Enterprise Ontology Driven Software Engineering

Steven J. H. van Kervel[2], Jan L. G. Dietz[1], John Hintzen[2], Tycho van Meeuwen[2] and Bob Zijlstra[2]
[1]*Delft University of Technology, Delft, The Netherlands*
[2]*Formetis Consultants BV, Hemelrijk 12 C, Boxtel, The Netherlands*

Keywords: Model Driven Software Engineering, Enterprise Ontology, Demo, Adaptive Case Management Systems.

Abstract: Model Driven Engineering (MDE) has been with us for quite some time, the most well known MDE approach being OMG's Model Driven Architecture. Current MDE approaches suffer from two major shortcomings. First, they are unable to deliver domain models that comprise all functional requirements. Second, the models to be produced during the system development process, are not formally defined. One of the theoretical pillars of Enterprise Engineering (EE) is the Generic System Development Process (GSDP). It distinguishes between the using system and the object system, and it states that any software development process should start from the ontological construction model of the using system. In addition, EE's systemic notion of Enterprise Ontology provides us with a formalized ontological model of an organization that satisfies the C4E quality criteria (Coherent, Consistent, Comprehensive, Concise, and Essential). An operational application software generator is presented that takes this ontological model, with some extensions, as source code input and executes the model as a professional software application. Changes in the software, as required by any agile enterprise, are brought about 'on the fly', through re-generation, based on the modified ontological model of the enterprise's organization.

1 INTRODUCTION

In this paper, the scope of interest is enterprises; cooperatives of human beings for delivering valuable results to other human beings, and their supporting IT systems. Enterprises are social systems, i.e. the elements are social individuals (human beings). The operating principle of enterprises is that these individuals, commonly called actors, make commitments and communicate regarding the bringing about of products for the benefits of actors in the environment of the enterprise. This notion of enterprises is based on the theory of enterprise ontology (Dietz, 2006). Enterprise ontology is a "formal, explicit specification of a conceptualization shared between stakeholders" (Gruber, 1993). It provides a shared vocabulary represented as a set (of limited size) of symbols of objects, used to model enterprises.

Enterprise ontology is also an ontology with empirical evidence of a high degree of ontological appropriate and truthfulness (Mulder, 2008), notions that are describes by Guizzardi (2005).

Enterprise Information systems (EIS) are those information systems that support the *operation* of an enterprise. There are other information systems in organizations that are not an EIS, for example accounting systems, CRM systems, etc.

Current state of the art engineering methodologies that should result into high quality IT systems, fail too often. We focus on three major problems but ignore management, political and strategic problem sources.

The *first* problem is the mismatch between the specified functionality of IT systems and the expectations of the stakeholders; a lack of business-IT alignment [ITGI]. Empirical evidence of the Standish Group (2009), Tata and others confirms this. We are obviously unable to produce high quality IT system functional specifications.

The *second* problem is that IT projects do not deliver IT systems within controllable financial budgets and resources (Sauer and Cuthbertson, 2003). Delivered software components need often overhaul cycles and programming is too demanding.

The *third* problem is that the costs of maintaining IT systems grow exponentially over time (Lehman, 1985). This problem is clarified and addressed by the theory of Normalized Systems (Mannaert and Verelst, 2008).

J. H. van Kervel S., L. G. Dietz J., Hintzen J., van Meeuwen T. and Zijlstra B..
Enterprise Ontology Driven Software Engineering.
DOI: 10.5220/0004080902050210
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 205-210
ISBN: 978-989-8565-19-8

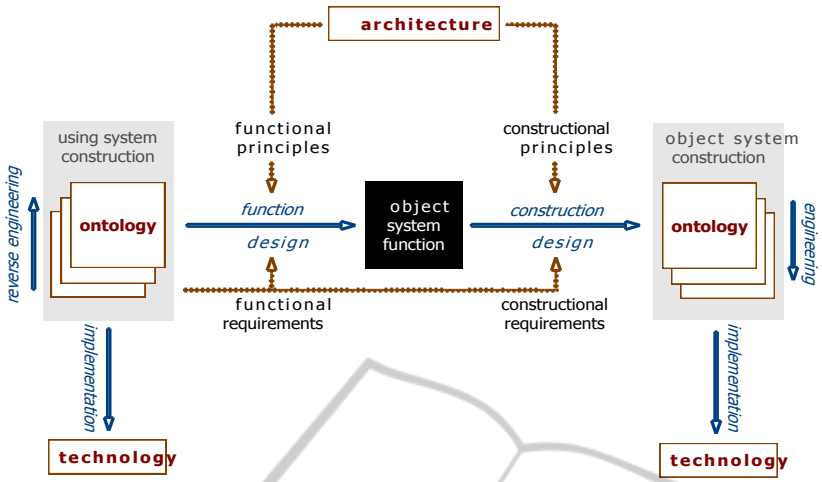

Figure 1: The generic System Development Process (GSDP).

If software engineers are provided with high quality specifications, they are usually able to construct high quality software systems for complex domains, for example GPS systems, cell phone systems and the Internet. EIS specifications apparently lack quality. Consequently, programmers have to take design and implementation decisions they should not take because they are not qualified to do this. There is unwanted design freedom and ambiguity.

In general, EIS specifications lack *comprehensiveness*: "everything" that should be included must be there. Other shortcomings are irrelevant information or lacking *conciseness;* hidden anomalies and inconsistencies or lacking *consistency*; and specifications that do not match semantically, lacking *coherence*. Programmers need software specifications matching the four quality criteria, comprehensiveness, conciseness, consistency and coherence (C4-ness) to deliver high quality IT systems.

2 SOFTWARE ENGINEERING PROBLEMS AND CHALLENGES

As a framework for discussing the problems mentioned above, we adopt the Generic Systems Development Process (GSDP) (Dietz, 2008). The GSDP is a generic model for developing any kind of artifact. It comprises all basic steps that need to be taken in a development process. The GSDP supports two distinctive perspectives on systems, the construction perspective and design principles, represented by white box models, and the function perspective and design principles, represented by black box models. This is especially important for software engineering.

Fig. 1 exhibits the basic steps in a system development process. The starting point is the need by some system, called the *using system* (US), of a supporting system, called the *object system* (OS). By nature, this need stems from the construction of the US, so one starts with a white-box model of the US. Ideally, this starting model is a purely ontological model of the US. This includes that the model is an instance of an appropriate metamodel, and that it is fully implementation independent. The first development step is function design, resulting in the functional specifications of the OS. By nature, these specifications constitute a black-box model of the OS, expressed in 'the language' of the US, i.e. in terms of the construction and operation of the US.

We consider them to include the so-called non-functional requirements, regarding various performance and quality aspects.

The next step is construction design, resulting in the constructional model of the OS. It can usefully be divided into ontological design and implementation design (engineering in Fig. 1). By the ontological design we mean the highest level white-box model of the OS; ideally it is purely ontological, meaning that it does not contain implementation choices.

A thorough analysis of this white-box model must guarantee that building the OS is feasible, given the available technology. There is the important recognition that designing is an iterative process, which is, however, not indicated in the figure. The end result of a design process is a balanced

compromise between (reasonable) requirements and (feasible) specifications. Regarding IT systems, this model is the source code of the system. This understanding of engineering is fully compliant with MDE. Both function design and construction design are 'fed' by requirements and principles.

An important advantage of applying ontological models, both of the US and of the OS, is that these models possess C4-ness quality. An ontological specification that is completely implementation independent may be used at an early stage for validation and assessment. This helps to avoid expensive re-implementation cycles.

3 ONTOLOGY BASED MODEL DRIVEN ENGINEERING

Model driven engineering (MDE) (OMG, 2001) is generally considered to address the problems we have discussed above. In MDE, software is developed in a series of model transformations. The process starts from the functional specifications and ends with the 'source code' in some formal language. However, a major cause of problems is still present. Projected on the GSDP, MDE proceeds in three phases: function design, construction design, and implementation design (engineering in Fig. 1). Each of these phases is prone to errors. Regarding the first one, it means that the functional specifications (the black-box model of the OS) are different from the user expectations. Regarding the second phase, it means that the ontological construction specifications do not fully match the functional specifications. Regarding the third phase, it means that the resulting implementation model does not fully match the ontological model of the OS. Another major cause of failures in applying MDE seems to be the lack of rigor in (formally) defining the distinct kinds of models, including the ignorance of the fundamental differences between functional and constructional models. One cannot just 'transform' a functional model into a constructional one.

In the MDE approach we propose, these drawbacks are excluded. This is achieved first by noting that any complete and correct implementation of the ontological model of a US, makes the US 'alive', once put into operation. Therefore we will consider any EIS, which is completely supporting some enterprise, as a possible implementation of the ontological model of the enterprise. By this, obviously correct, observation, we actually make the ontological model of the OS (our EIS) isomorphic to the ontological model of the US (the enterprise). Informally stated: everything in the US ontology exists also in the OS ontology, and everything in the OS ontology exists also in the US ontology.

By doing this, we skip the function design phase and the construction design phase.

Second, we consider the operating EIS as a real-time simulation of its ontological model. By doing this, we skip the engineering phase (implementation design) in the GSDP. For the MDE approach we have in mind, it means that we need a software system that executes ontological enterprise models. We call this system the DEMO processor. Concluding, we have fully addressed the first major cause of problems in applying MDE, as discussed above.

The second major cause (the lack of formalization) is also fully eliminated, by basing our MDE approach on the quality criteria as developed by Guizzardi (2005). Fig 2 exhibits Guizzardi's conceptual framework, slightly adapted to our purposes. Guizzardi distinguishes between a collection of real world phenomena *R* and a collection of real world phenomena *m*, which are

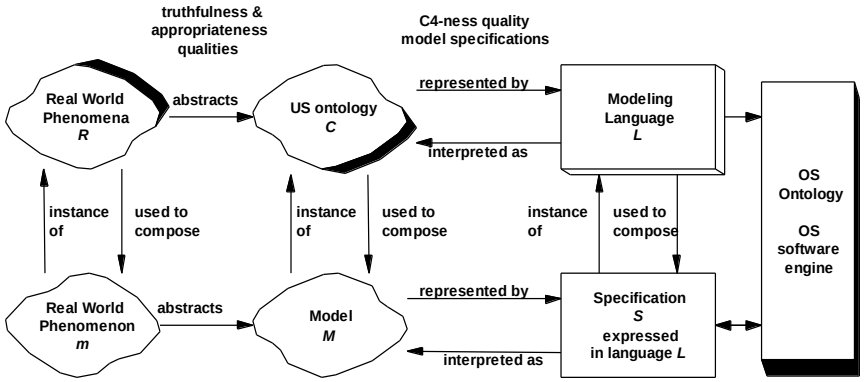

Figure 2: The GSDP MDE approach in Guizzardi's framework.

instances of the things in *R*. Applied to our purposes, *R* is the collection of all enterprises, and actors, transactions etc. from *R*. A conceptual model of an *R* is called a *C*. In our case, C is the notion of enterprise ontology, as defined by the ψ-theory (Dietz, 2006). Thus, such a model consists of transaction kinds, actor roles, and the connections between them. Next, every *M* is a conceptual model of a particular enterprise *m*, as well as an instance of *C* (by definition). In our case, this means that every M is an ontological enterprise model according to the ψ-theory. Guizzardi requires that the mapping from *R* to *C*, and consequently, from every *m* to its corresponding *M*, satisfies the qualities of truthfulness and appropriateness. Truthfulness refers to the extent to which the concepts of the ontology are able to represent phenomena in reality in a *truthful* way for all stakeholders (Krogstie, 2000). The *ontological appropriateness* quality (Krogstie, 2000) refers to how well and useful the ontology supports understanding and shared reasoning between stakeholders. A lack of truthfulness renders a model expressed in the ontology useless. A lack of appropriateness renders a model less valuable.

Because we know that our US ontology (which is the notion of enterprise ontology according to the ψ-theory) has the C4-ness qualities, it is possible to design a high quality modeling language *L*, in which enterprise ontological models can be expressed. So, *L* comprises the diagrams, tables, and other model representations of DEMO. The specification (*S*) of the ontological model of an enterprise (*M*) in *L* is called the DEMO model of the enterprise. For every (DEMO) model *M* there is one and only one specification *S* in *L*. Every specification *S* is interpreted as one and only one (DEMO model) *M*. Put differently, *S* is equivalent to one and only one specific DEMO model, and vice versa.

We have also developed an XML-based language, called DMOL (DEMO Modeling Language) of which the metamodel is isomorphic to the metamodel of *L*. Specifications in our *L* are automatically transformed into DMOL specifications, after a specification *S* in *L* has been input into the DEMO processor. This process is model translation, not programming. Subsequently, the DEMO processor (Section 4) can execute the specification and make it operational.

In order to arrive at high quality specification languages, Guizzardi (2005) postulates a cardinality law that guarantees that anomalies such as construct excess and construct overload are eliminated:

m : M : S = 1 : 1 : 1 [Cardinality law]

It states that for every phenomenon *m* (in our case: an enterprise), there is one and only one model *M* (in our case: the enterprise ontological, i.e. the DEMO, model of the enterprise). Next, every model *M* is represented by one and only one specification *S* in *L* (in our case: the specification of the DEMO model in the DEMO language, later on transformed into DMOL). Conversely, every model specification in *L* represents one and only one model *M* (in our case: one DEMO model). Guizzardi states that if the cardinality law applies to the US ontology *C* and the language *L*, then there are valuable advantages; lucidity, ontological clarity and the elimination of construct overload and construct excess. Every model *M* can directly be mapped to a specification S, which is a straightforward process. This applies also to the atomic model elements and relations of C and the language primitives and constructs of L.

4 THE DEMO PROCESSOR

The DEMO processor (the OS software engine in fig 2), executes (reads, writes, constructs, destroys) DMOL (XML DEMO Modeling Language) representations of the four DEMO aspect models 1; the Construction Model, the Process Model, the Action Model, and the State Model (cf. (Dietz, 2006)).

The DEMO model that executes a model is an EIS. The implementation is precisely in line with section 3, meaning that any DEMO model is equivalent - isomorphic to the software of the EIS of the enterprise and also isomorphic to the enterprise producing a production *instance*. The essence is that the ontological model is the executable software.

4.1 Operation

The DEMO processor executes the enterprise model dynamically and delivers simulation results for model validation and production. Dynamic simulation means that any changes of the dynamic state of the enterprise over time are immediately reflected in equivalent dynamic state changes of the model under execution, vice versa.

At any moment the model under execution can be rendered as an DMOL XML file with full state information and temporarily stored in a model

1 Here we use the term 'model' also for a 'specification expressed in a language'. From the context is clear what is meant.

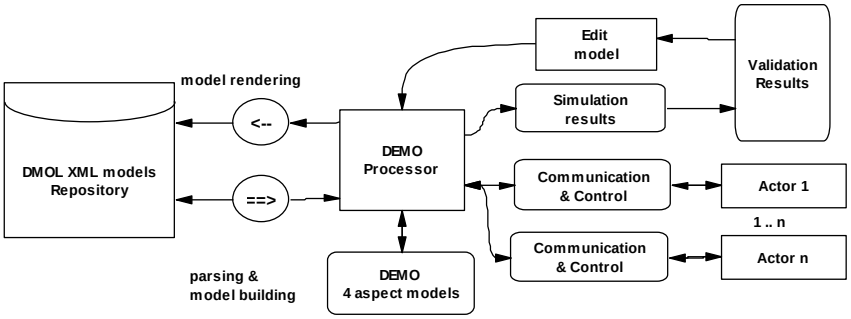

Figure 3: DEMO processor simulation and validation of an enterprise model.

instance repository. DEMO models under execution can be linked / aggregated at runtime into aggregated models that represent production chains of enterprises of unlimited size.

4.2 Enterprise Model Compliance Enforcement

Model execution delivers also a *prescriptive* specification of all allowed actor actions and prevents not allowed actor actions for each individual actor. DEMO models under execution are hence also a BPM (Business Process Model) execution machine, similar to a workflow system. Guerreiro (2011) investigated the DEMO processor as a role-based access control (RBAC) system. This is a control system enforcing model compliance of the enterprise to the model; the operaton of the enterprise cannot deviate from the model under execution. The workflow capability is completely calculated from the DEMO models, any BPM(-like) modeling is obsolete. At early development stages the models can be simulated for model validation, which supports a better business-IT alignment.

4.3 Model Instance Driven Case Management Systems

The EISs are characterized by the fact that the DEMO processor executes a *model instance* that is precisely related to a specific *production instance*. The model instance under execution represents in a truthful way the current production instance; any changes of the production instance, which is a state change of the production instance, is precisely calculated for the model instance under execution. In a large enterprise with many production instances, each of these production instances is in a unique state, more or less further advanced in production. The DEMO processor supports therefor model instance driven IT systems, also described as "adaptive case management systems".

4.4 Support for the Agile Enterprise and Evolving Information Systems

An agile enterprise is defined as an enterprise that continuously scans its environment and adapts itself whenever the need occurs. Enterprise agility is supported by the adaptation of the EIS through redesigned enterprise models, enabled by this approach.

5 CONCLUSIONS

The first DEMO processor application in full production is an adaptive case management system for a public utility company. The production consists of complex document-based contracts, subjected to business rules. This casus shows feasibility as a production IT system.

The business-IT alignment problem is addressed using an ontology with evidence of appropriateness. Direct execution of ontological models provides therefor a good degree of business-IT alignment of the EIS.

The possibility of subjecting models to validation at a very early project stage supports also a good degree of business-IT alignment.

The problem of uncontrolled software programming costs is addressed by the direct execution of DEMO models as the executable source code of the DEMO processor. In the first professional DEMO processor application, software programming is limited to interfaces with legacy systems etc; the core application is generated from models.

The deterioration of software quality during ongoing modifications is eliminated since the

DEMO processor is strict Normalized Systems (Mannaert, Verelst, 2008) compliant.

The DEMO processor offers also full workflow capabilities, elimination of anomalies such as deadlocks (Nuffel, 2009), and any BPM(-like) modeling is obsolete.

ACKNOWLEDGEMENTS

The authors thank the Endinet staff, especially Mrs. Marjolein Sigmans, for their cooperation in the first professional DEMO processor based IT project.

REFERENCES

Ciao Consortium; online: www.ciaonetwork.org

DEMO Knowledge Centre, Enterprise Engineering institute; http://www.demo.nl.

Dietz J.L.G. Enterprise Ontology. 2006, ISBN-10 3-540-29169-5, Springer Berlin Heidelberg New York.

Gruber, T. R 'A Translation approach to portable Ontology Specifications', 1993, Knowledge Acquisition, 5(2): 199-220, 1993. Knowledge Systems Laboratorium, Computer Science Department, Stanford University.

Guerreiro, S. Vasconcelos, A. Tribolet, J. Adaptive Access Control Modes enforcement in Organizations. Proceedings CENTERIS 2010, Springer-Verlag Berlin Heidelberg, Part II, CCIS 110, pp. 283–294, 2010.

Guizzardi, G. Ontological Foundations for Structural Conceptual Models. 2005, PhD thesis, University of Twente, The Netherlands. ISBN 90-75176-81-3.

ITGI, The IT Governance Institute. (URL: http:// www.itgi.org/).

Krogstie, J. 'Evaluating UML: A practical Application of a Framework for the understanding of Quality in Requirements Specifications and Conceptual Modeling'. Norwegian Informatics Conference (NIK) 2000.

Lehman M, 'Program Evolution – Processes of Software Change'. 1985. MM Lehman LA Belady,

Mulder, J. B. F. Rapid Enterprise Design. PhD thesis, 2008. ISBN 90-810480-1-5.

Mannaert H, and Verelst J, 'Normalized Systems', 2008. ISBN: 978 90 77160 008. J. Koppa, Kermt Belgium.

Nuffel van D, Mulder H, van Kervel, S. 2009. 'Enhancing the formal foundations of BPMN using Enterprise Ontology.' CAiSE CIAO 2009.

Object Management Group Inc, 2009. Business Process Modeling Notation (BPMN) Specifications. http:// www.omg.org/spec/BPMN/1.2/PDF/ (2009)

Sauer, C., and Cuthbertson, C., November 2003, "*The state of IT project management in the UK*." Templeton College, Oxford University.

Standish Group, CHAOS Summary 2009, (http:// www1. standishgroup.com/newsroom/chaos_2009.php)