Contract N° 034824

Workpackage 2
Automatic Code Generation from Models

Deliverable 2.1
Design of Software Generation Prototype

Project funded by the European Community under the "Information Society Technology" Programme

Contract number: 034824
Project acronym: OPAALS
Title: Open Philosophies for Associative Autopoietic Digital Ecosystems

Deliverable Nº: D2.1
Due date: May 31, 2007
Delivery date: July 23, 2007

Short description:
This report elaborates mainly on automatic code generation from SBVR models.

Author: Raimund Eder, Thomas Kurz, Thomas J. Heistracher, Victor Bayon, Mario Russo, Antonella Filieri
Partners contributed: UCE
Made available to: OPAALS Consortium and European Commission

Versioning		
Version	**Date**	**Author, Organisation**
0.1	30/04/2007 (first submission)	SUAS, UCE
0.2	15/06/2007 (comments of 1^{st} review)	SUAS, UCE
0.2	20/07/2007 (comments of 2^{nd} review)	SUAS, UCE

Quality check
1st internal reviewer: Paul Krause, University of Surrey
2nd internal reviewer: Stan Hendryx, Hendryx & Associates
3rd internal reviewer: Frauke Zeller and Josef Wallmannsberger, Kassel University

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License. To view a copy of this license, visit:
http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

creative commons

COMMONS DEED

Attribution-NonCommercial-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Contents

List of Figures

List of Tables

Executive summary

This work mainly intends to pave the way for a feasible code-generation prototype that is based on natural-language modeling artifacts. It exhibits three main characteristics of *introduction*, *conceptualisation*, and *realisation* as follows: i) brief presentation of fundaments, current concepts and frameworks in model-based code-generation , ii) conceptual study of model-driven automatic code-generation, and iii) specification of the design of the software generation prototype This report should support the 'diving-into' the challenges and difficulties in model-driven code generation for the computing partners as well as the business domain and the science domain partners in OPAALS. This report also puts the related OPAALS research into context of preceding research in that field. It provides a short overview of state-of-the-art in business modeling and tries to interlink that with the potentials of SVBR as a natural-language based notation of (business) workflows. To achieve this the model-to-model transformation is explained at first together with a short recap on Model Driven Development (MDA) fundamentals. Existing frameworks suitable to the aimed goal are described shortly and brought into the context of code generation and generative programming. After that business modeling is explained and different languages, standards and technologies supporting business modeling are listed together with an outline of their application area. As part of this chapter Semantics of Business Vocabulary and Business Rules (SBVR) is introduced as a language for business vocabulary and rule definitions. Afterwards Business Modeling in the context of Digital Ecosystems is explained including a recap of the preceding research project *Digital Business Ecosystems*[1]. Next the design of the demonstrator is outline and the reader gets a basic overview about the showcase that be used as a first test of the functionality of the code generation demonstrator. The paper concludes with further research topics that have to be considered for i) the deliverable D2.2 and ii) first lines of research that will be worked on in phase II. In the appendix the reader

[1] http://www.digital-ecosystem.org/

can find the detailed internal review of this deliverable which adds many critical comments and suggestions for further elaboration on natural language based code and workflow generation. We added the comments in order to give a clear understanding of the known issues of the work presented here and it outlines therefore, also an abstract for future research within WP2.

1 Introduction

1.1 Aims of this paper – success criteria

As the paper at hand is the first deliverable in WP2 we want to state clearly the intention of WP 2 *Automatic Code Generation from Models* and what we want to achieve regarding code generation and natural language. Semantics for Business Vocabulary and Business Rules (SBVR) is the approach of choice for code generation and business modelling in WP2 and the initial work which has been done in the Digital Business Ecosystem (DBE) project can be utilised for this workpackage.

Although, there were already results out of SBVR work in the DBE, the focus was on business modelling and search for these business models in a distributed network. Beside working further on elaborating the business modelling capabilities of SBVR, the focus of WP2 and this deliverable in particular, is the potential of SBVR as a basis for automated code generation and workflow manipulation.

Consequently, the aim of this paper is not only to give a design of a prototype for code generation, but also discuss in depth the underpinning technologies and languages. The reader is introduced to the basic principles behind the model driven approach as well as model transformations. An overview about state of the art business modelling is given and both, DBEs BML 1.0 and SBVR are discussed in the context of OPAALS. Furthermore, the modeling tools for SBVR are presented and suggested for the implementation of the software generation prototype. After this mainly theoretical work, a design and use case is scetched which will be implemented for D2.2 *Automatic code and workflow generation from natural language models*(month 18). Additionally, the practical point of view especially for small and medium sized enterprises can be seen as another aim of this paper.

1.2 Chapter Overview

The first part of this deliverable provides a short recap of model driven architecture in order to show the necessary foundation for the understanding of model transformation. After that the model-to-model transformation is introduced as the approach chosen for the software generation demonstrator. Different frameworks for model transformations are introduced at a very high level. This chapter finishes with a section on generative programming that currently mainly is achieved by template engines or stylesheet transformations.

Chapter 3 is dedicated to business modeling in general. It provides an introduction to the concepts of business modeling and why it is important. It summarizes the state of the art of business modeling techniques and introduces SBVR as a natural language business modeling facility. The role of SBVR within the OPAALS research project is outlined and finally the current status of the *ISUFI Business Modeler* as a free and open source tool is given.

After that the Business Modeling Language is explained in chapter 4. This part of the deliverable is focuses on the DBE outcomings regarding business modeling and also provides insights in simple techniques of information aggregation like RSS, RDF and dynamic languages used in the Web 2.0 environment.

Finally, the leveraged standards, frameworks and the design of the demonstrator is summarized in chapter 5. Firstly, information about the scenario the demonstrator should be able to cope with is introduced to the reader. After that the high level architecture is outlined. This includes the description of the possible model transformation frameworks available for the demonstrator development. Additionally, the rapid application development frameworks are introduced for which it is easy to create platform specific code.

2 Software Generation using Model Transformation

In 2001 the Object Management Group (OMG) proposed the Model Driven Architecture (MDA) with the intention to consolidate a number of trends in the software generation processes at that time [2]. The key principle of MDA is to separate the specification of the design and the architecture of software. The architecture includes the low level aspects of software such as programming platform and frameworks. The design is the high-level view of the software that can be seen as independent of the architecture. Without using the MDA principles the design and the architecture often go hand in hand and cannot exist without another. But if the software is built upon the MDA approach these two can be clearly separated. In MDA a model is a simplified description of a real world business. Although MDA is agnostic of any modelling format it often is used together with OMG's modelling standard stack.

This chapter provides a short overview about techniques and standards necessary to create a demonstrator to generate software out of a model described in natural language. It also tries to leverage the MDA and Model Transformation.

2.1 Meta Object Facility

According to [3] the primary problem of data exchange is incompatible metadata across systems in a distributed software environment. Metadata is the collection of all data describing the semantics and structure of data e.g. in databases. Proprietary metadata brings limitations. For example databases managements systems have individual metadata formats preventing the seamless metadata exchange. (Fortunately most DBMS use DDL as the language of choice to create the metadata structures in the data dictionary.) The OMG claims that resolving these differences will help the integration in distributed software environments and therefore adopted

the Meta Object Facility (MOF) as an OMG standard, as a common management framework for metadata and metadata services.

MOF borrows the basic building blocks to define abstract metamodels from UML. Therefore, MOF metamodels look similar to UML class models and common modelling tools can be used to create metamodels. Although the definition of metamodels in MOF is similar to modelling class diagrams, MOF is not limited to object oriented modelling. [2] [4]

2.2 Metalevels

The OMG suggests different levels of abstractions in the MDA approach. The list of the four abstraction levels is summarized in table 2.1. The most abstract level is M3 also known as the metametamodel. The metametamodel is the model used to describe how a metamodel has to look like. The metametamodel is self describing which means that this metametamodel is also used to model the metametamodel. The OMG answer to the metametamodel is the Meta Object Facility (MOF) which is recommended to use for the modelling of all metamodels.

The metamodels are instances of MOF constructs and build the M2 metalevel. This are the metamodels used to create models (e.g. UML). Models are instances of metamodels and according to the OMG build the M1 layer (e.g. Class diagrams reflecting the structure of software). The lowest abstraction level are the instances of the M1 model. The M0 instances are then instances of classes (e.g. a instance of a person-class with the name "Bob").

M3 is the end of the line because MOF (as the M3 layer) is defined using MOF. Frankel states in [2] that MOF defines a MOF-compliant model of its own constructs.

Metalevel	**Description**	**Also known as**
M3	MOF	Metametamodel
M2	Metamodels, consisting of instances of MOF	Metamodel
M1	Models, consisting of instances of M2 metamodel	Model
M0	Objects and Data	Instances

Table 2.1: MDA Metalevels. Adopted from [2]

2.3 Metamodels

Metamodels can be seen as models of a model or description of a model. A metamodel describes the terms and structure of a model. An example for a well known metamodel is the Unified Modeling Language (UML) which is used for software modelling. Classes defined in UML are then the model of the software. Currently there exists a high number of different metamodels for different purposes. But often these purposes are overlapping. For example the information contained in one model is partially also contained in another model (an ER diagram includes to a certain extent business object information just as an UML class diagram does, but both models can include information that can only be expressed in one specific models out of the two). If this behavior is extended it is then possible to generate another model out of the information of one or more different other models - or models can be merged together. This process is called Model Transformation (MT).

2.4 Model-to-Model Transformation

Model-to-model (M2M) transformation is one of the fundamental parts of software development based on MDA. Using the metamodel of a model, a M2M transformation can be performed. For example an Entity Relationship Model (ER) can be generated out of an class diagram. As this is a fundamental part of MDA the OMG issued model transformation in the *MOF Queries/Views/Transformations (MOF QVT)* standard. Figure 2.1 summarizes the M2M transformation where a model A conforming to metamodel A is transformed to a model B conforming to a metamodel B. The transformation is done using a model transformation T which itself is also compliant to a metamodel T. All metamodel itself conform to a common metametamodel for example MOF. There is a large number of engines for model transformations which will be introduced in the following section. In section 5.2.3 of this paper the introduced engines will be explained in more detail.

2.5 Model Transformation Engines

This section discussed a selected number of available model transformation engines available today. It has to be mentioned that no engine can be seen as mature yet because although MDA exists since 2002 the M2M approach is fairly new and more

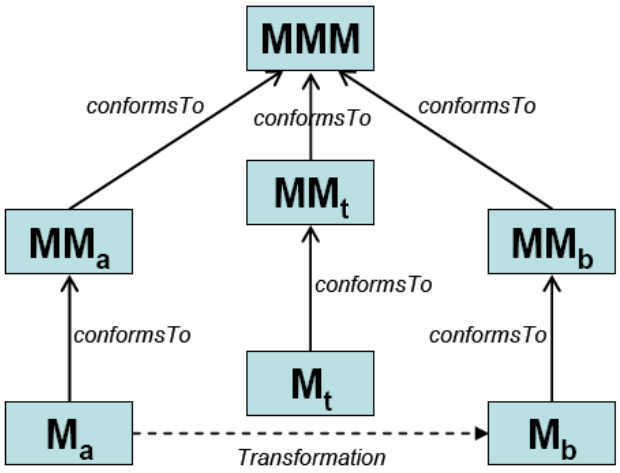

Figure 2.1: Model-to-Model Transformation. [1]

research is needed in that area. The following subsections provide examples for model transformation frameworks. There currently exist more then these but not as freely available and usually part of bigger MDA toolsets like Borland Together[1], or OptimalJ[2] offered by Compuware. It is expected that all MDA tools in near future will include model transformation to a certain extent.

2.5.1 Model Transformation Framework

The Model Transformation Framework (MTF)[3] is a prototype for working with EMF based metamodels. Among other goals the MTF aims to support two-way and many-way model transformations. Through plugin mechanisms it should be easy to integrate different model transformation or expression languages and it is open to text- and graphical syntaxes. MTF was developed to prototype concepts out of the QVT specification but does not aim to be an QVT implementation in the strict sense.

[1] http://www.borland.com/together
[2] http://www.compuware.com/products/optimalj
[3] http://www.alphaworks.ibm.com/tech/mtf

2.5.2 SmartQVT

SmartQVT[4] is an open source implementation of QVT's operational language. Transformations are compiled into java code to do the transformations. SmartQVT is like most other model transformation tools based on the Eclipse Modelling Framework (EMF) which offers a capable framework for modelling tools. Additionally a Eclipse plugin is provided which offers a code editor. The project is licensed under the Eclipse Public License (EPL). SmartQVT is being developed by France Telecom R&D and has been partly financed by the European IST Modelware project.

2.5.3 ATLAS Transformation Language

The ATLAS Transformation Language (ATL) is a model transformation language developed at the *Institut national de recherche en informatique et en automatique (INRIA)* ATL was developed to create an implementation of the QVT Request For Proposal. It also leverages the EMF framework and offers a large number of ready-to-use transformations [1].

As ATL is licensed under the EPL, has a large amount of available resources and is also the framework of choice from other OPAALS partners (IITK) it will be used as the tool for the software generation demonstrator and therefore discussed more detailed in section refsec:atl.

2.6 Object Constraint Language for Business Rules

Object Constraint Language (OCL) is a formal language to describe expressions on UML models. It was originally developed by IBM and now officially is part of the OMG UML standard. OCL is a text language for objects constraints in class diagrams, conditions in sequence diagrams, pre- and postconditions in method specifications and queries on MOF compliant models and metamodel. OCL is heavily used in Model to Model (M2M) transformation and is a basic building block for ATL which is discussed in 5.2.3. Details of OCL can be found in the OMG specification[5].

[4]http://smartqvt.elibel.tm.fr, http://sourceforge.net/projects/smartqvt/

2.7 Generative programming

Creating software using automatic source code generation is called *Generative Programming* [6]. The part of generative programming relevant for the software generation demonstrator is code generation.

Code generation is a major part in the model driven development and the most common form is the generation of source code out of models. There is a large number of source code generation utilities, but most of them aim to create stubs of for example classes modeled in UML. A common approach to this kind of source code generation is using template engines that generate code parts out of defined amount of input data.

A template engine is a software component that combines a data model with a textual template and outputs a resulting document. In the template the places that are reserved for the input data are marked using template specific notation and can vary according to the template engine.

The Extensible Stylesheet Language Transformation (XSLT) is a widespread template mechanism specific for transforming XML data to all kinds of output data. Although the main purpose is to transform one XML representation of data in another XML representation it is used for any XML to document transformations.

3 State-of-the-Art of Business Modeling

This chapter introduces business modelling in general and also the latest movements in the business modelling standards and technologies. SBVR and it's role in natural language as a business modelling tool is introduced and a short overview about its impact on the project.

3.1 Introduction to business modelling

The fast development and adoption of Information and Communication Technologies as well as the rapid growth and diffusion of the World Wide Web are determining the emergence of a new economic paradigm, known as *network society*.

This new context has enabled the shift from traditional companies' activities toward an electronic dimension, allowing an increase in efficiency and a reduction of coordination costs and enabling, at the same time, the creation of relationships and interactions among firms distributed worldwide. Transient networks constitute new organisational forms, which exhibit unmatched connectivity, flexibility and adaptability.

In this perspective companies try to adopt business and technological solutions that can allow them to easily configure and reconfigure themselves, to reach effectiveness and efficiency in organising their internal or collaborative processes and, at the same time, to enhance the value proposition for their customers. Anyway, the digitalisation of business activities often does not allow to reach the expected benefits. At least 350 Billion USD is lost each year on IT worldwide[1] due to abandoned projects, rectification of errors and consequential losses. Not all of this could be saved by a better development process, because of operational errors or changing

[1]Sources: Giga Group, Gartner Group, Standish Group, CCTA, BrunelUniversity

business conditions, but potential savings are at least 200 Billion USD per year. This requires a radical re-think of the way systems are built [7]. One of the reasons behind this problem is a recurrent misalignment between IT and business. In this perspective, modelling the business in order to support and drive the design of effective technological solutions emerges as a key factor to configure a successful system in the new business environment.

Another interesting point is closely connected with the emergent networked organisational forms. Competing in this context poses in fact a number of integration problems that enterprises have to tackle, such as the integration with customers, among suppliers, between design and manufacturing sites or between partners. Such integration is not required only on a technological level. The possibility to establish a communication among different electronic systems, even though necessary, is not a sufficient condition to establish successful collaborations and partnerships. What is required is in fact the possibility to interconnect business models, in terms of convergent objectives, activities and interests. In this perspective, thus, integration does not refer only to the interconnections among physical sites and software applications, rather it is focused on the overall business and all its aspects. In other words, building and strengthening relationships among firms, as well as just implementing economic transactions, require a high knowledge about how firms work, about the resources they need, the processes they implement, the rules they apply and so on. In this perspective, the first fundamental step is, once again, a clear definition and representation about what the firm is, how it operates and how it can communicate and co-operate with other firms. This means that the first fundamental requirement for integration is business modelling. The more a business model is effective, the easier is the definition of communication, coordination, control and exchange mechanisms.

3.2 State-of-the-Art of Business Modelling

A business model can be defined as a computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of a business, a government or other organizations [8]. Enterprise modelling was born in the United States at the beginning of the 80's and emerged through large Computer Integrated Manufacturing projects. In the mid-80's, Europe launched several projects on enterprise modelling giving birth to several enterprise modelling

languages. As a result, in the 90's many commercial tools dealing with enterprise modelling or business process modelling appeared on the marketplace, as well as a myriad of workflow systems, each one with its own modelling environment. This intensive production of tools has led to a situation where the many tools are unable to interoperate and can hardly or not at all communicate and exchange models [9]. Currently, enterprise modelling is a wide and complex domain containing many different methodologies, languages, tools and techniques, often developed in different context for different scope.

The considerations in the previous section describe a situation where business and IT perspectives are often considered as completely separate entities in modelling and executing business activities. This view has clearly affected the development of enterprise and business modelling approaches, that could be roughly divided into two main categories. From one side, there are the approaches mainly based on a business perspective; on the other side, it is possible to consider languages and methodologies mainly related to designing and developing IT applications.

In details, in the first group there are many languages and frameworks aimed at modelling specific characteristics of the enterprise. The most general and wide range ones are the Zackman Framework [10], that structures the enterprise model in order to implement a supporting system in the short term and, to grant coherence among system and model in the long term for enterprise architecture modelling, and the GERAM, Generalised Enterprise Reference Architecture and Methodology, consisting of many components such as methodologies, modelling languages and concepts, partial and global models, tools [11]. More operational frameworks and languages are:

- Integrated Enterprise Modelling (IEM), developed at Fraunhofer Institute of Berlin [12]. It is a holistic methodical basis for modelling of business processes, based on the object-oriented modelling technique. It allows analysis and optimisation of processes and organisational structures of enterprises [13].

- Integrated DEFinition methodology (IDEF), developed at Saarbrucken University. It is a group of modelling methods, each with its own syntax, used to describe several operations in enterprise [14]. The most important methods are IDEF0 (Function Modelling), IDEF1 (Information Modelling), IDEF1X (Data Modelling), IDEF2 (Simulation Model Design), IDEF3 (Process Description Capture) and IDEF4 (Object-Oriented Design).

- Architecture of Integrated Information Systems (ARIS). It is based on an integration concept derived from a holistic analysis of business processes and ensures a consistent description from business management-related problems all the way down to their technical implementation [15].

- Process Specification Language (PSL), developed at National Institute of Standards and Technology. The goal of PSL is to create a process interchange language that is common to all manufacturing applications, generic enough to be decoupled from any given application, and robust enough to be able to represent the necessary process information for any given application [16].

- Workflow Process Definition Language (WPDL), defined by Workflow Management Coalition and oriented to the definition and exchange of workflow [17]. It was established as a meta-language for the exchange of build-time workflow process models through a batch procedure (import/export of process models).

- Business Process Modelling Language (BPML), developed by the Business Process Modelling Initiative and oriented to the business process representation [18]. BPML provides an abstract execution model for collaborative transactional business processes based on the concept of a transactional finite-state machine.

- Business Rules approach, developed by the Business Rules Group. It is a methodology to model business and domain knowledge through facts and rules, in a language that is understandable and simply to learn by the business expert [19]. Through this approach it is possible to identify and explicitly describe facts and rules defining enterprise organisation and to define structure and scope of its operational activities.

The second category of modelling methodologies includes approaches and languages developed in the technological area. Among them, there are:

- UMM, UN/CEFACT Modelling Methodology, defined as a UML profile. It is an incremental business process and information model construction methodology that provides levels of specification granularity suitable for communicating the model to business practitioners, business application integrators, and network application solution providers [20]. Its main goal is to understand and formalise the dependencies between partner processes for a problem domain.

- ebXML, developed by OASIS and UN/CEFACT to support e-business, overcoming the EDI approach. Its objective is to provide a standard that enables enterprises of any size, in any location, to meet and conduct business through the exchange of XML-based messages. In this way, ebXML provides a XML meta-model based upon prior UN/CEFACT work, specifically the meta-model behind the UMM. The main result should be the creation of a single global electronic marketplace.

- Rosetta Net, developed to promote the creation of open and standard business space among IT sector companies. It offers a robust non proprietary solution, encompassing data dictionaries, implementation framework, and XML-based business message schemas and process specifications, for e-business standardisation.

  In the same group are many industry initiatives and de-facto standards, such as those promoted by the Object Management Group (OMG). A very interesting framework developed by this standardisation body is the Model Driven Architecture (MDA), conceived as a framework for software development that aims at improving productivity, quality, and longevity of the software product [21]. It promotes the creation of highly abstract and machine readable models, designed independently from technology of implementation that could be translated, through specific tools, in low level design model, code and script for different platforms. In this wide context, several OMG' standards can be applied for software designing and development. The most important are:

- Unified Modelling Language (UML), a language for specifying, constructing, visualizing and documenting the models of software systems [22]. It provides a rich set of modelling concepts and notations that have been designed to meet the needs of typical software modelling projects.

- Meta Object Facility (MOF), a set of standard interfaces that can be used to define and manipulate interoperable meta-models and their corresponding models. From a modelling viewpoint, the MOF is used to define an information model for a particular domain of interest. From the data viewpoint, the MOF (or more accurately, a product of the MOF) is used to apply the OMA-based distributed computing paradigm to manage information corresponding to a given information model.

- XML Metadata Interchange (XMI) is an XML application that facilitates the standardized interchange of object models and metadata over the Internet among groups working in team development environments using tools and applications from multiple vendors. XMI can also be used to exchange information about data warehouses. XMI is based on three industry standards - XML, UML, and MOF (an OMG modeling and metadata repository standard). Since it is based on the MOF metamodel, UML models can be passed from tool to tool using XMI.

- Semantics of Business Vocabulary and Business Rules (SBVR) is a metamodel for vocabulary, propositions, and projections. Its purpose is to describe the meaning of concepts, propositions, and questions. The descriptive elements are concepts of formal logic. SBVR is a Computer Independent Modelling language and is thought to be the most business oriented language of the MDA framework [23].

  The main standards in metadata definition and exchange, resource annotation and domain modelling are related with W3C works. The W3C Consortium is currently responsible for the development and maintaining of many standards for modelling and exchange of information:

- eXtensible Markup Language (XML). It is used to format structured documents and data on the Web. XML is an efficient means to exchange data between Enterprise Modelling tools. In general, XML is extensible, platform-independent and it supports internationalisation and localisation.

- Resource Description Framework (RDF). It is the first recommendation on which the Semantic Web will be built. RDF provides a standard vocabulary and encoding techniques to attach metadata to any resource on the Web.

- Ontology Web Language (OWL) is a language for ontology definition. An OWL ontology in the abstract syntax is a sequence of axioms, facts, imports, and annotation.

These languages are closely connected each other, since they consitute the three main layers on which the Semantic Web is built. XML, as basic layer for serialization, underpins RDF in the definition of data structures, while the upper layer is represented by OWL, that allows to define logic relations among these structures.

3.3 Requirements for business modelling and SBVR

Section 3.2 describes the complex landscape of business modelling approaches and languages. The high degree of complexity in this field implies some difficulties in defining a general set of requirements that characterises such languages. This is explained by the specific needs and purposes each of them aims to address. For this reason, this section starts from a basic assumption related to the main objective we want to address, that is *providing support to businesses in the new economic environment represented by the network society, where digitalisation of business processes and inter-firm collaboration assume a central role*, as described in section 3.1. The pressing need for an effective business modelling in the emerging economic context is driven by the necessity to provide a bridge between the business perspective and the technological perspective in the implementation of IT systems. Using an enterprise architecture that bridges between strategy and corporate planning and IT planning and project management is one way to align IT organizations, priorities and development processes with company strategies in a systematic way in order to assure that IT efforts support company goals [24]. This implies that, in the new scenario characterised by networking enterprises, a business modelling approach has to address the creation of a close connection among these two different environments. As a consequence, the first requirement for a modelling language is the *capacity to capture and represent enterprise business knowledge and make it available for the IT experts that have to develop technological solutions for the enterprises* (Req.1). Several other requirements directly come from Req. 1. First of all, in order to effectively capture enterprise knowledge, *a business modelling language should allow business people to express such a knowledge in terms of business concepts* (Req. 2) that is, in terms of the actual concepts, actions and events that they have to deal with as they run their businesses. Another problem is related to software requirement gathering. In the traditional software development approaches, the problem owner has in his mind some ideas about what the systems should do in order to reach its objectives. These ideas are collected, usually by an analyst, in some specifications that guide the design and development phases. There are several problems in this approach. First of all, the semantics are in the mind of the reader and writer; moreover, the semantics are unreliable, because dependent on human perspectives; finally, search tools are not geared to semantics [7]. One possible solution is represented in Figure 3.1. Capturing requirements and representing them in a machine-readable form can enable mechanisms of transformation that allow to implement an Information

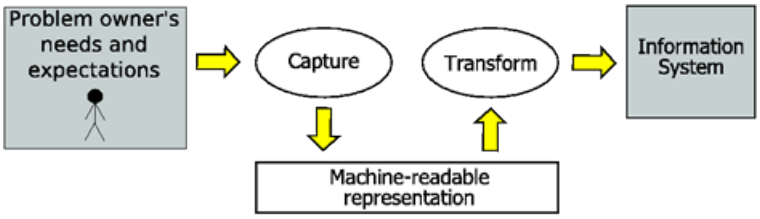

Figure 3.1: New approach for requirements transformation

System fully aligned to business objectives. This approach is a simplistic way to describe how business requirements can feed the process of generating system specifications, anyway it allows to explain how a strong alignement between these two different perspectives can be reached. This consideration implies that, for our purposes, a business modelling language has to enable and support the definition of requirements in order to make straightforward the process towards the implementation phase. In this perspective, it is desirable that a business modelling languages is integrable in a model driven approach for software development. Thus, *the business knowledge captured should be represented in, or easily converted into, a machine readable format* (Req. 3) in order to grant compatibility with future development of IT systems. This should allow, from one side, the development of technological solutions to digitalise internal processes and procedures; on the other hand, this should enable an easier development of electronic collaborative processes of different nature, such as e-business transactions as well as integrated information flows between partners.

The necessity to define a bridge between business and technology in the digitalisation on business activities poses some interesting basis for taking into consideration the concept of process. A business process is in fact the set of activities that a firm performs to reach a specific objective. Understanding how technology could support a specific process represents a first step in order to operatively bridge the gap between business strategy and IT solutions. Thus, in the perspective of enabling an easier business and software modelling in a network environment, focusing on processes could represent a very interesting perspective. As a consequence, *a business modelling language should enable the possibility to easily define connections between concepts upon which business models are built and more specific concepts, related to executable processes and the technological components that realise them* (Req. 4).

An interesting approach in order to satisfy these requirements is represented by the adoption of the Semantics of Business Vocabulary and Business Rules (SBVR)

as linguistic base for creating business models [25]. This standard provides a general linguistic metamodel that is mapped to formal logics, especially first order predicate logic, modal logic, basic arithmetic, and set theory [23].

Concerning Req. 1, it is important to highlight how SBVR is conceived with the aim of introducing the business perspective in the methodology for software development, creating a seamless process that starts from more abstract (business) models to specific models and code. Formalizing the Computing Independent Model and integrating it fully into the MDA process is in fact the principal way the OMG is contributing to building the bridge between business and IT, described above [23]. Another interesting perspective is that business people can be directly involved in the definition of models. This allows to overcome some communications problems between modellers and business people, that use the same language based on a shared and accepted vocabulary. In this way, it is possible to avoid misunderstanding and different interpretations of the same terms and concepts.

Concerning the second requirement, SBVR represents a framework aimed at providing linguistic support to business analysts in order to define the way by which "they run their business in their own language, in terms of the things they deal with in the business" [26]. In other words, SBVR allows to define ad use actual business concepts in order to create artefacts that capture the business knowledge. This characteristic allows to address Req. 2. A possible representation proposed by SBVR specification is in fact a *Structured English* that allows to define business concepts in a way that is familiar for business people.

The third requirement is well addressed by SBVR thanks to the formal logic that underpins it. The base is first-order predicate logic (with some restricted extensions into higher-order logics), with some limited extensions into modal logic - notably some deontic forms, for expressing obligation and prohibition, and alethic forms for expressing necessities [25]. This enables the possibility to use logical language to represent the knowledge captured using SBVR models. Moreover SBVR enables an easy interchange of business models between business modelling tools or metadata repositories in distributed heterogeneous environment. In fact, it uses MOF capabilities defining an XMI schema for the interchange of business vocabularies and business rules among organizations and between software tools.

Concerning the last requirements about process orientation, SBVR does not explicitly address this aspect. Anyway, it is conceived as a foundation of a more general approach, the Architecture of Business Modeling [23], that explicitly ad-

dresses the need for process modelling. Moreover, being an OMG standards opens several possibilities to create connections and define mapping to other languages for process representation, such as the Business Process Modeling Notation (BPMN). In this perspective some efforts have been started by different teams and task forces. Another important characteristic of SBVR is related to the use of rules within information systems aimed at supporting business activities. One of the SBVR design objectives is "to make the business rules accessible to software tools of several kinds, including [...] software tools that support the information technology experts in converting business rules into implementation rules for automated systems" [26]. This implies a basic support to process modelling, that can be addressed exploiting the potentialities of rules as means for modelling and constrain business activities.

3.4 SBVR as building block in OPAALS

The reasons to adopt SBVR within OPAALS are closely related to the main objectives of the project. The main claim that it makes is that, in order to achieve sustainable digital business ecosystems of SMEs and software components, it is fundamental to understand in depth the collaborative processes and ICTs that underpin the continuous creation, formalisation, and sharing of knowledge in the form of business models, software infrastructure for e-Business transactions, and new formal and semi-formal languages. In this perspective, SBVR can be conceived as a means to address some of these aspects. First of all, it represents a powerful tool to represent business knowledge in a language that is close to business people. This allows to share the knowledge not only from a technological perspective, through accessible repositories, rather it can be shared in terms common understanding and convergent meanings. Moreover, the introduction of SBVR is a first step to create a first mapping from human natural languages to structured representations of software requirements specifications, since it is founded on the idea to formalise and give a structured representation to business knowledge, in order to make it available for IT experts. More in detail, SBVR is conceived to provide support in developing software components starting from models based on a business perspective that can be translated with a reduced effort into models for technological solutions. SBVR represents, in fact, one of the core elements within the Business Modelling Architecture, intended in its widest definition as the set of systemic decisions relevant for businesses.

Another interesting point is the possibility to exploit emergent synergies coming from convergent efforts in the enhancement and experimentation of SBVR. Some international communities of repute, such as the OMG and the Business Rules Group, are currently working on this standard to improve it. The project should provide such communities with its interesting results giving them some suggestions and guidelines for their work. On the other side, the project could benefit from the collaboration and support coming from these external parties.

Finally, other existing standards play an important role in choosing SBVR as building block for OPAALS. The OMG is currently working on the creation of bridges with other standards for software development, such as ontologies and approaches for process representation and execution. This represents an additional advantage for using SBVR within the project, that could be strengthened by this continuous effort on SBVR enhancement and integration.

3.5 ISUFI Business Modeler

The Business Modeller is an integrated tool which allows business people to create business or domain models in the SBVR language.

It is a text-based tool which allows the modeller to create a description of a business model by typing structured sentences and business rules through a user-friendly graphical interface. The editor guides the modeller in the process of creating a business model in a computation-independent fashion, avoiding technical modelling formalisms typically based on the object oriented modelling paradigm as used by IT system designers and technical people.

Moreover, in order to take advantage of the high expressiveness of the SBVR language (mainly regarding formal logic), the editor needs to be improved with several additional features (add-ons, plug-ins, ..) which will allow a richer and more powerful automatic interpretation of models. Such supplemental tools (e.g. validators, parsers, query tools, verbalizers, wizards, content assistance processors, helpers,) will improve the usability and the effectiveness of the editor allowing a full exploitation of the SBVR language.

The main functionalities for the Business Modeller are:

- *Model creation* (to be developed): the user creates a new model. When creating a new model, the user can specify (if needed through a wizard) a vocabulary description and/or a rule set definition.

<Rule Statement or Clarification Statement>
Name:
Guidance Type:
Description:
Source:
Synonymous Form:
Note:
Example:
Enforcement Level:

Figure 3.2: Rule structure

- *Vocabulary Entry creation* (available): the user creates a new vocabulary entry (i.e. a new Object Type, Individual Concept or Fact Type). Each entry is represented by a concept's designation or form of expression, followed by some specific attribute-value pairs.

- *Business rule creation* (partially implemented): the user creates a new business rule (i.e. an entry in the rule set). Each entry includes the statement itself and optionally includes other information labelled by captions shown in Figure 3.2. A rule statement or clarification statement can be expressed formally or informally. A statement that is formal uses only formally styled text - all necessary vocabulary is available (by definition or adoption) such that no external concepts are required. Such a statement can be represented as a logical formulation.

- *Import Vocabulary* (available): this functionality allows to import external vocabularies in order to reuse their entries for a new vocabulary or rule-set definition.

- *Export Vocabulary* (available): this functionality allows to export, in a specific formalism, vocabularies created by means of the Business Modeller in order to allow interchange and reuse their entries.

- *Import Rule Set* (to be developed): Such functionality allows to import exter-

nal rule-sets when they are formalized in a standard format such as XMI or RuleMI.

- *Export Rule Set* (to be developed): this functionality allows to export, in a standard formalism such as XMI or RuleMI, rule-set created through the Business Modeller.

- *Integration with Worldnet for supporting process definition* (available): this functionality allows to refer to WordNet during design time in order to prompt the user with term definitions, synonyms and eponyms.

The Business Modeller is designed and developed as an Eclipse plug in. Currently a web based version of the Business Modeller is under design. Such new version will be based on Ajax technology through DVR libraries.

The components currently under development and shortly available in the SBEAVER community are:

- *An API implementing the dictionary module* (soon available). Such component provides a model for vocabulary specification and an API that allows external tools or modules to access it.

- *Rule export engine module* (soon available). This component will allow the exporting, to a standard formalism such as XMI or RuleMI, of rule-sets defined by the Business Modeller.

- *Rule parsing engine module* (soon available). This component will provide an API in order to allow external modules to parse SBVR rules and to represent them in a specific logical formalism.

Note that the whole developed code is not commented or documented at all, thus the work of enhancement and integration of the components will be quite difficult and time consuming. The actual implementation of the parsing engine underpinning the Business Modeller has been automatically generated by ANTLR v2.5.5. Such tool requires the designer to create a formal grammar by means of a specific scripting language and to provide such grammar to a parser generator, which automatically generates the parser engine. Since the current grammar supports only vocabulary entry recognition and lacks in flexibility and modularity in order to support also business rules, it is necessary to rewrite the grammar and thus create a new parsing engine. It is important to point out that a new and more powerful version of

ANTLR (v3.0) has been released recently, thus the new grammar will be written in this updated scripting-language.

4 Business Modelling in Digital Ecosystems

4.1 Introduction

BML 1.0 (from now on BML) was the technical framework developed through the Framework 6 DBE project. The main goal of BML was to help businesses to describe their operating environment in order to facilitate the linkage and alignment between the business view of a small enterprise (business models, business plans, business processes, business workflows, policy, governance, rules, usage, deployment and so on) and the IT view of the business (technical infrastructure, software services, etc) in order to offer seamless business to business (B2B) capabilities among different types of Small and Medium Enterprises (SMEs).

Within the technical infrastructure, DBE/BML proposed to investigate the marriage of the syntax of service implementation (technical part) with the service semantics (business part) and to code-generate parts of the service implementation from the service semantics. This can be generally viewed as:

- The service technical implementation view: what are the service method signatures such as method names, classes, requirements, parameter lists and their order; how to invoke/execute them, etc.
- The business service semantics view: what service to call, who can call it, when, how long, within what context, etc.; business processes and workflows and how the semantic view maps into the technical view.

As stated in DBE Deliverable D15.1: "DBE Business Modelling Language describes main characteristics of SMEs and of the business transactions they can support. The BML allows the representation of information as: service offered and

requested, resources, processes, business model and motivation, policies and agreement, location and event related to business and so on."

It is necessary to note that the marriage between the syntax on the technical side and the semantics on the business side within enterprise/business software is a very interesting, challenging and currently active area of research as already discussed in chapter 3.

The generic use-case of BML and BML models was to help businesses adopt a specific BML model so that many companies within the same sector could use similar models and, therefore, mapping the companies to specific business domains and software services could be achieved.

BML/DBE made a distinction between real world services (such as *"I am a restaurant, I offer food, this is my price list"*) versus software services (*"I am a restaurant, this is my booking interface, this is my interface to my price list"*). BML as a real world service could be used as a simple "yellow page" type of service.

To achieve this functionality and level of description BML proposed to have 5 specific areas in different parts of the business:

- Business Organization
- Business Process
- Business Motivation
- Business Event
- Business Location

Each package contained a set of elements and relationships that allowed a business analyst to construct more complex models. Each element could also have attribute/value pairs and also ontology concepts. Figure 4.1 illustrates the definition in a UML type of diagram - the BML Organization package and its elements.

BML packages as such dealt mainly with data structure definition. While some process information was encoded in the BML Process package, the Process package lacked enough logic and technical granularity (i.e. sequences, activities[1] or structure programming constructs[2]) to be able to create, for example, models in such a way

[1]Named after the terminology used in UML diagrams such as sequence, activity, interaction diagrams

[2]If-then-Else, while, sequence of commands and flow of commands (parallel execution)

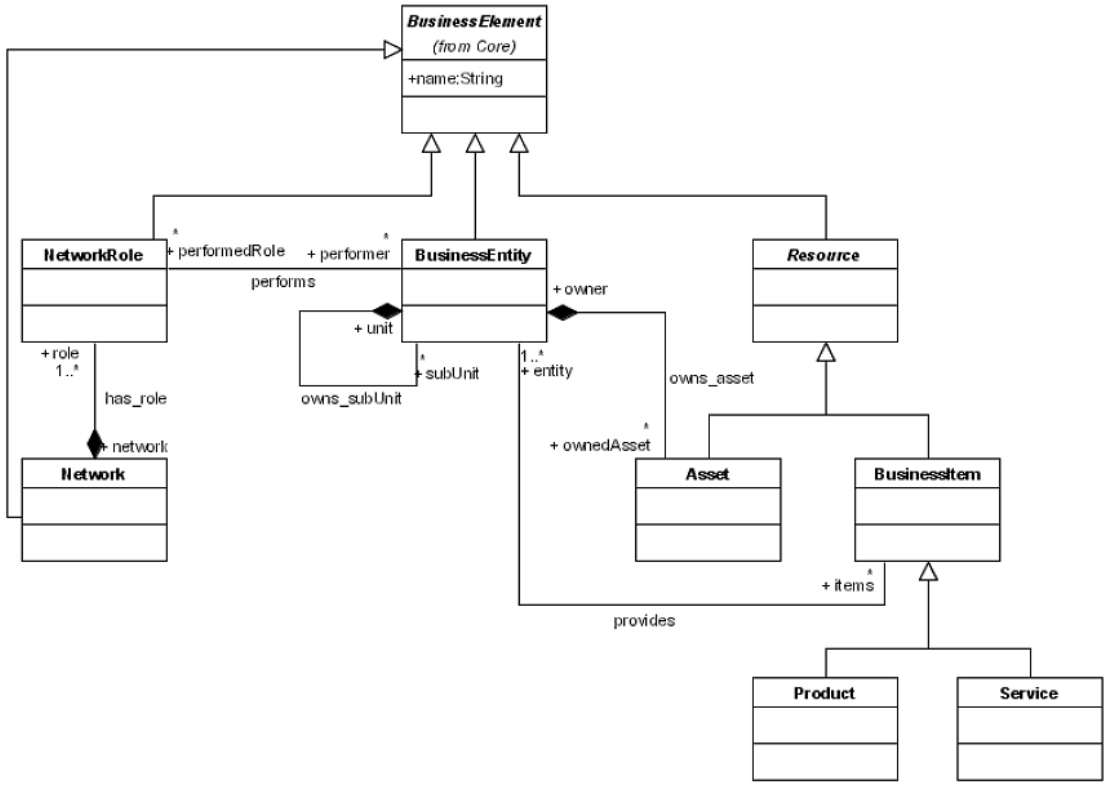

Figure 4.1: BML Organization package

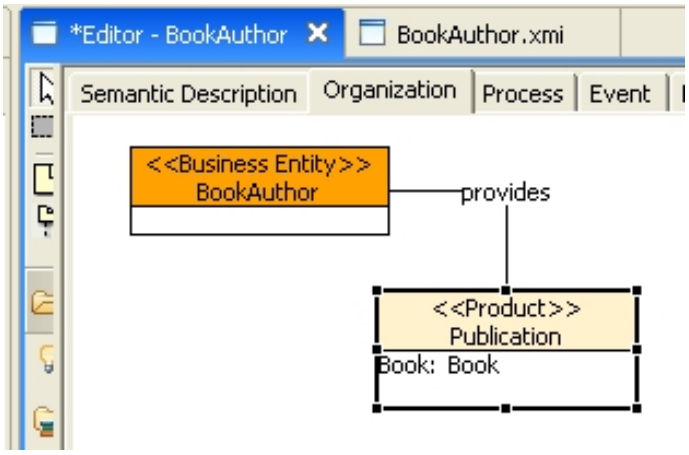

Figure 4.2: BookAuthor BML model

that were useful for complete code generation of business workflows and business process interoperability among different enterprises.

Figure 4.2 illustrates a simple BookAuthor BML model, linking a <<Business Entity>> "**BookAuthor**" as the *provider* of a <<Product>> "**Book**". Book itself is an attribute referring to an ontology concept "Book". Figure 4.2 illustrates the BML data and ontology concept associated with the BookAuthor BML model illustrated previously.

4.1.1 BML and SSL

Due to the lack of any form of formal logic in terms of how the different BML elements could be organised to form logical constructs that could be used for code generation, BML was mainly used to define business data attributes, fulfilling the initial requirement of real world and software service search and discovery.

The Semantic Service Language (SSL) was an extra package added to the BML framework to help the business to define the service technical specifications from a platform independent perspective. At the SSL level, it is possible to define method signatures with name, its inputs and its outputs. Figure 4.3 illustrates a Location Based Service SSL model with the method's signature.

The SSL part of the BML model was used to generate Service Description Language (SDL) and from there used to code service stubs [27].

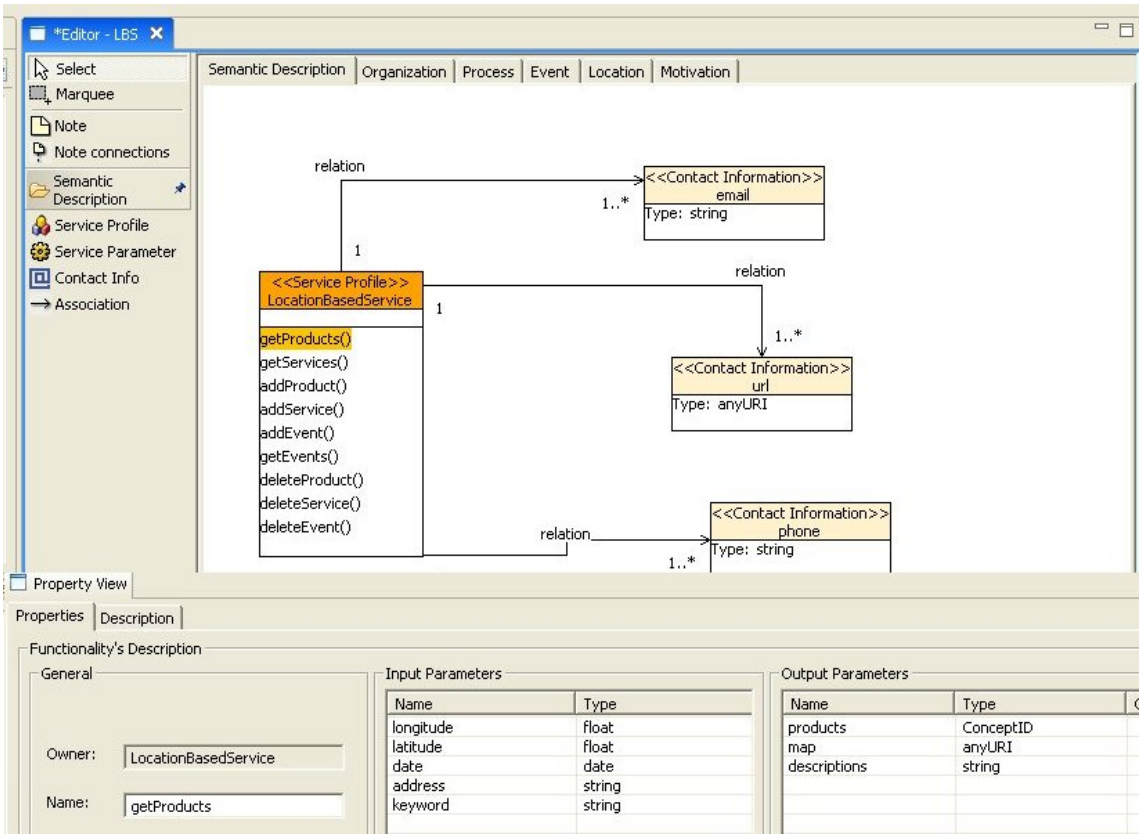

Figure 4.3: Location Based Service SSL model

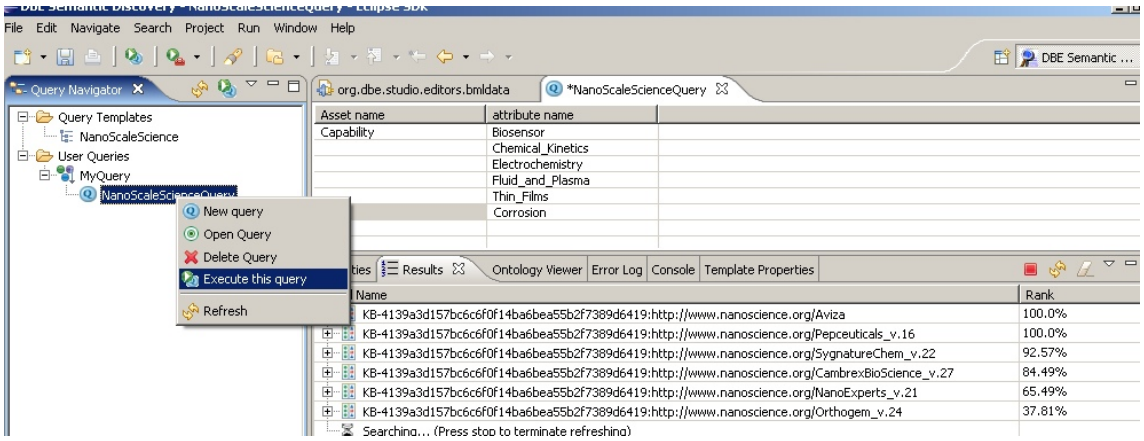

Figure 4.4: BML search framework

4.1.2 BML Search Framework

To enable the search and discovery capabilities within the DBE, BML models together with BML data were stored on the DBE Semantic Registry together with the SSL generated SDL model and other information such as the Service Manifest ID that contained the unique identifier of the service forming what it was called the ServiceManifest (SM).

Once the SM was stored the services could be searched by using a specific DBE devised query language. The image below illustrates a BML based search using a template. As BML was defined as a structure, powerful structured queries could be constructed against the DBE SR.

The BML search framework [28] was complemented with a recommender service [29] that matched BML models based on the principle that similar or identical BML models associated to different services could yield to a chance in service provider, given that the technical interfaces of the services were identical.

Also the DBE implemented a framework designed to provide service matching and recommendation based on BML service descriptions and service usage and context.

4.2 Shortcomings of BML

Although the BML was a powerful framework, it had a series of shortcomings that made it difficult for users to take full advantage of it and more importantly its

adoption in terms of community of practice. The following section outlines some of those issues.

4.2.1 BML Usage in Practice

First of all, BML required SME users to learn a new set of tools and structures in order to take advantage of it. For BML to work as envisioned, many users/models/services had to be present on the DBE so that the network effects of its usage would overcome the initial technology adoption/bootstrap problem. Also specific workflows had to be followed that required changing how developers worked and designed their systems.

This situation is very similar to the situation of the semantic web in general and OWL/RDF based ontologies in particular [30]. Some BML issues were:

- Without generic BML models that can be quickly reused and deployed many users will not use BML.
- Without practical applications that can provide "instant satisfaction" users will not see the benefits of investing time and resources into further adoption and/or development
- Without many users the potential benefits network effects of BML models usage cannot be realised.
- Complexity of the model versus expressiveness of the model. However, it is possible to build complex systems with simple pieces of technology.
- Current development practices among SMEs and the overall knowledge required to use the frameworks and level of "thinking abstract" and modelling skills required.

Also, on a pragmatic level users were not keen on disclosing the information that made BML models more interesting and powerful (i..e Business Processes/Motivation) and chose instead simple data (i.e. address, contact details). An analysis of the initial BML usage within models was performed here [31].

4.3 Related Technologies

Another issue was the specific technologies employed to implement the framework and its adoption. During the period of time that DBE lasted similar approaches achieved greater level of community adoption and maturity (i.e. EMF[3]) rather than the technologies initially adopted for implementing BML (i.e. MDA[4] MOF[5] , JMI[6]).

The DBE OMG/MDA conceptual adoption meant to re-implement many of the concepts that already existed in other different application domains. For example while DBE could re-use some ontology concepts from OWL[7]/RDF[8], it meant that it was required to translate from RDF to a metamodel (XMI) base. As the data models were different, it also meant that the development of new knowledge representation and search frameworks was required instead of using an already existing one (i.e. SPARQL storages).

Within the DBE many such efforts took place to adopt the MOF/MDA approach and development time had to be spent on model and metamodel transformations rather than the applications themselves[9].

Also SME users were usually more interested in learning current standards that have a proven track record on the market place rather than new unproven frameworks. As the BML and indeed the DBE introduced a different approach to web services, it was difficult for users to make informed decisions.

4.3.1 Lean and Agile versus Architecture Heavyweight

Over time within the IT industry there is a constant need to innovate and adopt fast market needs and new market opportunities. Current trends such as agile development producing fast iterations with customers, test oriented development and short iteration cycles promote potentially quicker reactions to changes than the more top-down full-on architectural designs.

[3]Eclipse Modelling Framework. http://www.eclipse.org/modeling/emf/

[4]Model Driven Architecture. http://www.omg.org/mda/

[5]Meta Object Facility. http://www.omg.org/mof/

[6]Java Metadata Interface. http://java.sun.com/products/jmi/

[7]OWL Web Ontology Language. http://www.w3.org/TR/owl-features/

[8]Resource Description Framework. http://www.w3.org/RDF/

[9]In OMG MDA/MOF, it seems that every problem is solved by one more metamodel/model transformation.

If two enterprises want to interoperate, at a pragmatic level one of the first issues to deal with is data interoperability rather than business logics. Each enterprise might not want to expose their business processes and only expose the endpoints that allow access to the data and perhaps an API to access such data. From a Digital Ecosystem perspective (DE) a lightweight data exchange is a crucial feature. BML took care mostly of SME at the search & discovery rather than on the data exchange aspects of the transaction.

Within the so called "Web 2.0" approach [32] the usage of lightweight programming approaches often in the form of dynamic languages and powerful data model generators allows, for example, the fast creation web applications; and by focusing on the endpoint data interoperability, it is also possible to build innovative composite applications such as web "Mashups" [32].

Keeping it lightweight and focusing on resource (data) sharing rather than workflow allows developers to innovate faster. Recently a movement for standardisation has started to create standard frameworks for data interchange[10] [11], indicating the trend towards more lightweight models and architectures among enterprise software vendors.

4.3.2 The rise of the RSS: Really Simple Structured Data Web

The previous sections have illustrated some of the issues of BML adoption, which in turn are very similar to the adoption of semantic web technologies. Many experts argue that the semantic web is too complex for any internet-scale practical use.

However, the question that we need to consider is not whether the semantic web will be adopted or not. The question should be reformulated in terms of - if a technology like the semantic web is not adopted, what are the alternatives technologies?

As we are observing, the steps towards the SW will go through a structured data web cobbled together with simple formats first, rather than a straight jump to semantics such as OWL DL or OWL Full. BML shared in common many of these aspects of simple data modelling (i.e. BML models without ontologies). However, it required a set of heavyweight MOF toolsets and libraries that made its usage

[10] Open Service Oriented Architecture. http://www.osoa.org/display/Main/Home

[11] Similar concepts to the original UNIX philosophy http://en.wikipedia.org/wiki/Unix_philosophy#Mike_Gancarz:_T

difficult for platforms outside the Java realm[12].

RSS stands for Really Simple Syndication[13]. Although many versions of this data format exist (RSS RDF based 1.0, RSS/Atom 2.0, etc), RSS has become the de-facto standard for data syndication on the web. Initially oriented towards information only weblog sites, it has been extended to be used for simple data publishing more towards a platform for data sharing and data publishing frameworks such as the. Atom RSS 2.0 based Google Base[14] including its own simple query language specification [15], Yahoo Pipes[16], and Really Simple Services Bus[17].

To work towards potential DBE adoption, the first step needed is work on the lowest common denominators, and that is interoperable data and endpoints. Image 4.6 illustrates a sample DBE mashup type of application where BML was used to power structured search and then BML was exported to static webpages to provide search engine friendly permalinks, structured vCards and RSS feeds.

The application was a composite application using the Google maps API Google Base for profile publishing. The application showcased the search & discovery of web designers (real world service).

Diagram 4.6 represents an approximation of the hypothetical number of users (Y axis) plotted against the simplicity of the framework use for data modelling. Empirically it can be illustrated that complex frameworks (i.e. RSS RDF 1.0) had less adoption than simpler ones (RSS item/attribute/value 2.0) and the outlook is more. In practice wider adoption and bigger communities come with a reduction of complexity.

4.3.3 A note on Dynamic Languages

During the last few years dynamic languages such as Ruby and Python have gained more and more adoption among developers, even to the point that static languages vendors such as Microsoft (.NET framework) and Sun Microsystems (Java) have

[12] A simple test is: Can a busy PHP developer do something interesting with it in 10 minutes? For example OSOA implements PHP bindings from the start. PHP is often seen as the easiest way to start developing web applications. For a very interesting podcast regarding the points of simplicity on the web and supporting simple approaches always work better: [http://www.itconversations.com/shows/detail571.html]

[13] http://en.wikipedia.org/wiki/RSS_(file_format)

[14] http://base.google.com/base

[15] http://code.google.com/apis/base/query-lang-spec.html

[16] http://pipes.yahoo.com

[17] http://www.rssbus.com/

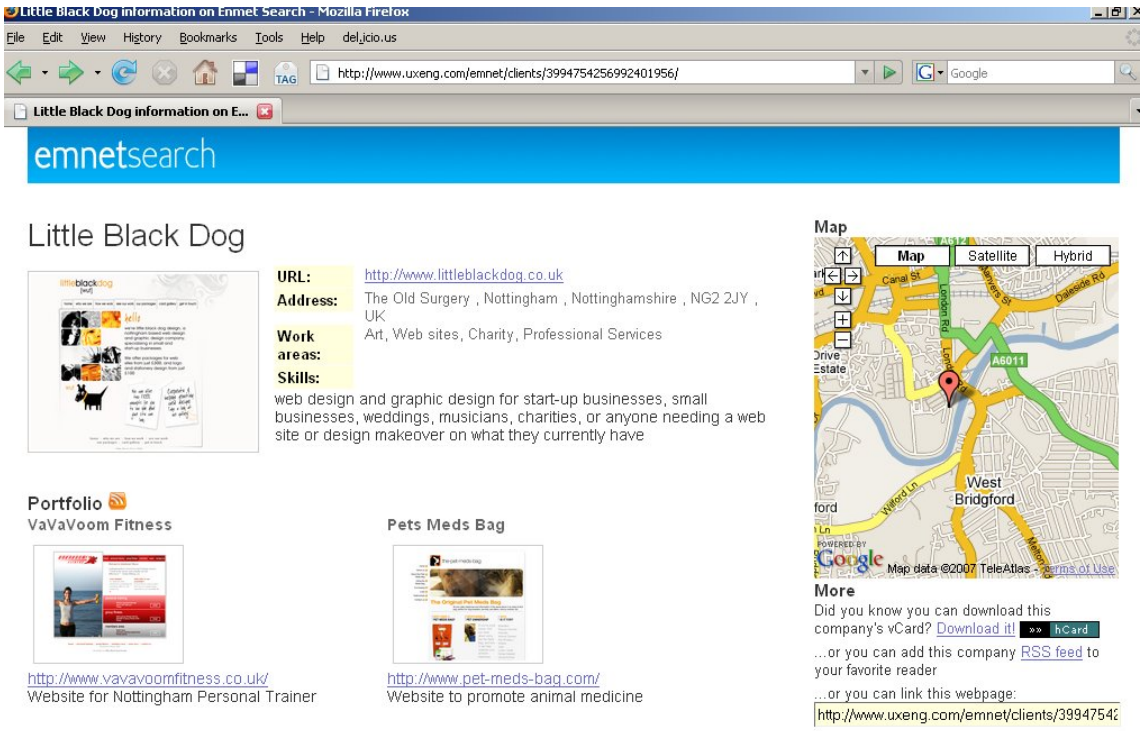

Figure 4.5: Showcase: Search and discovery web designers

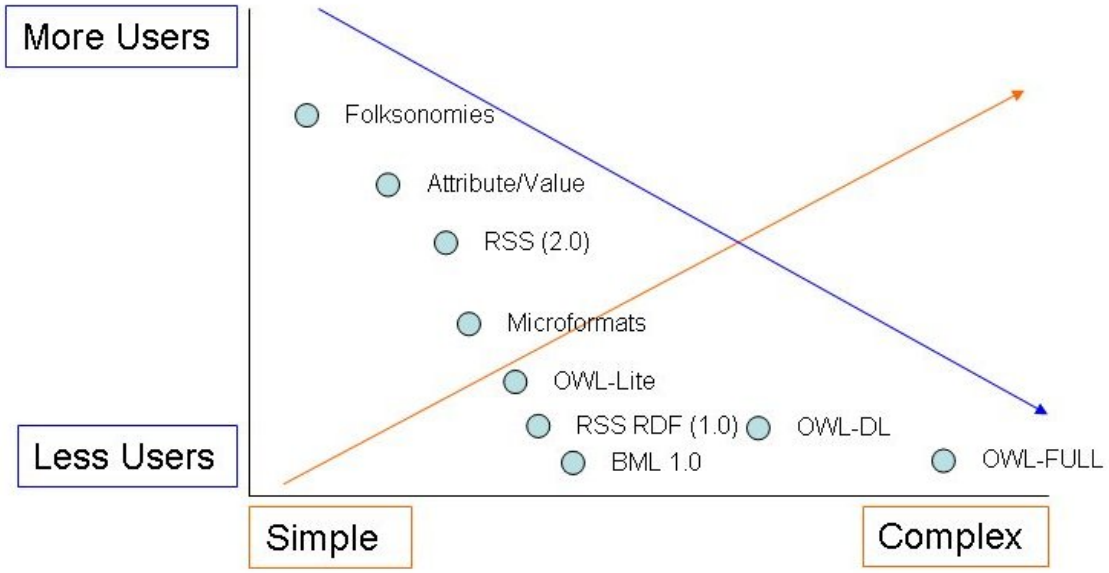

Figure 4.6: Reduction of complexity leads to wider adoption

started to integrate dynamic language capabilities within their frameworks (IronPython, IronRuby, Jython, JRuby). Arguably these programming languages are getting closer and closer to provide the capabilities of functional languages [33] [18].

DBE and BML were implemented in Java, a statically typed language. However the capabilities of dynamic languages such as the ability of changing the behavior of code runtime and the potential functional programming features are very interesting for the Digital Ecosystems[19] approach.

For example[20], if two different SME systems need to send an object of type "Client" with a static language such as Java it is necessary to know before hand all the object interfaces (proxy in DBE and distributed system terminology) to access the properties of the object.

Although Java reflection can be used to inspect the objects, if this is not the case, a runtime exception might arise. In a dynamic type language this situation can be avoided as a runtime can determine their own methods and attributes (known as

[18] Several chapters on these book discuss functional programming versus other approaches to other types of programming, including also a discussion about the benefits of dynamic languages

[19] These type of features are very likely not to be present on code generated from models as modelling in principle requires strong static typing

[20] This is a very simplistic approach of the problem

Duck Typing or dynamic typing).

4.3.4 A note on Licensing, Open Standards Open Systems

A very important aspect for community building is in what terms the technology is licensed, especially when it comes to open source backed by big organisations and standards bodies. For open source communities to grow, it is not only necessary that the source code is licensed under a properly approved Open Source license, but also that the terms in which intellectual property rights associated to the standards or specifications can be use and re-used by the community and also the existence of open implementations of the standard in order for grassroots open sources to emerge. Also, reference open implementations are critical for standard adoption and ensure interoperability[21]. An open standard with closed API implementations are not good for systems interoperability.[22]

[21] It has been suggested that implementations and specifications must go hand in hand rather than just architecting and providing an specification for implementation in the void is often not a good idea. http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium14_OMGStandards.pdf

[22] http://www.computing.co.uk/vnunet/news/2189545/red-hat-sets-limits-microsoft

5 Design of the Demonstrator

This chapter introduces the design of a software generation demonstrator from two sides. The first side is a high level architecture with the focus on suitable standards but a technology agnostic viewpoint. The second part discusses the technical details and how existing software projects are bundled together to demonstrate the feasibility of software generation out of structured natural language. As the high-level architecture is technology agnostic it is possible to reuse it leveraging other technology platforms which corresponds to the MDA way separating the design from the architecture.

The overall goal of the demonstrator is to show the feasibility to generate code out of natural language with basic functionality. As natural language processing is a big research area on its own, SBVR as a structural approach to natural language processing will be used. SBVR as a MOF compliant model for creating business vocabularies and business rules has the advantage that it is supported by the OMG and it is expected that it will be adopted as a standard by this organization soon.

The aim is to show that it is possible to generate software out of SBVR vocabulary and rules using standardized model transformation all the way along with primitive statements. The following section provides an introduction to the use case which will be implemented as the software generation demonstrator. After that the high level architecture of this approach, and how this approach directly corresponds to the MDA way propagated by the OMG, will be explained. Finally this chapter concludes with the technical details of the demonstrator.

5.1 Showcase

This section introduces a very basic application that finally should be created using the demonstrator. The application is based on the *EU-Rent* case study of the SBVR specification. As the case study - although limited in it's scope too - is quite

detailed, only a subpart as a basis for the demonstrator is explained in this section. The detailed example can be found in [34].

Figure 5.1 outlines the structure of the EU-Rent subpart of the car model. The first section lists the concepts used in the example. In the *Facts* subsection the definition of how the concepts relate to each other are provided. Finally a limited number of structural rules are given which have to be translated to constraints enforcing consistent data entry.

5.1.1 Concepts

```
car model.
car group.
engine capacity.
passenger capacity.
upgrade car group.
car manufacturer.
fuel type.
car name.
```

5.1.2 Facts

```
car manufacturer has car manufacturer name.
car group has passenger capacity.
car group has upgrade car group.
car model is included in car group.
car model has engine capacity.
car model is supplied by car manufacturer.
car model has fuel type.
car model has car name.
```

5.1.3 Structural rules

```
Each car model is included in exactly one car group.
Each car model is supplied by exactly one car manufacturer.
Each car model has at least one fuel type.
Each car group can have one or more car model.
```

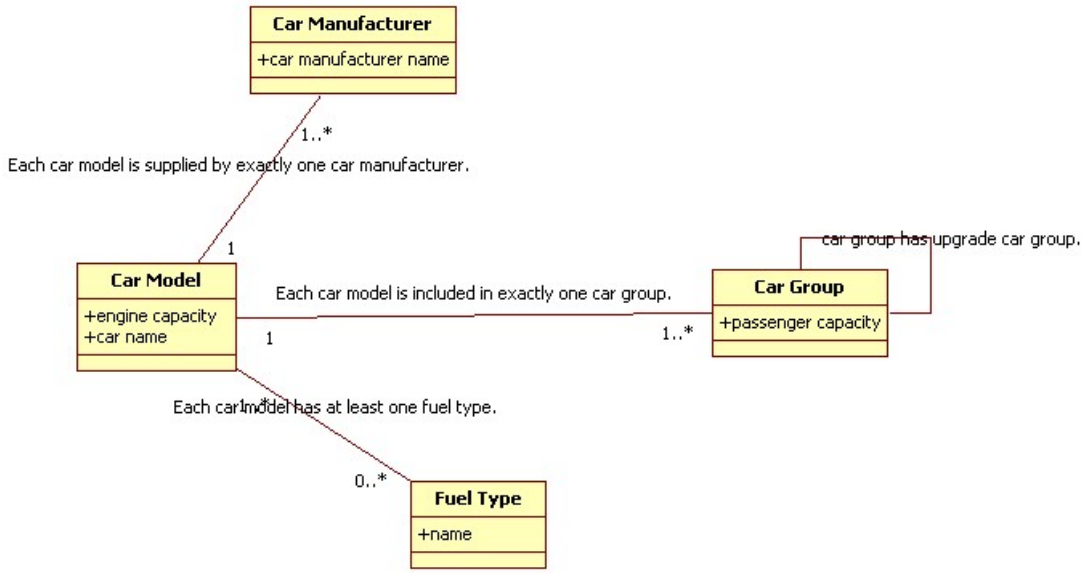

Figure 5.1: EU-Rent Class Diagram

To demonstrate the flexibility of the combination of business modelling and model transformation updates of the model should result in an updated application. Therefore an appropriate rapid development framework has to be chosen that is able to provide the necessary functionality. This selection is discussed later in this document in section 5.2.4. The feasibility of runtime application changes will be another outcome of the demonstrator.

5.2 High-Level design

The design of the prototype is based on a iterative model transformation approach. Basically one ore many input models are transformed to one or many output models. This transformation can then be repeated using output models from the first transformation additionally or instead of the input models of the first transformation. All models, the input and the output models, are stored together in a model repository. In each transformation cycle additional information can be fed to the transformation process in order to enrich the model with necessary information. This information can either gathered automatically by using knowledge bases or manually provided by business analysts and/or engineers responsible for the software.

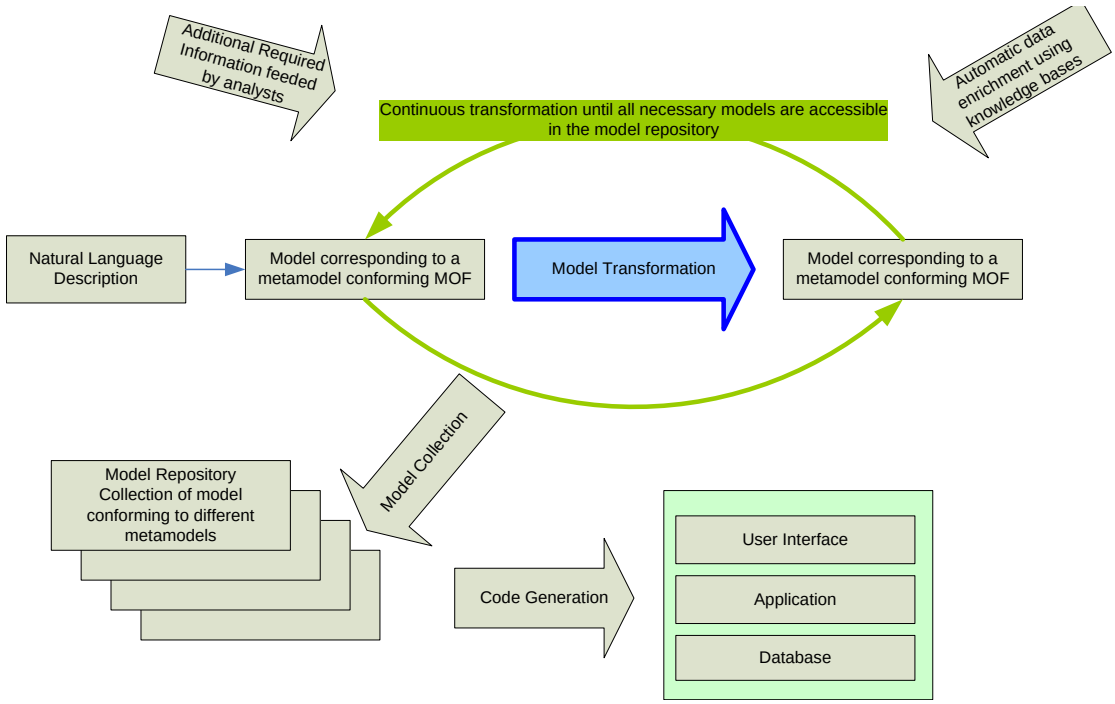

Figure 5.2: High-Level Architecture.

This is repeated until all necessary models for our purpose exist in the model repository. All these models can then be used as a basis for a platform specific model which themselves finally can be used to generate platform specific code - e.g. DDL statements to create a physical database model or Java classes to generate source code. Figure 5.2 shows the overall structure of this software generation approach.

This section explains the individual abstract steps that have to be taken from a natural language software description to a automatically generated code on an abstract meta level. The specific implementation of this theoretical approach will be discussed in section 5.3. The starting point is the natural language description which is used as a basis of the model transformation which is introduced in the following section.

5.2.1 Natural Language Model

According to [35] the knowledge of language needed to engage in complex language behavior can be separated into six distinct categories:

- **Phonetics** and Phonology - The study of linguistics and sounds.

- **Morphology** - The study of the meaningful components of words.
- **Syntax** - The study of structural relationships between words.
- **Semantics** - The study of meaning.
- **Pragmatics** - The study of how language is used to accomplish goals.
- **Discourse** - The study of linguistic units larger than a single utterance.

As the input format is already text, there is no need to know about phonetics. Additionally it is the aim to avoid knowing about semantics, pragmatics and discourse because it would make the generation of code out of natural language unnecessarily complex for this first approach - these characteristics can be issued at a later point. The final two characteristics are morphology and syntax.

Morphology is the information about the shape and behavior of words in context. This includes knowing that *doors* is the plural of *door* [35]. As the demonstrator will focus on a single domain the context is assumed to be clear. To find information about words having the same meaning lexical databases like WordNet[1] can be used to reprocess the input model which will be omitted for the demonstrator.

The final characteristic is the syntax of the language. As for a demonstrator it is too complex to care about all aspects of a real natural language the focus is switched to a standardized structured natural language already explained earlier in this paper: OMG's *Semantics for Business Vocabulary and Business Rules (SBVR).*

5.2.2 Eclipse Modelling Framework

The Eclipse Modelling Framework[2] (EMF) is a modelling framework and code generation facility openly available and therefore used as the basis for many MDA tools. As EMF is a shared component between a large number of tools it provides the foundation for tool-interoperability. [36]

The EMF provides the following components for software engineering based on model driven approaches: Firtsly **Ecore** as the core component of EMF which is the metamodel for describing models. As Ecore is modeled using itself it also is its own metametamodel. EMF is a subset of MOF and in the current MOF 2.0 proposal a similar subset is encapsulated in the Essential MOF (EMOF) specification. EMF

[1] http://wordnet.princeton.edu/
[2] http://www.eclipse.org/modeling/emf

can transparently read and write serializations of EMOF. Secondly, the **EMF.Edit** framework provides classes and functionality for building editors and visual tools for creating EMF compliant models. And finally the **EMF.Codegen** package offers different capabilities for code generation and the creation of graphical user interfaces for edition EMF models.

As EMF was born out of the Eclipse project is primarily focuses on the support of software development. Therefore, EMF delivers a set of low level tools for increasing software engineering productivity and comfort. Among them are persistence support including XMI serialization, model change notification and a reflective API for manipulating EMF objects. Additionally the Eclipse Graphical Modelling Framework (GMF) supports developers creating graphical editors based on EMF and the Eclipse Graphical Editing Framework (GEF). Most modelling tools based on Eclipse leverage these frameworks as a reliable software basis.

The use of EMF based tools for the development of the software generation demonstrator provides the advantage that these tools will, to a certain extent, be able to linked together through the common shared denominator EMF. Additionally tools based on EMF and hosted on the Eclipse project homepage are often also available under the Eclipse Public License which in terms of the OPAALS project frees us from proprietary licensed software and closed source offered by commercial companies. The last part is particularly important because the research project's aim is to support SMEs which often cannot afford high priced modelling tools.

5.2.3 Model Transformation

The Eclipse M2M[3] as a subproject of the Eclipse Modelling Project[4] provides an open framework for model-to-model transformation languages with the exemplary implementations of:

- Queries/Views/Transformations (QVT)
- Atlas Transformation Language (ATL)

ATL is a pragmatic approach to QVT created by the Open Source Community. Both, QVT and ATL will be discussed in the following subsections.

[3]http://www.eclipse.org/m2m/
[4]http://www.eclipse.org/modeling/

Queries/Views/Transformations (QVT)

The MOF Queries/Views/Transformations (QVT) specification [37] addresses the need for a standardized way to transform models between each other. QVT is OMG's answer which tries to solve this issue using three domain specific languages named *Relations*, *Core* and *Operational Mappings*. The QVT specification uses the term *candidate model* which is defined in the following way:

> A candidate model is any model that conforms to a model type, which is a specification of what kind of model elements any conforming model can have, similar to a variable type specifying what kind of values a conforming variable can have in a program. [37]

A transformation is successful if all declared equality expressions become true.

Atlas Transformation Language (ATL)

The Atlas Transformation Language (ATL) is a model transformation language (see section 2.5.3 in this paper). It is specified as a metamodel and a textual concrete syntax. ATL supports declarative and imperative programming but the declarative approach is preferred. Models are transformed by a composition of rules used to match source model elements. The matches then are used to create elements in the target model. Additionally ATL defines querying mechanisms to extract information out of models and finally mechanisms for code generation are provided.

ATL can be used as an Eclipse plug-in written in Java. It contains a compiler that is able to translate the transformation language to intermediate code which is interpreted by an execution machine similar to a virtual machine also written in Java. This intermediate code contains the elementary transformation operations. Additionally development tools based on eclipse are freely available. Both, the engine and the tools, are available under the Eclipse Public License (EPL) and can be downloaded on the Eclipse homepage [1][38].

As ATL focuses on model-to-model (M2M) transformation each transformation is seen as a module. As these transformations are collected there, a broad repository of transformations is already openly available on the ATL homepage[5]. Being heavily dependent on the Eclipse Modelling Framework Project (EMF) ATL uses *Ecore* as the metametamodel which is the basis of the EMF and is a subset of OMG's MOF.

[5] http://www.eclipse.org/m2m/atl/atlTransformations/

The ATL language can be used to defined three kinds of units: ATL transformation modules, ATL queries and ATL libraries. These units itself are combinations of ATL helpers, attributes, matched and called rules. For data types and declarative expressions the OMG Object Constraint Language (OCL) is used.

openArchitectureWare

openArchitectureWare[6] (oAW) is another MDA/MDD framework based on EMF. Although it is EMF based it supports arbitrary models including UML. oAW is based on a workflow engine that is used as the underlying mechanism for model transformation chains. It's capabilities include model validation and a template engine for building code generators. Additionally multiple models and even metamodels can be used for a single model transformation run. oAW is part of the Eclipse Generative Modelling Technologies[7] (GMT) Project and the sourcecode is available under the Eclipse Public License.

5.2.4 Application Frameworks

During the creation of the software generation demonstrator time can be spared using already existing powerful application frameworks that take care about persistence and the user interface. Currently there exist many freely available platforms that can be leveraged for software generation. The most important requirement is that the framework is capable of creating applications including graphical user interfaces (GUI) and the application and persistence layer to a certain extent. Additionally the framework should be Open Source Software to enable us to publicly spread the demonstrator without restrictions or necessary license fees.

The following three examples are discussed in this section and checked for their capabilities they can contribute to the demonstrator.

OpenLaszlo

OpenLaszlo[8] is an platform for creating rich web applications which means that the web applications created with OpenLaszlo are more capable than usual web applications. This is achieved by either using Macromedia Flash to render the user-interface

[6] http://www.openarchitectureware.org/
[7] http://www.eclipse.org/gmt/
[8] http://www.openlaszlo.org

or leveraging DHTML and AJAX. Applications are created with XML configuration files and ECMAScript. Although OpenLaszlo is promoted as a framework for rapid development it does not provide a capable persistence layer out of the box which is a basic building block if the frameworks that will be introduced in the following sections. The source code is openly available under the Common Public License. The main purpose is to speed up and enrich the GUI development for web applications which is not enough for the demonstrator development. As this does not meet the main criteria for a application framework OpanLaszlo will not be chosen for the demonstrator.

OpenXava

The OpenXava[9] framework is a stack of software used for the rapid development of database-based web applications. It is J2EE compliant an can be deployed on any standard Java application server. OpenXava is licensed under the GNU Lesser General Public License (LGPL) and therefore easily usable as the bases of own software development. This framework basically can be used to generate Create-Read-Update-Delete (CRUD) applications fast and with very little programming as the definition of the business entities is done in an XML definition file which supports Crossreferences between tables/entities as well. This makes it easy to generate a model transformation creating OpenXava readable output. OpenXava generates the views for the user interface automatically but these views can be configured according to the needs. The style of the user interface cannot be altered.

Groovy on Rails

Another framework for creating CRUD applications is Groovy on Rails[10] (Grails) which is a Ruby-on-Rails port to Groovy/Java. Groovy is a mature scripting environment in Java cover in the JSR 241[11]. The key difference between Grails and OpenXava is that Grails uses no XML configuration to define the business entities but classes defined in the Groovy scripting language. This means that the business entities or domain classes are defined in a programming language which can be the target of the code generation and therefore Grails will be used to develop the software generation demonstrator. The views for the model elements can be created

[9] http://www.gestion400.com/web/guest/openxava
[10] http://grails.codehaus.org/
[11] http://jcp.org/en/jsr/detail?id=241

individually using a the common template approach.

Grails is the only evaluated application framework that also supports the automatic generation of the database structures. As this is a powerful feature that diminishes the need for a DDL generation this framework can be considered as helpful in the demonstrator as the underlying framework.

Additionally Grails supports, although in an experimental state, online update of business entities which means that changes to a class are propagated to the user interface and the persistence layer. This functionality can be the foundation of more advanced demonstrator capabilities as real time model to application transformation. Finally Grails also offers a validation framework that can be leveraged for enforcing simple business rules or constraints in a prototypical way.

Feature	**OpenLaszlo**	**OpenXava**	**Grails**
Configuration	XML	XML	Code
Business Objects	-	XML	Code
Persistence	-	Configurable	Automatic and Configurable
User Interface	Manual	Semiautomatic	Semiautomatic

Table 5.1: Comparison of Application Frameworks

5.3 Technical Details of the Demonstrator

MDA suggests Model-to-Model transformation for software generation. The most important advantage is the broad applicability of this approach. The model transformation procedure is generic and can be applied for different source metamodels and target platforms. Another important advantage is that there is a large number of open source tools from the Eclipse Modelling Project and open standards from the Object Management Group that can be leveraged. Additionally there already exist a large number of already implemented transformations that can either be reused or taken as examples.

A risk of M2M is that the standards around it are relatively new and therefore the tools are immature. This may lead to solutions that soon have to be re-engineered because of changed interfaces or even approaches.

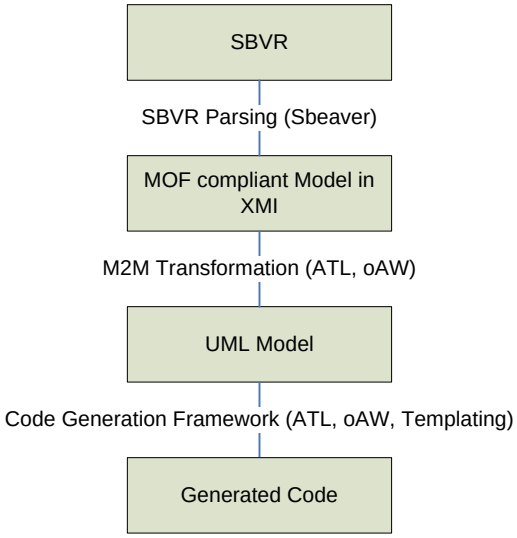

Figure 5.3: Components used in the demonstrator (redraw needed)

5.3.1 Aim of the Demonstrator

For implementing the demonstrator a bottom-up-approach will be used. This means that the feasibility will be proven by the implementation of a basic use case in order to study the advantages and limitations of model transformation in this context. Using an organic bottom-up-approach enables us to find weak spots in the design and eventually replace these spots with more applicable solutions. The next section introduces the use case and provides a overview how the individual steps will be processed.

5.3.2 Model Transformation Chain

Using MDA usually means leveraging UML to specify for example a class model and generate code out of this UML model. But because this demonstrator a structured natural language metamodel is used to specify parts of the software one transformation will not be sufficient. At least two or more transformations have to be used as can be seen in figure 5.2. As these model transformations are applied in a serial way, all transformations together will be referred as the *Model Transformation Chain* in this paper. It has to be mentioned that serial is a simplification as a Model Transformation Chain can also contain parallel transformations, splits and joins (as M2M can have one or more input and output models).

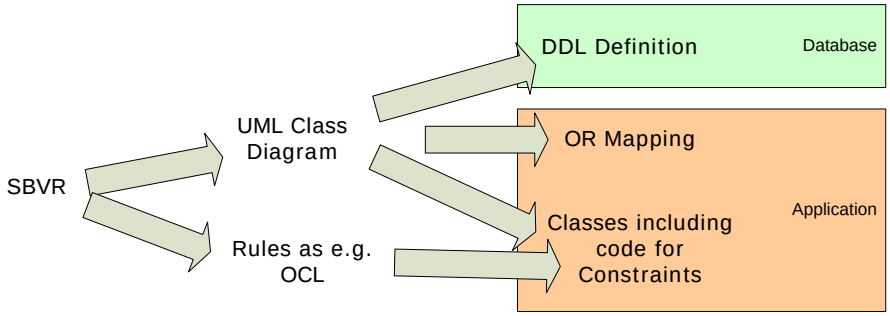

Figure 5.4: Demonstrator Model Transformation Chain. (redraw needed)

A *Model Transformation Chain* is the combination of all model transformations in whatever order and concurrence needed applied to all source information available to get a desired output in whatever form necessary to achieve a predetermined result.

Using this definition the Model Transformation Chain used in the software generation demonstrator can be analog to figure 5.4. In this example the source model is compliant to the SBVR metamodel which is transformed to a UML class model and to a model representing the business rules in parallel. The UML class model is used for generating the DDL statements to create a suitable database and the Object-Relational-Mapping used for the persistence layer (these two transformations can be transformed via a entity relationship model). Additionally, the UML class model and the rule model are used as the source models for a transformation generating platform specific classes in this case enriched with the code to enforce the constraints expressed as rules.

5.4 Business Modeller

Section 3.5 outlines the latest information about the *Business Modeller* developed in the FP 6 research project DBE. As the information arrived late in the writing phase of this document and a meeting between the OPAALS researchers and the ISUFI developers is scheduled after the finalisation of this document the following section provides only a draft usage of the Business Modeler within the demonstrator design.

The Business Modeler includes a SBVR to XMI component which can be leveraged in order to create the input model used for the transformation to platform specific code. As mentioned above it has to be clarified if the capabilities of the

modeler match the requirements. Additionally it has to be studied if the better solution is to leverage the related components out of the Business Modeler or as an alternate solution to extend the business modeler with the desired features to generate the output code desired for the rapid application development framework.

6 Conclusion and outlook

We started our work in WP 2 by clarifying the state of art in modelling and theories of model transformations and by defining the vision of automatic code generation. Three years of research in the DBE project ended up with a BML 1.0 which seems to work fine for many practical applications and with an more natural language based approach using SBVR. As SBVR seems to be the next step toward a natural language approach for business modelling WP 2 and especially the partner University of Kassel can evaluate SBVR from a linguistic point of view. In parallel, the potential for code generation out of SBVR descriptions will be assessed by compare SBVR to BML 1.0 and implement a prototype for code generation and manipulation out of SBVR business models. The first steps toward this vision is shown in this report.

Bringing together natural language approaches and generative software development is a challenging subject of research. Nevertheless we provided with this report the underpinning knowledge about the principles of business modelling which is necessary to understand the processes within the upcoming prototype. Furthermore, a set of tools was introduced which ease the further work on SBVR immensely and lessons learned from BML show the practical needs of business modelling. SBVR or any given business modelling language should be able to cope with the needs of small and medium sized enterprises and the issues of BML in the best way possible.

The design and use case for the prototypical implementation will lead to a prototype due in Month 18 and will be documented in D2.2 *Automatic code and workflow generation from natural language models*. Based on this prototype and the viewpoints of linguists, programmers, SMEs and computer scientists we can evaluate then the practical relevance and focus for further research on natural language based business modelling in general and SBVR in particular.

Based on this prototype and the viewpoints of linguists, programmers, SMEs and computer scientists we can evaluate then the practical relevance and focus for further research on natural language based business modelling in general and SBVR in

particular. The following section will introduce the scenarios for workflow generation and further research steps for OPAALS phase 2.

Bibliography

[1] ATALS group, LINA, and Nantes INRIA. *ATL User Manual*. February 2006. Version 0.7, `http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf`. Last accessed on 04/05/2007.

[2] David S. Frankel. *Model Driven Architecture*. Wiley Publishing, Indianapolis, 2003.

[3] OMG. *Meta Object Facility (MOF) 2.0 Core Specification*, January 2006. Version 2.0, `http://www.omg.org/docs/formal/06-01-01.pdf`. Last accessed on 04/03/2007.

[4] Nektarios Gioldasis, Fotis G. Kazasis, Yiannis Maragoudakis, Stavros Christodoulakis, Angelo Corallo, and Maurizio De Tomassi. DBE Knowledge Representation Models. Deliverable D14.1 Digital Business Ecosystems Project, 2005.

[5] OMG. *Object Constraint Language*, May 2006. Version 2.0, `http://www.omg.org/docs/formal/06-05-01.pdf`. Last accessed on 04/10/2007.

[6] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming - Methods, Tools, and Applications*. Addison-Wesley, 2000.

[7] Tony Morgan. Expressing business semantics, 2005.

[8] Francois B. Vernadat. *Enterprise Modelling and Integration: Principles and Applications*. Chapman & Hall, London, UK, 1996.

[9] Petit M. and Doumeingts G. Report on the state of the art in enterprise modelling. Technical report, UEML Project, 2002.

[10] John A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292, 1987.

[11] P. Bernus and L. Nemes. The contribution of geram to consensus in the area of enterprise integration. In K. Kosanke and editors J. Nell, editors, *a*. Springer, 1997.

[12] G. Spur, K. Mertins, and R. Jochem. *Integrated Enterprise Modelling*. Beuth Verlag GmbH, Berlin, Germany, 1996.

[13] Kai Mertins and Roland Jochem. *Quality-Oriented Design of Business Processes*. Kluwer Academic Publishers, 1999.

[14] NIST. Idef0 method report. Technical report, National Institute of Standards and Technology, 1981.

[15] Scheer A. W. *Architecture of Integrated Information Systems - Bases for Company Modeling*. Berlin, 2nd edition edition, 1992.

[16] NIST. Psl project, 2002.

[17] WfMC. Workflow Process Definition Language, 2002.

[18] BPMI. Business process modelling language, 2002.

[19] BRG. Defining business rules what are they really?, final report, revision 1.3, July 2000.

[20] UN/CEFACT. UN/CEFACT Modeling Methodology (UMM) User Guide, 2003.

[21] David Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[22] OMG. Unified modelling language specification. Technical report, Object Management Group, 2003.

[23] S. Hendryx. Architecture of business modeling, 2003.

[24] Buchanan R. D. and Soley R. M. Aligning enterprise architecture and it investment with corporate goals, 2002.

[25] OMG. Semantics of business vocabulary and business rules specification, 2006.

[26] OMG. Business semantics of business rules - request for proposal, 2004.

[27] Pierfranco Ferronato. D21.3 - DBE Architecture Requirements, March 2005.

[28] George Kotopulos, Christos Alatzidis, and Kostas Krommydas. Query formulator and semantic discovery tool. Final Release. DBE Deliveable WP24, D24.6., 2006.

[29] George Kotopoulus, Yiannis Kotopoulos, and Fotis Kazasis. Dbe knowledge base. Final Release of the Recommender. DBE Deliverable WP14, D14.6, 2006.

[30] Martin Hepp. Possible ontologies: How reality constrains the development of relevant ontologies. *IEEE Internet Computing*, 11:90–96, 2007. Retrieved February 15, 2007 from: http://www.heppnetz.de/files/IEEE-IC-PossibleOntologies-published.pdf.

[31] Victor Bayon. BML Data Editor, BML Wizard. DBE Project Deliverable WP20, D20.8., 2006.

[32] Tim O'Reilly. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. webpublished, 09 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[33] Paul Graham. *Hackers and Painters. Big ideas from the computer age.* OReilly, 2004.

[34] OMG. *Semantics of Business Vocabulary and Business Rules Specification (SBVR)*, March 2006. Interim Convenience Document, `http://www.omg.org/docs/dtc/06-03-02.pdf`. Last accessed on 04/05/2007.

[35] Daniel Jurafsky and James H. Martin. Speech and language processing. Prentice Hall, 2000. International Edition.

[36] Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, 2003.

[37] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, November 2005. Final Adopted Specification, `http://www.omg.org/docs/ptc/05-11-01.pdf`. Last accessed on 04/03/2007.

[38] Frederic Jouault and Ivan Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.*, 2005.

A First Review

by Paul Krause, University of Surrey

Overall it's good as a report that provides the context for what you are doing, and an outline of the basic design for the generator. Where it is weaker is in addressing any of the issues around expressing requirements in natural language. Here the report is a little lacking in depth, I think. That worries me a little bit, but this is a Big Unsolved Requirements Problem - that needs a whole research project in its own right. Nevertheless, talking to Stan and gaining a little more focus on this would be an important next action.

2.3 I dont think this is a good example. I think the relationship between Class models and ER diagrams is too strong. Whitness for example, Leon Starr's use of standard database normalisation rules for critiquing class diagrams, and also NetBeans' direct mapping of Entity Classes (a class model) into Database tables when building a persistence layer automatically.

2.7 I think you should mention that some tools, such as iUML and Rhapsody claim generation of complete applications. But, of course, the modelling has to be done in detail using an action language which always seems to me to be so close to adding in code fragments.

3.2 It is perhaps a little strange to itemise these three separately. I tend to view MOF as a tool for extending UML in the context of MDA?

3.2 In line with the previous comment, there is more structure needed here compared with the linear list as presented. One approach could be to supplement the bullet points with a diagram to represent the relationships between the different languages?

B Second Review

by Stan Hendryx, Hendryx & Associates

In this section the internal review of this deliverable is included without any further modifications except the addition if comments which are typed in green letters. As the critiques of the second internal reviewer are fundamental it was not possible to address all of the points. If changes ware performed it is stated as a comment and a reference to the changed section is included as well. Points that could not be addressed as such have to be discussed for deliverable D2.2.

1 Overall assessment: the paper is very weak. It does not have a clear purpose. Content does not correspond well to the title. It is not coherent. It is not focused. The proposed design does not demonstrate what the authors give as its overall goal, and will probably not work to show code generation from natural language. "The overall goal of the demonstrator is to show the feasibility to generate code out of natural language with basic functionality." (p.36) "The paper does not address this goal at all. "Basic functionality" is not defined. The demonstrator design is lacking in detail. The paper provides a survey of model transformation technology for transforming formal models, and is dismissive of natural language issues (5.2.1). The demonstrator user interface is much too technical for a SME.

Out of the fact that the SBEAVER project was not updated during the last year we supposed that the project is not developed further. At the DBE final review many new insights and an additional tool prototype was shown which forced us to change our original plans regarding SBVR. Out of that reason and because of a difficult interdisciplinary and international environment we had to change and adapt our strategy toward the new basic conditions. As we wanted to stick with the planned date for submitting D2.1, we decided to only draft our ideas and write an SBVR discussion paper which first, summaries the activities of all WP2 related partners and second, give us the opportunity to define and redefine the prototypes and tools

along the development process. The result was of course not fully defined as regards details of implementation and also not all collaboration points were visible down to details, but it was essential to have a first document which is on time for starting the discussion. In the very interdisciplinary WP2 it seemed to be the best way of establish a structured, target oriented and at the same time collaborative way of working.

2 To address the feasibility to generate code out of natural language, the paper must demonstrate an understanding of how meaning expressed in natural language can be formally represented, e.g. by using SBVR, and how these formal representations of meaning map to formal languages, e.g. UML, Java. The choice of technology and issues about programming languages, modeling and code generator tools, MOF repositories, and such are secondary to understanding the semantic mapping problem. [Must have.]

For the deliverable 2.2 more effort has to be taken to understand how meaning expressed in natural language can be formally represented.

3 The "New approach for requirements transformation" shown in Figure 3.1 (p.17) and figure 5.3 (p.39) is simplistic. Too many concerns are conflated into an "iterative model transformation approach" (p.37), details of which are not explained. The emphasis on "lean and agile" as opposed to "full-on architectural designs" (p.30) is risky. The chances are very low that a robust, scalable, working demonstration of natural language to code generation can be produced using this approach. Too much magic is assumed to occur in the "SBVR Parsing (Sbeaver)" step (Figure 5.4 p.46.) The paper should clearly separate business modeling concerns from data concerns from computational concerns from implementation technology and deployment concerns. [Must have]

As discussed in the next paragraph a clear way to separate business modeling concerns from data concerns from computational concerns from implementation technology and deployment concerns should be introduced. This is done by studying the applicability of RM-ODP as guideline to separate these concerns in way that is already successfully used in real world enterprises.

4 Consideration should be given to using the ISO/IEC Reference Model of Open Distributed Processing, ISO 10746, as a reference model for separation of concerns. In doing so, a family of models, each representing a different set of concerns, can

be identified. Correspondences between different concerns provides a firm basis for discussing model transformation requirements. Some of the models are generic to a particular infrastructure (DBE), while others, particularly the Enterprise Viewpoint Model, is unique to each SME. Various indusry library Enterprise Viewpoint Models can be created and provided to SMEs to simplify their model creation efforts and make their models and resulting system compatible with the DBE and interoperable with other SMEs in the DBE. [Optional]

The ISO/IEC Reference Model of Open Distributed Processing (ISO 10746) will be studied as a formalized way for separating the different viewpoints in the analysis phase of the business model.

5 The library example of Figure 5.1 does not explain how the UML model relates to the metamodel of Figure 4.1 (p.25), or any other model template, nor does it give a natural language representation of the UML model. It should. [Must have]

Chapters 4 and 5 were authored by different persons and this point will be considered for the next deliverable.

6 "Semantics of Business Vocabulary and Business Rules (SBVR) is a natural language based logic language thought to be used as concrete syntax in order define business model and to capture requirements" (p.16) SBVR is not a concrete syntax. It is a metamodel for vocabulary, propositions, and projections. Its purpose is to describe the meaning of concepts, propositions, and questions. The descriptive elements are concepts of formal logic. [Must have]

Corrected in section 3.2.

7 A natural language representation of the Library example (figure 5.1):
Vocabulary:

```
publisher
book
author
library
university
series
customer
author writes book
```

```
publisher publishes book
book belongs to library
book belongs to series
library belongs to university
book is borrowed by customer
```

Rules:

```
Each book belongs to exactly one library.
Each book belongs to exactly one series.
Each book is borrowed by exactly one customer.
Each library belongs to exactly one university.
```

As suggested in a latter comment (paragraph 20) the example was changed to the EU-Rent example from the SBVR specification [34]. The intention of the change was to better align with other researchers working on SBVR. Additionally structural rules were included in the basic SBVR vocabulary given. Optional the showcase can be enriched by adding more example vocabulary out of the SBVR specification to the example. The adatption can be seen in section 5.

8 The vocabulary can be described in terms of the SBVR Meaning and Representation Vocabulary. The rules can be described in terms of the SBVR Logical Formulation of Semantics Vocabulary. These descriptions can be represented in the form of MOF/XMI for interchange between tools, and for storage in a MOF repository.

This point relates to the point discussed in paragraph 12 and should be considered when trying to find ways to create and manage business metadata in a distributed environment. At this stage it is not possible to speculate about the best way how to distribute and store this data.

9 Figure 5.1 is naive in its treatment of "Book". There are three concepts of "book," at least two of which could beneficially be represented in the model:

- book: abstraction of the content that is contained in the book, irrespective of its form; what the author wrote
- book form: publication of a book that is identified by an international standard book number (ISBN) (This is probably the closest to "Book" in the figure.)

- book copy: physical instance of a book form, typically printed, bound and packaged in the manner specified by the book form (A library may have multiple copies.)

Implicitly solved with the change to the EU-Rent example already outlined in the comment to paragraph 7 in this section.

10 This points up a common shortcoming of technical models: the concepts the model purports to represent are undefined. Each noun concept must have a definition; some verb concepts require a definition. [Must have]

Implicitly solved with the change to the EU-Rent example already outlined in the comment to paragraph 7 in this section.

11 The natural language modeling problem is, "How does a SME produce a vocabulary and rule set like the one above?" The code generation problem is, "What does the code you want to generate from a natural language model look like? How do you create it, given such a vocabulary and rule set?" These are the two grand questions for OPAALS to address.

The Conclusion and outlook chapter was enriched by the *Scenarios of Code and Workflow Generation from SBVR*, a paper created during a workshop on these topics. This section ow includes different scenarios for what code is possible to create and how this can be achieved.

12 The answer to the first question is highly pragmatic. It involves how people use their language, their skill in writing about their business, their ability to be consistent in the terminology they use, their skill in writing unambiguous sentences, their skill in abstract thinking. This is where sociology, psychology, and linguistics comes into play. This process could be aided by tools that analyze what SMEs write and produce a vocabulary from their writings. The process could be further aided by providing SMEs with pre-defined vocabularies relevant to their industry, the words of which (in various standard morphological forms) they can use in their writings, introducing new words only when existing ones do not suffice to express what the author has in mind.

This point will be explicitly addressed in OPAALS Phase 2 by studying ways how to generate and manage business knowledge in a decentralized environment. This will on the one hand be done by having a look at the sociology, psychology and

linguistic part and on the other hand by studying how helpful tools can be in this process.

13 The answer to the second question is more technical. However, at the first level of description and approximation, it does not have anything to do with the technology talked about in the paper. It has to do with mapping the semantic representation of a business model expressed in terms of a business metamodel (e.g. SBVR), to a specified target metamodel, e.g. UML or OWL. Once you get to UML or OWL or some other target formal language, you can begin to think about the technology that the paper surveys. We should be thinking in terms of mapping between metamodels, the logic and rules involved, in such mappings, not the technology to do it. This should be the subject of the paper. Once we understand what needs to be done, we can select technology that will do the job, buy it or build it, and write about that.

14 Use of parsers generated with tools like ANTLR which assume essentially context free grammars have limited success parsing natural language statements. Natural language grammars are not context free. Ambiguity is a significant problem to be dealt with. Anaphora, especially long distance reference, is another. Coordination is another. There are many ways in which natural language grammars differ from the formal grammars of computer science, and require different approaches to parsing. We need first to understand the range of grammatical constructs needed to write rule sets needed by SMEs, to define an English grammar, before choosing parser technology. [Must have]

In the opinion of the authors it is not necessary to have a full parser in the beginning of the project as this is probably a whole research project in its own. But what is necessary is to be able to computationally process simple SBVR sentences to find out what can be done with SBVR and what cannot be done. ANTLR is not a perfect choice but on the one hand it is a well established starting point and on the other hand there already exists SBVR grammar code created in the *Digital Business Ecosystem*[1] research project which can be reused. This topic still is discussed internally as there a many opinions covering parser and parser technology to use for the creation.

[1] http://www.digital-ecosystem.org/

15 Note that a description of the library, above, is NOT a description of any information system. To specify a system, it is necessary to say what the system is supposed to do. In this paper, the system is presumed to do CRUD. However, the business model does not say "do CRUD;" the programmers assume that is what is wanted. "CRUD" means something in relation to a UML model, because some of the tools described in the paper can read standard UML models and produce CRUD interfaces according to some software architecture. Note that the concerns of software architecture to do CRUD are entirely absent from the business model. Software architecture is a different set of concerns, and should be treated as such, modeled and analyzed as such, separately from the business model. The software architecture of a DBE can be designed once and for all, and be applied mechanically to thousands of SME models to produce the desired information system. A lot of work has gone into designing the software architecture of a DBE. These software architecture models must involve all of the DBE infrastructure architecture, all of which is hidden from the SME, except what the SME sees from the user interface that is produced. User interfaces designs should be careful not to expose the metamodel unnecessarily to the users, because each such exposure requires the user to understand it, and detracts from acceptance.

Demonstrator can be refocused to cope with structural rules at first. These rules can enrich the datamodel and be used as metadata to build an example application that performs CRUD operations. To have a CRUD application is not the main target but a side effect of transforming the vocabulary of a business (e.g. the EU-Rent example) to platform specific code. It was decided to use the Grails platform as the target for the code according to the described reasons but it could be any other platform and according to the MDA philosophy can be done even later (the optimistic approach). But for a demonstrator it might be a step too far to include all infrastructure components at first but feasible to stick with a step-by-step approach.

16 No business rules are given in the library example. There should be some. The rules represented by the UML multiplicity constraints are alethic (structural rules); business rules are deontic (obligations). There are probably business rules about borrowing and lending of books, the number of books that can be on loan to a customer at a time, due dates, fines, library hours of operation, etc. It is relatively easy to produce DDL from the models. Harder to decipher the rules and decide how to enforce them, if the system is to enforce them, how, technically, to create program

logic from the rules. A system requirement to enforce a rule (structural or business) is another rule that states the obligation of the system design and operation to enforce the rule in a prescribed manner. For each business rule, there needs to be other rules about the consequences of failure to follow the given rule.

Implicitly solved with the change to the EU-Rent example already outlined in the comment to paragraph 7 in this section. Structural rules are included out of the examples in the SBVR specification.

17 Note that the system requirements model is distinct from the business model. Business input is required to say what the business wants the system to do. Write the business model first, describing just the business domain to be treated by the system, then write the system requirements using the vocabulary and rules of the business model. The SBVR metamodel is applicable to both of these models.

18 The OPAALS effort of code generation from natural language would benefit from a refocusing from technology to the problems outlined above. [Must have]

Too late to follow this advise for this deliverable. This has to be done targeting the next deliverable D2.2.

19 A rich example natural language model, including vocabulary, structural rules, business rules, should be created and used as a target for research and development into SME modeling tools and code generation tools. Having such an example would help significantly to focus the OPAALS development effort. [Must have.]

20 Consider using the EU-Rent example contained in the SBVR specification. [Optional]

As explained in the comment to paragraph 7 the showcase was changed to the suggested EU-Rent example.

C Third Review

by Frauke Zeller and Josef Wallmannsberger, University of Kassel

The overall goal/intention of deliverable 2.1 is depicted as a general introduction to WP 2 "Automatic Code Generation from Models" and what SUAS wants to achieve regarding code generation and natural language in her work package (page 3). Accordingly, the authors mention Semantics for Business Vocabulary and Business Rules (SBVR) as the approach of choice for code generation and business modelling in WP2, as well as work which had been carried out in the Digital Business Ecosystem (DBE) project and to be utilised for this work package. It is also mentioned that prior results from SBVR work in the DBE focused on business modelling and search for these business models in a distributed network. Therefore it makes sense that the focus of WP2 and deliverable 2.1 aims to describe the potential of SBVR as a basis for automated code generation and workflow manipulation. The deliverable also promises in the introduction:

- to give a design of a prototype for code generation
- to discuss in depth the underpinning technologies and languages
- to introduce the reader to the basic principles behind the model driven approach as well as model transformations
- to provide overview about state of the art business modelling is given and both, DBEs BML 1.0 and SBVR are discussed in the context of OPAALS
- to present and suggest the modelling tools for SBVR for the implementation of the software generation prototype
- to implement a design and use case which will be implemented for D2.2 Automatic code and workflow generation from natural language models(month 18)

- to provide a practical and adaptable point of view for SMEs.

To summarise, the introduction depicts a clear purpose of the deliverable with a number of smaller sub-goals that all appear to lead consistently and logically to the overall goal. A minor critique could be that the various goals appear in its details very broad (i.e. the various focal points could also be split up into two or more deliverables).

In terms of coherence, the table of contents mirrors the different goals depicted in the introduction. The title "Design of Software Generation Demonstrator" definitely implies the particular interpretation and strategy SUAS has decided on for the deliverable. However, the actual software engineering part, which will also be necessary for the later work in work package 2 and other work packages, is rather small.

Chapter 3 lacks an integrative discussion regarding natural language and its role in state-of-the-art business modelling. It would be particularly interesting to integrate natural language issues/paradigms/aspects into the list of operational frameworks and languages at the end of chapter 3.2.

From a computational linguistic point of view, SBVR should not be defined as a linguistic base for creating business models as it rather claims to be a metamodel for vocabulary, propositions, and projections. Its purpose is to describe the meaning of concepts, propositions, and questions. The descriptive elements are concepts of formal logic (see Stan's feedback).

Chapter 3.3 would need more elaboration (which should or is already planned ??? to be conducted in deliverable 2.2). Moreover, a clear-cut distinction between business modelling concerns from data concerns from computational concerns from implementation technology and deployment concerns regarding the requirements explanations already at this point would be useful (see also Stan's feedback) as it would provide a more elaborate and detailed basis for the following chapter "SBVR as building block in OPAALS" which in turn could be improved/amended accordingly.

Chapter 4 should be stronger aligned with the foregoing chapters. Vice versa, the clearly stated focus on Digital Ecosystems in chapter 4 should also be integrated into chapters 2 and 3. So far, this had been done more or less indirectly, which might cause confusions and the impression of a too strict rupture regarding focus and overall scope of the deliverable when starting to read chapter 4. It would be useful, therefore, to use the DE concept as a red line and Leitmotiv, which can also

be easily connected to more detailed descriptions/amendments of natural language aspects/issues.

Chapter 5.2.1 might need a more detailed analysis and definitely a concrete outlook for the necessary steps to follow. As the restriction on syntax and morphology (2 out of six necessary categories for natural language processing) is not sufficient in order to address all requirements depicted in chapter 3.3.