Next Generation of Modelling Platforms

Dimitris Karagiannis and Niksa Visic

University of Vienna, Knowledge Engineering Research Group, Brünnerstr. 72,
A-1210 Vienna, Austria
{dk,nv}@dke.univie.ac.at

Abstract. Future enterprise systems require an elaborate conceptual foundation that promotes a tight mutual alignment between information systems and business to effectively support business operations and managerial decision-making. Thus a growing number of groups around the world show interest in modelling methods – either standard or individual ones – that satisfy the requirements of their domain and comply with the conceptual foundations. In order to analyze modelling methods in different domains, we introduce a generic modelling method specification framework that describes modelling methods on three major parts: (i) the modelling language that describes the syntax, semantics and notation, (ii) the modelling procedures that describe the methodology as well as (iii) algorithms and mechanisms that provide "functionality to use and evaluate" models described by a modelling language. Simultaneous to the raise of modelling methods there is a need for re-use, integration or combination of different modelling methods. The metamodelling approach is considered to provide the required concepts and mechanisms to combine different modelling methods for the so called Hybrid Method Engineering. The Next Generation Modelling Framework (NGMF) supports Hybrid Method Engineering, both on a conceptual and on a technical integration level. On conceptual level the NGMF provides mechanisms to encapsulate modelling methods and enable the hybrid use of different modelling languages. On technical level the Next Generation Modelling Platform provides functionality for modelling method engineering and modelling method application. The organizational framework is provided by the Open Model Initiative (OMI), which supports users in realizing and applying this approach based on three pillars: community, projects and foundations.

Keywords: Metamodelling Platforms, Hybrid Modelling, Modelling Methods, Modelling Languages, ADOxx.

1 Introduction

Modern day system developers have some serious problems to contend with. The systems they develop are becoming increasingly complex as customers demand richer functionality delivered in ever shorter timescales. To add to that, nothing stays still: today's "*must have*" technology rapidly becomes tomorrow's legacy problem that must be managed along with everything else. Languages are the primary way in which system developers communicate, design and implement systems. Languages provide abstractions that can encapsulate complexity, embrace the diversity of technologies and

J. Grabis and M. Kirikova (Eds.): BIR 2011, LNBIP 90, pp. 19–28, 2011.

design abstractions, and unite modern and legacy systems. The benefit of *metamodelling* is its ability to describe these languages in a unified way. This means that the languages can be uniformly managed and manipulated thus tackling the problem of language diversity. For instance, mappings can be constructed between any number of languages provided that they are described in the same metamodelling language. Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain [19]. As a result productivity is greatly improved.

Uses for a metamodel can be summarized as follows: define the syntax, notation (sometimes also called visual or graphical syntax) and semantics of a language, explain the language, compare languages, specify requirements for a tool for the language, specify a language to be used in a meta-tool, enable interchange between tools, enable mapping between models.

1.1 Basic Definitions

The notion of model goes beyond the narrow view of semi-formal diagram thus requiring much more precise definitions. The following definitions help us in understanding the concept of SUS, model, conceptual model, metamodel, and meta-metamodel.

A *System under Study* (SUS) is a delimited part of the world considered as a set of elements and interactions. A *model*, representation of a given SUS, is a directed multigraph that consists of set of nodes, a set of edges, and a mapping function between nodes and edges, where nodes may be connected with more than one edge, and is such that its reference model is a metamodel [24]. *Conceptual model*, also known as domain model, represents concepts (entities) and relations between them, and is independent of design or implementation concerns [26]. The aim of a conceptual model is to express the meaning of terms and concepts used by domain experts to discuss the problem, and to find the correct relationships between different concepts. The conceptual model attempts to clarify the meaning of various, usually ambiguous terms, and ensure that problems with different interpretations of the terms and concepts cannot occur. A *metamodel* is a model such that its reference model is a meta-metamodel [24]. In its broadest sense, a metamodel is a model of a modelling language, and it must capture the essential features and properties of the language that is being modelled. Thus, a metamodel should be capable of describing a language's syntax, notation and semantics. A *meta-metamodel* is a model that is its own reference model (i.e. conforms to itself) [24]. It is the key to metamodelling as it enables all modelling languages to be described in a unified way, i.e., all metamodels are described by a single meta-metamodel.

Concepts mentioned here represent different tiers of abstractions of the real world, where SUS can be viewed as lowest or tier zero, and meta-metamodel as highest or tier three.

1.2 DSLs vs. GPLs

There is a variety of categories of languages. A distinction is often made between programming languages and modelling languages, but this distinction is currently

becoming more and more blurred since programs are treated as models, and some modelling languages may have the executability property. Another distinction is between General Purpose Languages (GPLs) and Domain Specific Languages (DSLs). UML, Java, and C# are examples of GPLs. SQL, HTML, and Excel are examples of DSLs. A DSL is a language designed to be useful for delimited set of tasks, i.e., they have a clearly identified, concrete problem domain, in contrast to GPLs that are supposed to be useful for much more generic tasks, crossing multiple application domains. Domain-Specific Modelling Language (DSML) is a special case of DSL that is used in domain of modelling (as outlined in [28] for Service Modelling).

2 Metamodelling Platforms: An Overview

Metamodelling approaches are an active research field and in the past 20 years serious application areas in the software and information technology industries have been found. Some of them are Enterprise Model Integration (EMI) [8] in the context of Enterprise Application Integration (EAI) [9], Model Integrated Computing (MIC) [10], modelling languages such as the Unified Modelling Language (UML) [11] based on Meta Object Facility (MOF) [12], and model driven development approaches such as Model Driven Architecture (MDA) [13].

Applying the research results of metamodelling approaches metamodelling platforms are developed, like ADO*xx*, industrial software like ADONIS [14], MetaEdit+ [15], modelling frameworks like Eclipse Modelling Framework (EMF) [7], and toolkits like Generic Modelling Environment (GME) [17].

ADO*xx* is an extensible, repository-based metamodelling platform, which offers a three-step modelling hierarchy with a rich meta-metamodel. ADO*xx* can be customized using metamodelling techniques and extended with custom components to build a modelling environment for a particular application domain. ADONIS is a modelling tool based on ADO*xx* for the domain of business process management [14]. The ADO*xx* platform kernel provides basic modules for managing models and metamodels. In addition, the ADO*xx* generic components for graphical and tabular model editing, for model analysis, for simulation, or for model comparison can be reused and customized in all solutions derived from ADO*xx*. Each ADO*xx*-based solution contains a solution-specific modelling language and may have additional set of solution specific components. The scripting language AdoScript provides mechanisms to define specific behavior and functionality. Mechanisms such as *simulation* or *analysis* are defined on meta-meta level and can be redefined on the metamodel level.

MetaEdit+ is a completely integrated environment for building and using individual Domain-Specific Modelling (DSM) solutions [15]. Same as ADO*xx*, it offers a three-step modelling hierarchy. The meta-metamodel forms the *GOPRR* model, offering the basic concepts *Graph*, *Object*, *Property*, *Relationship* and *Role*. A diagram editor, object & graph browsers, and property dialogs support the definition of a new modelling language without manual coding.

OMG's MOF [12], the open source EMF [7] and the Graphical Editor Framework (GEF) [16] are no meta-CASE tools themselves. With MOF the OMG created a meta-metamodel standard, which provides a basis for defining modelling frameworks. UML [11] is an example of instantiated metamodel of the MOF. The EMF which was

influenced by MOF is an open source Java based modelling framework and code generation facility for building tools and other applications based on a structured data model. Together with the GEF it provides a possibility to create a new modelling tool.

The GME [17] is a configurable toolkit for creating DSM and program synthesis environments. The configuration is accomplished through metamodels specifying the modelling language of the application domain. The metamodelling language is based on the UML class diagram notation and OCL [18] constraints. The metamodels specifying the modelling language are used to automatically generate the target domain-specific environment. The generated domain-specific environment is then used to build domain models that are stored in a model database or in XML format. GME has a modular, extensible architecture that uses MS COM for integration. GME is easily extensible; external components can be written in any language that supports COM (C++, Visual Basic, C#, Python etc.).

3 Hybrid Modelling

The fundamental integration problem among metamodels (modelling languages) emerges when we try to join together vertically and/or horizontally different metamodels. Metamodels are (i) vertically different, when they vary in the level of details they describe, (ii) horizontally different, when their concepts on the same abstraction level describe different aspects of the system or the same aspect in a different way and (iii) both vertically and horizontally different, when they show characteristics of the previous two. No matter what kind of integration orientation is considered, there is a need to overcome *syntactical*, *structural* and *semantic* discrepancy of metamodels, in order to join their concepts together [25].

Syntactical heterogeneity [25] represents the difference in formats intended for the serialization of metamodels. Two metamodelling platforms can base their serialization mechanisms on different proprietary formats or even paradigms, e.g. having diverse relational, object oriented or XML based schemas.

Structural heterogeneity [25] can be expressed through representational and schematic heterogeneity. Metamodels are represented using different metamodelling languages, i.e. meta-metamodels, each of them showing difference in its expressive power of available modelling primitives (classes, attributes, supported relationship types, etc.). Even when agreed on the common meta-metamodel, metamodels vary schematically when the same concepts being described are modelled in a different way (thus having different conceptual schemas). There are two primary reasons for schematic conflicts: equal concepts are modelled either with different modelling primitives or with different number of primitives.

Semantic heterogeneity [25] includes differences in the meaning of the considered metamodel concepts. Concepts coming from different metamodels can use the same linguistic terms to describe different concepts or use different terms to describe the same concept etc.

A modelling method consists of two components: a modelling technique, which is divided in a modelling language and a modelling procedure, and mechanisms & algorithms working on the models described by the modelling language (see Figure 1). The modelling language contains the elements with which a model can be described. A

modelling language itself is described by its syntax, semantics, and notation. The modelling procedure describes the steps applying the modelling language to create results, i.e., models [2]. The amount of requirements concerning defined syntactical rules and modelling steps is influenced by the automated processing that is planned on the created models. This processing is done with the help of mechanisms & algorithms that provide "*functionality to use and evaluate*" models. Basically, when such functionalities, enabling structural analysis (e.g. queries that return activities that meet some defined criteria like costs, delivery times) as well as simulation of models (e.g. prediction of cycle times or staff requirements) are defined for existing modelling techniques, the modelling methods are formed [1].

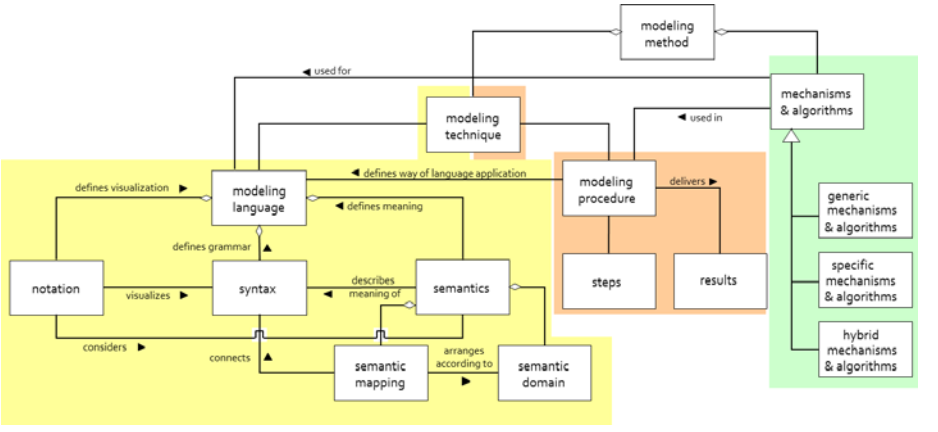

Fig. 1. Modelling methods, mechanisms and algorithms (Karagiannis & Kühn, 2002)

The issue of merging two or more modelling methods (as outlined in [27]) into one can be addressed as *Hybrid Method Engineering* – a combined modelling approach, based on meta-modelling that takes into consideration the different perspectives of modelling languages (metamodels), and results in a comprehensive modelling framework. For this issue to be solved appropriately we need to go through several steps: (i) *integration of modelling methods*, (ii) *support of standards*, and (iii) *merging of different modelling concepts*. In the integration step we need to make sure that different modelling methods, or method chunks are integrable on a common platform. If modelling languages (metamodels) that are a part of modelling methods can be integrated, modelling methods can be integrated as well. In the *support of standards* step we need to check if available standards for modelling aspects are supported, i.e., compliance of models and metamodels to a standard, e.g. ITIL compliance validation. In the last step, *merging of different modelling concepts*, prerequisite is that the platform supports a combined view on concepts from different disciplines [3].

To support the concepts mentioned in this chapter, metamodelling platforms should be realized on a component-based, distributable, and scalable architecture [2]. An important element of metamodelling platform architecture is the meta-metamodel [5]. The meta-metamodel [2] defines general concepts available for method definition and method application such as *metamodel*, *classes*, *relations*, *attributes*, *model types*, etc.

According to Karagiannis & Kühn, other important architectural elements of the metamodelling platform are: *metamodel base*, *model base*, *mechanism base*, *persistency services*, and *access services*. Metamodel base, model base and mechanism base are all based on meta-metamodel and they store, respectively, metamodels, models, and mechanisms. Persistency services support the durable storage of the various bases. These services abstract from concrete storage techniques and permit filling of modelling information in heterogeneous databases, file systems, web services, etc. Access services provide the open, bidirectional exchange of all metamodelling information with other systems, and cover all aspects concerning security such as access rights, authorization, en-/decryption, etc.

A strong model repository is composed of the metamodel, model & mechanism base, persistency services, access services, version control, and validation & verification mechanisms. Furthermore, the model repository needs to be designed to accommodate the reuse of already developed modelling method constructs, and to support pruning and slicing algorithms. If there are such prerequisites, hybrid modelling methods can be easily developed using chunks and pieces from the repository by binding them together into a new coherent whole using appropriate mapping and integration rules.

4 MCG vs. MAS Metamodelling Platforms

Every metamodelling platform is built around a concrete meta-metamodel. This can be a custom meta-metamodel, specifically developed for that platform, or it can be already specified meta-metamodel like MOF. Because most of the modelling platforms have similar foundation, i.e., meta-metamodel, we need to find other key characteristics for metamodelling platform comparison.

Generally, we can divide metamodelling platforms in two groups: the ones that specialize in *model based code generation* (MCG), and the ones that specialize in *model analysis & simulation* (MAS). The other comparison can be based on *value added* to the metamodelling platforms, that is, extra features that are distinguishing one platform from the other.

Most of the metamodelling platforms have additional features that are distinguishing them from other metamodelling platforms. After doing research on most popular metamodelling tools, including ADO*xx*, MetaEdit+, GME, and EMF, a list of important features was compiled (see Table 1). The most influential factors for defining this list of features are: (i) productivity (or rather rise in productivity), (ii) usefulness, and (iii) quality of the modelling tool produced.

Model based code generation (MCG) metamodelling platforms support Model-Driven Engineering (MDE) methodology. MDE is a software development methodology focused on creating and exploiting domain models. In MDE, we use models as the primary artifacts in the development process – we have source models instead of source code [20]. MDE raises the level of abstraction and hides complexity. Truly MDE uses automated transformations in a manner similar to the way a pure coding approach uses compilers. Once models are created, target code can be generated and then compiled or interpreted for execution. From a modeler's

Table 1. List of features used to compare metamodelling platforms

Feature	Description
Language Definition Approach	Graphical, form-based, hybrid
Specifying Notation	Graphical, text-based, hybrid
Syntax Highlighting & Debugging	Support for highlighting, autocompleting, debugging
Scripting	Scripting support for advance customizing
Import & Export	Import & export of metamodels and models
Integration with Other Tools	APIs, Client-Side, Server-Side Integration
Rich Notation	More than simple symbols for nodes and arcs
Dynamic Symbol Change	Symbols change dynamically when model data changes
Different Modelling Views	Modelling, matrix, tabular view

perspective, generated code is complete and it does not need to be modified after generation. For this approach to work, knowledge is not just in the models, but in the code generator and underlying framework. To raise the level of abstraction in MDE, both the modelling language and the generator need to be domain-specific, that is, restricted to developing only certain kind of applications. Focusing on a narrow area of interest makes it possible to map a language closer to the actual problem and makes full code generation realistic – sometimes that is difficult, if not impossible, to achieve with general-purpose modelling languages (UML, etc.).

Model analysis & simulation (MAS) metamodelling platforms support Enterprise Modelling (EM) methodology, including areas like Enterprise Model Integration (EMI) and Enterprise Application Integration (EAI). EM is the abstract representation, description and definition of the structure, processes, information and resources of an identifiable business, government body, or other large organization [22]. It deals with the process of understanding an enterprise business and improving its performance through creation of enterprise models. This includes the modelling of the relevant business domain, business processes, and information technology. In BPM (Business Process Management) models are primarily used for analysis and simulation of the business processes, to find the means to improve their efficiency and quality. Another example of using enterprise models is sharing of knowledge between two or multiple parties (people, departments, companies, etc.). Because, enterprise models are primarily used for analysis & simulation, MAS metamodelling platforms are specialized for creating modelling methods, which are an upgrade on modelling languages, including modelling procedures, algorithms & mechanism (see Figure 1).

5 The Open Model Initiative

The Open Model Initiative (OMI) is an international scientific community, which focuses on the creation, design, evolution and processing of modelling methods and the models designed with them. The initiative is open-membership for all interested experts and organizations, and every *'model'* which is considered to be useful for a specific purpose by any application domain. The results are public.

OMI provides value both through the modelling and meta-modelling compiler ADO*xx* and also through the social and collaborative platform, by providing

knowledge and communities of practice for the development of modelling methods and tools. Adjacent services like OM-TV, the OM-Repository, OM-Apps, OMIverse and OMIpedia give additional features to the initiative.

OMI is structured through its activities in three pillars:

- Community: where groups of individuals share common values and follow common goals. Organized in communities of practice for different domains, they provide value through competence, joint activities, shared practices and resources, sustained interaction, experiences and tools.
- Projects; which can be either (a) modelling projects, thus creating model content for various domains and/or purposes (www.wikimodels.org) and (b) method engineering projects, where the conceptualization of new and further development of existing methods, development and deployment of IT-based modelling tools is realized (www.wikimethods.org), and
- Foundations: which provide modelling languages and algorithms for the processing of models as well as IT-based modelling environments. Additionally this pillar supports designers to choose the right algorithms for the processing of methods and models.

In other words, the OMI platform is a social computing platform which supports the communities through multiple different channels of communication, e.g., forums, blogs, wikis, etc. This model, as mentioned in [23] supports online human interaction and information flow so that communities are formed for ongoing collaboration and exchange of information and knowledge among their members.

6 Conclusion

A metamodel as an idea is introduced to raise the level of abstraction and to simplify the development of modelling languages, modelling methods, and finally, modelling tools. Raising the level of abstraction means making the metamodel flexible, that is, customizable by the metamodelling platform user. This user is sometimes called language engineer. The advantage of using flexible metamodels is direct mapping to the domain under study. There are also benefits that come in form of considerable savings in time and costs and increased quality of delivered solutions.

Due to rapid changing business requirements such as faster time to market, shorter lifecycles, increased interdependencies between business partners, and tighter integration of the underlying information systems, the complexity in developing applications which deliver business solutions is continually growing [2]. This is the reason why the elements of an enterprise are managed more and more model-based, and why are metamodelling platforms getting an integral part of business engineering strategies and approaches. Well-known examples are international standards UML [11] and MOF [12], and metamodelling platforms like ADO*xx* and MetaEdit+ [15]. Additionally, domain specific languages, model transformation approaches, and lifecycle management within large model bases are active research issues.

The other more specific research issue that needs attention is language engineering. Languages are hard to design. The effort that goes into producing a language definition can be overwhelming, particularly if the language is large or semantically rich. That is

why reusability is very important. By reusing, rather than re-inventing, it is possible to significantly reduce the time spent on development, allowing language designers to concentrate on the novel features of the language [19].

One of the less noticeable problems is that almost every modelling and metamodelling platform wants to do everything. Most underlying frameworks are very general. Additional functionality for a specific domain of application should be engineered upon the meta-metamodel of the metamodelling platform. That way a new generation of more specialized metamodelling platforms can be developed.

References

1. Karagiannis, D., Grossmann, W., Höfferer, P.: Open Model Initiative A Feasibility Study. University of Vienna, Department of Knowledge Engineering (September 2008)
2. Karagiannis, D., Kühn, H.: Metamodelling platforms. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, p. 182. Springer, Heidelberg (2002)
3. Xu, T., Ma, W., Liu, L., Karagiannis, D.: Hybrid Modelling: Strategic Model and Business Processes in Active-i*. In: WGBP 2010, Vitoria, ES, Brazil (2010)
4. Brumar, B.A., Popa, E.M.: Advanced techniques for metamodelling. In: Proceeding ICCOMP 2007 Proceedings of the 11th WSEAS International Conference on Computers World Scientific and Engineering Academy and Society (WSEAS) Stevens Point, Wisconsin, USA (2007)
5. Kühn, H., Murzek, M.: Interoperability Issues in Metamodelling Platforms. In: Proceedings of the 1st International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA 2005), Geneva, Switzerland. Springer, Heidelberg (2006)
6. Open Model Initiative, `http://www.openmodels.at/`
7. Eclipse Modelling Framework Project (EMF), `http://www.eclipse.org/modelling/emf/`
8. Kühn, H., Bayer, F., Junginger, S., Karagiannis, D.: Enterprise Model Integration. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2003. LNCS, vol. 2738, pp. 379–392. Springer, Heidelberg (2003)
9. Linthicum, D.S.: Enterprise Application Integration. Addison-Wesley, Reading (2000)
10. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason IV, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modelling Environment. In: Workshop on Intelligent Signal Processing at WISP 2001, Budapest, Hungary, May 17 (2001)
11. Object Management Group: Unified Modelling Language, `http://www.omg.org/spec/UML/`
12. Object Management Group: Meta Object Facility, `http://www.omg.org/mof/`
13. Object Management Group: Model Driven Architecture, `http://www.omg.org/mda/`
14. ADONIS, `http://www.boc-group.com/products/adonis/`
15. MetaEdit+, `http://www.metacase.com/`
16. Graphical Editing Framework, `http://www.eclipse.org/gef/`
17. Generic Modelling Environment (GME), `http://www.isis.vanderbilt.edu/projects/GME`
18. Object Constraint Language (OCL), `http://www.omg.org/spec/OCL`

19. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied Metamodelling - A Foundation for Language Driven Development, Version 0.1 (2004)
20. Kelly, S., Tolvanen, J.-P.: Domain-Specific Modelling - Enabling Full Code Generation. Wiley-IEEE Computer Society Press (2008)
21. Frank, U.: Multi-Perspective Enterprise Modelling (MEMO) - Conceptual Framework and Modelling Languages. In: Proceedings of the 35th Hawaii International Conference on System Sciences, HICSS 2002 (2002)
22. Fox, M.S., Gruninger, M.: Enterprise Modelling. AI Magazine 19(3) (1998)
23. Capuruço, R.A.C., Capretz, L.F.: A Unifying Framework for Building Social Computing Applications. In: Lytras, M.D., Damiani, E., Tennyson, R.D. (eds.) WSKS 2008. LNCS (LNAI), vol. 5288, pp. 11–21. Springer, Heidelberg (2008)
24. Kurtev, I., Bezivin, J., Jouault, F., Valduriez, P.: Model-based DSL Frameworks. In: Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2006), Portland, Oregon, USA (2006)
25. Zivkovic, S., Kühn, H., Karagiannis, D.: Facilitate Modelling Using Method Integration: An Approach Using Mappings and Integration Rules. In: ECIS 2007 Proceedings. Paper 122 (2007)
26. Moody, D.L.: Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. Data & Knowledge Engineering 55, 243–276 (2005)
27. Utz, W., Woitsch, R., Karagiannis, D.: Conceptualisation of Hybrid Service Models: An Open Models Approach. In: The 4th International IEEE Workshop on Service Science and Systems, Held in Conjunction with COMPSAC 2011 (in press, 2011)
28. Hrgovcic, V., Utz, W., Karagiannis, D.: Service Modeling: A Model Based Approach for Business and IT Alignment. In: The 5th International IEEE Workshop on Requirements Engineering for Services, held in Conjunction with COMPSAC 2011 (in press, 2011)