterative and incremental development cycles with integral testing and frequent, use case centric, adaptive requirements analysis.

Rapid prototyping⁴ is founded on time-boxed iterative and evolutionary development. The foundation is adaptive planning, that encourages rapid and flexible response to change, and frequent evolutionary internal and external releases. Iteration planning defines which functionalities will be implemented within the next iteration. The result of each of the iterations is an internal iteration release. Feedback from iterations leads to refinement and adaptation of the requirements and several iterations regularly lead to delivery of external releases to the end users – that means useful and valuable software for the end users and valuable end user’s feedback for the development teams. Figure 13 provides an overview of the rapid prototyping approach.

⁴ Bijay K. Jayaswal, Peter C. Patton (2006): Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software, Prentice Hall.

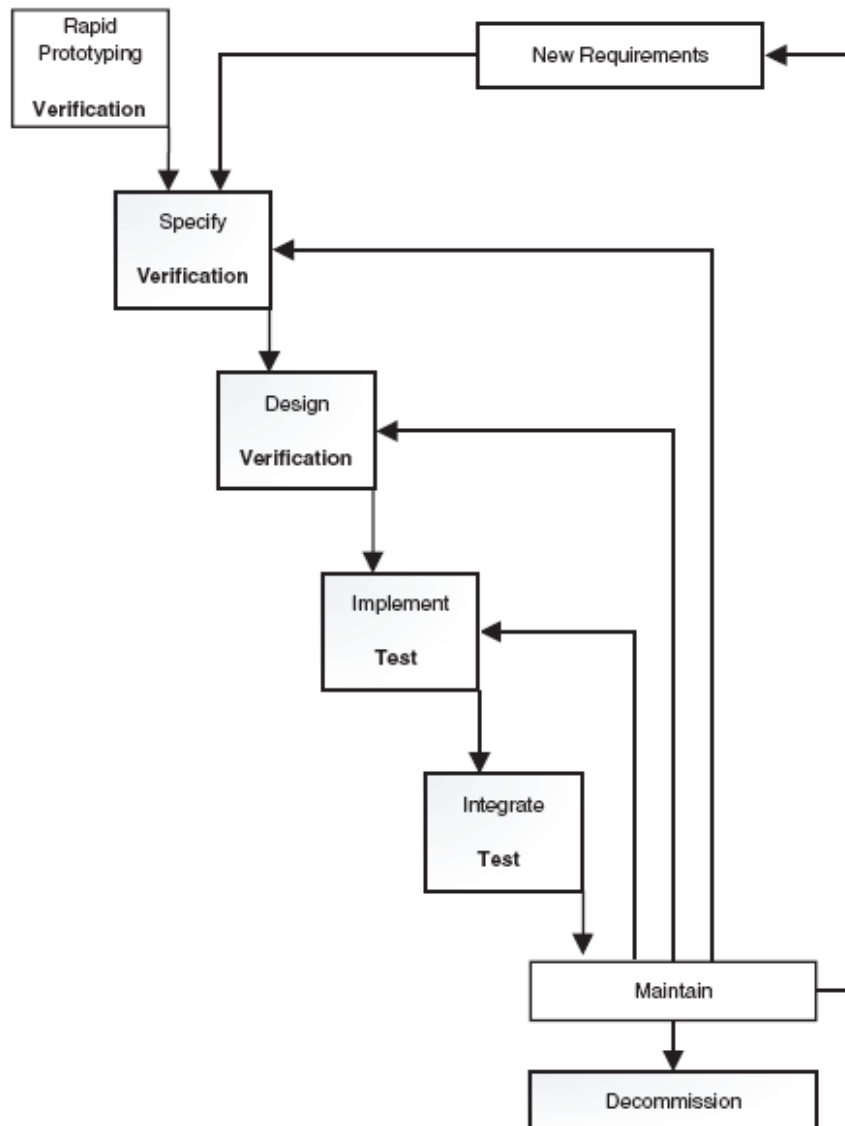


Figure 13: Overview of the Rapid Prototyping Approach (Bijay, 2006)

In MATURE we followed the rapid prototyping approach to identify technical systems and services already in use, which are applicable for maturing. A rudimentary architecture and demonstrator will be developed based on the findings. The goal is to establish a basic architecture on which we will build in the following years, as well as a test bed which can be used to test initial developments.

In the following the findings of the first iteration of the rapid prototyping cycle will be presented, namely the collected services which were analysed for their applicability within project and an initial integration scenario which was realized as a technical demonstrator in order to identify requirements for the MATURE system architecture.

4.1.2 Service Collection

As defined in the Description of Work (MATURE DoW, 2007) the MATURE system has to be able to integrate the following components:

- The wrapped services of already existing functionalities identified in WP2 and WP3
- The Personal Learning and Maturing Environment (PLME) which is developed in WP2

- The Organisational Learning and Maturing Environment (OLME) which is developed in WP3
- The Maturing Services which are developed in WP4
- The co-existing knowledge sources at the MATURE application partners, which may include learning management systems, E-Mail, document repositories, wikis, information about people like stored in LDAP (light-weight directory access protocol) data bases, competence descriptions or yellow pages.

As already mentioned before following a rapid prototyping approach, as a first step the existing services or functionalities of existing tools were gathered and analysed for their applicability within MATURE. Th

In order to gather the integration relevant aspects right from the start, the services integrated in the design studies were collected and integration relevant aspects were identified by the corresponding partners. The design studies were experiments that explored key aspects that needed to be validated prior to embarking on full scale requirements specification, which typically involved establishing that the conceptual, user, and system design aspects of the project could be integrated (see D6.1 (MATURE D6.1, 2009) for an overview of design studies).

The goals were, in the context of a focussed area, to: get feedback on initial ideas, to discover integration potential, to gain experience with supporting knowledge maturing processes and, by implication and to elicit any ‘early warning signs’ that needed to be considered and addressed as the project progressed. Specifically, design studies were well-focused on investigating existing tools with limited further developments and exploring both conceptual and software development foci. Within WP5 attention was paid to the integration and architecture aspects that these studies surfaced.

Table 9 depicts the generic design study template that was instantiated by each design team by the description of the services integrated in the design studies and the gathering of integration-relevant attributes for WP5. The filled tables for each design study can be found in Annex A.

Table 2: Service Integration Template for the Design Studies

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	Name of the Service	Service Description understandable for business users (technical non-savvy user)	Technical specification of service – architecture, technology used, service structure	[OLME Service, PLME Service, Maturing Service] – Multiple selection possible	[UI Layer, Logic Layer, Data Layer] – Multiple selection possible	Atomic, Service cluster[specification of sub-services necessary using IDs], Composite services [specification of sub-services and logical flow necessary] – based on SOMF 2.0 (Arsanjani, 2004)	If feasible, technical specification of required input for service (ideally Web-Service message specification or data types)	If feasible, technical specification of required output for service (ideally Web-Service message specification or data types)	Input/Output relation to other service available in the list

The service descriptions were refined in several iterations in order to describe the services on the same granularity level.

4.1.3 Integration Scenario

Based on the collected services an initial integration scenario was developed to identify requirements for the MATURE system architecture. To identify services that will be integrated, it was necessary to discuss the candidate services in more detail. Therefore a service fact sheet was created which was used as common template to describe the services and the provided features of each service. Please see Annex B for the Service Fact Sheet Template.

The developed integration scenario is related to the use case areas I “Learning by searching for and exploring artefacts for the task at hand” and IV “Creating, refining, developing, aggregating, structuring, and sharing artefacts”. This use case area supports knowledge maturing by improving the findability of existing artefacts, making use of them in practice and to make them more mature. An overview of the use cases can be found in the deliverable D6.1 (“Specification of Requirements”) (MATURE D6.1, 2009). A detailed description of the use cases and how they enable personal and organisational learning and maturing can be found in D2.1 (“Pedagogical and usability foundations and concept for a PLME”) (MATURE D2.1, 2009) and D3.1 (“Model of organizational requirements and of supporting services of the OLME”) (MATURE D3.1, 2009).

Based on this use cases the first integration scenario can be described as follows:

Taking the scenario at the MATURE application partner Connexions Kent as an example, a Personal Advisor P.A. has to provide a young person with the knowledge product career guidance. The P.A. uses knowledge access services to identify relevant sources of labour market information (LMI). The Soboleo service and the Data Persistence service can be used to identify knowledge sources that might be of relevance for the case at hand.

Then maturing services enable the P.A. to select sources of labour market information (LMI) that are reliable, valid and manageable based on certain criteria. The Maturing analysis provides the user with information about the readability of each knowledge item and is used to classify and recommend tags for each document. The subsequent page ranking provided by the rule engine is used to select the most suitable knowledge sources based on rules.

Afterwards the user aggregates the various knowledge sources, resolves contradictions and presents the information in a way that helps the young person in this scenario to understand her options. After using the various identified knowledge sources, the user aggregates the information and stores the document. In this scenario he/she uses the Soboleo and the Data Persistence service to store newly created documents or to store changes to the identified documents and to save the recommended tags for each document.

Figure 14 provides an overview of this scenario and highlights the relevant concepts as introduced in Chapter 3.

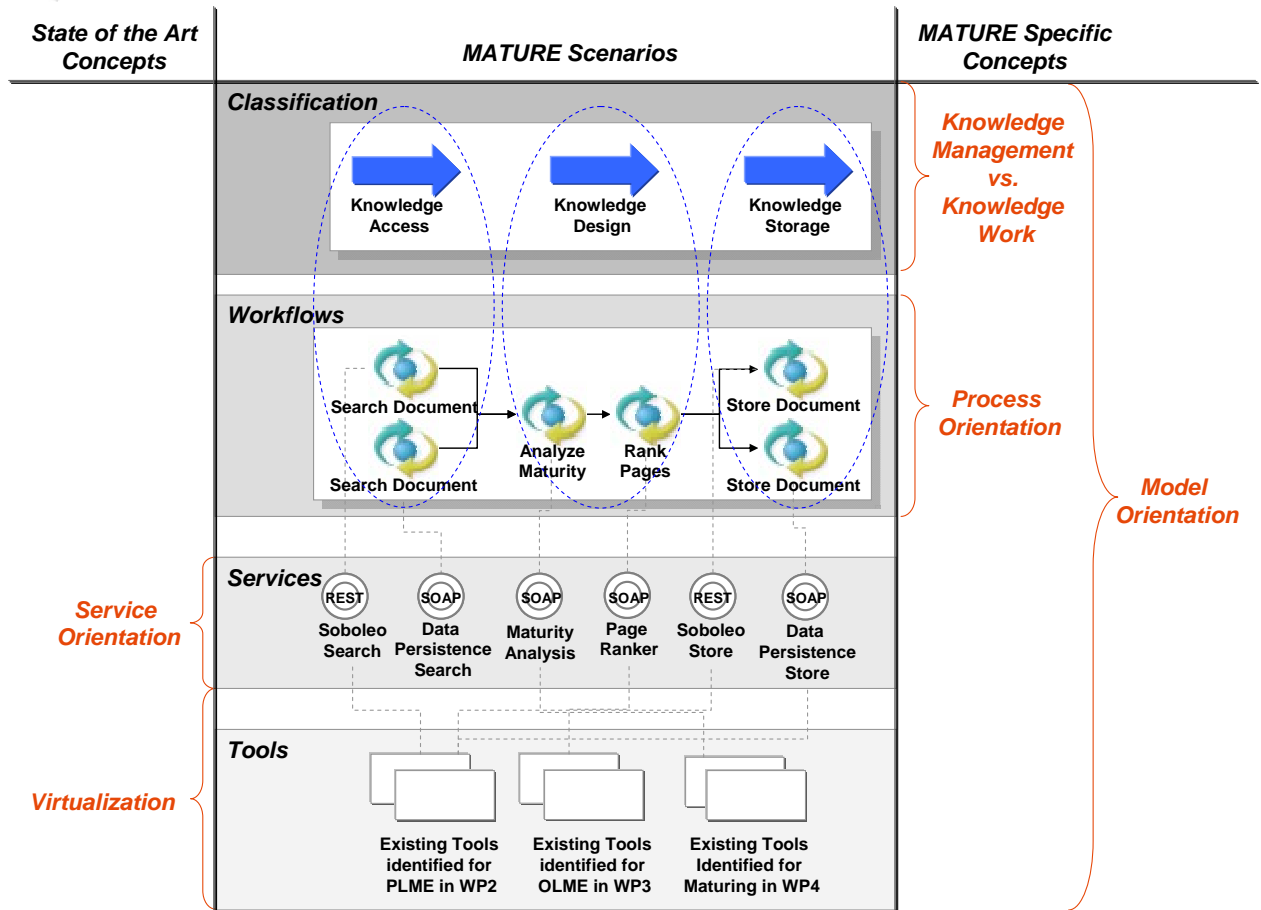


Figure 14: Bottom-Up Approach on the System Architecture – Integration Scenario

In the following the services that have been integrated in the first integration scenario will be briefly introduced by providing a table with a short summary, the responsible partner, the technology used and an overview of all features provided by the service. Please see the MATURE Wiki⁵ for a detailed description of each service.

⁵ MATURE Wiki, Service Fact Sheets, http://wiki.mature-ip.eu/index.php/Service_Fact_Sheets

Table 3 provides an overview of the Soboleo service.

Table 3: Soboleo Service - Overview

<i>Soboleo Service</i>	
Summary	This service offers functions related to a document store that is organized with a SKOS ontology or related to a domain description in a Semantic Media Wiki (depending on configuration). Further, the service provides functions related to a person store containing profiles generated from user activities and explicit user annotations (i.e. what users annotate with and how persons are annotated). Note that the system knows about more persons (the person store) than there are users of the system (user store).
Responsible Partner	FZI
Access	API
Technology	SOBOLEO is implemented as a Java Web application (runs in a current version of Tomcat). The services are currently implemented XML and POJO over HTTP (in the context of an AJAX/GWT applications). It is implemented as a RESTful Web-Service.
Features	The service allows to search and retrieve documents, retrieve persons, retrieve the ontology, change the ontology, change the document store (by adding or retrieving documents), change information about persons and to obtain a list of recent changes to ontology, document store, and person profiles. In the following rows only the features relevant for the prototype are documented.
<i>Search Documents</i>	This function searches the document store using the ontology as background knowledge. <i>Input:</i> A search string (in the sense of keyword search) <i>Output:</i> A list of results (document identifier) and possible query refinements (proposed alternative search strings)
<i>Change the Document Store</i>	Changes the document store <i>Input:</i> A document identifier and a document change event (such as add document, remove document, add annotation, remove annotation, change annotation), user identifier <i>Output:</i> Status message

Table 4 provides an overview of the RHEA rule engine service.

Table 4: Rule Engine Service - Overview

<i>RHEA Rule Engine Service</i>	
Summary	RHEA is a rule engine, which can be integrated in an internet based workflow engine. RHEA supports the adaptivity of knowledge intensive process parts using rules.
Responsible Partner	FHNW
Access	API
Technology	This service is implemented as a Web-Service and provides a WSDL interface.

Features	The service provides four methods for resource allocation, variable process execution, decision making and constraints checking.
<i>executePageRanking</i>	This method is used to execute a rule set for the ranking of pages at execution time. <i>Input:</i> Array of context relevant data and a link to a rule set <i>Output:</i> Ranked list of pages.

Table 5 provides an overview of the Maturing service.

Table 5: Maturing Service - Overview

<i>Maturing Service</i>	
Summary	The already available maturing service provides a set of functions in order to support the user in finding and creating knowledge items. In addition, the service facilitates the improvement of quality of knowledge items.
Responsible Partner	TUG
Access	API
Technology	This service is implemented as a Web-Service and provides a WSDL interface.
Features	All features are based on text processing, within the design study the services were used for evaluation of MediaWiki content. The service-functions can be divided into two groups, mark-up recommendation and maturing indicators. Markup recommendation: tag recommendation and classification Maturing indicators: reading score and semantic indicator In the following rows only the features that will be used within the integration scenario will be introduced.
<i>Tag recommendation</i>	Based on a given string the tag recommendation computes the most relevant terms. <i>Input:</i> String <i>Output:</i> String-Array containing terms, length=3
<i>Classify</i>	This function classifies a string to a given set of categories <i>Input:</i> String <i>Output:</i> Category
<i>Readability</i>	The Readability indicator provides several reading scores indicating the maturity of a given text. <i>Input:</i> String <i>Output:</i> String-Array containing reading scores, length=2

Table 6 provides an overview of the Data Persistence service.

Table 6: Data Persistence Service - Overview

<i>Data Persistence Service</i>	
Summary	This service is a persistence service to store the metadata of an object. Such objects are Twitter messages as the service has been created for the Widget Design Study, but can serve for other artefacts, too. It serves the metadata as an XML file and relates it to tags which are extracted of the message itself.
Responsible Partner	UPB
Access	API
Technology	The service uses the Web-Service implementation of Java6 and a Hibernate wrapper for MySql to access the database.
Features	The features provide functionality to store and get metadata of an object.
<i>Get Metadata by Tags</i>	Fetch the metadata XML file from the database, which is related to the tags. <i>Input:</i> string (tags) <i>Output:</i> byte array
<i>Store Tagged Metadata</i>	Store a metadata XML file by giving the byte array and the related tags. <i>Input:</i> string (tags), byte array <i>Output:</i> None

Several challenges can be identified analysing the services integrated in this scenario. To give a specific example related to the aforementioned scenario, one service may specify the documents in XML another just as a String which provides the location of the document or as a byte array containing the contents of the identified documents. Furthermore existing services are implemented to communicate in different ways using Representational State Transfer (REST) or the Simple Object Access Protocol (SOAP). Different semantics in messages further complicate the problem. As the number of interacting services with proprietary message models increases (when extending the prototype within this work package), the challenges of managing the necessary transformations to enable service interaction increase exponentially. For each requester, a separate transformation has to be specified to each provider. A solution to this problem is the common message model approach, where each requester needs a transformation to the common message model, and each provider needs a transformation to the common message model, in case they all have different models. Therefore, the total number of transformations is reduced. More details on the common message model can be found in Chapter 5. Within MATURE the discussion on a common message model for the MATURE purpose was started. A WSDL file implementing this MATURE Message Model can be found in Annex C. More details on the message model will be provided in section 5.2.2.2.

At the first technical partner meeting in Vienna the implementation of the aforementioned integration scenario was started. The existing services were wrapped in order to implement the MATURE Message Model. One possible approach to wrap a service is to integrate it using BPEL workflows. Figure 15 depicts a picture of the meeting and a BPEL workflow that was jointly created in the course of this meeting. This BPEL workflow wraps the maturing service, in order to implement the MATURE Message Model.

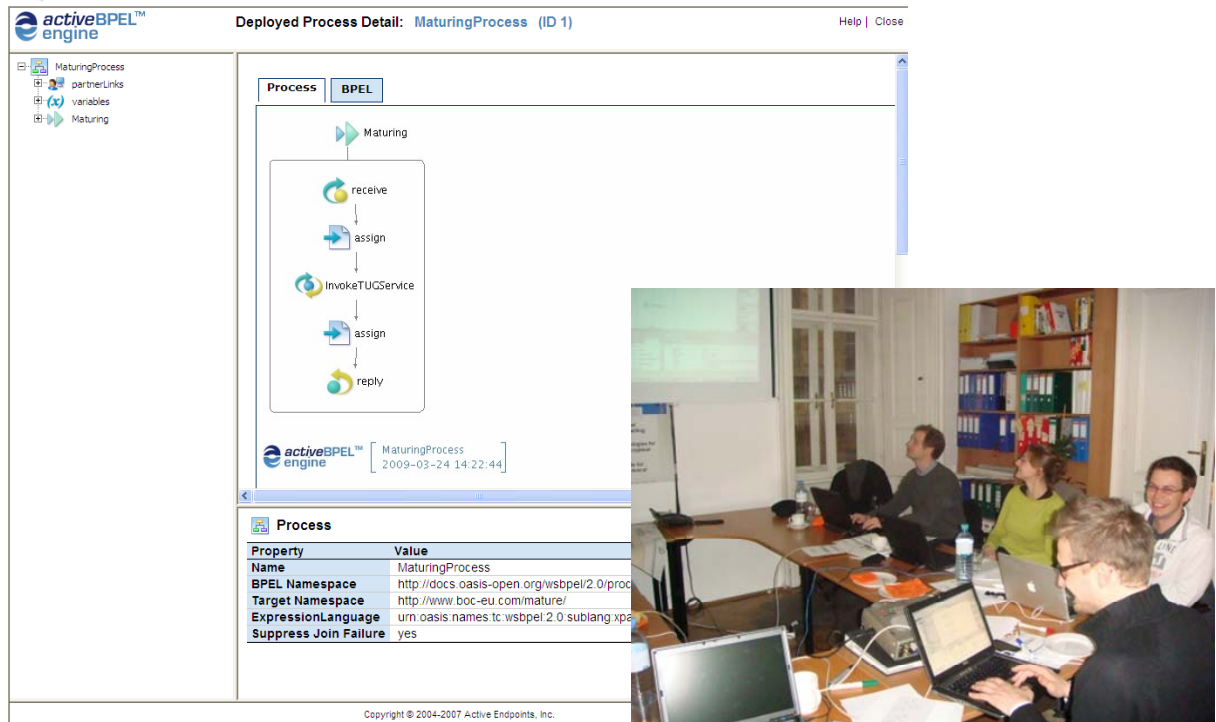


Figure 15: Implementation of the Integration Scenario at the 1st Technical Partner Meeting

At this point in time the introduced MATURE Message Model is only the first version that was created based on the integrated services. The MATURE Message Model will be refined in the course of this work package.

The wrapped services are registered, annotated and published using the WP5 infrastructure testbed as described in D5.1 (MATURE D5.1, 2009).

The MATURE system architecture has to integrate various components. As a first step existing services as identified in WP2, WP3 and WP4 will be integrated in the proposed integration scenario. As a next step newly developed PLME (WP2), OLME (WP3) and Maturing (WP4) services have to be integrated. Finally also co-existing knowledge sources in use at the MATURE application partners have to be integrated when the system is deployed at the application partners.

The MATURE system has to provide an integration layer which acts as a uniform interface for accessing various knowledge sources and for registry, discovery and invoking of services as well as for messaging between services. After this section provided a bottom-up view on the MATURE system by analysing the services to be integrated, the following section will provide a top-down view on the system by analysing the integration layer of the system from different view points.

4.2 Top-Down View on the MATURE System

This section provides a top-down view on the MATURE system. First the applied architecture view model will be described followed by an analysis of the system from multiple concurrent view points. The system has to provide the infrastructure for the registry, discovery and invocation of the services, as well as the messaging between the services developed in the different work packages and the registration of knowledge sources.

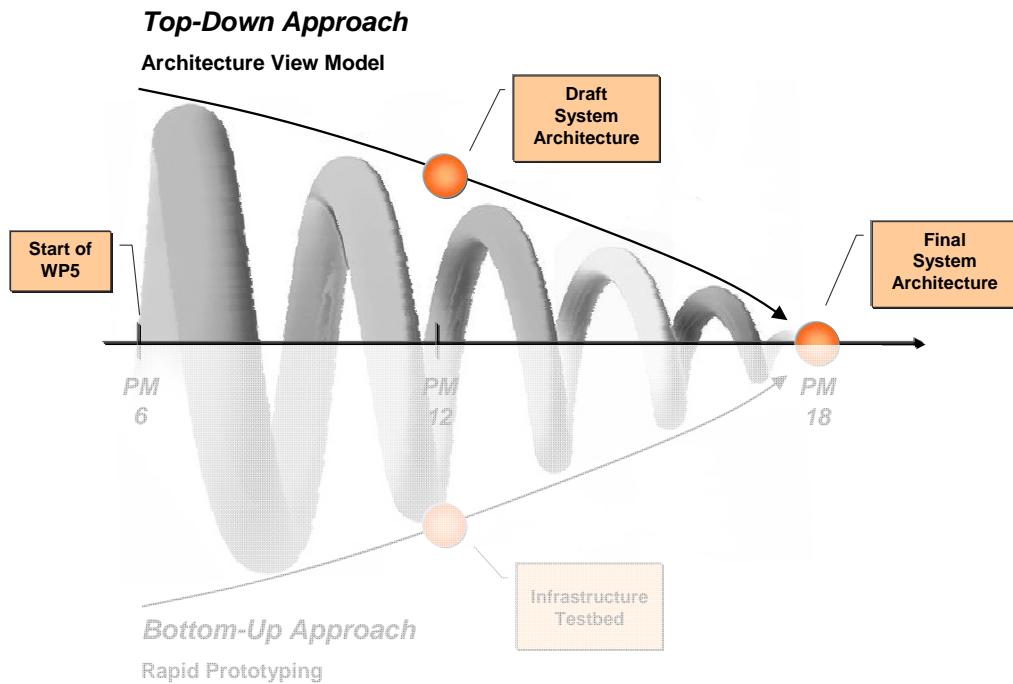


Figure 16: Top-Down View on the System Architecture

The top-down approach will be described in the following section.

4.2.1 Introduction to the Top-Down Approach - The Architecture View Model Approach

This section describes the model that has been used to specify the overall architecture of the MATURE system. The description of the architecture follows the 4+1 view model (Kruchten, 1995) by Philippe Kruchten. 4+1 is a view model which is designed for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. These views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers.

The four views comprise the logical, the development, the process and the physical view. In addition scenarios are used in order to make the system architecture more tangible. As a result the model contains 4+1 views as Figure 17 depicts.

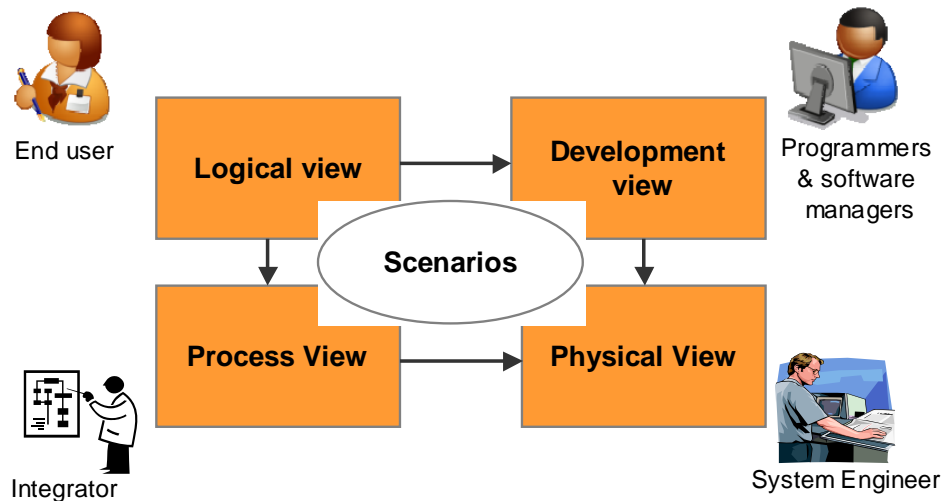


Figure 17: The MATURE System Architecture Design Method

In the following the different views will be described before this model is applied to describe the MATURE system architecture from these view points.

The **logical view** can be seen as a layer capable of providing the information on the necessary functional requirements that have to be addressed within a specific system in order to achieve the desired level of functional usability for the targeted group of end users. Mostly, these functional requirements are seen as services (dynamic or static), which are being provided to the end users of the system. The system itself, based on the Object Oriented approach followed in this case, is decomposed to atomic actions comprising the described services and presented in form of objects or objects classes. These classes are used to describe the use case scenarios presented in the following paragraphs.

The **development view**, also known as implementation view, provides an overview of the overall system from the programmer's perspective, focusing mainly on software management. It is used to provide the overall picture of the software landscape by grouping the software chunks (managed or developed by small groups of developers) in different layers which are connected through well defined interfaces.

The **process view** deals with the dynamic aspects of the system, in the first place with such requirements on the system as performance and availability / continuity. That is, this view is concerned with the issues arising in the software development area such as concurrency, deployment, integrity, fault-tolerance, effectiveness, etc. Based on the proposed requirements the description of the system can be made on different levels of abstraction, where each level is used to provide answers for different requirements.

The **physical (or deployment) view** depicts the system from the system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as communication between these components. This view provides an overview of the deployment scenarios for the MATURE system.

An overall **scenario** will be used to bring the different views together and to understand the whole system's functionality. In the scenario the main actors will be identified and the corresponding use-cases will be derived.

In the following sections the introduced architecture view model will be applied to describe the MATURE system from different view points, starting with the logical view.

4.2.2 Logical View on the MATURE Architecture

As introduced before, the logical view is an object-oriented decomposition of the system from an end user perspective. It considers functional requirements, basically what the system should provide in terms of services to its users. The MATURE system has to provide a central component for the configuration of the system, the registration, the publication, the discovery and the orchestration of services and the integration of different knowledge sources. This component which acts as a middle tier between the

various knowledge sources, the wrapped services, the PLME and OLME services and the Maturing services is called the Knowledge Bus. Figure 18 shows the logical view on the middle tier of the MATURE system by identifying all services that have to be provided to cover the required functionality.

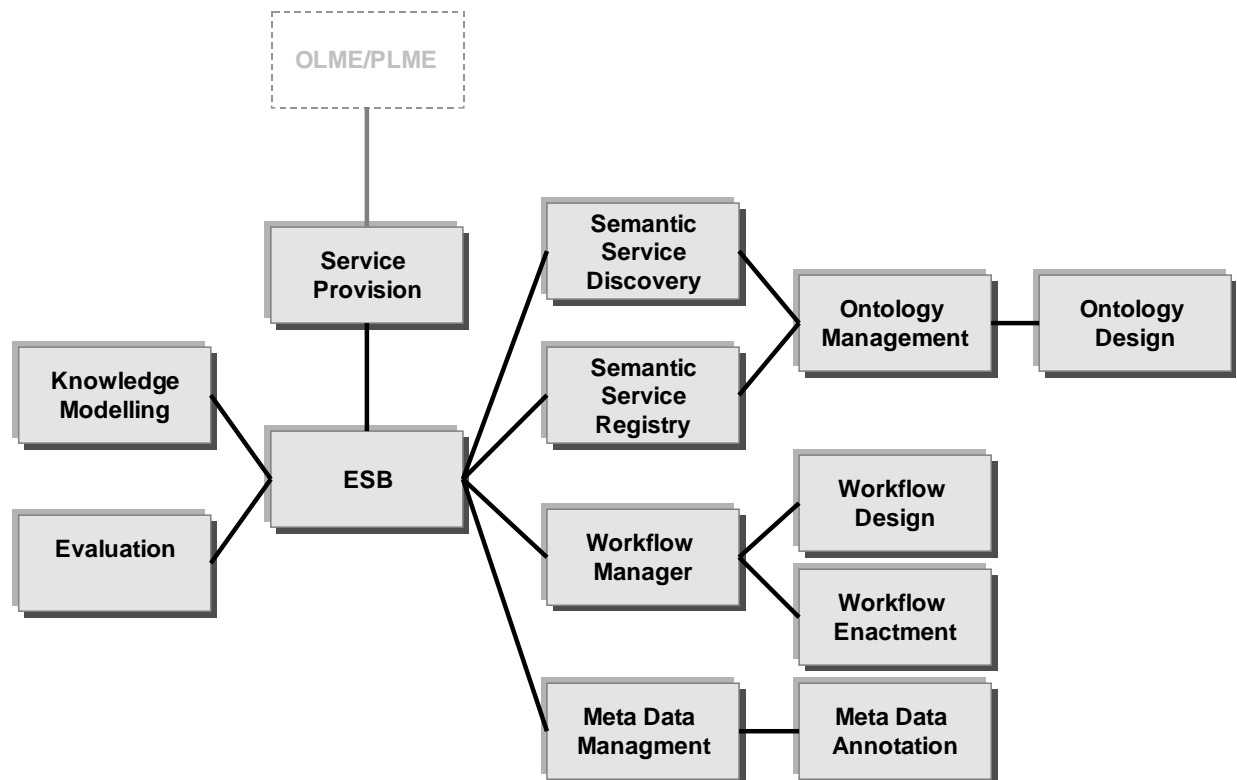


Figure 18: Logical View on the Knowledge Bus Architecture

In the following the components depicted in the above figure will be described.

Knowledge Modelling is used as the starting point for building up the MATURE system at a certain application partner. Using this component the requirements are acquired with domain expert involvement. The acquired models serve as the basis for the system configuration.

The main usage scenario for the Knowledge Worker is the use of the running system and its knowledge services once the system is setup. The OLME (see WP3) and PLME (see WP2) are only of importance for the actual use case. So they are not discussed in this section. Services are either offered through an OLME or PLME environment. In both cases the needed infrastructure for the service provisioning is the same:

The entry point to the infrastructure will be the **Service Provisioning** that is responsible for delivering a service to the requesting end user regardless of the internal structure (atomic service or workflow) and independent of its location. To fulfil a request it will forward the information to the different infrastructure elements, in order to find services or workflows, to execute them and to return the result.

The **Enterprise Service Bus** itself takes over the responsibility of transferring messages between the services taking part in the system. The bus is also responsible for the mediation between the services, which may use different message formats and communication protocols.

The **Semantic Service Registry** hosts the descriptions of the services registered at the ESB. These descriptions contain data like interface descriptions, endpoint addresses, and policies covering service level agreements, security relationships, and so on. The Semantic Service Registry covers the registration of services, the semantic annotation of them and the publishing of services.

The **Semantic Service Discovery** uses the Semantic Service Registry in order to discover the potential services/workflows for a request. To fulfil the task it will use the ontology management system to retrieve details on the concepts that were used for describing the service or to do simple reasoning.

The **Ontology Management** provides access to the semantics that are consumed by other subsystems. It is also composed of an **Ontology Design** system that deals with the creation ontologies, in the MATURE context a reference ontology that is used for the annotation of services and a meta data ontology is required.

The **Workflow Management** is used in case the request cannot be fulfilled with a single service. It is also responsible to “translate” abstract workflows into concrete ones, by searching for services for each activity of the workflow template. It can be split in the **Workflow Design** tool that is used to define the workflows and the **Workflow Enactment** that is responsible for the execution of them.

The **Meta Data Manager** holds detailed descriptions of the knowledge items, which are the smallest data elements exchanged between the sources and services. With the help of the meta data there is a common data format regarding the description of knowledge items from different sources and legacy systems. The **Meta Data Annotation** is concerned with “attaching” additional information to existing knowledge items in order to have a common format.

To enable a continuous improvement cycle of the system, the **Evaluation** system is used to identify potential problems and to show potential for improvement. The definition of the evaluation system is an ongoing task within this work package.

4.2.3 Process View on the MATURE Architecture

The process view is a process decomposition from the perspective of the integrators. There are several ways to represent this view. For the MATURE system this view concentrates on visualizing the involved components and the interdependencies between the components for the different usage scenarios of the system and the interfaces needed between the subsystems from a high level.

For the usage of the system we can distinguish between design time, execution time and monitoring. The process views for these phases will be presented in the following.

4.2.3.1 Design Time

The design time refers to the configuration of the system where services are registered and semantics are fed into the system. In this phase the following tasks can be executed:

Knowledge Modelling (see Figure 19): The first step in configuring the system is the acquisition of knowledge models representing the requirements from an end user and organisational viewpoint. This system is basically not dependent of other subsystems, but the general idea is to allow a certain degree of automation in the configuration of the system so that models created here can be reused for building the ontology or to derive the services required for a certain application scenario. Therefore the Knowledge modelling will also provide interfaces to exchange and to transform the models.

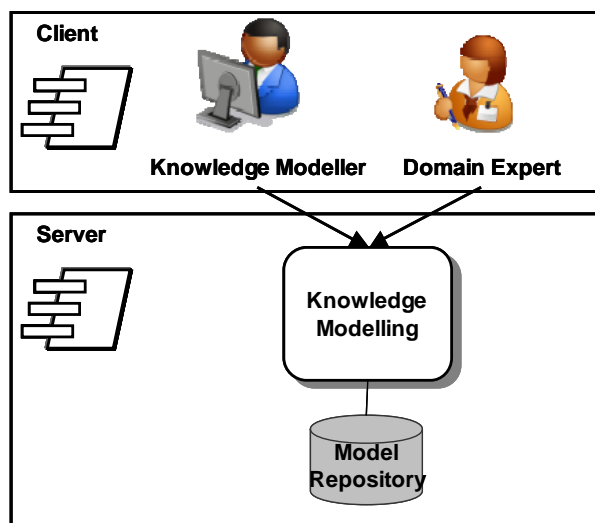


Figure 19: Process View on Knowledge Modelling

Service Registering (see Figure 20): Registering Services makes external services available within the system. Each Service and its methods are described through the annotation with concepts from the ontology. Therefore the service registry depends on the ontology management, which provides the concepts. The ontology management has to provide an interface to retrieve the concepts stored inside. The interface should allow retrieving the whole tree in e.g. XML format, but also allow retrieving the concepts stepwise by first getting the root concept and then selectively the child-concepts. After the service is registered and annotated the service can be published. From that point in time the service is available as a Concrete Service in the system and can be accessed by other components at execution time.

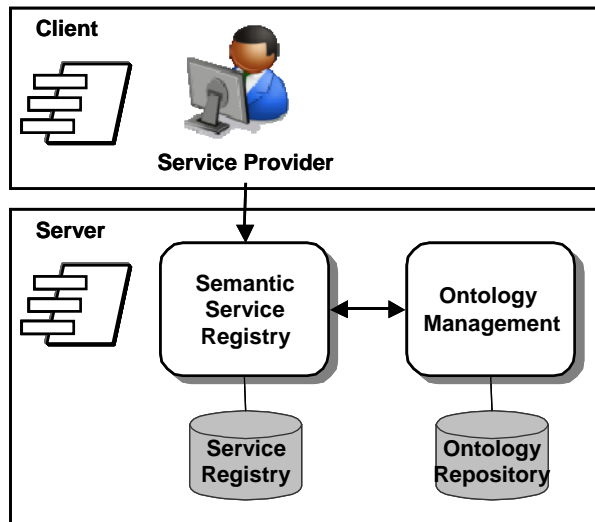


Figure 20: Process View on Service Registering

Workflow Design (see Figure 21): Designing an abstract workflow involves defining abstract activities and the control flow. Each abstract activity has to be described by ontology concepts, so that during the execution a concrete service can be discovered based on the description. That's where the ontology management component comes into play. In order to allow the annotation of the activities the concepts have to be retrieved from the ontology system. For the interface to the ontology management the same requirements as for the service registering applies. Since each abstract service is identified by a concept in the ontology and Concrete Services have been previously annotated with the same ontology, it is possible to check, for each Abstract Workflow, if each class of services has at least one concrete service already published that means if it is possible to substitute each Abstract Service with at least one Concrete Service.

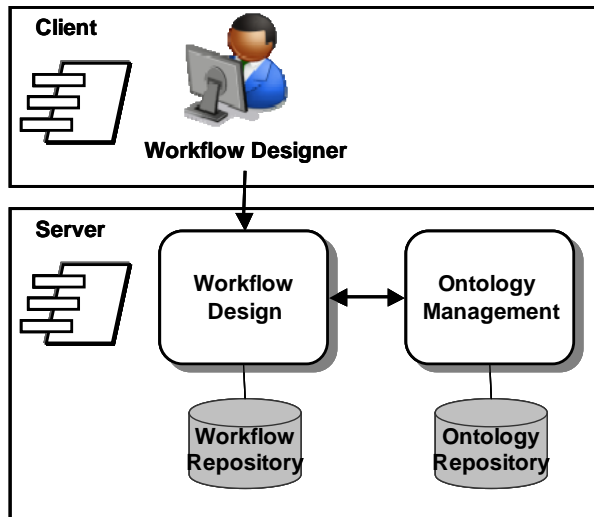


Figure 21: Process View on Workflow Design

Meta Data Management (see Figure 22): The meta-data management is necessary to describe knowledge items, which are the smallest data elements exchanged between the sources and services. Each knowledge item consists of content and ontological meta-data. Knowledge items that are described with the common meta-data format can be exchanged between the various integrated services and data sources. In order to allow the annotation of the activities the concepts have to be retrieved from the ontology system. For the interface the same requirements as for the service registering applies. The difference is the type of concepts, so the ontology system has to provide access to different ontologies.

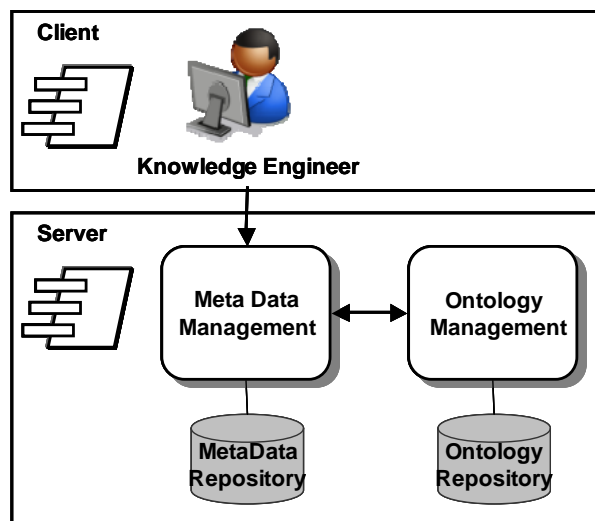


Figure 22: Process View on Meta Data Management

Ontology Management (see Figure 23): This task is usually done by an Ontology Engineer. For building the ontology there are no dependencies on other subsystems, but the domain knowledge gathered through knowledge modelling can provide valuable input. The ontology system has to store ontologies for different purposes and has to provide access to these ontologies for other components. The necessary functionality covers the typical ontology editing functionalities like creating, editing and deleting concepts and the possibility to create subclasses and instances of classes.

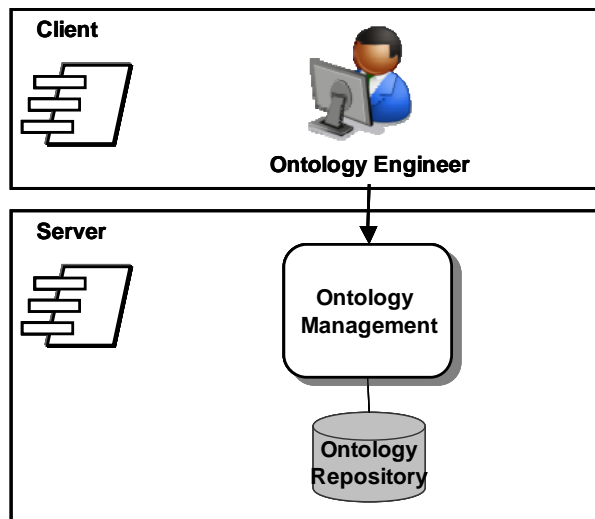


Figure 23: Process View on Ontology Management

4.2.3.2 Execution Time

The execution time refers to the actual system usage when the end user (Knowledge Worker) is actually using services. Depending on the actual service type and use case the scenarios have different process flows and complexity as depicted in Figure 24.

In the simplest case the Knowledge Worker sends a request which is processed by the Service Provisioning. The Service Provisioning forwards a query to the Semantic Service Discovery Subsystem, which itself has to query information or do some simple reasoning with the help of the Ontology system. The service discovery returns some candidate services from which the best matching is selected and provided to the user.

In a more complex case the returned service is not an atomic one, but an abstract workflow. In this case the service provisioning triggers the Workflow Enactment which takes over the responsibility of the execution. In order to resolve each activity to a service the service discovery is queried on basis of the semantic activity description.

The Ontology Management has to provide an interface to the Semantic Service Discovery, so that the Semantic Service Discovery can query for exact matching concepts, for similar concepts and for related concepts. There should be also possibility to perform simple reasoning queries (e.g. is concept A a concept related to concept B, is concept A a subclass of concept C, how similar is concept A to concept B, etc.). The Semantic Service Discovery might use this information for the ranking of the results.

The Semantic Service Discovery itself exposes interfaces to the service provisioning and workflow enactment. In both cases the Semantic Service Discovery has to provide the functionality to find a concrete service based on a semantic description. There should be the possibility to get a list of matching services ordered by relevancy and the possibility only to retrieve the best matching service. The search should allow for options such as exact match or also including similar concepts.

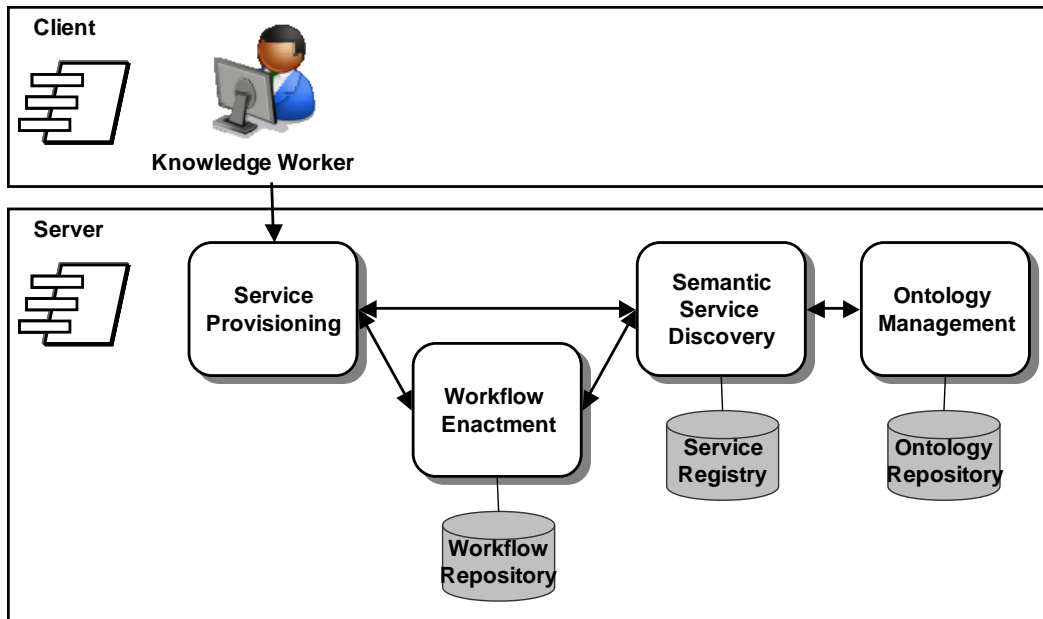


Figure 24: Process View on Service Provisioning

4.2.3.3 Evaluation/Administration

In the evaluation phase the performance of the system is monitored to identify potentials for improvement. This task is performed by the Knowledge Manager. He/She is responsible that the system fulfils the needs of the Knowledge Workers.

Apart from that, the Knowledge Manager may also manage users and their access rights for the system.

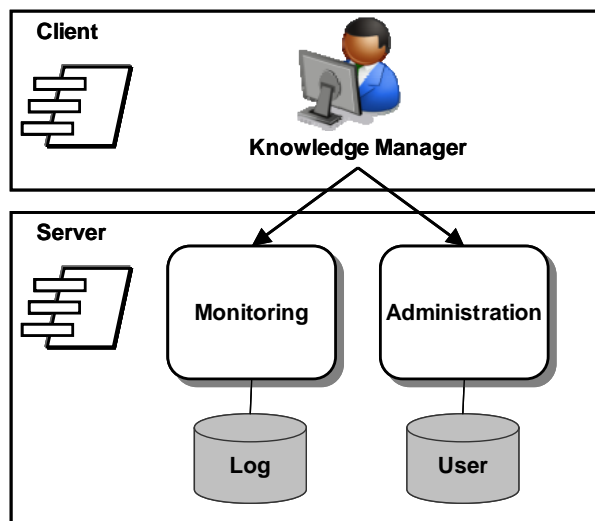


Figure 25: Process View on Monitoring and Administration

More details on the evaluation and administration part will be provided in the final version of this deliverable in PM 18.

4.2.4 Development View on the MATURE Architecture

The development view presents a subsystem decomposition that is usually done in a layered style. It refers to the viewpoint of programmers and software managers considering the software module organization, the hierarchy of layers, software management, reuse and constraints of tools.

As Figure 26 depicts, as in the process view we distinguish between the design time, execution time and evaluation /administration. The description of the subsystems corresponds to the one in the logical view, but this view additionally organized the different subsystems according to layers and visualizes the involved data sources.

The GUI layer is the top-level layer containing the user interface and responsible for user request and response handling. The application layer is pulled out from the GUI layer. It controls an application's functionality by performing detailed processing. The data layer provides the data storage and the data access for the result. Giving data its own tier also improves scalability and performance.

Finally, the External Services provide the services (PLME, OLME and Maturing services) that will be orchestrated by the Core System (the Knowledge Bus) to satisfy an End User request.

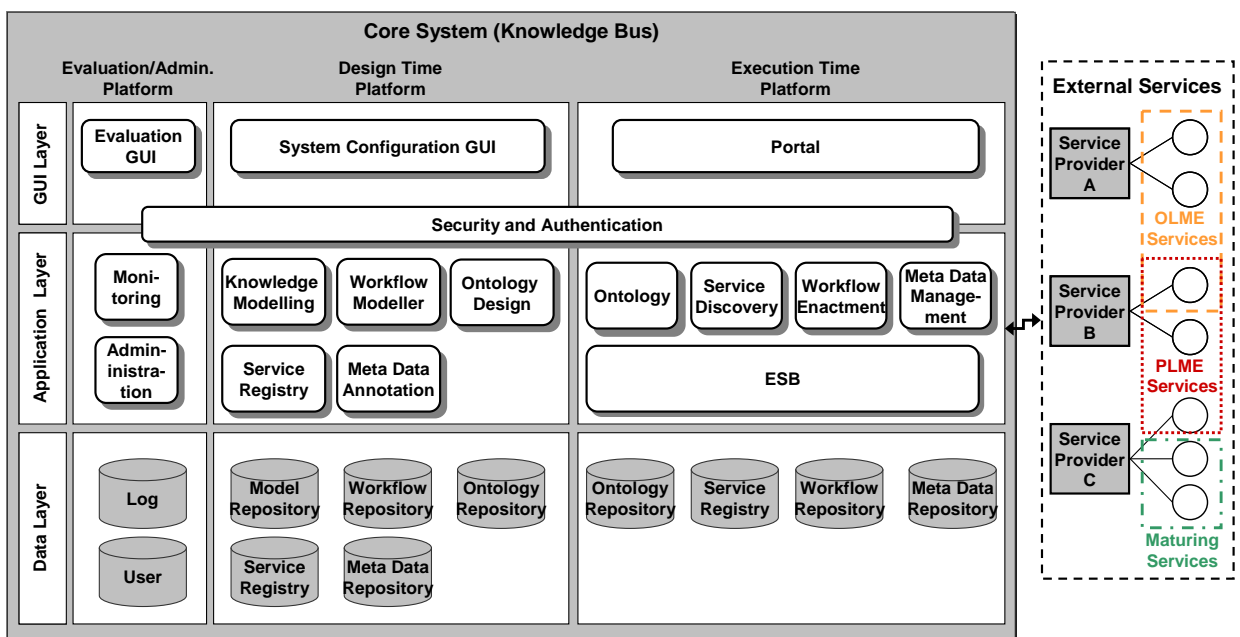


Figure 26: Development View on the Knowledge Bus Architecture

4.2.5 Physical View on the MATURE Architecture

The physical view provides the mapping of the software to the hardware from the System Engineer's viewpoint. It considers non-functional requirements regarding to underlying hardware (Topology, Communication) that can be represented in different forms.

This section is seen as input for the further WP5 tasks. Especially it is seen as input for task 5.4 where all components of the MATURE system are deployed at the application partner's sites to prove the applicability of MATURE in a real-world environment. The refinement of the system deployment is a task that lasts until the end of the project, when all components are fully developed, deployed and the platform is analysed. Nevertheless some preliminary considerations on the future deployment can be done at this stage. This view will show how, at a later stage, the system deployment will be done at the application partners. At this stage Figure 27 just presents a very general view on the system and its integration at the application partners. Here the integration with existing enterprise systems and legacy databases has to be addressed.

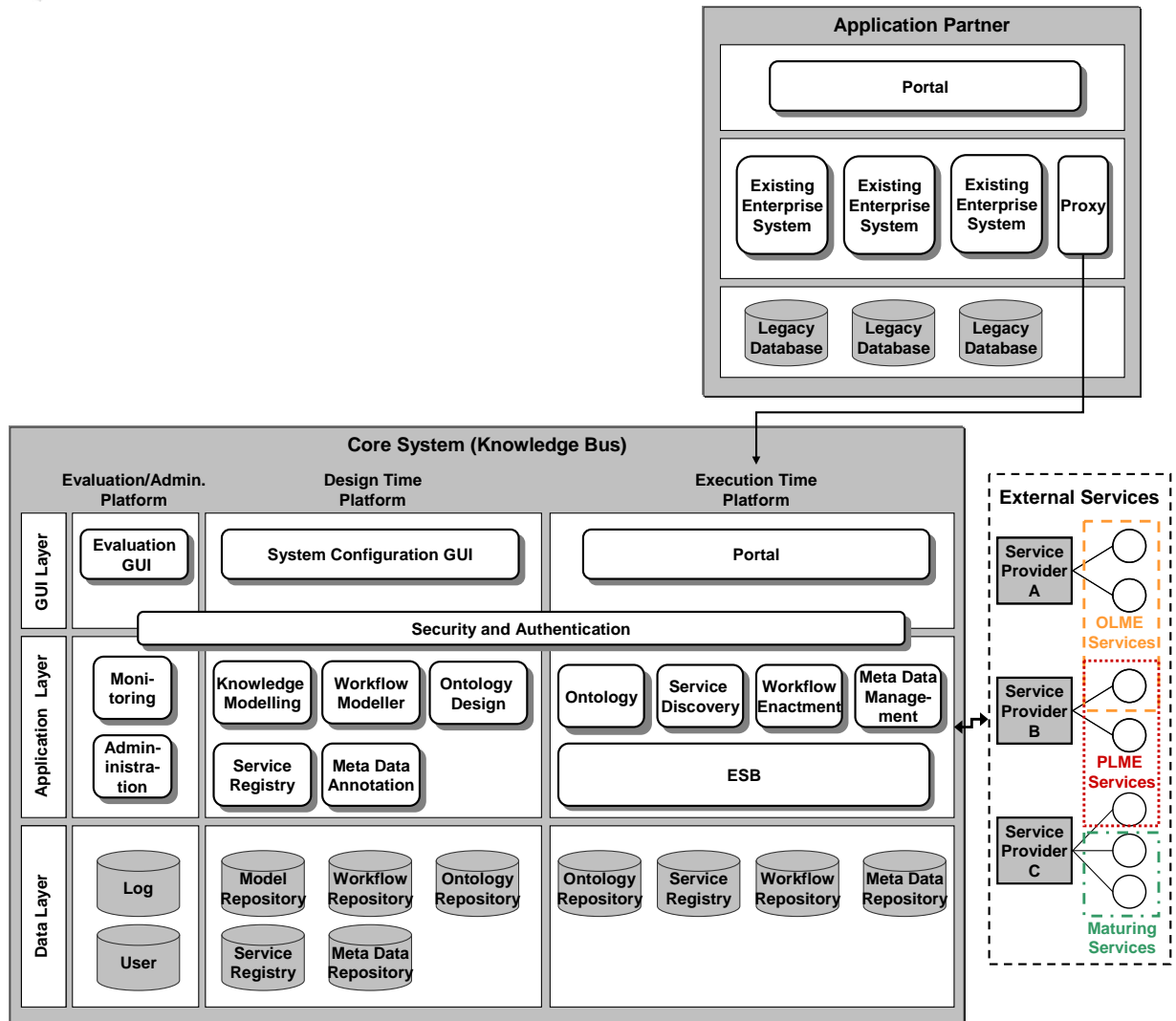


Figure 27: Physical View on the Knowledge Bus Architecture - Possible Integration at an Application Partner

Some high level choices have already been done in the early phases of the project and can be summarized as follows:

- The system is based on different logical platforms: the design platform, the execution platform and the evaluation/administration platform
- For the different components of the system, different technologies are used
- Deployment will be based on a web, multi-tier architecture
- Scalability is one of the most important features of the deployed system
- A SOA approach is followed by the MATURE system

Following the service oriented approach the subsystems identified in the development view can be deployed individually on different physical servers.

Figure 28 provides an overview of the deployment of the MATURE system. As the figure depicts and, as the MATURE system follows the SOA approach, the integrated services may be deployed distributed at different places. Services provided and deployed at the MATURE technical partners e.g. in Vienna, Graz or Olten have to be accessed by the users of the system. All the services need to be deployed in a secure environment that is trusted by all involved partners – service providers as well as end users.

As introduced in 3.2.4 a solution to achieve such a distributed but trusted environment is Shibboleth (Shibboleth, 2009). Shibboleth created an architecture and open-source implementation for a federated identity-based authentication and authorization infrastructure based on the Security Assertion Markup Language (SAML) (SAML, 2009). The federated identity allows information about users in one security domain to be provided to other organizations in a federation. This enables cross-domain single sign-on and removes the need for content providers to maintain user names and passwords. Identity providers (IdPs) supply user information, while service providers (SPs) consume this information and get access to secure content. Shibboleth is proposed because it is a powerful open source solution and has a huge supporting community.

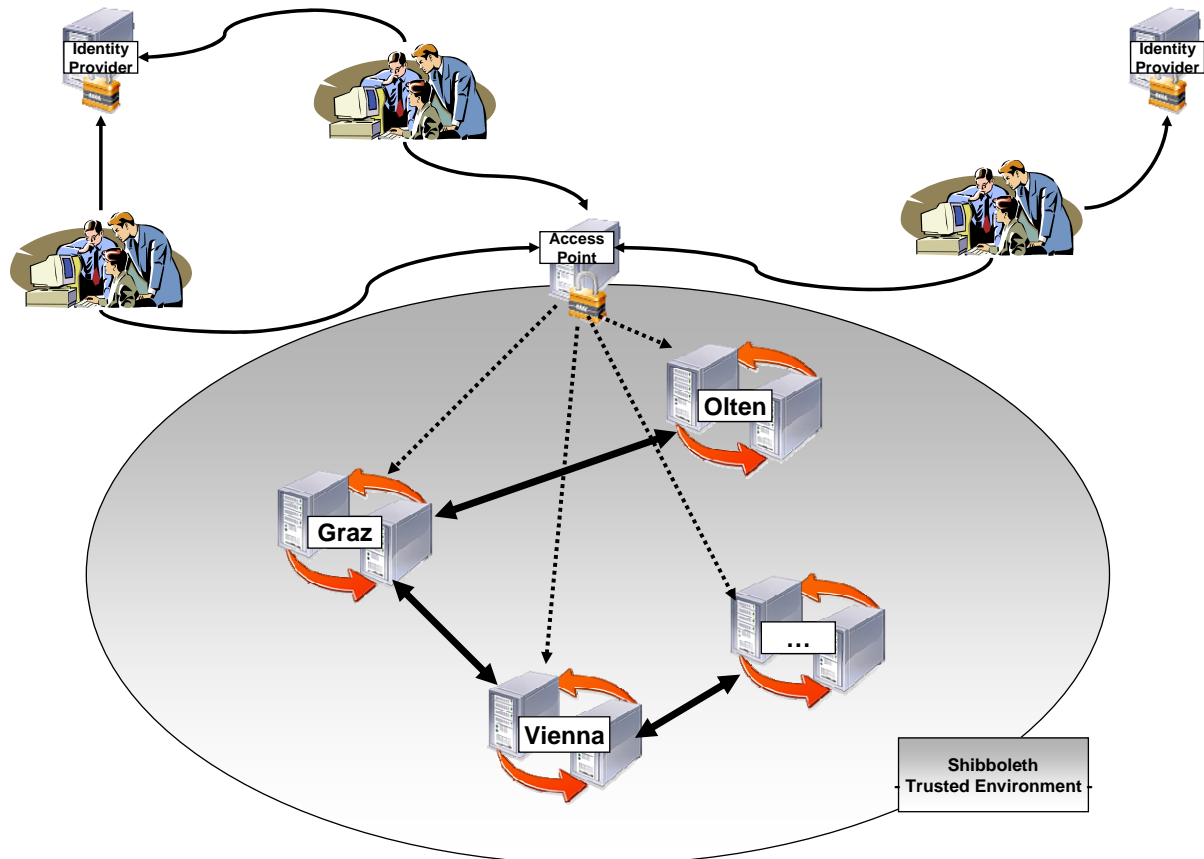





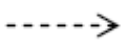
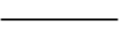
Figure 28: Deployment of the MATURE System in a Trusted Environment

4.2.6 Scenario View on the MATURE Architecture

The scenario view aims to bring the previous views together using UML (OMG UML, 2009) use case diagrams. Use case diagrams are used because they are a common method to identify and document requirements bridging the gap between business requirements and technical implementation (Larman, 2005). There are three different formats and levels of formality to describe use cases. A brief summary is used for requirement analysis. A casual description is used to show different scenarios. A fully dressed use case description is used to show all necessary steps and variants. Within this section the use cases will be described briefly on a high level. The description is technological neutral and will not investigate on exchange formats and interaction mechanisms. This will be specified in chapter 5.

Table 7 shows an excerpt of the UML Use Case Diagram notation introducing the elements that will be used in the following to describe the scenarios.

Table 7: UML Use Case Diagram Notation

<i>Element</i>	<i>Icon</i>	<i>Description</i>
actor		<p>An “actor” describes a role, which participants take concerning to the system. In modelling the system interface it is not important which concrete persons make demands. All participants are divided into groups with the help of their demands. A role is assigned to every group. Different participants within a group who make the same demands, have the same role and are modelled with the help of a single symbol.</p> <p>An actor can be a human actor, but also a computer system. Therefore an alternative notation can be used.</p>
use case		<p>A “use case” is the specification of a set of actions performed by a system, which yields to an observable result, which is typically of value for one or more actors or other stakeholders of the system.</p>
system boundary		<p>With the help of the “system boundary” use cases that logically belong together can be grouped.</p>
dependency		<p>The “dependency” relation is used to depict that a client-element depends on the supplier-element and that a change in the supplier affects the client. This relation is depicted as a broken line from the client to the supplier.</p>
association		<p>The “association” relation is used to relate a use case to the involved actors.</p>

In following sub-sections the design time, the execution time and the evaluation/administration platform of the MATURE system will be described in further detail by introducing scenarios.

4.2.6.1 Design Time Scenarios

Design time services are used by the Knowledge Manager or the Knowledge Worker to externalize their implicit knowledge. Therefore flexible and easy-to-use editors and design tools will be provided. The services of the design environment are accessible by execution time services through online interfaces using Web-Service technology and therefore allow “live” update and change of the MATURE system. In the following the design time scenarios will be introduced.

Knowledge Modelling Scenario

As already mentioned before, the Knowledge Modeller and the Domain Expert analyse the situation at the application partner before configuring the system using a model based approach (see section 5.1 for further details). The objective of this building block is to give full read and write access to the model repository for human end users, according to their access rights. The main use cases are depicted in Figure 29:

- **Acquire Models:** A Knowledge Modeller uses methods like document inspections (e.g. laws, regulations), existing models or interviews with Domain Experts and manually models within a graphical modelling tool.
- **Design Models:** This use case enables the Knowledge Modeller and the Domain Expert to refine and create new models.
- **Analyse Models:** Created models can be analysed to identify potential for improvement.
- **Search Models:** The model search provides a quick possibility to navigate to a model containing objects with the defined keywords, or browsing the models according to the term searched for.
- **Import/export models:** the import and export of models allows the exchange of models with external systems or between different installations of the system.

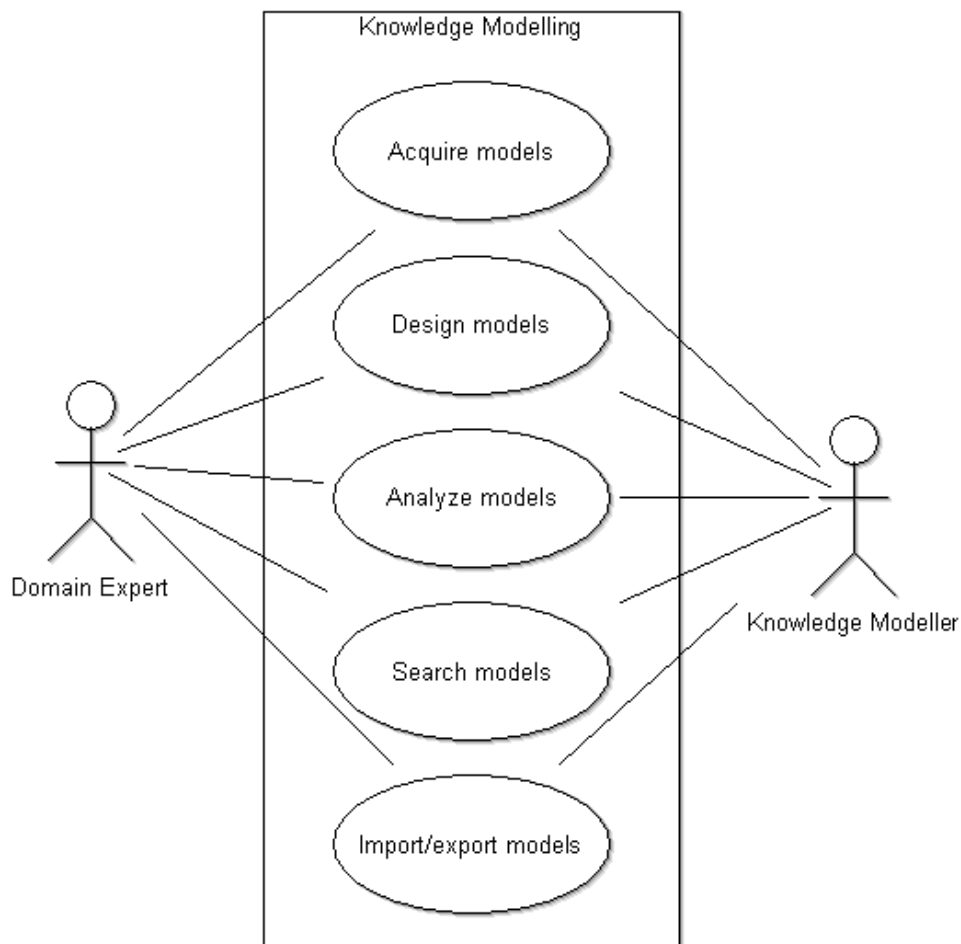


Figure 29: Knowledge Modelling Scenario

Semantic Service Registry Scenario

The Service Provider uses the Semantic Service Registry at design time for the following use cases (see Figure 30):

- **Register Service:** services are registered through the Semantic Service Registry Module using strictly defined templates. The Service Provider is using this template to provide all necessary information needed to register a new service with the MATURE System. After the successful completion of this step, the service has been registered but is still not accessible for the discovery and usage within the Execution Time Platform. For this it has to be annotated and published.

- **Annotate Service:** already registered services can be annotated with ontology concepts in order to be discovered by the Semantic Service Discovery later on. Existing annotations can be changed as well.
- **Publish Service:** After the service was registered and annotated the service provider has to publish the registered service to make it available for discovery and usage within the Execution Time Platform.
- **Edit Service:** The Service Provider may want to change the service description, its parameters or wants to unpublish or deregister the service.
- **Show/browse Services:** All services are listed, for the selected service the existing annotations are shown. It is also possible to display the services that are annotated with specific concepts.

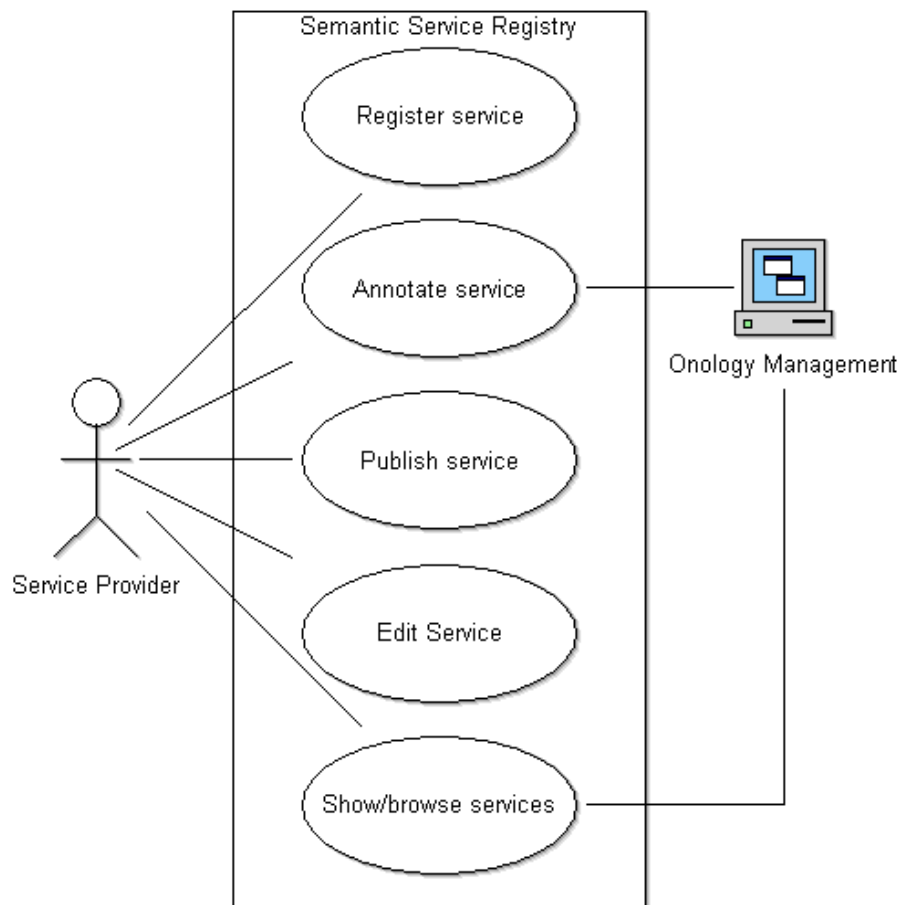


Figure 30: Semantic Service Registry Scenario

Workflow Modelling Scenario

The Workflow Modelling subsystem has to provide a set of functionalities to allow the definition of workflows. The use cases as depicted in Figure 31 are the following:

- **Import / Export Workflow Models:** The import and export of existing workflow models, in order to allow the exchange with existing systems and for deployment to the execution time system.
- **Edit Workflow:** This use case summarizes the creation of a new abstract workflow or the manipulation of an existing workflow by defining the activities and the control flow.
- **Remove Workflow:** This functionality allows the modeller to remove deprecated processes.

- **Annotate Workflow:** Activities of existing (abstract) workflows are annotated with concepts from the ontology, so that services for the activities can be discovered during execution time.

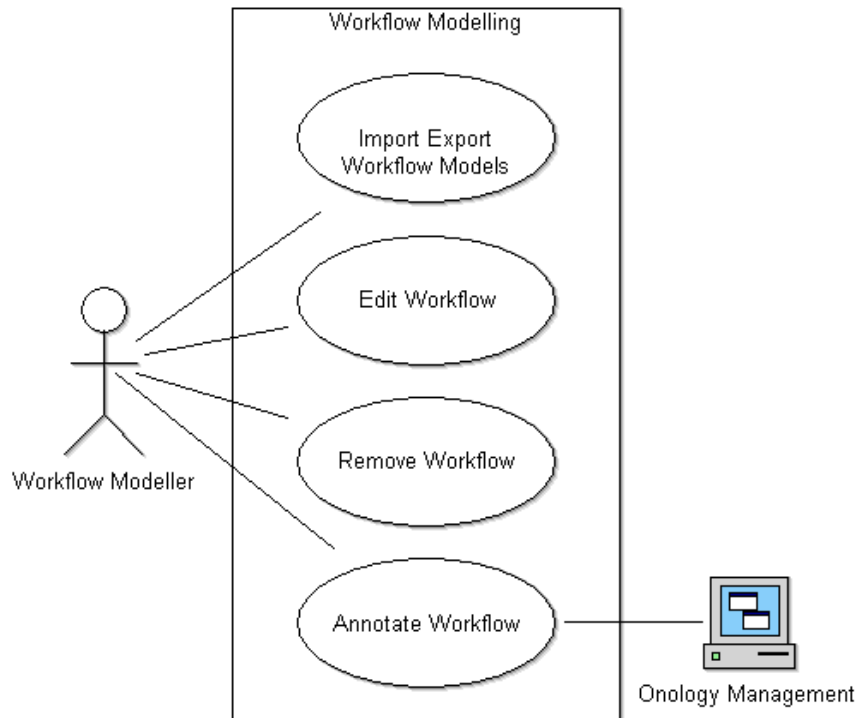


Figure 31: Workflow Modelling Scenario

Ontology Management Scenario

As presented in the following sections the Ontology Management component is accessed by other design time components for the annotations of services or abstract workflows. The Ontology Engineer and other components interact with the Ontology management system in the ways as depicted in Figure 32:

- **Import/Export Ontology:** The Ontology Engineer is provided with import and export functionalities of the ontology into an appropriate ontology language.
- **Edit ontology:** This use case covers adding, editing or removing concepts of the ontology. It provides the Ontology Engineer with functionalities for the maintenance of the ontology.
- **Provide ontology:** access to the ontology is provided to other accessing components (Knowledge Modelling, Workflow Modelling, Semantic Service Registry) that require access to the ontology e.g. for annotation or discovery.

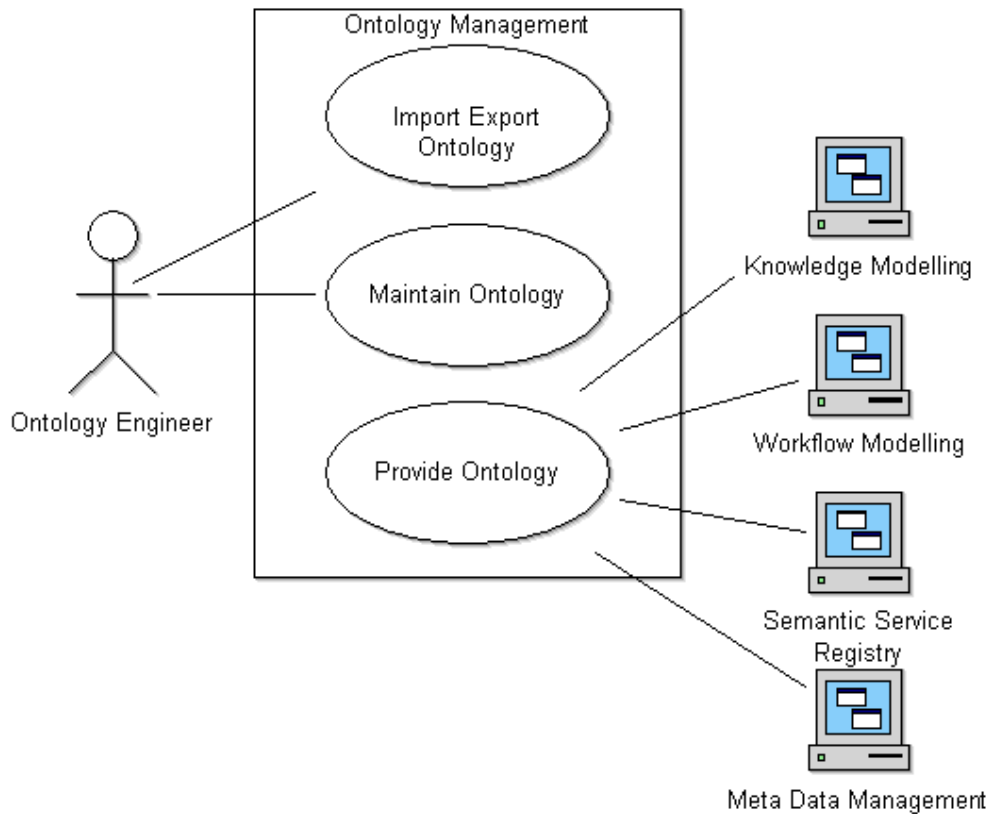


Figure 32: Ontology Management Scenario

Meta Data Management Scenario

The meta-data management is necessary to describe knowledge items, which are the smallest data elements exchanged between the sources and services. Knowledge items consist of contents and ontological meta-data. The system has to provide the Knowledge Engineer and the accessing systems with the functionalities as depicted in Figure 33:

- **Annotate Knowledge Item:** A knowledge item is extended with meta data information by annotating it with attributes like Author, Date, Maturity level, etc. Therefore the Ontology Management has to be accessed, which provides the concepts for the annotation.
- **Provide Meta Data:** Provides information about already existing annotations for each knowledge item to accessing external services (PLME, OLME or Maturing Services).

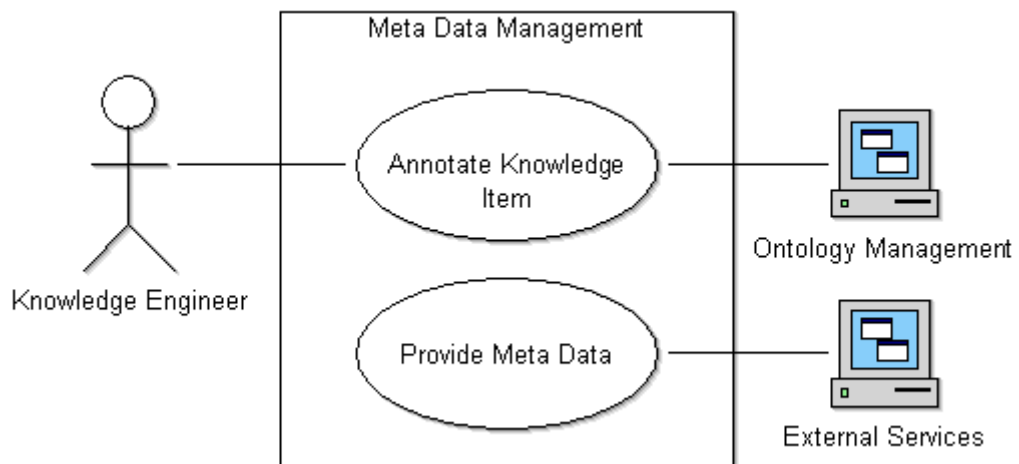


Figure 33: Meta Data Management Scenario

4.2.6.2 Execution Time Scenarios

This part of the MATURE system consists of services integrated within the running MATURE system. They use information from the design environment. These services execution is triggered and used by the Knowledge Worker. From a high level view the Knowledge Worker uses the system as depicted in Figure 34.

- Find service: The Knowledge Worker wants to find a service that conforms to his needs. He has to provide a service description which is then used to query the service discovery. Further systems (Semantic Service Registry and Ontology Management) interact with the Service Discovery to provide appropriate search results.
- Invoke service/workflow: This use case can be either a follow up of the previous use case, meaning that the service is invoked after discovering it or the Knowledge Worker might already know which service he would like to use and therefore he only wants to invoke the service or the workflow. The invocation of a service might be simple in the case of an atomic service or rather complex if the service is an abstract workflow.

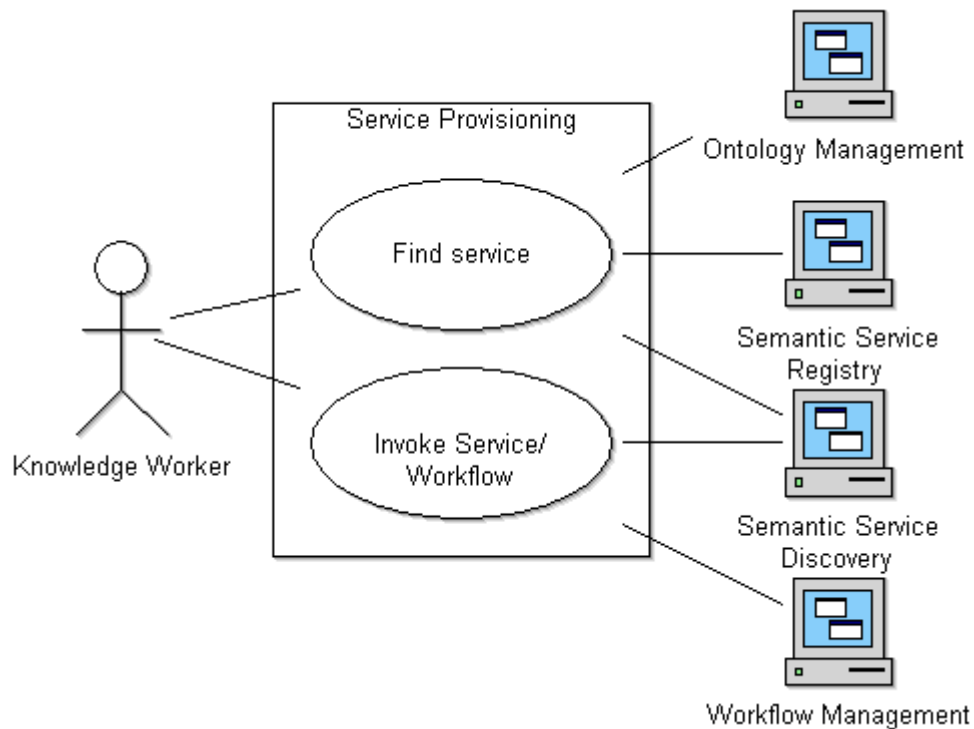


Figure 34: Service Provisioning Scenario

4.2.6.3 Evaluation and Administration Scenarios

The Evaluation and Administration environment provides different services that are both relevant for the design and for the execution time. It provides basic functionality for the user administration and the monitoring of the system.

The main use cases for the Knowledge Manager are:

- Administrate user and rights: This involves the creating, editing or removing of users and defining their access rights. Users may only have access to a limited set of services or be able to use the full set. There are also more fine grained access policies for the data (models, ontologies, knowledge items).

- **Show log:** This functionality allows access stored log data. It provides a view on the raw data of the log which might be necessary to track a certain issue in details. This functionality is supposed to be used by an expert user.
- **Monitor system:** In contrast to the previous use case the monitoring system also provides a more high level view on the system. It visualizes data from different sources like log data (hard facts), or questionnaire data (soft facts) in an aggregated form. This allows the Knowledge Manager to identify potentials for improvement.

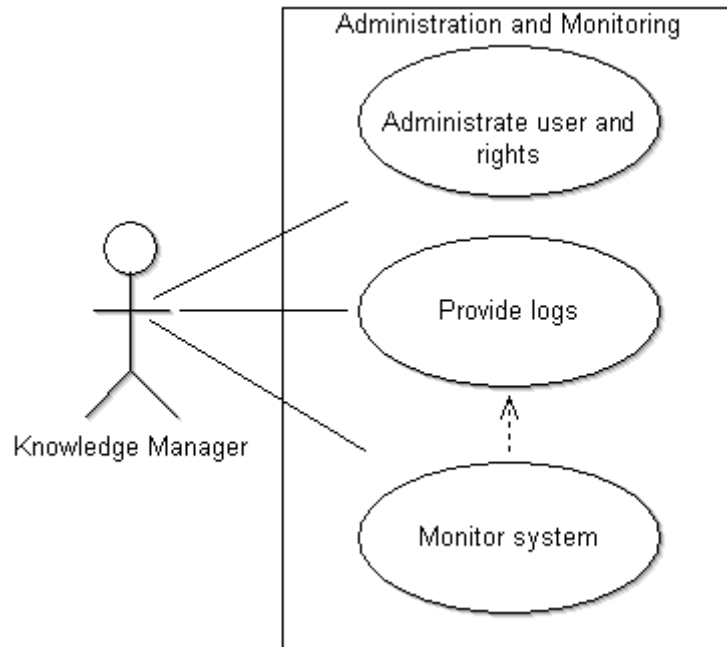


Figure 35: Administration and Monitoring Scenario

This chapter provided a first high level view (from different perspectives) on the MATURE system following a bottom up and a top down approach. The MATURE system has to integrate wrapped services of already existing functionalities as identified in WP2 and WP3, PLME services developed in WP2, OLME services developed in WP3, maturing services developed in WP4 and finally, co-existing knowledge sources at the application partners. To enable this integration within WP5 a central component will be developed. The next section is dedicated to this central component – the Knowledge Bus.

5 Knowledge Bus as Integration Tool

The Knowledge Bus as introduced before is the middle tier between the various knowledge sources, the wrapped services of already existing functionalities, the learning and maturing environments (PLME in WP2 and OLME in WP3), the maturing services (WP4) and the co-existing knowledge sources at the application partners.

Figure 36 provides a conceptual overview on the knowledge bus as central component of the MATURE system. It consists of two layers, the infrastructure layer and the integration layer. The infrastructure layer provides the basic functionality for the registration, execution and orchestration of services. The integration layer enhances the basic functionality by introducing semantics to describe services and messages exchanged between the services and involves the domain experts in the acquisition of system requirements. The two layers are influenced by MATURE-specific (Model Orientation, Process Orientation, Knowledge Management vs. Knowledge Work) and SOTA concepts (Service Orientation, Virtualisation, Semantics and Security & Trust), which were already introduced in chapter 3.

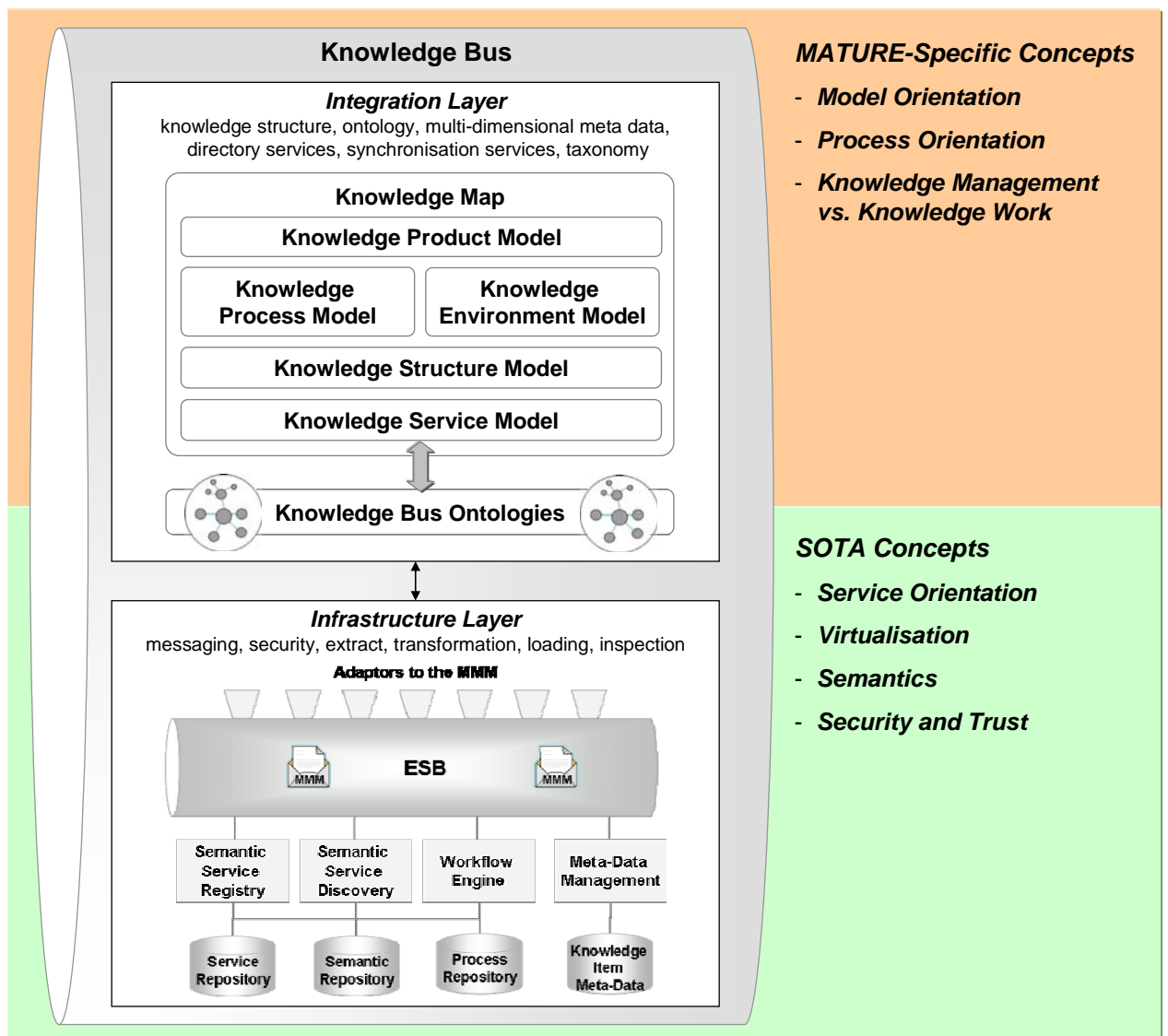


Figure 36: Layers of the Knowledge Bus

In the following section each of the two layers of the Knowledge Bus will be described in more detail both from a conceptual and from a technical perspective.

5.1 Knowledge Bus Integration Layer

This section focuses on the integration layer of the Knowledge Bus. First, details on the conceptual view will be provided, followed by a implementation view on this layer.

5.1.1 Conceptual View on the Knowledge Bus Integration Layer

Figure 37 provides an overview on the integration layer of the Knowledge Bus. The challenge of the integration layer of Knowledge Bus is to bridge the gap between the demands of the business-oriented end user and the technology-oriented service developer, thus the alignment of the conceptual and technical level.

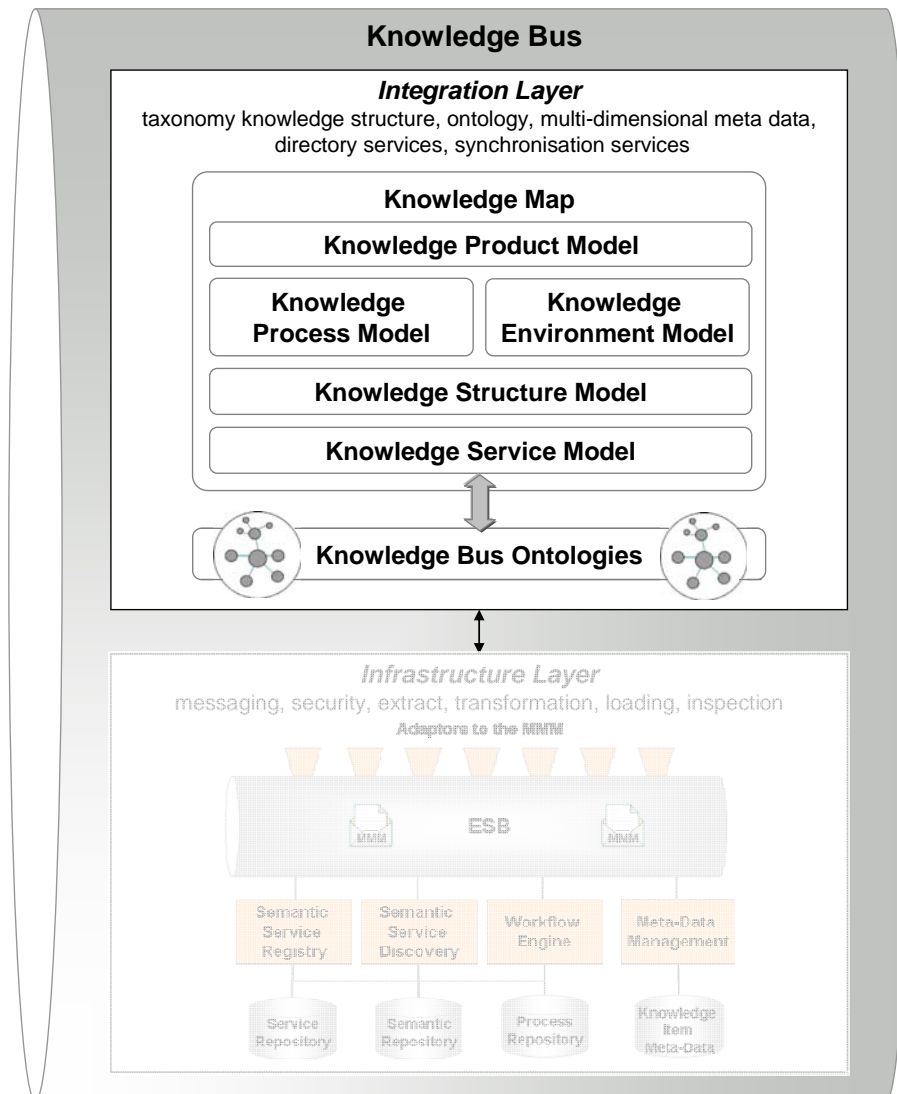


Figure 37: The Integration Layer of the Knowledge Bus

In the following the an overview on knowledge modelling to gather the requirements of the business-oriented end user is provided, followed by an introduction to ontologies for the knowledge bus.

5.1.1.1 Knowledge Modelling

This section introduces the model-based approach that has been selected to realise the Knowledge Bus and point out some basic principles. Model-based approaches became a base technology over the last years, as they proofed to simplify complex real situations to make it understandable for humans and enable a formalisation to be interpreted by machines. Before the modelling language has been selected to configure the Knowledge Bus, there are some initial statements required that are formulated as three axioms:

- ***Axiom 1: Knowledge Bus can be configured model-based:*** The first axiom states that the Knowledge Bus can be configured model-based where a model is seen as an immaterial reflection of reality into a model system for the purpose of an individual (Kühn et al, 2003). This means, that modelling languages can be used to describe the system and all related aspects in MATURE within a modelling system to reduce the complexity, enable flexibility for end user changes and provide formalisms for machine interpretation. The model boundaries are identified according contextual and modelling restrictions. Model objects concerning the same context are grouped in model types under the restriction of a reasonable number of modelling objects and modelling instances.
- ***Axiom 2: Formal model as requirement for machine interpretable models:*** The second axiom says that a model needs to be formalised for machine interpretation. This means that the models consist of business graphs, execution graphs and evaluation graphs. The business graph defines the concept; the execution graph defines the technological mapping between the concept and the IT-infrastructure whereas the evaluation graph defines the monitoring of the execution.
- ***Axiom 3: Meta model as implementation approach:*** The third axiom is to use the meta model concept for the implementation of the model. The new models are therefore defined on three layers. The meta 2 model layer defines the basic modelling constructs defined in Cedif, MOF, GOPRR or UML Profiles. The meta model layer implements the formalised business graph, execution graph and the evaluation graph using the constructs of the meta 2 model. The integration of the meta models followed the reference patterns for meta models (Kühn et al, 2003). The third layer implements an instance of a model.

There are three different modelling scenarios that can be observed in knowledge modelling: First, the knowledge is modelled for documentation purpose. This means that knowledge is modelled to communicate between workers, find an agreement, as well as to work-out details. The goal of this scenario is to make knowledge explicit. Second, the knowledge is modelled for management purpose. This means that knowledge is modelled to ensure quality, efficiency as well as to reduce cost and time. The management scenario has the goal to identify knowledge as an object that has to be managed. Third, the knowledge is modelled for configuration purpose. This means that knowledge enables a tool and technology independent approach. The models are exported into the infrastructure where the models are seen as tool configuration. Beside the tool independency the models from two different modelling languages can be integrated, analysed, simulated and adapted. The configuration scenario has the goal to configure a technical infrastructure via models.

The above mentioned modelling scenarios are often combined. A typical approach is to start with the documentation scenario and improve the models in the second step for a management or configuration scenario. In MATURE the knowledge will be modelled for documentation and configuration purpose, but also for the purpose of management. Before the models can be specified in detail, a documentation approach is applied to find a common understanding between the MATURE partners on the available models.

In the following the knowledge management modelling language PROMOTE[®] will be introduced. PROMOTE[®] is a holistic modelling approach for process-oriented knowledge management that has been developed in the EC-Project PROMOTE (IST Project 11658) (Woitsch, 2004) and improved in the recent years during commercial and research projects. PROMOTE[®] has been successfully used and extended in the projects Akogrimo (Woitsch et al, 2006), AsIsKnown (Woitsch et al, 2007) and Brein (Woitsch and Leutgeb, 2008), as well as in the Austrian military within the central documentation department (Mak and Woitsch, 2005) and the ABC-Abweherschule, which is the school to defence against nuclear, biological and chemical weapons. This method has also been used in MATURE to document the ethnographic studies in WP1 (see D1.1 for details)

The overall goal of knowledge management is to support the business processes within an organisation. This assures that knowledge management has a direct link to performance improvements in order to achieve the business goals. Although PROMOTE[®] does not distinguish if the business process is graphically modelled or not, the overall assumption is that the provided knowledge supports a business process.

As a direct linkage between business process and provided knowledge is difficult – due to the fact that business processes are modelled with different notation and in different granularities– PROMOTE® provides an indirect coupling between business processes and knowledge products.

This entry point into knowledge management has found useful, as it provides a concrete structure for knowledge. Starting with the identification of knowledge products, there are four dimensions that describe knowledge management.

The top dimension is the knowledge product dimension that structures the way knowledge is provided to the business process.

In order to generate the knowledge product, the well known process-oriented view is applied, which sees the product as the result of process. The so-called Knowledge Management Process is identified that produce the knowledge products.

Beside the knowledge management process – that specifies the logic sequence of knowledge interactions to finally produce the knowledge product - the Knowledge Environment needs to be observed. PROMOTE® interprets skills, content-oriented roles and knowledge-depending access rights as the so-called knowledge environment.

The last dimension of knowledge is the knowledge resource. This dimension is concerned with tools, content and knowledge that is available. Typically this dimension is well supported by the ICT department, as traditionally knowledge management has been defined.

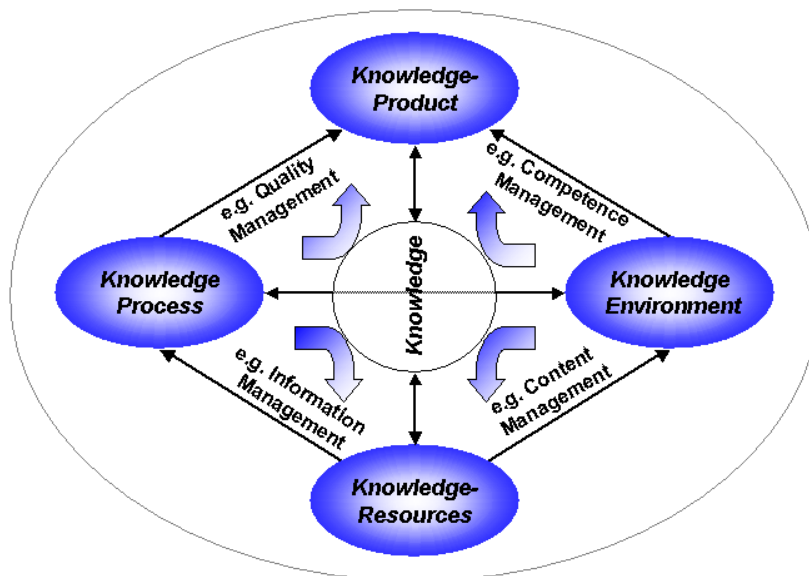


Figure 38: Knowledge Dimensions of PROMOTE®

Figure 38 introduces the four knowledge dimensions of PROMOTE® and indicates different management approaches in the area of knowledge management that can be allocated to the knowledge management dimension.

PROMOTE® has the philosophy to support knowledge management with ICT. Here it has to be pointed out that in contradiction to most of the other knowledge management approaches, PROMOTE® supports the Knowledge Manager in managing the knowledge and not the Knowledge Worker in using the knowledge. Hence the ultimate goal of PROMOTE® is to support the knowledge management in order to make the knowledge usage more efficient.

In order to support the Knowledge Manager with ICT, PROMOTE® offers a modelling language that describes the knowledge management approach. The aforementioned four dimensions are described in graphical models.

Figure 39 introduces the PROMOTE® model stack used to represent the real world at the MATURE application partners in models. On the conceptual layer there are knowledge products (Knowledge

Product Model). An example of knowledge products would be the career guidance provided by the application partner Connexions Kent. In order to maintain and provide such a knowledge product different knowledge management processes (Knowledge Process Model) are executed. For instance an employee at Kent who gives career guidance has to search for relevant information. Each knowledge management process consists of several activities. These activities can be classified (Knowledge Structure model) using the codes introduced in WP1. Knowledge services (Knowledge Service Model) e.g. a full text search or yellow pages may be useful to support certain activities. As the knowledge services are also classified using the codes, candidate services to support certain activities can be identified.

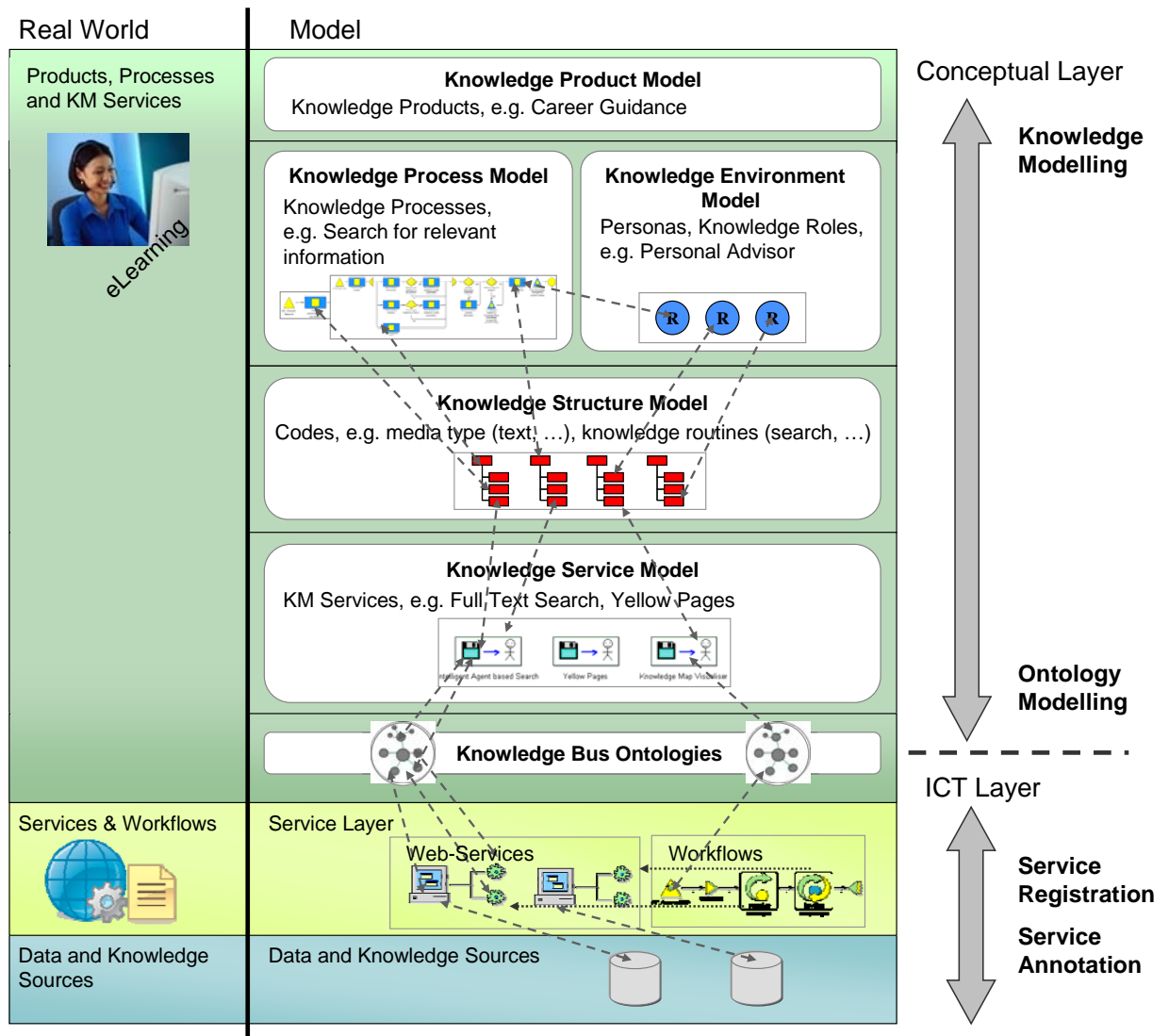


Figure 39: From the Conceptual to the Technical Level

On the ICT (technical) layer each of the services as identified on the conceptual layer has to be provided in order to be executable at execution time. This may be a Web-Service or an orchestration of Web-Services (workflows). In order to align the conceptual and the ICT layer a semantic layer is introduced, which is used to annotate services on the conceptual level as well as executable services on the technical level using the Knowledge Bus ontologies.

5.1.1.2 Knowledge Bus Ontologies

The alignment of the conceptual level which describes the application scenario on a semi-formal level and the technical and thus executable level is the challenge to be solved by the Knowledge Bus. Therefore semantics will be introduced in form of the Knowledge Bus ontologies. As depicted in Figure 40 different

integration patterns (Kühn et al, 2003) (based on semantic technologies) are feasible to address this challenge. The *extension pattern* enlarges a part of a meta-model with new concepts to broaden the expressiveness of the meta-model. The *reference pattern* can be regarded as a hyperlink which references parts of another meta-model. Applying the reference pattern results in navigation paths from one meta-model to another independent meta-model. The *transformation pattern* provides rules to transform parts of source meta-models to concepts provided by a target meta-model.

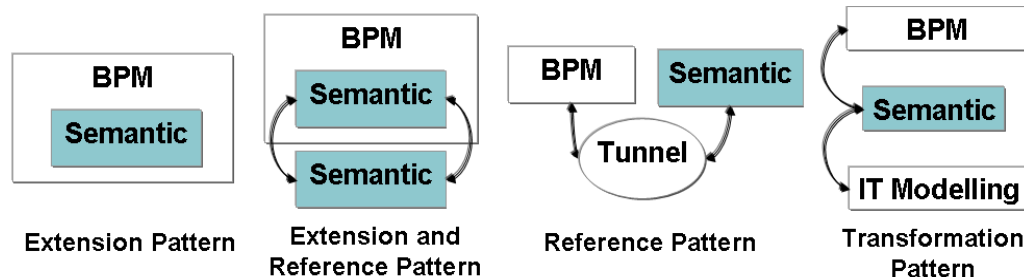


Figure 40: Integration Patterns

Within the MATURE project the most appropriate integration pattern of the integration of the conceptual and ICT layer will be selected. Details on the selected integration pattern will be provided in the final version of this deliverable in project month 18.

After providing a conceptual overview of the integration layer in this section, the following section deals with an implementation related view on this layer.

5.1.2 Implementation View on the Knowledge Bus Integration Layer

This section introduces the implementation view on the knowledge bus integration layer. The Knowledge Bus integration layer enables a model-based configuration and initialization of MATURE.

Three implementation approaches will be followed:

1. The meta-modelling approach will be used to implement a method-independent and flexible modelling environment,
2. A service-oriented modelling framework will be used, in order to realise a flexible modelling infrastructure as well as
3. Web-application technology will be used to realise the modelling framework and to enable a transparent access to the modelling system.

These implementation approaches will be followed by proposed model-based knowledge management design framework as depicted in Figure 41. It is implemented following the reference architecture of the ESB (which will be introduced in 5.2.2.1).

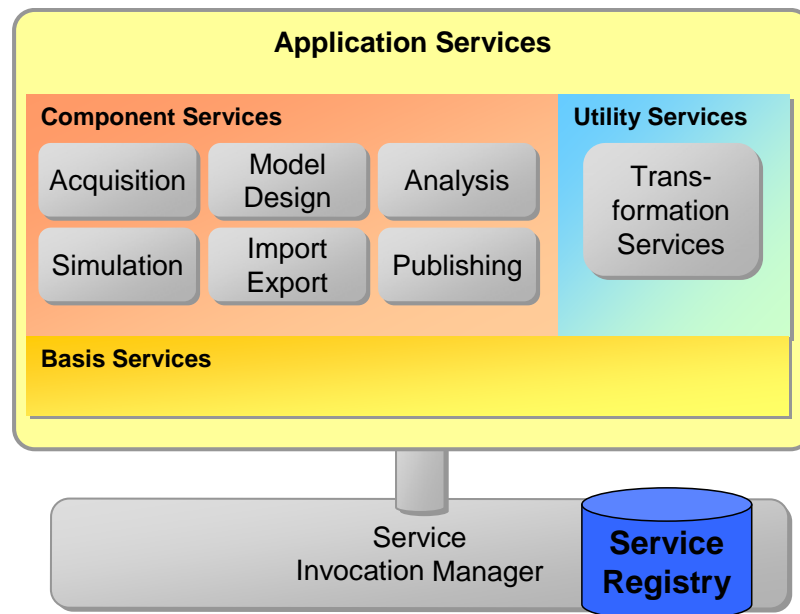


Figure 41: Model-Based Knowledge Management Design Framework

The Service Invocation Manager is the implementation of the communication of the communication protocol, the application services communicate over. The Service Registry holds the descriptions of the deployed application services.

The application services themselves are the services supporting the work in the domain, in this case the graphical knowledge modelling. They can be further distinguished using the categories “Basis Service”, “Utility Service” and “Component Service”.

Basis Services offer therefore access to the model and the meta model repositories etc. Basis services are chosen based on the specifications of the OGSA framework (Globus OGSA, 2009). The basis services offer standard functionality necessary for graphical modelling, which can be used by component services.

Utility services are more general in their functionality and are not necessarily specific for the graphical modelling domain. Such services may be sorting algorithms, XML parsers or transformation mechanisms. Utility services may be used by various component services.

Component services comprise all modelling services that can actually be used by an end user and can be regarded as autonomous, as they offer a benefit to the user even if deployed autonomously. They comprise a well-defined set of functionality, which may or may not be based on basis- and utility services. The component services comprise services for the acquisition, design, analysis, simulation, import/export, publishing of the knowledge base. This includes knowledge models and ontologies.

This section presented details on the integration layer of the Knowledge Bus. In the following the infrastructure layer will be presented.

5.2 Knowledge Bus Infrastructure Layer

This section focuses on the infrastructure layer of the Knowledge Bus. Details on the conceptual and on the implementation view will be provided in the following.

5.2.1 Conceptual View on the Knowledge Bus Infrastructure Layer

Figure 42 gives an overview of the infrastructure layer of the Knowledge Bus. The infrastructure layer provides basic functionality to enable the execution of knowledge services. As the figure depicts the main components of this layer are the enterprise service bus, several adapters, the semantic service registry, the semantic service discovery, the workflow management and the meta-data management component.

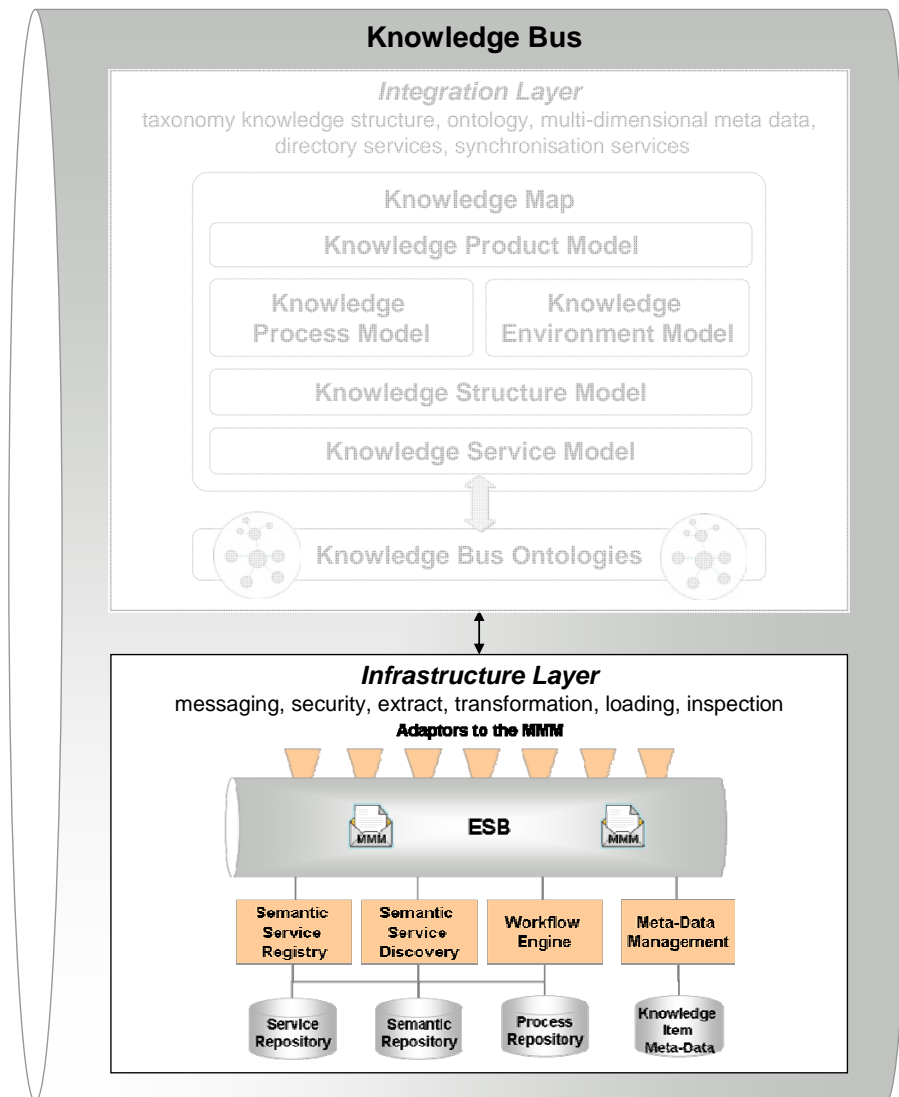


Figure 42: The Infrastructure Layer of the Knowledge Bus

This section provides an overview of the components of the knowledge bus from a conceptual level, before the following section provides further implementation details. In the following each component of the infrastructure layer will be briefly described before the following section provide more details on the implementation view.

Enterprise Service Bus: The Enterprise Service Bus is the part of the infrastructure which is in charge of transporting the messages from senders to receivers. It offers interfaces for attaching Adapters, which allows the mediation between different message formats used by the services. In order to identify senders and receivers the Enterprise Service Bus needs to cooperate closely with the Semantic Service Registry and the Semantic Service Discovery.

Adapter: The Adapters are pieces of software, attached to the Enterprise Service Bus. Their purpose is to translate the different services' messages to and from a common format, which can be transmitted through the Enterprise Service Bus. The following transformations are possible: among Web-Services, between legacy applications and Web-Services and between human services and Web-Services.

Semantic Service Registry: The Semantic Service Registry holds the descriptions of the deployed services. The descriptions contain not only syntactic information like in pure WSDL, but also semantic information used to identify the functionality of the services. It provides functionalities to register, annotate and publish services.

Semantic Service Discovery: The Semantic Service Discovery operates on the semantic data in the descriptions of the services. Its task is to identify services matching a request. It informs the client-services whether there are provider-services that can deliver the functionality required. Moreover it decides which service to pick, if there is more than one available.

Workflow Management: The Workflow Engine serves the purpose of creating new functionality by orchestrating available services. For this reason the Workflow Engine must interact with the service discovery component.

Meta-Data Management: The meta-data management is necessary to describe knowledge items, which are the smallest data elements exchanged between the sources and services. Each knowledge item consists of content and ontological meta-data. Knowledge items that are described with the common meta-data can be exchanged between the various integrated services and data sources.

After this section provided a conceptual overview on the infrastructure layer the following section a look at the implementation view.

5.2.2 Implementation View on the Knowledge Bus Infrastructure Layer

This section provides an overview on the infrastructure layer from an implementation view. As already mentioned before, different components are necessary to enable the registration and execution of services. A high level overview has already been provided in chapter 4.

5.2.2.1 Enterprise Service Bus

The Enterprise Service Bus (ESB) is based on the message bus concept. An application or service that sends messages through the message bus must prepare the messages so that they comply with the type of messages the bus expects (see the section 5.2.2.2). Similarly, an application or service that receives messages must be able to understand (syntactically, although not necessarily semantically) the message types. If all applications or services in the integration solution implement the bus interface, adding applications/services to or removing applications/services from the message bus incurs no changes. This is realized as all services that will be integrated will implement a common message model, which will be introduced in the following when introducing the adaptors.

An ESB is based on the message bus concept, but provides additional infrastructure services (e.g. transformation and routing). It is the implementation backbone for a loosely coupled, event-driven SOA. Further details on the ESB have been provided in the conceptual background section 0.

An open source ESB will be used in the MATURE project, as it will provide enough flexibility to be extended when needed during the course of this work package. The open source jBoss ESB has been selected, Figure 43 depicts its architecture. It uses a flexible architecture based on SOA principles such as loose-coupling and asynchronous message passing, emphasizing an incremental approach to adopting and deploying a service oriented infrastructure (SOI). It is a pluggable architecture where infrastructure services (e.g. transformation, routing or event notification), business services and event listeners and actions can be integrated. The figure depicts, which parts of the ESB architecture are realized now (highlighted in blue), are provided by partners (highlighted in grey) or are seen as developments that will be realized for future versions of the ESB (highlighted in green).

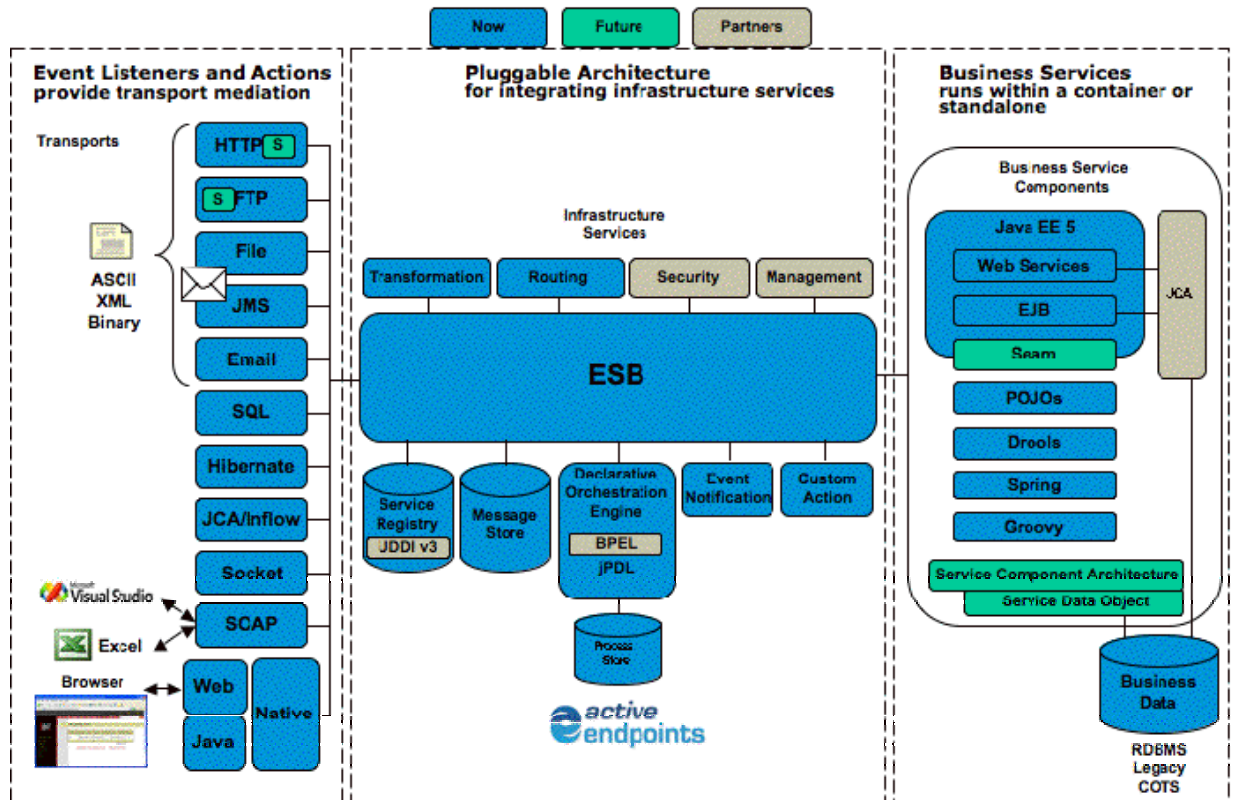


Figure 43: jBoss ESB Architecture (jBoss ESB, 2009)

As shown in the figure the ESB is extendable as different infrastructure services can be plugged into it, like for example the adaptors, the semantic service registry and discovery or the workflow engine. All this components that that will be plugged into the ESB to transform it into a Knowledge Bus will be provided in the following.

5.2.2.2 Adapter

Due to the legacy environment that most organizations have to deal with, applications often use proprietary models and meta-models (syntax) to describe information (semantics) in the messages they send or receive. An example is pointed out in (Selvage et al, 2008): Even if all participants use the same meta-model, for example, XML Schema Definition Language (XSD), they can still use a different message model, such as a different schema. Challenges arise when there are multiple proprietary message models and meta-models from different applications describing the same information, such as customer. Even if they have the exact same meta-model, such as XSD, you might still need to exert great effort to align and map these models.

To give a specific example, one application might provide the name of a person in a single attribute, while another application might structure it in two attributes: one attribute for first name and another for the last name. Potentially different semantics in messages further complicate the problem: A marketing campaign application may define a customer differently (for example, including potential customers) than a call centre application (where a customer is a person that has purchased a product).

The role of the ESB is to establish loosely coupled connectivity between services. That role includes transformation between the different message models and meta-models. However, as the number of interacting applications with proprietary message models increases, the challenges of managing the necessary transformations to enable service interaction increase exponentially.

Figure 44 illustrates the problem of performing direct transformation between service requesters and providers. In this example, n service requesters interact with m service providers. Each service uses its proprietary message model for communication. The message model in general reflects the application's internal data model. For the requesters to invoke the service of the providers, the message models from the requesters need to be transformed into the message models of the providers. In this scenario the ESB

has to understand all the proprietary message models and transform between the message models. Therefore the ESB must provide transformations for each possible interaction. For each requester (n in this scenario), a separate transformation has to be specified with respect to each provider (m in this scenario). That means m transformations for the first requester, m transformations for the second, and so on. The result is a total number of transformations of $n * m$. If one new provider is added and all requesters need to interact with it, again n new transformations need to be added. If one new requester is added and needs to interact with all m providers, then again m transformations need to be added. The consequence, of such an integration solution is the poor extensibility.

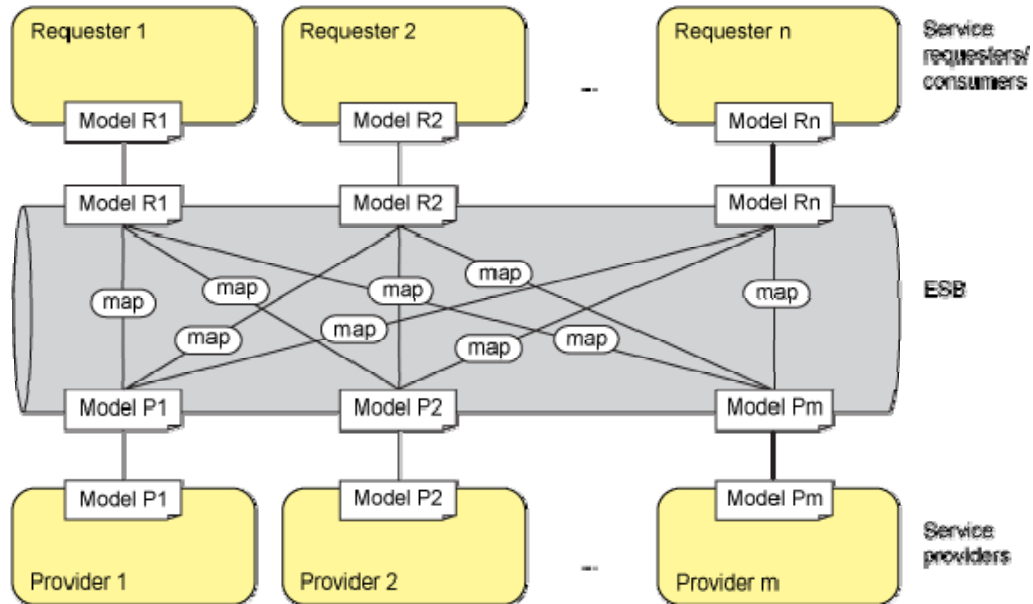


Figure 44: ESB – Direct Transformations between Service Consumers and Providers

A solution to this problem is the CMM (common message model) approach, where each of the requesters (n of them) needs a transformation to the CMM, and each of the providers (m of them) needs a transformation to the CMM, in case they all have different models. Therefore, the total number of transformations is $n + m$. If a new service, as a service requester or provider, is being introduced and this service uses a proprietary message model, only the transformation between the CMM and the application-specific message model needs to be created, regardless of the number of applications that already participate. This reduction in the number of transformations illustrates the core benefit of using the CMM pattern and helps to reduce the number of required transformation in this heterogeneous environment from $n * m$ to $n + m$. The service consumers and providers have to translate their original proprietary models into the CMM, which means that they have to build adaptors.

In an ideal scenario the messages used by the service requesters and the service providers have all been defined using the CMM. Because all applications use the CMM, therefore no transformations are required — neither in the applications nor in the ESB. This approach is unrealistic when connecting legacy applications like in MATURE when integrating the legacy applications at place at the MATURE application partners. Nevertheless this approach is ideal for new applications, because they can directly adopt the CMM as their internal data model and, therefore, reduce the effort of developing transformations that might have to take place when mapping between an internal data model and an external message model. This will be the case for services that will be developed from scratch during the project, like in WP2, WP3 and WP4. The approach followed in MATURE is depicted in Figure 45, where legacy applications of the application partners and existing services need to be integrated by implementing adaptors, while some services will be newly implemented. The common message model within the MATURE project is called the MATURE Message Model (MMM).

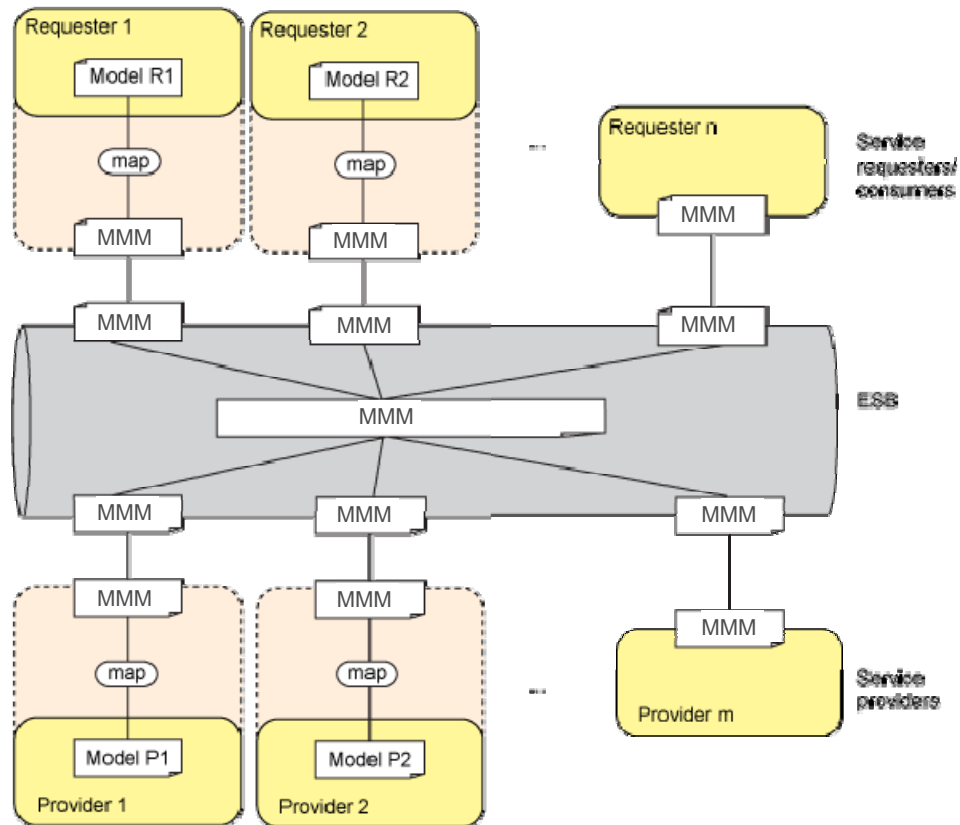


Figure 45: ESB – Integration of Applications and Services in MATURE

Adaptors have to be developed in order to implement the MMM. The following transformations are possible:

- Among Web-Services: Transformation may be necessary if the Web-Services already exist and their implemented message model differs from the agreed MMM
- Between Legacy Applications and Web-Services: Legacy applications may be anything, from Minesweeper to a “DBMS”. In order to attach them to the Bus they must be coupled with adapters that allow them to communicate over the MMM. In most cases this concretely means that they need to be wrapped up into Web-Services implementing the MMM.
- Between Human Services and Web-Services: Special attention is also required when integrating virtualized “Human Services” into the Bus. Human Services are those typically used among natural persons for communications, like Email or short messages.

Recalling the integration scenario introduced in section 4.1.3, this means that adaptors have to be built for each existing service that does not implement the MMM. Services that will be implemented during the project can implement the MMM as part of the service specification right from the start so that it will not be necessary to develop an additional adaptor.

5.2.2.3 Semantic Service Registry

The Semantic Service Registry is the component for the registration and semantic annotation of the services. Current standards describe Web-Services using syntactic notations such as WSDL. Since these descriptions are machine readable but not machine understandable, only IT personnel can carry out most of the tasks associated with creating and maintaining Web-Service-based applications such as Web-Service discovery, composition, and invocation. These tasks can be automated to a great extent by applying semantic technologies (such as OWL-S, WSMO, WSDL-S).

First, the service has to be registered by describing it on a non-functional level. This is done by providing information about the service provider as well as quality of service (QoS) aspects like security, time, cost or accessibility. The structure of this non-functional description is the following:

- **ID:** an identifier given by the system at registration time;
- **Label:** the name of the service;
- **Description:** a natural language description;
- **Type (WS, HUMAN):** is an enumeration field whose values are: WS for services provided as Web-Service and HUMAN for services provided by a human being;
- **HTTP Address:** the http address where the Web-Service implementation can be invoked from;
- **Input:** the list of required parameters;
- **Output:** what the service releases;
- **Cost:** the price of the service expressed in euro (e.g., 2.5);
- **Time:** the estimated time for the service execution (e.g., 0.5 sec);
- **Service Provider:** provider of the service - it contains a reference to a Service Provider instance (e.g., SP1)

A Service Provider is a real instance of an institution or organization that provides certain services. The structure of the Service Provider template is the following one:

- **ID:** an identifier given by the system at registration time (e.g., SP1);
- **Name:** the label used for referring to the Service Provider;
- **Description:** a natural language description of the Service Provider;
- **Contact Details:** address information;
- **Contact Person:** information like name, telephone number, e-mail address of the contact person.

In order to be discovered by the semantic service discovery the service and its parameters have to be semantically annotated on a functional level (Stollberg et al, 2007). Therefore the services' operations and corresponding input and output parameters will be annotated using the concepts of the previously introduced MATURE Message Model (MMM) are used (see section 5.2.2.2 for further details on the MMM) which is provided as an ontology.

After the service has been registered and annotated it can be published. Published Web-Services are accessible by other systems as the Semantic Service Registry will provide functionality to find published services through Web-Service interfaces. Figure 46 provides an overview on the architecture of the semantic service registry and semantic service discovery (which will be introduced in the following section). A human service provider can access the service registry GUI to register, annotate and publish his/her service. A model of the service and its annotations are stored in the model repository.

See D5.1 for a user manual on the service registration, annotation and publication using the semantic service registry component.

5.2.2.4 Semantic Service Discovery

The Semantic Service Discovery operates on the semantic data (according to the MMM) in the descriptions of the services. Its task is to identify services matching a request. It informs the client-services whether there are provider services that can deliver the functionality required. Moreover it decides which service to pick, if there is more than one available.

Once a service was successfully published using the semantic service registry (as explained in the previous section), accessing services can find the service using the provided API to the semantic service

discovery as Figure 46 depicts. For testing purposes, a discovery GUI is provided to allow a service administrator to evaluate if the published service is found successfully.

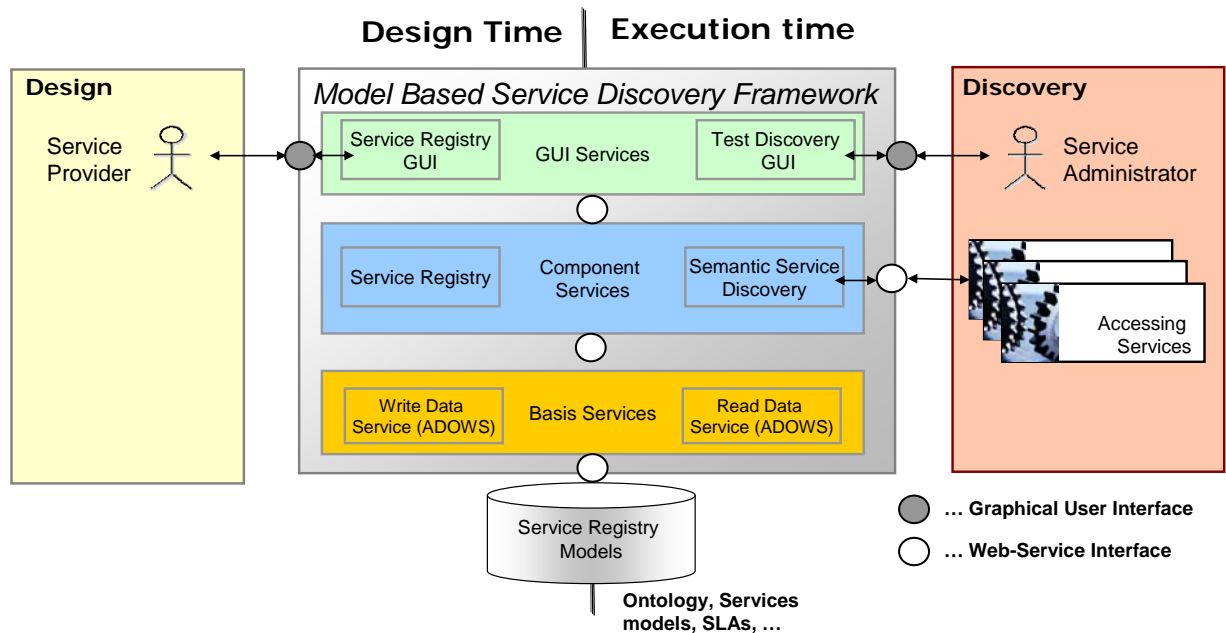


Figure 46: Implementation View on the Semantic Service Registry and Discovery

When the Semantic Service Discovery receives a request by an accessing service it first searches for potentially usable Web-Services. Therefore semantic matchmaking concerning the functional description of the service is executed. If no service is found that can be accessed directly, a combination of Web-Services will be composed. As a next step the found services will be weighted according to the non-functional description (e.g. service provider, time, cost, quality) and the most appropriate service is selected. Finally the information is exchanged between the requester and the selected Web-Service, thus the Web-Service is executed.

5.2.2.5 Workflow Management

The Workflow Engine serves the purpose of creating complex services by orchestrating available services, thus enabling complex services. For this reason the Workflow Engine has to cooperate with the service discovery component. The orchestration problem needs technologies that enable its real and effective implementation. Such technologies should address the two most important aspects of orchestration: How is the orchestration between services modelled and described? How is the orchestration model interpreted and executed?

The first question is addressed by selecting a language that can describe the control and data flow between different entities, possibly specified in both an abstract way (i.e. without referring to real implemented piece of software or service) and a concrete way (i.e. binding the entity specification to a precise service or resource). From this point of view the Web-Services business process execution language (WS-BPEL) has been selected. Its specification enables the description of highly complex workflows, within which parties exchange information by following the control and data flows descriptions. WS-BPEL is a fully working orchestration language that supports abstract and concrete specification of services and that provides a very wide range of constructs for flow control, data binding and variable definition.

As defined in the specification, “*WS-BPEL provides a language for the specification of executable and abstract business processes. By doing so, it extends the Web-Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.*” (OASIS BPEL, 2009)

The detailed specification on WS-BPEL can be found in the official WS-BPEL specification (OASIS BPEL, 2009) developed by OASIS (OASIS, 2009).

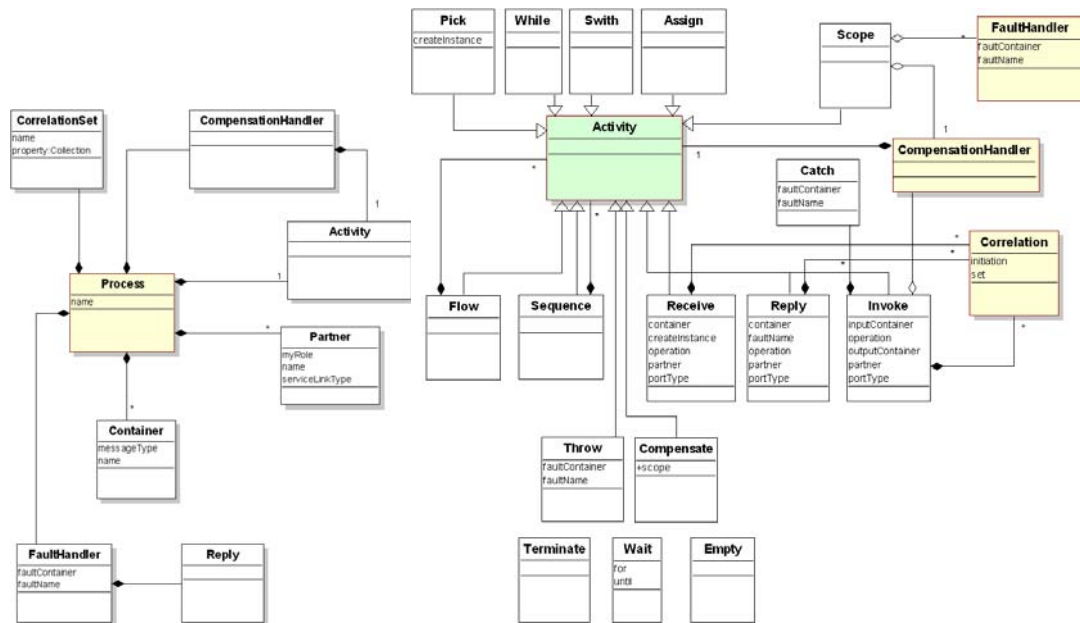


Figure 47: BPEL Meta-Model (ebPML BPEL, 2009)

Figure 47 gives an overview on the structure of WS-BPEL and the corresponding meta-model. WS-BPEL utilizes and integrates several XML specifications and standards. Details on each standard can be found at the reference provided:

- WSDL 1.1 and XML Schema 1.0 are used to build the data model
- XPath 1.0 and XSLT 1.0 are used to add data manipulation support

It is important to note that all external sources in WS-BPEL are addressed and represented using WSDL definitions.

An engine should be selected in charge of effectively parsing and executing the workflow. Such an engine needs to maintain adequate data-structures and information about a workflow so that execution can be steered, monitored and managed in a flexible way. Moreover, such an engine should also provide all the necessary APIs, languages, protocols and standards for making it possible to invoke service’s methods, passing them inputs and retrieving, when necessary, outputs. A wide range of WS-BPEL engines exist. An open source WS-BPEL engine will be selected for use in the MATURE project to enable orchestration of services. Implementing an SOA based approach the WS-BPEL compliant open source workflow engine provided by Active-Endpoints⁶ is implemented as the runtime environment.

The Active-Endpoint engine is a Java-based implementation of a WS-BPEL workflow engine, available under the GPL licence⁷. The update on WS-BPEL 2.0 and an open architecture based on web/application server and using the Apache Axis implementation as a Web-Service container allow flexible adaptation and use within the project. All functions and information needed regarding the workflow engine are accessible via administrative Web-Services, allowing a high level of integration in any domain.

⁶ Active Endpoints Homepage. Access: <http://www.activevos.com/> [10.04.2009]

⁷ General Public License (GPL). Access: <http://www.gnu.org/copyleft/gpl.html> [10.04.2009]

5.2.2.6 Meta-Data Management

This section specifies the knowledge items, which are the smallest data elements exchanged between the sources and services. Each knowledge item consists of content and ontological meta-data. There are a number of domain-independent initiatives to standardize meta-data, e.g. Dublin core (Dublin Core Metadata Initiative, 2009), Digital Object Identifier (Digital Object Identifier System, 2009) or the Text Encoding Initiative (Text Encoding Initiative, 2009). For MATURE the most relevant initiative is the Dublin Core Metadata Initiative which mainly aims at the description of text documents. Apart from the domain-independent meta-data standards there are a number of domain-specific ones. The most relevant one for the project is the Learning Object Metadata (LOM) (LOM, 2009), which is used for the description of learning objects. LOM is a data model (based on Dublin Core), usually encoded in XML, and used to describe a learning object and similar digital resources for supporting learning. The purpose of learning object metadata is to support the reusability of learning objects, to aid discoverability, and to facilitate their interoperability, usually in the context of online learning management systems. Most of the LOM concepts can also be applied in the context of the MATURE system. Figure 48 shows a schematic representation of the hierarchy of elements in the LOM data model. Extensions to the model will be made to include MATURE specific aspects, e.g. the maturity level of knowledge items.

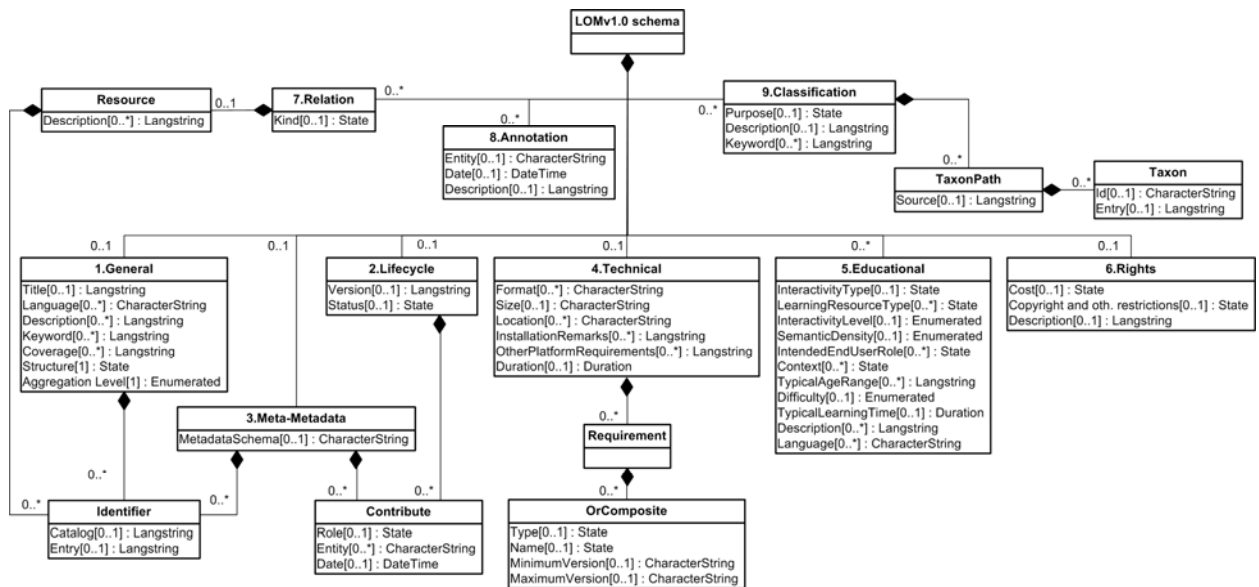


Figure 48: Schematic Representation of the Hierarchy of Elements in the LOM Data Model (LOM, 2009)

In the following the base schema structure of the meta-data used to describe knowledge items will be detailed. Extensions of LOM to include MATURE specific concepts are included and highlighted (orange-coded) in the tables. Table 8 presents details on the root tag – the knowledge item tag.

Table 8: Knowledge Item Meta-Data - Knowledge Item Tag

<i>Knowledge Item Tag</i>			
<i>Nr</i>	<i>Name</i>	<i>Description</i>	<i>Datatype</i>
1	General	This tag groups the general information that describes the knowledge item as a whole	GENERAL TAG
2	Lifecycle	This tag groups features related to the history and current state of the knowledge item and those who have affected this knowledge item during its evolution.	LIFECYCLE TAG

		Here MATURE specifics like the “Maturity Level” will be included.	
3	Meta-Metadata	This tag groups information about the metadata instance itself (rather than the knowledge item that the metadata instance describes)	METAMETADATA TAG
4	Technical	This tag groups technical requirements and technical characteristics of the knowledge item.	TECHNICAL TAG
5	Rights	This tag groups the intellectual property rights and conditions of use for the knowledge item.	RIGHTS TAG
6	Relation	This tag groups features that define the relationship between the knowledge item and other related knowledge items.	RELATION TAG
7	Annotation	This tag provides comments on the use of the knowledge item and provides information on when and who created the comments. Here MATURE specifics like a “Rating” of Knowledge Items are included.	ANNOTATION TAG
8	Classification	This tag describes this knowledge item in relation to a particular classification system.	CLASSIFICATION TAG

Each of the mentioned tags and the corresponding attributes is presented in Annex D. This chapter described the Knowledge Bus as an integration tool. All components of the Knowledge Bus were described from a conceptual and from an implementation view.

6 Summary and Outlook

The present deliverable D5.2 (Specification of the System Architecture) aimed to guide the integration of the services which will be developed in WP2 (PLME services), WP3 (OLME services) and WP4 (Maturing services) by specifying the MATURE system architecture. In the following the followed methodology to reach this goal will be summarized.

After the followed integration philosophy was introduced the relevant MATURE specific and SOTA concepts were introduced. Based on this conceptual foundation the system was described from a high level. The approach followed was a hybrid approach, analysing the system from the bottom up and the top down.

From the bottom up a rapid prototyping approach was followed to analyze existing services (PLME, OLME and Maturing services) and application for the application within MATURE and to integrate them in a first integration scenario.

From the top down the necessary infrastructure that enables the integration and execution of services and legacy applications at the MATURE application partners was derived. Therefore a view model was applied to analyse the system from different viewpoints.

The Knowledge Bus as central component of the MATURE system was described in more detail. It consists of two layers, the infrastructure layer and the integration layer. The infrastructure layer provides the basic functionality for the registration, execution and orchestration of services. The integration layer enhances the basic functionality by introducing semantics to describe services and messages exchanged between the services and involves the domain experts in the acquisition of system requirements. Both layers were described in detail thus emphasizing its role as an integration tool within the project.

The present deliverable is at this point in time in a DRAFT status. The further procedure followed in WP5 will be introduced in the following section.

6.1 Outlook to the Further Procedure in WP5

This section concludes this deliverable by providing an outlook to the further procedure followed by WP5. This deliverable provided an overview of the MATURE system architecture, which is at this point in time in a DRAFT status. Figure 49 summarizes the further procedure in this work package.

The system architecture will be refined within the related task 5.1 (“System Architecture Design”), the final version will be available in project month 18. The infrastructure required to realise the system architecture is implemented within task 5.2 (“Infrastructure”). The developed test bed will be used by developers to integrate their knowledge sources, use already available services and test their own services. Refinements of the system architecture will be continuously realised in the infrastructure, if affected. The knowledge sources and services have to be prepared so that they can be connected to the Knowledge Bus using adaptors. This will be part of task 5.3 (“Integration”). Finally, within task 5.4 (“Deployment”), the MATURE system will be deployed at the application partners’ sites and evaluated in order to demonstrate the success of MATURE in a real world environment. As the figure depicts during the duration of this work package several prototypes will be realized.

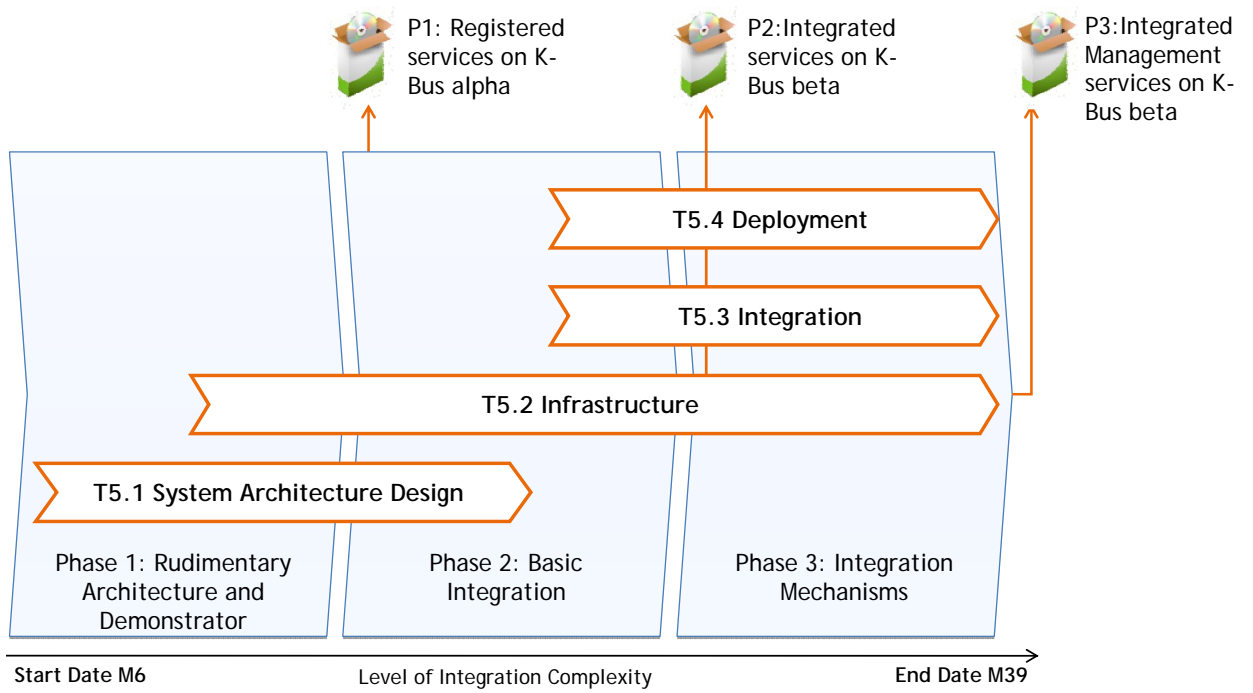


Figure 49: Further Procedure in WP5 – From the Initial Prototype to the MATURE System

7 References

- Aberer, K. and Despotovic, Z. (2001). Managing trust in a peer-2-peer information system. In: Proceedings of 10th International Conference on Information and Knowledge Management, pages 310–317, 2001.
- Active Endpoints Engine Architecture. Access: <http://www.active-endpoints.com/open-source-architecture.htm> [17.03.2009]
- Akkiraju, R., et al. (2005): Web-Service Semantics - WSDL-S, W3C Member Submission, 7 November 2005. Access: <http://www.w3.org/Submission/WSDL-S/> [12.03.2009]
- Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004): Web-Services: Concepts, Architectures and Applications, Springer Verlag
- Ali Arsanjani (2004): Service-oriented modeling and architecture. IBM Online article, 09 Nov 2004.
- Bijay K. Jayaswal, Peter C. Patton (2006): Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software, Prentice Hall.
- Blaze, M., Feigenbaum, J. and Lacy, J. (1996): Decentralized trust management. In: Proceedings of IEEE Conference on Security and Privacy.
- Bonatti, P. A. and Olmedilla, D. (2005): Driving and monitoring provisional trust negotiation with metapolicies. In: 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), pages 14–23, Stockholm, Sweden, IEEE Computer Society.
- BREIN Deliverable D4.1.2, Overall Architecture, 2008. Access: http://www.eu-brein.com/index.php?option=com_docman&task=doc_view&gid=30&Itemid=31 [17.03.2009]
- Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C. (2006): IRS-III: A Broker for Semantic Web-Services based Applications. In: Proceedings of the 5th International Semantic Web Conference (ISWC 2006).
- Cabral, L., Domingue, J., Motta, E., Payne, T. R. and Hakimpour, F. (2004): Approaches to Semantic Web-Services: An Overview and Comparison. In: Proceedings of the European Semantic Web Conference.
- Cardwell, L. (2005): AJAX – Bridging the Thin-Client Performance Gap. Access: <http://www.ironspeed.com/articles/AJAX-Bridging%20the%20Thin-Client%20Performance%20Gap/Article.aspx> [16.03.2009]
- DAML Homepage: Releases of DAML-S / OWL-S. Access: <http://www.daml.org/services/owl-s/> [16.03.2009]
- Digital Object Identifier System. Access: <http://www.doi.org/> [13.03.2009]
- Domingue, J. and Fensel, D. (2008): Toward a Service Web: Integrating the Semantic Web and Service Orientation, January 2008, IEEE Intelligent Systems
- Drucker, P. (1973): Management: Tasks, Responsibilities, Practices. Harper & Row, New York.
- Dublin Core Metadata Initiative. Access: <http://dublincore.org/> [13.03.2009]
- Dublin Core Meta-Data Initiative: Dublin Core - The Elements. Access: <http://dublincore.org/documents/usageguide/elements.shtml> [16.03.2009]
- ebPML BPEL. BPEL Meta-Model. Access: <http://www.ebpml.org/bpel4ws.htm> [10.04.2009]
- Erl, T. (2005): Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall/PearsonPTR
- Fielding, R. (2000): Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California.

- Flurry, G. (2007): Exploring the Enterprise Service Bus, Part 1: Discover how an ESB can help you meet the requirements for your SOA solution, IBM Software Group. Access: <http://www.ibm.com/developerworks/library/ar-esbpat1> [17.03.2009]
- Foster, I. and Kesselman, C. (2004). The Grid – Blueprint for a New Computing Infrastructure. Second Edition, Morgan Kaufmann Publishers.
- Foster, I., Kesselman, C., Tuecke, S. (2001): The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3)
- Gartner Group (2008): Gartner Identifies the Top 10 Strategic Technologies for 2009. Access: <http://www.gartner.com/it/page.jsp?id=777212> [17.03.2009]
- Globus OGSA, The Open Grid Services Architecture. Access: <http://www.globus.org/ogsa/> [10.04.2009]
- Google Gadgets Homepage. Access: <http://www.google.com/ig/directory?hl=en&synd=open> [13.03.2009]
- Hemingway, C.J. and Breu, K. (2003): From traditional to virtual organisation: implications for work unit boundaries. In Proceedings of the Eleventh European Conference on Information Systems (Ciborra CU, Mercurio R, de Marco M, Martinez M, Carignani A eds.), 778-787, Naples, Italy.
- Hinkelmann, K.; Karagiannis, D.; Telesko, R. (2002): PROMOTE - Methodologie und Werkzeug zum geschäftsprozessorientierten Wissensmanagement. In: Geschäftsprozessorientiertes Wissensmanagement, Springer-Verlag
- KMI, Knowledge Media Institute, IRS - Internet Reasoning Service. Access: <http://technologies.kmi.open.ac.uk/irs/> [16.03.2009]
- Kruchten, P. (1995): Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50
- Kühn et al. (2003): Enterprise Model Integration, In: Bauknecht, K.; Tjoa, A M. Quirchmayer, G. (Eds.): Proceedings of the Fourth International Conference EC-Web 2003 – DEXA 2003, Prague, Czech Republic, September 2003, LNCS 2738, Springer-Verlag, Berlin, Heidelberg, pp. 379-392.
- Kühne, T. (2005): Understanding metamodeling. In: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), 716-717
- Larman C. (2005), UML 2 und Patterns angewendet – Objektorientierte Softwareentwicklung, mitp-Verlag/Bonn, 2005, pp. 98ff.
- Lee, J., Upadhyaya, S. J., Rao, H. R., and Sharman, R. (2005): Secure knowledge management and the semantic web. Communications of the ACM 48, 12 (Dec. 2005), 48-54.
- LOM, Learning Object Metadata. Access: http://en.wikipedia.org/wiki/Learning_object_metadata [13.03.2009]
- LOM, Draft Standard for Learning Object Metadata. Access: http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf [13.03.2009]
- Maier, R. (2007): Knowledge Management Systems, Information and Communication Technologies for Knowledge Management, Springer.
- Mak, K. and Woitsch, R. (2005): Der Einsatz des prozessorientierten Wissensmanagementwerkzeuges PROMOTE® in der Zentralkommunikation der Landesverteidigungsakademie, Schriftenreihe der Landesverteidigungsakademie, 19/2005
- Martin, D., Domingue, J., Brodie, M. and Leymann, F. (2007): Semantic Web-Services, Part 1. In: IEEE Intelligent Systems
- MATURE D1.1: Results of the Ethnographic Study and Conceptual Knowledge Maturing Model, MATURE deliverable, April 2009

MATURE D2.1: Model of pedagogical requirements and of supporting services of the PLME, MATURE deliverable, April 2009

MATURE D3.1: Model of organizational requirements and of supporting services of the OLME, MATURE deliverable, April 2009

MATURE D4.1: Definition of Maturing Services, MATURE deliverable, April 2009

MATURE D5.1: Infrastrucutre Testbed, MATURE deliverable, April 2009

MATURE D6.1: Requirements specification and Evaluation plan, MATURE deliverable (Draft), March 2009.

MATURE DoW: Description of Work, November 2007

Merrill, D. (2006). Mashups: The new breed of Web app, Access: <http://www.ibm.com/developerworks/xml/library/x-mashups.html> [17.03.2009]

Mika, P., Oberle, D., Gangemi, A. and Sabou, M. (2004): Semantic Web-Services: Foundations for service ontologies: aligning OWL-S to dolce. In: Proceedings of the 13th international conference on World Wide Web, ACM Press

Milke, J.-M., Schiffers, M., Ziegler, W. (2006): Virtuelle Organisationen in Grids: Charakterisierung und Management, PIK Praxis der Informationsverarbeitung und Kommunikation, 2006, Saur-Verlag.

Moran, M., Kopecky, J. and Mocan, A. (2005): WSDL-S (LSDIS and IBM) & WSMO, WSMO Working Group Presentation. Access: <http://www.wsmo.org/papers/presentations/WSDL-S.ppt> [16.03.2009]

OASIS (2006): Reference Model for Service Oriented Architecture 1.0, OASIS Standard. Access: <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html> [16.03.2009]

OASIS BPEL, BPEL Specification. Access: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html> [16.03.2009]

OASIS Homepage. Access: <http://www.oasis-open.org/home/index.php> [17.03.2009]

OASIS SAML, SAML V2.0 Specification. Access: <http://saml.xml.org/saml-specifications> [17.03.2009]

OASIS UDDI, UDDI Specifications. <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm> [16.03.2009]

OCML, Operational Conceptual Modelling Language Homepage. Access: <http://technologies.kmi.open.ac.uk/ocml/> [10.04.2009]

OMG, Model Driven Architecture. Access: <http://www.omg.org/mda/> [16.03.2009]

OMG MOF, OMG's Meta Object Facility Homepage. Access: <http://www.omg.org/mof/> [16.03.2009]

OMG UML, UML Specification v2.2, Accessible: <http://www.omg.org/technology/documents/formal/uml.htm>, [10.04.2009]

Operational Conceptual Modelling Language. Access: <http://technologies.kmi.open.ac.uk/ocml/> [16.03.2009]

O'Reilly, T. (2005): What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software. Access: <http://oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1> [16.03.2009]

OWL-S Specification. Access: <http://www.daml.org/services/owl-s/1.1/> [16.03.2009]

Polleres, A., Lausen, H., and Lara, R. (2006): Semantische Beschreibung von Web-Services. In: Semantic Web - Wege zur vernetzten Wissensgesellschaft. Springer

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, Ch., Fensel, D. (2005): Web-Service Modeling Ontology, Applied Ontology, pp. 77 – 106

SAML Specification. Access: <http://saml.xml.org/saml-specifications> [17.03.2009]

- Selvage. et al (2008): Exploring the Enterprise Service Bus, Part 3: Four approaches to implementing a canonical message model in an ESB. Access: <http://www.ibm.com/developerworks/architecture/library/ar-esbpat3/> [13.02.2009]
- Shibboleth, Shibboleth® Enabled Applications and Services. Access: <https://spaces.internet2.edu/pages/viewpage.action?pageId=11484> [17.03.2009]
- SOA4All D1.1.1 Design Principles for a Service Web v1, SOA4All Deliverable, August 2008. Access: <http://www.soa4all.eu/resources.html?func=startdown&id=25> [16.03.2009]
- Solazzo, T., Handschuh, S., Staab, S., and Frank, M. (2002): Semantic Web-Service Architecture - Evolving Web-Service Standards toward the Semantic Web. In: Proceedings of the 15th International FLAIRS Conference. Pensacola, Florida, May 16-18, 2002. AAAI Press
- Stachowiak, H. (1973): Allgemeine Modelltheorie. Springer.
- Stevens, M.: The Benefits of a Service-Oriented Architecture. Access: <http://www.developer.com/design/article.php/1041191> [16.03.2009]
- Stollberg, M., Hepp, M. and Fensel, D. (2007): Semantic Web-Services – Realisierung der SOA Vision mit semantischen Technologien. SWS – MKE conference
- Text Encoding Initiative. Access: <http://www.tei-c.org/index.xml> [13.03.2009]
- W3C HTTP, HTTP - Hypertext Transfer Protocol. Access: <http://www.w3.org/Protocols/> [16.03.2009]
- W3C OWL. Access: <http://www.w3.org/2004/OWL/> [10.04.2009]
- W3C SAWSDL, Semantic Annotations for Web-Services Description Language Working Group Homepage. Access: <http://www.w3.org/2002/ws/sawSDL/> [12.03.2009]
- W3C SOAP, SOAP Version 1.2. Access: <http://www.w3.org/TR/soap/> [16.03.2009]
- W3C WSDL, Web-Services Description Language (WSDL) 1.1. Access: <http://www.w3.org/TR/wsdl> [16.03.2009]
- Warner, C. and Crupi, J. (2008): Enterprise Mashups Part II: Why SOA Architects Should Care, The SOA Magazine. Access: <http://www.soamag.com/I21/0808-1.asp> [17.03.2009]
- Woitsch, R. (2004): Process Oriented Knowledge Management: A Service Based Approach. PhD Thesis, University of Vienna, July 2004
- Woitsch, R. and Leutgeb, A. (2008): The BREIN-Roadmap with PROMOTE®: A Use-Case of a Service-Based Knowledge Management Approach. In: Proceedings of I KNOW '08, Graz, Austria
- Woitsch, R., Karagiannis, D., Fill, H.-G. and Blazevic, V. (2007): Semantic Based Knowledge Flow System in European Home Textile: A Process Oriented Approach with PROMOTE. In: Proceedings of I KNOW '07, Graz, Austria
- Woitsch, R. and Utz, W. (2006): Roadmap to Akogrimo Convergence, A Sample of Process Oriented Knowledge Management with PROMOTE. In: Proceedings of I KNOW '06, Graz, Austria
- WS-I, Web-Services Interoperability Organisation (2006): Basic Profile Version 1.1. Access: <http://www.ws-i.org/Profiles/BasicProfile-1.1.html> [16.03.2009]
- WS-I, Web-Services Interoperability Organisation (WS-I) homepage. Access: <http://www.ws-i.org/> [10.04.2009]
- XML-RPC, XML-RPC Homepage. Access: <http://www.xmlrpc.com/> [10.04.2009]



Annex A Design Studies - Integration Relevant Aspects

This section provides an overview of the services and the integration relevant aspects gathered for each design study. Table 9 depicts the template that was filled out by each design study leader. Further details on the design studies can be found in D2.1 (for PLME related design studies), D3.1 (for OLME related design studies) and D6.1 for an overview on the design studies.

Table 9: Service Integration Template for the Design Studies

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	Name of the Service	Service Description understandable for business users (technical non-savvy user)	Technical specification of service – architecture, technology used, service structure	[OLME Service, PLME Service, Maturing Service] – Multiple selection possible	[UI Layer, Logic Layer, Data Layer] – Multiple selection possible	Atomic, Service cluster[specification of sub-services necessary using IDs], Composite services [specification of sub-services and logical flow necessary] – based on SOMF 2.0 ⁸	If feasible, technical specification of required input for service (ideally Web-Service message specification or data types)	If feasible, technical specification of required output for service (ideally Web-Service message specification or data types)	Input/Output relation to other service available in list (mention ID of service)

⁸ Ali Arsanjani (2004): Service-oriented modeling and architecture. IBM Online article, 09 Nov 2004.

Annex A.1 Design Study “DS1: OLMEWiki” Service Collection

Responsible Partner: TUG

Table 10 specifies the identified services in detail.

Table 10: Design Study “DS1: OLMEWiki” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	KnowMiner Service	KnowMiner provides a framework of services for knowledge discovery from unstructured content. This includes meta-data extraction, knowledge relationship discovery as well as indexing, clustering and classification tasks.	This service is designed as a set of back-end services, so it can be easily integrated in various types of services and applications. It is permanently improved and expanded by TUG.	Maturing Service	Data	Framework	Unstructured Content (Text)	Meta-data, Associations, Classes, Clusters, Key Terms	
2	Maturity Analysis Services	A set of services that provides indicators which are related to the level of maturity of a certain knowledge artefact. These services will be available in the sectors content, semantic and processes.	These services implement several metrics for the maturity level of a knowledge artefact. In addition, the analysis services provide graphical indicators to refer to the current maturity of a certain artefact.	Maturing Services	UI, Logic	Framework	Content, structure, processes	Maturity indicator	1
3	Consolidation Services	According to the SER model, during the reseeded phase the task of consolidation	The task of consolidation covers reflection services, collaboration services	OLME	Data, Logic	Framework	Content, Semantics, Processes	Content, Semantics, Processes (consolidate	

		is required. The consolidation services provide support for the task of consolidation of (semantic) structures, content and processes	and services for revision of knowledge artefacts.					d)	
4	Information Retrieval Service	This service provides a search interface which helps the user to aggregate information related to a certain topic without the need to use multiple search engines.	Using different search facilities of various search engines (yahoo, YouTube, Flickr, local and shared databases etc) this service provides a combined interface.	PLME	Data	Framework	Keywords	List of data	6

Annex A.2 Design Study “DS2: Dialogue Games for Ontology Maturing” Service Collection

Responsible Partner: LTRI/FZI

Development of a knowledge maturing dialogue game through mashing up SOBOLEO and Interloc, to examine: the role of dialogue in knowledge maturing, knowledge maturing as a social learning process, the technical realisation of ‘loosely coupling’ two related technologies and the application of this mashup in AP scenarios. Table 11 specifies the identified services in detail.

Table 11: Design Study “DS2: Dialogue Games for Ontology Maturing” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	SOBOLEO Ontology Editor Service	The ontology editor service allows collaborative creation, update, maintenance, and operationalization of a SKOS model	The editor is an AJAX application (based on Google Webtoolkit, i.e. implemented in Java) that allows creation, update, maintenance, and operationalization of a SKOS model for multiple users	PLME, OLME	UI, Data	Atomic	Domain Expert Knowledge; SKOS ontology description	SKOS ontology description; logging of all operations and chat messages as text file	
2	InterLoc Dialogue Game Editor Service	The dialogue game editor service allows to modify the textual contents of the Menu (move categories + openers) and interaction rules		PLME, OLME	UI, Data	Atomic	XML file	XML file	
3	InterLoc Dialogue Game Interface Service	The InterLoc Dialogue Game interface service allows for guided dialogue games for multiple users.	The Dialogue Game interface is a collaborative Java application (using Sun Java Webstart) with an HTML/CSS based UI using XMPP protocol for message exchange between the users.	PLME, OLME	UI	Atomic	User input; Pre-defined move categories & openers from 2 as XML file;	Dialogue protocol as HTML file	2



Annex A.3 Design Study “DS3: Interacting Widgets” Service Collection

Responsible Partner: UPB

Development of a widget mashup with interacting widgets on a persistent layer and an additional integration layer. Table 12 specifies the identified services in detail.

Table 12: Design Study “DS3: Interacting Widgets” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	Widget Hosting and Administration	The widget server allows uploading of user-created content (widgets), integrating them with the existing widget infrastructure.	A function of the web server, an archived widget is uploaded and installed according to the included XML configuration.	PLME, OLME	UI, Logic, Data	Atomic	Widget Archive (ZIP)		1.2
1.2	Widget Discovery / Advertising	The server can be queried for available widgets to allow for easy, automated integration into third-party web pages or applications.	Regular HTTP request, returns XML of all installed and public widget types.	PLME, OLME	Logic, Data	Atomic	HTTP Request	XML containing widget information	1.1
1.3	Widget Communications	This service handles widget communication and storage.	Widgets communicate exclusively via AJAX (DWR framework), using the server and its database as a relay. Internally, communication is managed through a “channel” metaphor, linking related widgets together on one or more channels.	PLME, OLME	Logic, Data	Service Cluster - Channel Management - Widget Yellow Pages - Widget Real-	AJAX Request	Success / Error / requested data	1.5

						Time Messages - Widget Data Pool Storage			
1.4	Logging	In addition to the widgets' own information being persistently stored, this service allows for on-the-fly logging of knowledge-building data (e.g. user interactions) while the widgets are in use.	Logging calls are made through AJAX and initiated on demand by the widget. Logs are stored in the server database.	PLME, OLME	Data	Atomic	AJAX Request	Success / Error	1.5
1.5	Data Export	This service is an interface to external clients, preparing and exporting knowledge and usage data in various formats.	The service generates data from the server's database and exports via a generalized interface.	OLME	Logic, Data	Atomic	Request (HTTP, RPC, ...)	Formatted and collated data	1.4, 1.3, Knowledge Bus
1.6	Proxy	This service allows Widgets the communication to other services which is necessary as browsers permit cross-scripting	The proxy allows a http communication channel through the server	PLME, OLME	Data	Atomic	HTTP Request	HTTP Response	

Annex A.4 Design Study “DS5: OLMentor” Service Collection

Responsible Partner: FHNW

The OLMentor design study develops a demonstrator for knowledge maturing from an organisational point of view. The goal is to examine the usefulness of the intended support by:

1. providing relevant knowledge artefacts (e.g. documents or information on experts) automatically depending on tasks (process steps) to be performed
2. providing relevant knowledge artefacts (e.g. documents or information on experts) automatically based on (additional) information gathered through process execution
3. providing the possibility of rating the provided artefacts and using the rates to rank the knowledge artefacts with respect to relevance
4. applying knowledge maturing services to automat annotation of various knowledge artefacts (blog, wiki, notes, documents etc.)
5. making suggestions for learning (e.g. learning/reading a document, talking to an expert, attend e-learning course, participate in community of practice etc.)

Table 13 specifies the identified services in detail.

Table 13: Design Study “DS5: OLMentor” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	Case Based Reasoning (CBR) Service	The case based reasoning service provides historical (closed) cases related to tasks (process steps) and cases in process	The CBR service uses ontologies and application information	OLME, PLME	Logic, Data	Atomic	Context relevant data and task description	List of matching historical (closed) cases	
1.2	Retrieval Service	Service for retrieving knowledge artefacts (e.g. documents or information on experts) depending on context information	The retrieval service uses ontologies and application information	OLME, PLME	Logic, Data	Atomic	task description	List of knowledge artefacts	

1.3	Case Retrieval Service	The case retrieval service provides detailed description about the case	The service retrieve additional information about the case, like who has performed which task, which data was set when	OLME, PLME	Logic, Data	Atomic	Case id	Information about the case	
1.4	Work list handler	The work list handler provides a graphical user interface with which the human worker interacts. Depending on the case different decision support can be shown. (S)he can log in. After it, (s)he gets a list of her/his tasks. Automatically the Retrieval Service is called and (s)he gets all relevant information related to the task, e.g. a list of experts, historical cases, documents, websites or other knowledge-artefacts	Adaptive graphical user interface, Ajax	PLME	UI	Atomic			1, 1.2, 1.3, 1.5, 1.6
1.5	Storing Service	The storing service is responsible to store knowledge-artefacts	The service provides a user interface to upload a new artefact (file, website) and	OLME, PLME	UI, Logic, Data	Atomic	Knowledge-artefacts	True, if storing was successful	

			stores the knowledge-artefacts and their metadata						
1.6	Adaptive Workflow Engine	The workflow engine is responsible for the execution of the structured part of the process and also for the assignment of tasks, based on the case (e.g. difficulty of a task)	The service controls the execution of a workflow.	OLME	Logic, Data	Atomic	Workflow model		1.4, 1.7
1.7	RHEA	RHEA is a rule engine, which can be integrated in an internet based workflow engine. RHEA supports the adaptivity of knowledge intensive process parts using rules.	For the execution of the knowledge-intensive process part, the rule engine, RHEA, is invoked as a Web-Service, which executes rules. Depending on the context relevant data flexible resource allocation, decision support, constraint checking or planning is supported.	OLME	Logic, Data	Atomic	Rule set, Context-relevant data	Consequences of the fired rules	1.6
1.8	ATHENE	The modelling environment is used for modelling knowledge artefacts (e.g. adaptive process models)	For modelling a knowledge artefacts ATHENE is used. ATHENE is a modelling environment which can be accessed via browser. The models are stored using ontologies.	OLME, PLME	UI, Logic, Data	Atomic		Workflow model	
1.9	Monitoring	Observe the	The monitoring	OLME, PLME	UI, Logic, Data	Atomic	Workflow	Workflow	

	service	availability of knowledge artefacts related to tasks and the behaviour and interaction of users	service observes the user during his tasks and tries to figure out if knowledge artefacts are available to help the user.				execution, Feedback	model	
2.0	Assembling service	Various knowledge artefacts stored in the knowledge base are assembled automatically (e.g. the generation of a report)	The service assembles needed artefacts for and merges them, e.g. for the generation of a report	OLME, PLME	Logic, Data	Atomic	Artefacts	Artefact	
2.1	Share service	Based on the cases a user worked on, new information will be automatically sent to users for whom it might be useful	The knowledge base links new artefacts with existing ones and is therefore able to detect related artefacts	OLME, PLME	UI, Logic, Data	Atomic	Artefacts	Artefact	1.9
2.2	Mining Service	Mining of process instances to identify individual and organisational goals.	Monitoring whether tasks have been change or added. Mining of it and adapt the processes.	OLME, (PLME)	Logic, Data	Atomic	Use, changing, managing of tasks.	Suggestions for matured workflows	1.9



Annex A.5 Design Study “DS6: APOSDLE” Service Collection

Responsible Partner: TUG

Table 14 specifies the identified services in detail.

Table 14: Design Study “DS6: APOSDLE” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	Associative Network Service	This service facilitates the representation of associative networks. In order to search for information it provides retrieval mechanisms based on spreading activation mechanisms.	This service is developed as stand-alone service and is permanently further developed at TUG. In order to create meaningful associations between nodes an additional service providing the extraction of meta-data is required. Network representation is based on document (textual) similarity and on concept (ontological) similarity measures.	Maturing Service	Logic, Data	Atomic	Nodes (Documents, Persons...), Associations between Nodes	Nodes which are associated to a given node	1, 3, 4, 5, 6
2	User Profile Service	These services are responsible for the gathering and representation of user related data and inferences based on this representation. This facilitates the context aware behaviour of the system in particular context sensitive information retrieval and task recognition.	The user profile services are a set of services covering services for user data representation as well as services for gathering user data and retrieval of user related data, and inference mechanisms.	Maturing Services	Data	Framework	User actions and attributes	Static and dynamic user data	1, 2
3	Domain Modelling Service	Services for informal modelling of structured data by non-expert users.	These services facilitate the graphical or textual creation and revision of models	OLME	UI, Logic, Data	Framework		(semantic) models	

Annex A.6 Design Study “DS7: Kasimir” Service Collection

Responsible Partner: SAP

Kasimir provides a framework for integrated and collaborative task management. The design study investigates a special feature of Kasimir, namely task patterns which are especially interesting in the context of process knowledge maturing. Table 15 specifies the identified services in detail.

Table 15: Design Study “DS7: Kasimir” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	STMF (Semantic Task Management Framework)	The STMF offers a number of task management-related functions for developers of personal information management / personal task management applications.	STMF service implemented in Java, deployed as OSGI service in the Nepomuk Social Semantic Desktop, invoke exposed STMF service interface via Java, XML/RPC or as Web-Service.	OLME, PLME	Logic, Data	Atomic service	User task data passed on from service #2 or request for repository data (again from service #2)	Persistent storage of data in RDF repository or data read from repository and passed on to service #2	2
2	Kasimir	Service for manipulation of task information and further task-related personal information by users.	Swing GUI	OLME, PLME	UI, Logic	Service cluster (uses service #1)	User interaction, data from repository	Graphical representation of task data, data to be written to repository (via service #1)	User, 1

Annex A.7 Design Study “DS8: SOBOLEO” Service Collection

Responsible Partner: FZI

Table 16 specifies the identified services in detail.

Table 16: Design Study “DS8: Soboleo” Service Collection

#	Service Name	Service Description (non-technical)	Technical Service Specification	MATURE Service Type	Technical Service Type	Granularity	Input	Output	Interaction
1	SOBOLEO Ontology Editor Service	The ontology editor service enables collaborative creation, update, maintenance, and operationalization of a SKOS model	The editor is an AJAX application (based on Google Webtoolkit, i.e. implemented in Java) that allows creation, update, maintenance, and operationalization of a SKOS model for multiple users	PLME, OLME	UI, Data	Atomic	Domain Expert Knowledge; SKOS ontology description	SKOS ontology description; logging of all operations and chat messages as text file	11
2	SOBOLEO Concept Adding Service	Allows users to add a new concept with a specific preferred label as prototypical concept to the ontology	Allows authenticated users to add a new concept via an HTTP request. The preferred label is passed parameter. If such a concept does not exist yet, a new one is created and put as narrower concept of prototypical concept.		Data	Atomic	HTTP request with user name, password, and preferred label	Success message	1, 11
3	SOBOLEO Ontology Export Service	The ontology export service provides an output of the current state of the ontology	Provides via HTTP request a dump of the current state of the ontology in SKOS format with turtle notation		Data	Atomic	HTTP request	Ontology in SKOS format in turtle notation	

4	SOBOLEO Ontology & Resource Browser	The Ontology & Resource Browser enables the user to explore the current state of the ontology, persons and the annotated web resources.	For each concept, provides a web page with the description of the concept, links to related concepts and information about the most recent annotations. Further it shows information about the persons most connected to the concept.		UI	Atomic	HTTP request + concept id if needed	Web page describing concept and related resources and persons.	11
5	Atom Feed	Provides machine readable information about the newest annotations for a particular concept (or the entire ontology)	An ATOM feed with the most current annotations. A parameter allows the user to receive only the annotations for particular concepts.		Data	Atomic	HTTP request + concept id if needed	ATOM feed	
6	SOBOLEO Annotation Service	enables users to save bookmarks of web resources and to annotate them with concepts from the ontology or with arbitrary terms that are automatically added to the ontology	is an AJAX tool that enables the users to save the current web page as bookmark and to annotate it with concepts from the ontology or with arbitrary terms that are automatically added to the ontology as prototypical concept. It can be stored as bookmark within the browsers that opens a popup with url and title filled out for the current web page.	PLME, OLME	UI	Atomic	HTTP request with document title + url		1, 11

7	SOBOLEO Annotation Adding Service	Allows to add an annotation that is made from another application than SOBOLEO but with the same ontology	Allows for authenticated users to add an annotation, e.g. done with another annotation tool that uses the same ontology, via HTTP request. URL and title of the annotated document and the concept labels the document is annotated are passed as parameters		Data	Atomic	HTTP request with user name, password, document url, title, and concept labels to annotate with	Success message	
8	SOBOLEO Semantic Search Service	allows users to search for annotated web resources and for persons	Provides a web page that enables users to enter a search string in order to find annotated web resources and for persons.		UI, Logic		search string	web page with result set of annotated web resources and persons and suggestions for query relaxations or refinements	11

8.1	SOBOLEO Web resources Search Service	allows users to search for annotated web resources	Provides a web page that enables users to enter a search string in order to find annotated web resources. The search string is analyzed for occurrence of concept labels. Then the service looks for indexed web resources annotated with these concept labels or with narrower ones. This result is combined with a full text search over all annotated web resources. Query refinements and relaxations are also proposed.		UI, Logic	Atomic	search string	web page with result set of annotated web resources and suggestions for query relaxations or refinements	11
8.2	SOBOLEO People Search Service	users can search for who might be knowledgeable about a specific topic	The SOBOLEO People Search Service enables users to enter a search string in order to find other persons for a specific topic based on their activities. The search string is analyzed for occurrence of concept labels. The search engine looks for users who already used these concepts or narrower ones (e.g. for annotation)		UI, Logic	Atomic	search string	web page with result set of users and suggestions for query relaxations or refinements	



9	SOBOLEO Logging Service	logging of user activities	logging of all user activities with timestamp within a text file			Atomic	activity details	log as text file	
10	SOBOLEO Statistics Service	provides an aggregated view of the activities of each user	provides an aggregated view of the activities of each user; i.e. count an activity is performed		Logic	Atomic	HTTP request	comma separated list of activities per user	9

Annex B Service Fact Sheet

This section presents the service fact sheet that was used to specify the already existing services and to identify services to be integrated for the first prototype of the MATURE system. The service fact sheet consists of three parts:

1. Service Requirement Fact Sheet: Provides an high level view on the service and the implemented features.
2. Service Solution Fact Sheet: Provides a description of the selected implementation approach.
3. Service Implementation Report: Provides information about the implementation status of each feature, dependencies on technologies or standards and information on how the service was tested (on which platform, which test input / output).

The template is presented in the following.

1. Service Requirement Fact Sheet

<u><Service Name></u>	
<i>Service Overview</i> ²	<it is a brief description of the service underlining what high level functionalities will be available>
<i>Features Summary</i> ¹⁰	<it provides the description of each service feature, we could associate a feature to a method of the service>
<Feature 1>	<Describe the feature in term of what it does. If it is associated to a method details about input parameters and results should be provided>
<Feature 2>	
...	
<Feature N>	
<i>Comments</i> ¹¹ :	<it provides optionally graphics, comments or references to the full service description>

⁹ Service Overview is a brief description of the service underlining what high level functionalities will be available

¹⁰ Feature Summary provides a description of each service feature (public method)

¹¹ Comments might be provided like graphics, references to the full service description etc.

2. Service Solution Fact Sheet

<u><Service Name></u>	
<i>Solution Overview</i> ¹²	<it is a brief description of the selected implementation approach>
<i>Reviewer Comments</i> ¹³ :	<it provides optionally the possibility to state comments, hints and advises from an experienced MATURE service developers regarding the above approach>

¹² Solution Overview is a brief description of the selected implementation approach

¹³ Comments might be filled in by an experienced programmer to advise or provide hints

3. Service Implementation Report

<u><Service Name></u>	
Implementation Status¹⁴	<p><For each feature provides info about the implementation status</p> <ol style="list-style-type: none"> 1. design available -> link, copy or attached preliminary sequence and class diagrams 2. interface available -> link, copy or attached description of interface 3. implementation started -> start-, end-date of implementation 4. implementation completed -> code available in CVS tested -> reference to the final code>
<Feature 1>	
...	
<Features N>	
Dependencies¹⁵	<here it should be clarified which technologies or standards the service depends on.>
<Technology 1>	<For example WSRF.NET, GT4 core or a feature provided by a specific OS: a brief description of the technology should be provided with reference to find out detailed information>
...	
<Technology N>	
Tested Platform¹⁶	<describes on which OS the service has been tested>
<Platform 1>	<provides information about the results of testing. If some specific setting should be done, it has to be described here. If the testing has not been successfully, it should be outlined the problem that has to be faced (if identified)>
...	
<Platform N>	
Test Input/Output	<provides test inputs and outputs for each feature>

¹⁴ This is an optional description of the service implementation:

1. design available -> link, copy or attached preliminary sequence and class diagrams
2. interface available -> link, copy or attached description of interface
3. implementation started -> start-, end-date of implementation
4. implementation completed -> code available in CVS
5. tested -> reference to the final code

¹⁵ Dependencies clarify which technologies or standards the service depends on. For example WSRF.NET, GT4 core or a feature provided by a specific OS: a brief description of the technology should be provided with reference to find out detailed information

¹⁶ Tested platforms describes on which OS the service has been tested. Further the test settings or problems that have been faced should be outlined.

<u><Service Name></u>	
<Feature 1>	<provides sample input and output for each feature for testing reasons>
...	
<Feature N>	



Annex C MATURE Message Model

This annex introduces the first version of the MATURE Message Model. Table 17 depicts the WSDL interface implementing the first version of the MATURE Message Model.

Table 17: WSDL implementing the MATURE Message Model (First Version)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:mature="http://www.boc-eu.com/mature/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="MATURE"
  targetNamespace="http://www.boc-eu.com/mature/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.boc-eu.com/mature/">
      <xsd:element name="search">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="searchString"
type="xsd:string" maxOccurs="1" minOccurs="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="searchResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="weblink" type="xsd:anyURI" />
            <xsd:element name="weblinklist"
type="mature:weblink"> </xsd:element>
            <xsd:element name="content"
type="xsd:string"></xsd:element>
            <xsd:element name="conceptlist"
type="mature:concept"></xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="weblink">
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
          <xsd:element name="weblink"
type="xsd:anyURI"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="concept">
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
          <xsd:element name="concept"
type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="maturing">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="fulltextcontentString"
type="xsd:string"></xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="maturingResponse">
        <xsd:complexType>
```

```

        <xsd:sequence>
            <xsd:element
name="readabilityscoreintarray" type="mature:readabilityscore"></xsd:element>
            <xsd:element name="conceptlabelstring"
type="xsd:string"></xsd:element>
            <xsd:element name="tagsetstringarray"
type="mature:tagset"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

    <xsd:complexType name="readabilityscore">
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="readabilityScore"
type="xsd:string"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="tagset">
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="tagset"
type="xsd:string"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="pageranking">
        <xsd:complexType>
            <xsd:sequence>

                <xsd:element name="weburreadability"
                    type="mature:weburreadability">
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="pagerankingResponse">
        <xsd:complexType>
            <xsd:sequence>

                <xsd:element name="orderedweburllist"
type="mature:weburll"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="weburreadability">
        <xsd:sequence maxOccurs="unbounded" minOccurs="1">
            <xsd:element name="weburll" type="xsd:anyURI"
maxOccurs="1" minOccurs="1"></xsd:element>
            <xsd:element name="readabilityscore"
type="mature:readabilityscore" maxOccurs="1" minOccurs="1"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="storage">
        <xsd:complexType>
            <xsd:sequence>

                <xsd:element name="content"
type="xsd:string"></xsd:element>
                <xsd:element name="weburll"
                    type="xsd:anyURI">
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

```



```

        <xsd:element name="conceptlist"
            type="mature:concept">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="storageResponse">
    <xsd:complexType>
        <xsd:sequence>

            <xsd:element name="statusmessage"
type="xsd:boolean"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>
<wsdl:message name="searchRequest">
<wsdl:part element="mature:search" name="input"/>
</wsdl:message>
<wsdl:message name="searchResponse">
<wsdl:part element="mature:searchResponse" name="output"/>
</wsdl:message>
    <wsdl:message name="maturingRequest">
        <wsdl:part name="input" element="mature:maturing"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="maturingResponse">
        <wsdl:part name="output" element="mature:maturingResponse"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="pagerankingRequest">
        <wsdl:part name="input" element="mature:pageranking"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="pagerankingResponse">
        <wsdl:part name="output"
element="mature:pagerankingResponse"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="storageRequest">
        <wsdl:part name="input" element="mature:storage"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="storageResponse">
        <wsdl:part name="output" element="mature:storageResponse"></wsdl:part>
    </wsdl:message>
    <wsdl:portType name="MATURE">
<wsdl:operation name="search">
<wsdl:input message="mature:searchRequest" />
<wsdl:output message="mature:searchResponse" />
</wsdl:operation>
        <wsdl:operation name="maturing">
            <wsdl:input message="mature:maturingRequest"></wsdl:input>
            <wsdl:output message="mature:maturingResponse"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="pageranking">
            <wsdl:input message="mature:pagerankingRequest"></wsdl:input>
            <wsdl:output message="mature:pagerankingResponse"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="storage">
            <wsdl:input message="mature:storageRequest"></wsdl:input>
            <wsdl:output message="mature:storageResponse"></wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
<wsdl:binding name="MATURESOAP" type="mature:MATURE">

```

```

<soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="search">
  <soap:operation
    soapAction="http://www.boc-eu.com/mature/search" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="maturing">
  <soap:operation
    soapAction="http://www.boc-eu.com/mature/maturing" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="pageranking">
  <soap:operation
    soapAction="http://www.boc-eu.com/mature/pageranking" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="storage">
  <soap:operation
    soapAction="http://www.boc-eu.com/mature/storage" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="MATURE">
<wsdl:port binding="mature:MATURESOAP" name="MATURESOAP">
<soap:address location="http://www.boc-eu.com/mature/" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Annex D

Knowledge Item Meta Data

This Annex provides details on the first version of the knowledge item meta data format. This format will be refined for the final version of this deliverable. Mature specific parameters are orange highlighted.

Table 18 presents details about the General Tag that describes the knowledge item as a whole.

Table 18: Knowledge Item Meta-Data - General Tag

General Tag					
Nr	Name	Description	Value space	Datatype	Example
1.1	Identifier	Unique label for the knowledge item			
1.1.1	Catalogue	Name of the identification scheme			“URI”
1.1.2	Entry	Value of the identifier.			
1.2	Title	Name given to this knowledge item.			
1.3	Language	The primary language used within this knowledge item.			
1.4	Description	A textual description of the content of this knowledge item.			
1.5	Keyword	A keyword or phrase describing the topic of this knowledge item.			
1.6	Structure	Underlying structure of this knowledge item.	<p>atomic: an object is indivisible</p> <p>collection: a set of objects with no specified relationship between them</p> <p>networked: a set of objects with relationships that are unspecified.</p> <p>hierarchical: a set of objects whose relationships can be represented by a tree structure.</p> <p>linear: a set of objects that are fully ordered.</p>		

Table 19 presents details about the lifecycle tag that describes features related to the history and current state of the knowledge item and those who have affected this knowledge item during its evolution. To include also MATURE specific aspects, the maturity level has been included as an attribute.

Table 19: Knowledge Item Meta-Data - Lifecycle Tag

Lifecycle Tag					
Nr	Name	Description	Value space	Datatype	Example
2.1	Version	The edition of this knowledge item.			
2.2	Status	The completion status of this learning object.	draft final revised		
2.3	Contribute	Entities that have contributed to the state of this knowledge item			
2.3.1	Role	Describes the kind of contribution	Author Publisher Unknown Initiator Terminator Validator Editor Graphical designer Technical implementer Content provider Technical validator Educational validator Script writer Instructional designer Subject matter expert		
2.3.2	Entity	Information about the contributing entity.			
2.3.3	Date	Date of the contribution			
2.4	Maturity Level	Information about the maturity level of the	Level 1a: Expressing ideas		

		knowledge item	Level 1b: Appropriating Level 2: Distributing in communities Level 3: Formalizing Level 4: Ad-hoc training Level 5: Standardizing		
--	--	----------------	---	--	--

Table 20 presents detailed information about the meta-metadata tag. This tag groups information about the metadata instance itself (rather than the knowledge item that the metadata instance describes).

Table 20: Knowledge Item Meta-Data – Meta-Metadata Tag

Meta-Metadata Tag					
Nr	Name	Description	Value space	Datatype	Example
3.1	Identifier	A globally unique label that identifies this metadata record			
3.1.1	Catalogue	Name of the identification scheme for this entry.			
3.1.2	Entry	Value of the identifier within the identification scheme.			
3.2	Contribute	Those entities that have affected the state of this metadata instance during its lifecycle.			
3.2.1	Role	Kind of contribution	creator validator		
3.2.2	Entity	The identification of and information about entities.			
3.2.3	Date	Date of the contribution			
3.3	Metadata Schema	The name and version of the authoritative specification used to create this metadata instance.			“LOMv1.0”
3.4	Language	Language of this metadata instance.			

Table 21 presents detailed information about the technical tag. This tag groups technical requirements and technical characteristics of the knowledge item.

Table 21: Knowledge Item Meta-Data - Technical Tag

Technical Tag					
Nr	Name	Description	Value space	Datatype	Example
4.1	Format	Technical datatype of the knowledge item	MIME types		“video/mpeg” “text/html”
4.2	Size	The size of the digital knowledge item in bytes.			
4.3	Location	A string that is used to access this knowledge item.			
4.4	Requirement	The technical capabilities necessary for using this knowledge item.			
4.4.1	OrComposite	Grouping of multiple requirements. The composite requirement is satisfied when one of the component requirements is satisfied.			
4.4.1.1	Type	The technology required to use this knowledge item, e.g. hardware, software, network, etc.	operating system browser		
4.4.1.2	Name	Name of the required technology to use this learning object			“ms-windows” “firefox”
4.4.1.3	Minimum version	Lowest possible version of the required technology to use this knowledge item.			“4.2”
4.4.1.4	Maximum version	Highest possible version of the required technology to use this knowledge item			“6.0”
4.5	Installation remarks	Description of how to install this knowledge item			
4.6	Other Platform	Information about			“sound card”

	Requirements	other software and hardware requirements			
4.7	Duration	Time a knowledge item takes when played at intended speed.			

Table 22 presents information about the rights tag. This tag groups the intellectual property rights and conditions of use for the knowledge item.

Table 22: Knowledge Item Meta-Data - Rights Tag

Rights Tag					
Nr	Name	Description	Value space	Datatype	Example
5.1	Cost	Whether the use of this knowledge item requires payment			
5.2	Copyright and Other Restrictions	Whether copyright or other restrictions apply to the use of this learning object.			
5.3	Description	Comments on the conditions of use of this knowledge item			

Table 23 presents information about the relation tag. This tag groups features that define the relationship between the knowledge item and other related knowledge items.

Table 23: Knowledge Item Meta-Data - Relation Tag

Relation Tag					
Nr	Name	Description	Value space	Datatype	Example
6.1	Kind	Nature of the relationship between this knowledge item and the target knowledge item.	Based on Dublin Core: is part of has part is version of ...		
6.2	Resource	The target knowledge item that this relationship references			
6.2.1	Identifier	A globally unique label that identifies the target knowledge item			
6.2.1.1	Catalogue	The name of the identification for this entry.			“URI”

6.2.1.2	Entry	The value of the identifier within the identification that designates or identifies the target learning object.			
6.2.2	Description	Description of the target knowledge item			

Table 24 presents information about the annotation tag. This tag provides comments on the use of the knowledge item and provides information on when and who created the comments. The rating has been included for MATURE to enable the rating of knowledge items with predefined values.

Table 24: Knowledge Item Meta-Data - Annotation Tag

Annotation Tag					
Nr	Name	Description	Value space	Datatype	Example
7.1	Entity	Entity (e.g. person, organization) that created this annotation			
7.2	Date	Date this annotation was created			
7.3	Description	The content of this annotation.			
7.4	Rating	Rating of this knowledge item			

Table 25 presents information about the classification tag. This tag describes this knowledge item in relation to a particular classification system.

Table 25: Knowledge Item Meta-Data - Classification Tag

Classification Tag					
Nr	Name	Description	Value space	Datatype	Example
8.1	Purpose	The purpose of the classifying knowledge item	Discipline Idea Prerequisite Educational objective Accessibility Restrictions Educational level Skill level Security level Competency		
8.2	Source	The name of the classification system.			
8.3	Taxon	A particular term within a taxonomy. A taxon is a node that has a defined label or term.			
8.3.1	Id	The identifier of the taxon, such as a number or letter combination provided by the source of the taxonomy.			
8.3.2	Entry	The textual label of the taxon			
8.4	Description	Description of the knowledge item relative to the classification purpose.			
8.5	Keyword	Keywords and phrases descriptive of the learning object relative to the stated classification purpose.			