INFORMATICA, 2003, Vol. 14, No. 4, 455–470
© 2003 *Institute of Mathematics and Informatics, Vilnius*

The Role of Ontologies in Reusing Domain and Enterprise Engineering Assets

Albertas ČAPLINSKAS, Audronė LUPEIKIENĖ
Institute of Mathematics and Informatics
Akademijos 4, 2021 Vilnius, Lithuania
e-mail: {alcapl,audronel}@ktl.mii.lt

Olegas VASILECAS
Vilnius Gediminas Technical University
Saulėtekio 11, 2040, Vilnius, Lithuania
e-mail: olegas@fm.vtu.lt

Received: June 2003

Abstract. The main purpose of the paper is to compare ontology-based reuse techniques in domain engineering and enterprise engineering. It discusses attempts to combine classical domain engineering techniques with ontology-based techniques as well as the attempts to incorporate ontologies in enterprise engineering process and demonstrates that, on the one hand, both approaches still are not mature enough to solve practical reuse problems and, on the other hand, both propose ideas that can be used to develop more mature approach. The main contribution of the paper is the detail description of the problems of ontology-based reuse of enterprise engineering assets.

Key words: enterprise engineering, domain engineering, ontology-based engineering, reuse.

1. Introduction

There is an increasing trend towards acceptance of reuse as an approach to information systems development. Reuse is defined as "the employment of a previously used and explicitly defined systems development artefact in another information systems development process" (Brash, 2001). The rationale for reuse is the belief that it would increase the quality of information systems and developers productivity. A lot of reusing techniques including requirement, analysis and design patterns, component based development, frameworks, parameterisation, and partial evaluation have been proposed in scientific literature (Allen and Frost, 1998; Baxter, 1999; Fowler, 1997; Gamma, 1994; Johnson, 1997; Jones *et al.*, 1993; Masuhara, 1999; Robertson and Robertson, 1999). In recent years the philosophy of reusing has been influenced strongly by two important factors: *domain engineering techniques* (Cohen, 1997a; 1997b; 1999; Czarnecki, 1997; Czarnecki and Eisenecker, 2000; Eichman, 1997; Falbo *et al.*, 2002; Maccario, 1997; Masuhara, 1999; Sherif, 1997; Štuikys and Damaševičius, 2002) and *ontology-based approach* (Borst, 1997; Dobson and Martin, 1997; Hakimpour and Geppert, 2002; Johnstone and McDermid, 2001; Maccario, 1997; Uschold, 1998; Van Belle, 1996; Wand and Weber, 1990; Weber, 1997).

The main purpose of the paper is to compare ontology-based reuse techniques in domain and enterprise engineering, and to explore the problem of ontology-based reuse of enterprise engineering assets. The authors are working in the project that aims to develop an enterprise engineering framework and it is the reason for this paper. The paper discusses attempts to combine classical domain engineering techniques with ontology-based techniques as well as the attempts to incorporate ontologies in enterprise engineering process. It demonstrates that, on the one hand, both analysed approaches are not mature enough to solve practical reuse problems, and, on the other hand, both propose ideas that can be used to develop more mature approach.

The rest of the paper is organised as follows. Section 2 discusses main reuse levels in the context of enterprise engineering. Reusable assets and reuse techniques at different levels are analysed as well. Section 3 analyses the ontology-based approach to domain and enterprise engineering. Section 4 investigates the problems of ontology-based reuse in enterprise engineering context. Finally, Section 5 concludes the paper.

2. Reuse in the Enterprise Engineering Context

Systematic reuse is employed in various engineering areas. Enhancing reuse is an important issue of information systems engineering, too. Current approaches (patterns, frameworks, business components, etc.) mostly address this topic either at different phases of the software development process or at the level of software components (sub-systems, objects, functions, etc.). In information system engineering, however, it is necessary to consider reuse issues at higher abstraction levels, namely, at enterprise engineering (EE), domain engineering (DE), and application engineering (AE) levels because not only software components should be reused. According to (Cohen, 1997a), EE should support reuse of all artefacts created in the enterprise domain including data and information architectures, product line definitions, and business models. DE should support reuse of products created in certain domain including generic domain models and generic software architectures. The term “domain” denotes here “an area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area” (Cohen, 1997b). The purpose of AE is to produce products for delivery using assets created through enterprise engineering and domain engineering.

Let us discuss the approaches mentioned more detail.

2.1. *Enterprise Engineering*

Usually, an enterprise is thought of as one or more organisations sharing a definite mission, goals, and objectives to offer an output such as product or service. It is necessary to distinguish concepts of “enterprise” and “enterprise system”. The relation between an enterprise and enterprise system is similar to one between hardware and software. An enterprise is a collection of functional entities (organisational structures, employees, software systems, machines, etc.) capable to do certain work. An enterprise system (ES) is a kind of operational system (Fig. 1) designed to perform some business (by the term "business" we mean any commercial, industrial, governmental or other interest). ES is run by enterprise, more exactly, by its functional entities.

ES is a hierarchical operational system. It has three levels of hierarchy: business level, information processing level, and information technology level (Fig. 1).

At the business level operates a number of business systems. Any business system is a subsystem of ES. It is implemented as a set of interrelated business processes performed by an enterprise to achieve certain, usually, long duration goals while producing particular products or/and services delivered to clients. A business process, in turn, is implemented as a set of interrelated activities, which manipulate (acquire, create, store, transform, transport, deliver, etc.) instances of particular business entities. It is governed by a number of business rules, implements particular functionality and meets a number of certain non-functional constraints ensuring the rational usage of recourses, reliability and other preferable properties of this process.

At the information processing level operates a number information systems. Usually, any information system supports a particular business system but sometimes it may be shared by several business systems. It is implemented as a set of information processes performed by enterprise functional entities in order to provide a number of information services required to support business processes. Information process is a kind of supporting process. It is implemented as a set of interrelated activities, which manipulate (create, copy, store, transform, transport, disseminate, etc.) particular information objects, and is governed by a number of information processing rules. Information object is a piece of business knowledge. Usually it is represented as a record of socially constructed reality and models some instance of particular business entity. Enterprise's information systems cannot operate as autonomous systems. They should be integrated into Enterprise Information System (EIS) and operate as its subsystems.

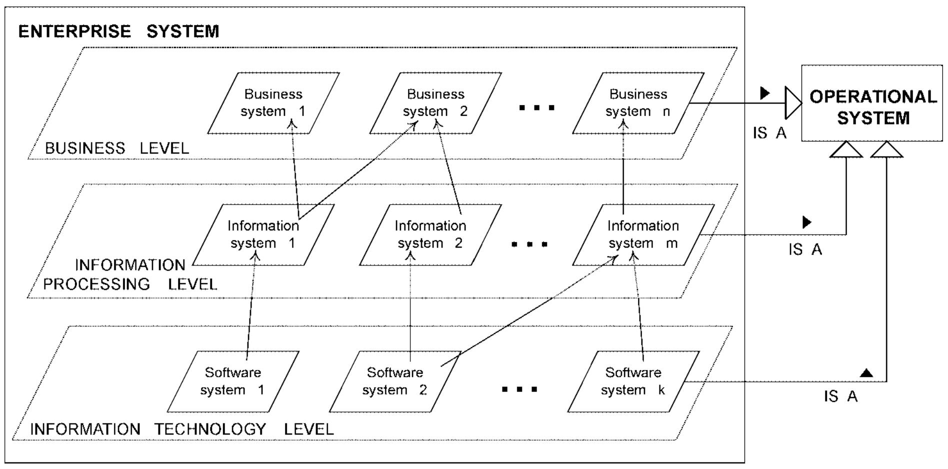

Fig. 1. Taxonomy of components of enterprise system.

At the IT level operates a number of software systems (applications, portals, data warehouses, expert systems, messaging systems, etc.). Any software system is a component of particular information system (sometimes it can be shared by several IS) and is used to implement, at least partly, a particular functional entity or a particular tool, which is used to manipulate software objects. Software objects are information objects represented in digital form.

So, architecture of any IS should be aligned with enterprise's business goals and mission. Architecture of software systems should be aligned with goals and mission of the related IS. To be effective all three levels of enterprise system should be integrated properly. Concepts should be mapped rightly from higher level system to lower level systems and lower level systems should be constrained by rules governing processes in higher level systems. Lower level systems should be aware of higher-level systems and all changes should be propagated correctly from top-level system to bottom-level one. Even systems of the same level should be aware of each other. Design and integration of ES levels is one of the most important tasks of EE.

According to (Liles *et al.*, 1998), enterprise engineering is a body of knowledge, principles, and practices having to do with analysis, design, implementation, and operation of an enterprise and its system. In other words, EE deals also with the design of enterprise itself. In this discipline the enterprise is viewed as a complex system created and maintained by a number of organisational processes. An example of organisational process is human resources management. Organisational processes should be supported by particular information systems that also should be integrated into EIS. The basic premise of EE is that key competitive advantages can be gained by rethinking business and organisation processes both externally, analysing the effectiveness of the enterprise from the customer's point of view, and internally, analysing the flow of work in the enterprise and the enterprise resources from the point of view of efficiency (Meerts, 2002). That is where information systems and information technology come in focus.

This paper concentrates on the development of EIS, disregarding issues of organisation and business design. In other words, we assume that enterprise itself and business level of ES are already designed and implemented. So the term "enterprise engineering" in the further text denotes activities related to design and implementation of information processing and IT levels and to integration of EIS components. It also encompasses the enterprise modelling (EM) activities because one must understand enterprise and business to design the information processing level. EM facilitates understanding of business processes and supports the reuse of business knowledge accumulated by enterprises. Enterprise models are used also in business engineering related EE activities, for example, in enterprise integration. As pointed out in (Bernus and Nemes, 1996), enterprise model is used in this field to "investigate how to perform processes better using a mix of technology and human recourses".

A variety of EM approaches have been proposed. They differ in intended purpose and in conceptualisations of an enterprise (ontological assumptions). For example, IDEF approach (NIST, 1993; Mayer, 1995) conceptualises business is in terms of input-output chains, business activities (functions), functional entities (mechanisms), and conditions required to produce correct output (controls). The approach proposed by Andreas Dietch (Dietsch, 2002) conceptualises business in terms of goals, input-output chains, and processes. Conceptualisation proposed in ODP Enterprise Language (ANSI, 1996) and used in project COMBINE (COMBINE, 2003) sees business in terms of behavioural, organisational, policy, and responsibility concepts. BPMI (Arkin, 2002) conceptualises business in terms of collaborations and interfaces.

As design and integration of ES levels is one of the most important tasks of EE, EM approaches should be revised to cover conceptualisation of the whole ES in uniform manner (Caplinskas *et al.*, 2002a). More, the common conceptualisation is necessary for reuse of business models, product lines, development processes in EE, which lacks of reuse techniques.

2.2. *Domain Engineering*

Domain engineering originated in the software engineering community and it puts emphasis on the formality and rationality. In the context of information systems, DE can be defined as a discipline that aims to support systematic reuse of business solutions and supporting software, "focusing on modelling common knowledge in a problem domain" (Falbo *et al.*, 2002).

Domain engineering aims at reuse assets created through EE, first of all, domain models and software architectures. Domain model represents domain concepts, objects, operations and their relationships (Maccario, 1997). It is used for understanding and classification of information within the domain (Cohen, 1997b). DE aims to reuse domain models through generalisation. Generic domain model is a generalisation of a family of domain models that have common aspects and predicted variability. The term variability is defined as "an assumption about how members of a family may differ from one another" and the term commonality as "an assumption that is true for all members of a family" (Robak, 2002). Generic domain model provides an abstract, independent and concise representation of common and variable parts of the family of domains (Kang *et al.*, 1990). Determining, what will constitute the common part and what the variable part, has more strategic than the technical nature. Commonality constraints the size of the family of domain models, variability increases the systems' complexity. The commonality and the variability of the family are defined within so called scooping activity (Cornwell, 1999). A variation point is a point identifying one or more locations, at which the variation will take place (Jacobson *et al.*, 1997).

DE constitutes three main phases: domain analysis, domain design, and domain implementation (Sherif, 1997). Generic domain model is produced during the domain analysis phase. Domain analysis is "the process where the knowledge of user/customer needs is identified, elaborated and organised in such a way that the applications planned inside a domain (or an application family) can be analysed and designed following modularity and reusability criteria" (Maccario, 1997). The outcome of the domain design phase is a generic system's architecture (domain architecture, architectural domain model) or, in other words, it is a description of the structural properties and elements of the system.

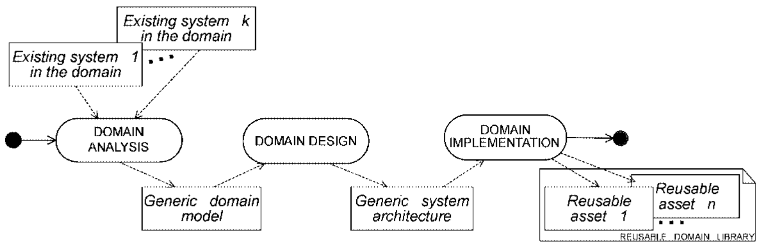

Fig. 2. Reuse in domain engineering.

It characterises the solution space versus the problem space characterised by the domain model. Generic architecture categorises architectural objects (interfaces, corporate data, processes, etc.) (Jacobson *et al.*, 1997). It describes particular family of systems (Robak, 2002; Weiss and Lai, 1999). So, generic architecture is the scheme that can be used for producing new domain applications or for re-engineering existing ones.

Starting from the previously defined generic domain model, the domain design activity extracts low-level components (object class, subsystems, etc.) and maps them to architectural objects that can be reused. Domain implementation refers to the development of reusable assets. The outcome of this phase and final product of the whole DE activity is repository of reusable components. During the domain implementation phase the generic architecture is implemented as a reusable asset (e.g., as an object framework). Automatic domain engineering (Frakes, 1997) means combining the domain analysis, domain design and domain implementation phase into a coherent whole. The outputs of one phase are mapped to inputs of another phase automatically and the whole DE process is supported by a set of compatible tools.

Since domain engineering originated in the software engineering community, it deals with family of systems but not with family of information processes. In other words, a domain model represents properties of the software, but not the common and variable properties of information processes. The sources of domain analysis include existing applications in that domain, their requirements, etc. So domain engineering supports reuse on IT level, but not on information processing level. From the information system engineering viewpoint, it is the shortcoming because it is desirable to reuse business models, too. We argue that, to satisfy the needs of enterprise engineering, domain engineering and IS engineering should be integrated more tightly.

2.3. *Application Engineering*

The purpose of application engineering (sometimes it is named reuse engineering (Agarwal *et al.*, 1999)) is to produce products for delivery using assets created through EE and DE (Cohen, 1997b). AE constitutes three main phases: requirement analysis, design analysis, and integration and testing. During the requirement analysis phase business needs using the features (reusable requirements) are described and additional requirements (needs not covered by the domain model) are formulated. Requirements are expressed as variations (so called deltas) to an established base. In other words, requirements describe a set of implementation constraints. Additional requirements are used to refine and extend the reusable assets. During the design analysis phase changes propagated by additional requirements are identified in generic design and the product configuration describing the architecture of the system under development is produced. During the integration and test phase the software system from delivered reusable components and additional custom-developed components is assembled. The input for AE is generic architecture together with the requirements for particular IS. DE and AE form together so called dual-lifecycle (Eichman, 1997).

It should be noted that currently application engineering has serious shortcomings. It almost always requires to customise reusable design and to write additional code. Thus additional attempts should be made to improve AE techniques in order to remove these shortcomings.

3. Ontology-Based Approach

3.1. *Reasons and Problems*

One of the reasons for building ontologies is the reuse capability they provide. A number of ontologies, for example, (Uschold *et al.*, 1996; Fox, 1992), has been developed for enterprises also. These ontologies conceptualise the enterprise domain and can be used to support various enterprise engineering processes.

According to Sowa (Sowa, 2000), "the subject of ontology is the study of the categories of things that exist or may exist in some domain". Uschold defines (Uschold, 1998) ontology as a vocabulary of interrelated terms, which "collectively impose a structure of the domain and constrain the possible interpretation of terms". Ontology may be informal, for example, specified by catalogue of types that are definitions stated in a natural language, or formal, for example, specified by axioms and definitions stated in a formal language[1] (Sowa, 2000). Axiomatised ontology can be seen as an axiomatic concepts theory and provide the basis for conceptual modelling. Choice of a particular modelling approach for IS development always is linked to ontological assumptions about the modelled social reality (Humphrey and Feiler, 1992; Weber, 1997). Even generic system architecture (system domain) is "more suitable modelled as an ontology within which a properly parameterised architectural framework is defined" (Cornwell, 1999). A choice of ontological categories is the first step in designing a particular application as well as a family of applications (Smith, 2002). For development of single application in highly specialised domains, so-called microworlds, limited ontologies can be used. The principal advantage of limited ontology is easy of design and implementation. However, it is difficult to reuse such ontology in other microworlds. So, domain engineering requires generic ontologies that can be reused in many microworlds. Enterprise engineering requires even

[1]In philosophy the term "formal ontology" has other meaning. A division is made between formal and material (or regional) ontology. Formal ontology is domain-neutral; it deals with those aspects of reality, which are shared in common by all domains (material regions). Material ontology deals with those features, which are specific to given domains (Smith, 2002).

more generic shared ontologies that can support applications across many domains, more exactly, across all areas of enterprise's business (Czarnecki and Eisenecker, 2000; Sherif, 1997).

Although theoretically ontologies can play an important role in enterprise and domain engineering, it is not so simple to use them practically.

First of all, it is slightly complicated to insert ontology-driven approach in domain engineering process because of object-oriented nature of domain engineering. Concepts defined by ontology should be mapped to classes or, in other words, object-oriented generic architecture should be derived from domain ontology. As pointed out Czarnecki and Eisenecker (Czarnecki and Eisenecker, 2000), concepts and object-oriented classes have several important distinguishes. Concepts define intensional sets, classes define extensional sets. Objects are more specific as instances of concept. They are aggregates of descriptive, operational and organisational properties. Objects have identity, state, and well-defined behaviour. They can interact. Instances of concepts do not have such properties. Concepts are defined on the basis of other concepts. They are aggregates of features, which describe qualitative properties of concepts. Concepts are not stable. They evolve over time. The meaning of the concept varies from person to person. It depends also on the particular context. So it is impossible to map concepts to classes directly. The problem is complicated also by so-called minimum ontological commitments criterion, or, in other words, by the requirement that "an ontology should make as few claims as possible about the world being modelled, allowing the parties committed to the ontology freedom to specialise and instantiated the ontology as needed" (Gruber, 1993). Application of this criterion, in general, causes that domain ontology is too abstract to be directly reused in software development. In addition one must deal with common problems of conceptual design: some concepts can be better mapped to attributes; some concepts should not be mapped to architecture because they are important only in business but not in the system or have been defined only to clarify some aspects of ontology; solution-specific classes that have any counterparts in domain ontology should be added to architecture; etc.

Second, to insert ontology-driven modelling into enterprise engineering process, domain ontologies must be embedded within a more general framework. In other words, they must be based on upper-level ontologies that form the superstructure to define lower-level categories (Sherif, 1997). The question, how to define this superstructure and which upper-level ontology it is better to use as a base, is still open.

3.2. *An Ontological Approach to Domain Engineering*

An ontology-based approach (ODE approach) to domain engineering has been proposed by Falbo, Guizzardi, and Duarte (Falbo *et al.*, 2002). The main philosophy beyond this approach is to use domain ontology as domain model and to derive object framework for them. In ODE domain analysis is considered as ontology development phase, domain design as mapping of the ontology to object model (infrastructure specification), and domain design as Java components (infrastructure implementation) development phase. ODE provides integration of the ontology under development with previously developed more general (modular) ontologies. It allows wider reuse of domain knowledge.

Domain ontology in ODE should be explicitly represented in a formal language. Formal axioms that constrain the interpretation of the domain terms should be written. ODE approach uses four kinds of axioms: epistemological, consolidation, ontological, and definitions. Epistemological axioms describe the structure of the concepts (whole-part relations, is-a relations, etc.), consolidation axioms impose constraints for relations, and ontological axioms represent basic declarative knowledge. Constraints mostly define preconditions that must be satisfied for a relation to be consistently established. Definition axioms are used to define new concepts and relations. First-order logic is used to specify the axioms.

Ontologies also should be represented graphically. The LINGO language (Falbo *et al.*, 2002) is used for this aim. LINGO defines the basic notations (concepts and relations) to represent a domain conceptualisation. Some types of relations have predefined semantic. This enables automatically generate the epistemological axioms. Authors see LINGO as meta-ontology. In our opinion it should be seen rather as a higher-level ontology used to define domain-specific concepts.

To derive objects from ontologies a set of directives, design patterns, and transformation rules is used in ODE approach. In general, directives guide the mapping from the epistemological structures to classes, design patterns are used to map ontological axioms, and transformation rules are used to map consolidation axioms. However, some exceptions bias general rules. For example, whole-part design pattern is used to map whole-part relations, defined by epistemological axioms.

A formalism that lies at an intermediate abstraction level, between first-order logic and object-oriented language, is used as intermediate formalism based, predominantly, on set theory. The primitives of this formalism are related to the constructs of LINGO language. Computational structure is generated in two steps. First step translates formal ontology to an epistemological conceptual model (conceptual view of class diagram) represented using intermediate formalism. Second step translates conceptual model to object-oriented design. So, intermediate formalism bridges conceptual and implementation levels, respectively represented by first-order logic axioms and object models.

The main shortcoming of ODE approach is that it is highly focused on structural aspects and, consequently, should be extended by the techniques to address the dynamic aspects of domain.

3.3. *Ontologies in Enterprise Engineering*

Several attempts have been done to incorporate ontologies in EE process. An integrated system of ontologies to support enterprise modelling has been developed in Toronto Virtual Enterprise (TOVE) project (Fox, 1992). This system consists of a number of generic core ontologies, including an activity ontology (Fox and Grüninger, 1994), resource ontology (Fadel *et al.*, 1994), organisation ontology (Fox *et al.*, 1995), and a product ontology (Baid, 1994; Lin *et al.*, 1996). It also includes a set of extensions to these generic ontologies that define additional concepts (cost, quality, etc.). Recently the attempt has been done to develop an additional business process ontology (Grüninger *et al.*, 2000).

The system of ontologies in TOVE project has been designed and implemented using the logical approach. Ontologies are described using formal language. Alphabet of the language consists of logical and non-logical symbols. Logical symbols (connectives, variables, quantifiers) are defined by lexicon of Knowledge Interchange Format (Genesereth and Fikes, 1992). Non-logical symbols (constants, functions, relations) refer to concepts in appropriate domain. The model theory provides for each ontology a rigorous mathematical characterisation of the terminology. The proof theory provides axioms for the interpretation of terms in the ontology. Such approach allows a precise and rigorous characterisation of the consistency and completeness of the ontology with respect to its intended application and supports inference for enterprise model. An enterprise in TOVE approach is seen as an engineering artefact, which is conceived, designed and then implemented, using specific methods and tools. It is modelled as a set of logical constraints represented using the TOVE system of ontologies. More exactly, an enterprise model is thought as a set of constraints on activities, resources required by activities, organisation, goals, products, and services. Constraints are represented as sentences in first-order logic that are formulated in terms of appropriate ontology. Enterprise engineering problems are formulated as constraint satisfaction or logical entailment problems.

The TOVE approach is criticised that it requires too many efforts to instantiate the model for a particular enterprise. In order to easy this task, the authors are working on the development of the automation tools.

Another important contribution to ontology-based EE has been done in The Enterprise Project at the University of Edinburgh (Uschold *et al.*, 1996). In this project ontology (the Enterprise Ontology) for enterprise modelling that is intended to support an EE environment has been developed. EE environment is thought as an integrated collection of methods and tools for capturing and analysing key aspects of an enterprise. The Enterprise Ontology provides five top-level classes for integrating the various aspects of an enterprise: meta-ontology, activities and processes, organisation, strategy, and marketing. Meta-ontology defines basic modelling concepts (entity, relationship, role, actor, state of affairs). So, the Enterprise Ontology also can be seen as a system of integrated ontologies. It is semi-formal. Terms are expressed in a restricted and structured form of natural language supplemented with a few formal axioms using Ontolingua. Consequently, it does not support automated reasoning.

4. Reusing Enterprise Engineering Assets: Problem Statement

The results of the analysis presented in above sections demonstrate that the current approaches to the reuse of enterprise engineering assets are not mature enough. They suffer from the number of shortcomings:

- the lack of the theoretical backgrounds how to construct superstructure used to define lower level ontologies in problem as well as in system domain;
- insufficient coverage of modelling views, especially in system domain, that is serious obstacle to develop many reusable assets (information architecture, data architecture, network architecture, etc.);

- significant semantic gap between the modelling of business processes, information processes, and applications;
- strong orientation of domain engineering techniques towards the reuse of software solutions and, as a result, inability to model commonalities and variabilities at business and information processing levels;
- too abstract level of models and, as a result, necessity to perform too many manual operations in customising and instantiating reusable design;
- too strong emphasis on structural aspects, and, as a result, inability to reuse dynamic aspects, especially at business and information processing levels;
- monolithic structure of problem models and, as a result, inability reuse assets related to vertical domains (e.g., generic processes) in different enterprise information systems.

In order to develop the approach free from above listed shortcomings, it is necessary to solve a number of problems. One of the most important problems is the separation of concerns. First of all it is required to separate problem and system concerns (Fig. 3) or, in other words, customer and designer perspectives. It may be done separating problem and system ontologies (Caplinskas *et al.*, 2002a; Caplinskas *et al.*, 2002b). System ontology should allow to describe all levels of the enterprise information system architecture in uniform manner. It is itself a difficult and complicated task. This task becomes even more complicated because of possibility to conceptualise the system domain in different ways.

Further, domain and task knowledge inside the problem domain should be separated. This can be done by replacing the monolithic structure of problem model with a layered modular one (Fig. 4). In such structure upper-level ontology introduces generic concepts that are shared by lower-level ontologies and reflect underlying theory about the nature of enterprise's social reality (discourse of interest). This ontology is functional or, in other words, designed for specific purposes of enterprise information system design. Concepts defined by upper-level ontology should be shared across domains (i.e., specialised area of expertise) including across all areas of enterprise's business (application domains), process domains, problem domains, and system domain.

In this framework application domains are seen as horizontal domains. It means that a number of processes including business processes can be located in the domain. In other words, domain knowledge and process knowledge is separated. Generic process is thought as an abstraction used to describe the family of particular process (process in-

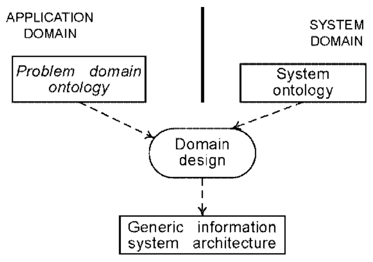

Fig. 3. Ontologies in enterprise engineering.

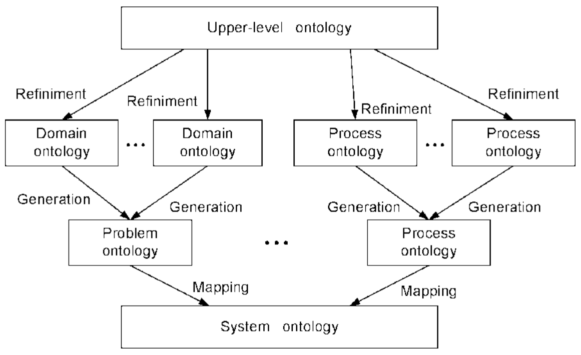

Fig. 4. Ontological framework.

stances). Process ontology introduces specific terms required to describe certain generic process including task structure, artefacts, resources, roles, enacting mechanisms, preconditions, and constraints. Collection of process ontologies specifies the behavioural view on enterprise systems. Finally, a problem is defined as a process, located in a particular application domain. When process is located in a particular application domain, entities from domain should be assigned to the process roles. A single entity may perform multiple roles and a single role may be performed by multiple entities. It is true in case of active as well as in case of passive roles. Process placing in application domain is one of the set of activities that should be performed in the domain design stage.

Second important problem is the development of approaches to address all aspects of enterprise, including dynamic ones. In order to do it, domain engineering techniques should be extended in a way that these techniques could be applied to the whole enterprise model and to develop appropriate directives, design patterns, and transformations, required to derive additional assets.

Finally, application engineering techniques must be enhanced in order automate design customisation and instantiation. One of the possible solutions is the development of generative methods not only for software components development but for the whole software system developing as well.

5. Conclusions

The paper analyses the problems related to reuse of enterprise engineering assets. The state of affairs in rough can be characterised as follows: reuse techniques in this field are at the very beginning. They focuses on reuse of business models and as a rule are ontology-based. On the other hand, domain engineering techniques became mature enough and are widely used in practice. The main challenge is to combine domain engineering techniques with ontology-based modelling and to extend these combined techniques to cover all aspects of enterprise.

The main problems that must be solved to realise this programme are as follows:

- development of shared ontologies that should support across all areas of enterprise's business;

- development of upper-level ontologies that should form the superstructure to define lower-level categories;
- development of integrated set of ontologies including application domain ontologies, process ontologies, problem ontologies, and a system ontology;
- development of reuse techniques for all enterprise engineering assets including data and information architectures, product line definitions, network architectures, etc.

References

Agarwal, R., B. Ghosh, A. Sarangi (1999). Designing reliable software using DAF. In *Proceedings of the Fast Abstracts and Industrial Practices of the IEEE 10th International Symposium on Software Reliability Engineering*. ISSRE Boca Raton, Florida, November 2–5.
http://www.chillarege.com/fastabstracts/issre99/99104.pdf.

Allen, P., S. Frost (1998). *Component-Based Development for Enterprise Systems: Applying the SELECT Perspective*^{TM}. Cambridge University Press, SIGS Books, Cambridge.

ANSI (1996). *Business Rules for the Enterprise Viewpoint of RM-ODP*. ANSI X3H7-96-07R2, 7 December.

Arkin, A. (2002). *Business Modelling Language*. BPMI.org.
http://www.bpmi.org/bpml-spec.esp.

Baid, N. (1994) *Toward an Ontology of Change for Concurrent Engineering*. M.A. Sc. Department of Industrial Engineering, University of Toronto.

Baxter, I.D. (1999). Transformation systems: domain-oriented component and implementation knowledge reuse. In 9th *Annual Workshop on Institutionalizing Software Reuse (WISR9)*. The University of Texas at Austin, USA, January 7–9, 1999. Full Position Papers.

Bernus, P., L. Nemes (1996). Enterprise integration – engineering tools for designing purposes. In P. Bernus, L. Nemes (Eds.), *Modelling and Methodologies for Enterprise Integration*. Chapman & Hall. pp. 1–9.

Borst, P. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD Thesis, University of Twente, Enschede, The Netherlands.

Brash, D. (2001). Reuse in information systems development: classification and comparison. In *Proceedings of the Twelfth Australasian Conference on Information Systems*. Coffs Harbour, NSW, Australia, December 5–7, 2001. Conference Proceedings CD.
http://infotech.scu.edu.au/ACIS2001/Proceedings/PDFS/14.pdf.

Caplinskas, A., A. Lupeikiene, O. Vasilecas (2002a). Unified enterprise engineering environment: ontological point of view. In H.-M. Haav, A. Kalja (Eds.), *Proceedings of the Baltic Conference, BalticDB&IS 2002*, vol. 1. Institute of Cybernetics at Tallinn Technical University. pp. 30–50.

Caplinskas, A., A. Lupeikiene, O. Vasilecas (2002b). Shared conceptualisation of business systems, information systems and supporting software. In H.-M. Haav, A. Kalja (Eds.), *Databases and Information Systems II, Selected Papers from the Fifth International Baltic Conference, BalticDB&IS'2002*. Kluwer Academic Publishers. pp. 109–320.

Cohen, S. (1997a). Object technology, architectures, and domain analysis – what's the connection? In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Position Papers.
http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.

Cohen, S. (1997b). Object technology, architectures, and domain analysis – what's the connection? Is there a connection? In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Working Group Reports.
http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.

Cohen, S. (1999). From use cases to domains and architecture. In 9th *Annual Workshop on Institutionalizing Software Reuse (WISR9)*. The University of Texas at Austin, USA, January 7–9, 1999. Full Position Papers.
http://www.opengroup.org/combine/documents.htm.

COMBINE (2003). *COMponent-Based INteroperable Enterprise System Development*. D32.1 – COMBINE Green Paper, Version 2.2 31/01/2003.

Cornwell, P. (1999). Scoping the task and application domain for knowledge acquisition. In 9th *Annual Workshop on Institutionalizing Software Reuse (WISR9)*. The University of Texas at Austin, USA, January 7–9,

1999. Full Position Papers.
http://www.opengroup.org/combine/documents.htm.

Czarnecki, K. (1997). Leveraging reuse through domain-specific software architectures. In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Position Papers.
http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.

Czarnecki, K., and W. Eisenecker (2000). *Generative Programming: Methods, Tools, and Applications*. Addison–Wesley.

Dietsch, A. (2002). Adapting the UML to business modelling's needs: experiences in situational method engineering. In *UML 2002*, *Lecture Notes in Computer Science*, **2460**, Springer–Verlag. pp. 73–83.

Dobson, J., and M. Martin (1997). The ontology of enterprise and information systems. In *Third Americas Conference on Information Systems in Indianapolis*. Indiana, August 15–17, 1997, online papers.
http://hsb.baylor.edu/ramsower/ais.ac.97/papers/dobson.htm.

Eichman, D. (1997). Representing knowledge in domain engineering. In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Position Papers.
http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.

Fadel, F., M.S. Fox, M. Grüninger (1994). A generic enterprise resource ontology. In *Proceedings of the Third Workshop on Enabling Technologies – Infrastructures for Collaborative Enterprises*. West Virginia University.

Falbo, R., G. Guizzardi, K.C. Duarte (2002). An ontological approach to domain engineering. In *Proceedings of the XIV International Conference on Software Engineering and Knowledge Engineering (SEKE-2002)*. Ischia, Italy, 2002. ACM Press.

Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Addison–Wesley.

Fox, M.S. (1992). The TOVE project: a common-sense model of the enterprise, industrial and engineering applications of artificial intelligence and expert systems. In F. Belli and F.J. Radermacher (Eds.), *Lecture Notes in Artificial Intelligence*, **604**. Springer–Verlag. pp. 25–34.

Fox, M.S., and M. Grüninger (1994). Ontologies for enterprise integration. In *Proceedings of the 2nd Conference on Cooperative Information Systems*. Toronto, Ontario.

Fox, M.S., M. Barbeceanu, M. Grüninger (1995). An organization ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. *Computers in Industry*, **29**, 123–134.

Frakes, B. (1997). Automating domain engineering. In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Position Papers.
http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.

Gamma, E., R. Helm, R. Johnson, J. Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison–Wesley.

Genesereth, M., and R. Fikes (1992). *Knowledge Interchange Format Reference Manual*. Report Logic-92-1, Department of Computer Science, Stanford University.

Gruber, T. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, **5**(2), 199–220.

Grüninger, M., K. Atefi, M.S. Fox (2000). Ontologies to support process integration in enterprise engineering. *Computational & Mathematical Organization Theory*, **6**, 381–394.

Hakimpour, F., and A. Geppert (2002). Global schema generation using formal ontologies. In S. Spaccapietra, S.T. March, Y. Kambayashi (Eds.), *Conceptual Modeling – ER 2002*. 21st *International Conference on Conceptual Modeling*. Tampere, Finland, October 2002, Proceedings. Lecture Notes in Computer Science, **2503**. Springer–Verlag, Berlin, Heidelberg, New York. pp. 307–321.

Humphrey, W.S., and P.H. Feiler (1992). *Software Process Development and Enactment: Concepts and Definitions*. Technical Report SEI-92-TR-4, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Jacobson, I., M.L. Griss, P. Jonnson (1997). *Software Reuse: Architecture, Process and Organization for Business Success*. Addison–Wesley Longman, New York.

Johnson, R. (1997). Frameworks = (components + patterns). *Communications of the ACM*, **40**(10), 39–42.

Johnstone, M.N., and D.C. McDermid (2001). Using ontological ideas to facilitate the comparison of requirements elicitation methods. In *Proceedings of the Twelfth Australasian Conference on Information Systems*. Coffs Harbour, NSW, Australia, December 5–7, 2001. Conference Proceedings CD.
http://infotech.scu.edu.au/ACIS2001/Proceedings/PDFS/46b.pdf.

Jones, N.D., C.K. Gomard, P. Sestoft (1993). *Partial Evaluation and Automatic Program Generation*. Prentice Hall International.

Kang, K.C. *et al.* (1990). *Feature-Oriented Domain Analysis (FODA)*. Technical report, CMU/SEI-90-TR-21, Software Engineering Institute, Pittsburgh, PA.

Liles, D.H., M.E. Johnson, L. Meade (1998). *The Enterprise Engineering Discipline*. Enterprise Engineering Homepage, Automation & Robotics Research Institute.
`http://arri.uta.edu/eif/`.

Lin, J., M.S. Fox, T. Bilgic (1996). A requirement ontology for engineering design. *Concurrent Engineering: Research and Applications*, **4**(4), 279–291.

Maccario, P.M. (1997). The domain analysis integrated in an object oriented development methodology. In 8th *Annual Workshop on Institutionalizing Software Reuse (WISR8)*. Ohio State University, USA, March 23–26, 1997. The WISR 8 Position Papers.
`http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html.`

Masuhara, H. (1999). *Architecture Design and Compilation Techniques Using Partial Evaluation in Reflective Concurrent Object-Oriented Languages*. PhD Thesis. University of Tokyo.

Mayer, R., Ch. Menzel, M. Painter, P. Witte, T. Blinn, B. Perakath (1995). *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*. Knowledge Based Systems Inc. (KBSI), September 1995.
`http://www.idef.com/Downloads/pdf/Idef3_fn.pdf`.

Meerts, J. (2002). *IT Management and Strategy: Enterprise Engineering White Paper*. Lessius Hogeschool, Antwerpen.
`http://www.zwelgjes.be/files/2lic/ICTMgm/Enterprise_Engineering_V1R1.pdf`.

NIST (1993). *Integration Definition for Function Modelling (IDEF0)*. National Institute of Standards and Technology.

Robak, S. (2002). Developing software families. In J. Desel, M. Weske (Eds.), *Proceedings of the PROMISE 2002 Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*. Hasso–Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, October 9–11.
`http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-65/11robak.pdf.`

Robertson, S., and J. Robertson (1999). *Mastering the Requirements Process*. Addison–Wesley.

Sherif, K. (1997). Software artifact reuse: a domain engineering approach. In *Third Americas Conference on Information Systems in Indianapolis*. Indiana, August 15–17, 1997.
`http://hsb.baylor.edu/ramsower/ais.ac.97/papers/sherif.htm.`

Smith, B. (2002). *Ontology and Information Systems*. *Stanford Encyclopedia of Philosophy.*
`http://ontology.buffalo.edu/smith//articles/ontologies.htm.`

Sowa, J.F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks/Cole, Thomson Learning, Pacific Grove, CA.

Štuikys, V., and R. Damaševičius (2002). Relationship model of abstractions used for developing domain generators. *Informatica*, **13**(1), 111–128.

Uschold, M. (1998). Knowledge level modelling: concepts and terminology. *The Knowledge Engineering Review*, **13**(1), 5–29.

Uschold, M., M. King, S. Moralee, Y. Zorgios (1996). The enterprise ontology. *Knowledge Engineering Review*, **13**(1), 31–90.

Van Belle, J.-P. (1996). A critique of current system engineering methods: the case for ontology-augmented methodologies. In K. Siau and W. Wand (Eds.), *Proceedings of the Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Crete, Greece.

Wand, Y., and R. Weber (1990). An ontological model of an information system. *IEEE Transactions on Software Engineering*, **16**(11), 1282–1292.

Weber, R. (1997). The link between data modeling approaches and philosophical assumptions: a critique. In *Third Americas Conference on Information Systems in Indianapolis*. Indiana, August 15–17, 1997. Online conference papers.
`http://hsb.baylor.edu/ramsower/ais.ac.97/papers/weber.htm.`

Weiss, D.M., and C.T.R. Lai (1999). *Software Product-Line Engineering: A Family Based Software Development Process*. Addison–Wesley, New York.

A. Čaplinskas is a principal researcher and a head of the Software Engineering Department at the Institute of Mathematics and Informatics in Vilnius, Lithuania. He also teaches as a part-time professor at the Vilnius University and at the Vilnius Gediminas Technical University. A. Čaplinskas graduated in mathematics from the Moscow Lomonosov State University. He defended his doctor thesis in computer sciences at the Vytautas Magnus Dux University (Kaunas, Lithuania). He is author of two and co-author of two books published over hundred research papers in information system engineering, software engineering, legislative engineering, knowledge engineering and related areas. A. Čaplinskas is member of AIS and of the Lithuanian Computer Society, permanently works in programming committees of different international and local conferences.

A. Lupeikienė is a researcher at the Institute of Mathematics and Informatics in Vilnius, Lithuania. She also teaches as a part-time assoc. professor at the Vilnius Gediminas Technical University. A. Lupeikiene received doctor degree from Vilnius Gediminas Technical University in 1999. Her research interests include information system engineering, software engineering, knowledge engineering. A. Lupeikiene is member of ECCAI and of the Lithuanian Computer Society.

O. Vasilecas is an associated professor in Information systems Department and a principal researcher and a head of the Information Systems Scientific Laboratory at Vilnius Gediminas Technical University. He also teaches as a part-time professor at the Informatics Department of Klaipeda University. O. Vasilecas defended his doctor thesis in physics and mathematics sciences at the Vilnius University. He is author and co-author of two books and published over one hundred twenty research papers in business, information and software systems engineering, knowledge engineering, intelligent information systems; process mathematical modelling and related areas. O. Vasilecas is member of ACM, AIS, and Lithuanian Computer Society, permanently works in programming committees of different international and local conferences. He is co-ordinator of FP5 and SOCRATES/ERASMUS projects.

Ontologijų vaidmuo pakartotinai panaudojant organizacijų ir dalykinių sričių inžinerijos artefaktus

Albertas ČAPLINSKAS, Audronė LUPEIKIENĖ, Olegas VASILECAS

Pagrindinis straipsnio tikslas yra palyginti ontologijų panaudojimu grindžiamas pakartotinio panaudojimo technikas organizacijų inžinerijoje ir dalykinių sričių inžinerijoje. Straipsnyje nagrinėjami bandymai apjungti klasikines domenų inžinerijos technikas su ontologijų panaudojimu grindžiamomis technikomis ir bandymai integruoti ontologijas į organizacijų inžinerijos procesus. Parodoma, kad, viena vertus, aptariamos technikos nėra pakankamai išvystytos, kad leistų spręsti praktines pakartotinio panaudojimo problemas, tačiau, antra vertus, pasiūlytos idėjos gali būti realiai panaudotos tokioms technikoms sukurti. Pagrindinis straipsnyje pateikiamas rezultatas yra problemų, kurias reikia spręsti norint sukurti ontologijų panaudojimu grindžiamą organizacijos inžinerijos artefaktų pakartotinio panaudojimo metodą, išvardijimas ir detalizavimas.