INFORMATICA, 2005, Vol. 16, No. 2, 175–192

Approach to Enterprise Modelling for Information Systems Engineering

Saulius GUDAS, Audrius LOPATA
Information Systems Department, Kaunas University of Technology
Studentų 50, Kaunas, Lithuania
Kaunas Faculty of Humanities, Vilnius University
Muitinės 8, Kaunas, Lithuania
e-mail: gudas@soften.ktu.lt, audrius.lopata@if.ktu.lt

Tomas SKERSYS
Information Systems Department, Kaunas University of Technology
Studentų 50, Kaunas, Lithuania
e-mail: skertoma@pit.ktu.lt

Received: December 2003

Abstract. The paper deals with Knowledge-based Information Systems (IS) engineering. The Enterprise management functions, processes and their interactions are considered as the major components of the domain knowledge. This is the peculiarity of this approach to Enterprise modelling for IS engineering. The resulting framework for Enterprise modelling and Knowledge-based IS engineering – Enterprise meta-model (EMM) – is developed and presented in this paper. The architecture of the advanced CASE system is also discussed in this paper.

Key words: information systems engineering, CASE system, knowledge-based, enterprise meta-model.

1. Introduction

Nowadays efficient Information Systems (IS) engineering and Enterprise modelling are directly interrelated issues. Enterprise modelling is treated as a source of Enterprise knowledge that adds value to the business process and also influences methods of IS engineering. Some of the IS engineering approaches involve Enterprise models as a source of structured knowledge about the real world (business domain) in IS development life cycle stages, such as user requirement analysis and specification, development of detailed IS project solutions and other.

There is a great number of Enterprise modelling methods and approaches (such as CIMOSA, GERAM, IDEF suite, GRAI) (Schekkerman, 2003), standards (ISO 14258, ISO 15704, PSL, ISO TR 10314, CEN EN 12204, CEN 40003) and supporting Enterprise modelling tools (Totland, 1997). Moreover, CASE tools, that appear in contemporary market and are intended for the IS development, include graphical editors for business process modelling and analysis techniques.

Different types of Enterprise models became the main component of the CASE systems; among most widely used models one can distinguish dataflow diagram, workflow diagram, organization (hierarchy) diagram, process (hierarchy) diagram, business goal's diagram, business interaction diagram, UML and IDEF diagram sets, etc. The problem is that these diagrams are weakly integrated, which leads to the inconsistency of the whole IS development life cycle and cannot give expected results.

Therefore, in order to reach a higher degree of integration between IS engineering systems and Enterprise modelling techniques, one requires a common logical structure for classifying and organizing the descriptive representations (i.e. models) of an Enterprise that are significant to the management of the Enterprise as well as to the development of the Enterprise systems including Information Systems (Persson, 2001).

To achieve this goal one must develop a normalized definition of the Enterprise meta-model that would be a generalized structure of integrated core constructs, selected from different Enterprise modelling approaches and methodologies. Some definite results in this area are developed by the IFAC-IFIP Task Force, which works on the Architecture for Enterprise Integration. The goal of this Task Force is the development of the Unified Enterprise Modelling Language (UEML, 1999).

Object Management Group (OMG) has also proposed principles for the standardization of a model-driven process of IS engineering that could potentially incorporate Enterprise modelling constructs (Stephen and Kendall, 2004). The implementation of MDA technology in UML-based approaches, that are capable to process Enterprise modelling activities, is highly desirable. However, the UML itself does not satisfy the needs and requirements for the domain knowledge modelling in the area of IS engineering. Information Systems design languages require business-specific constructs and the Enterprise meta-model (accepted by users as business domain experts and IS developers) from which the particular models of specific business domain could be mapped.

Both MDA approach and the UEML constructs are aimed at the integration of the Enterprise modelling and the processes of Information Systems engineering. Nevertheless, this paper concentrates more on the UEML as it is particularly aimed at the unification of the Enterprise modelling approaches and is certainly important in the enhancement of IS engineering methods and tools.

It should be mentioned that Enterprise modelling introduces one important concept – that is Knowledge-based Enterprise – which expands the horizon of the Information Systems development itself.

The Knowledge-based Enterprise from the IT point of view has to be connected to the shared Enterprise Repository that represents the Enterprise Knowledge Base common for both business Enterprise and IS engineering activities. From the perspective of IT, the unified framework for the Enterprise modelling and IS development is a background that warrants relevant technology solutions, delivered to the business (Popkin, 2002).

Enterprise models and languages mentioned above do not satisfy the needs and requirements for Enterprise modelling in the area of IS engineering. The paper deals with a formal framework aimed at the development of the Enterprise Knowledge Base. This formal structure is called an Enterprise meta-model (EMM).

2. The UEML Constructs

One of the attempts to integrate approaches of Enterprise modelling is Unified Enterprise Modelling Language (UEML). Fig. 1 depicts basic constructs of the UEML core (UEML, 1999). The UEML core assumes the CEN ENV 12204 (ENV 12204, 1996), CEN ENV 40003 (ENV 40003, 1990), also Enterprise-modelling standards and languages such as IDEF, OMT, UML, CIMOSA, ARIS (Vernadat, 2001; UEML, 1999; Lopata, 2002).

Constructs of the UEML core (UEML, 1999) are as follows:

$$\begin{aligned}\textbf{UEMLcore} = \{&\mathit{Objects}(\mathit{Event, Time, Agent, Process, Activity, Function},\\ &\mathit{Input\ Object, Output\ Object, Environment});\\ &\mathit{Relations\ (Raises, Triggers, Produces},\\ &\mathit{Happens\ at, Made\ of, Performs, Used\ by})\}.\end{aligned}$$

The refinement and definition of the concepts *Process* and *Function* is one of the background points in Enterprise modelling. The UEML core includes *Function* and *Process* as separate constructs of Enterprise model, yet the collaboration (interaction) of the *Activity, Process* and *Function* is not defined in UEML (UEML, 1999). Fig. 1 depicts that the construct *Function* includes the construct *Process*, and the construct *Process* consists of a set of *Activities*. The transactions (information flow) among *Activity, Process* and *Function* are also not declared in this UEML core.

The next UEML version (UEML 1.0) involves a wider set of the constructs (Vernadat, 2001). The formalized description of the UEML 1.0 is as follows:

$$\begin{aligned}\textbf{UEML\_model} = \{&\mathit{UEML\_Object}(\mathit{UEML\_model}, \mathit{Object}(\mathit{Information\ Object},\\ &\mathit{Resource}(\mathit{HumanResource, MaterialResource})), \mathit{Port}(\mathit{ResourceRole},\\ &\mathit{Anchor}(\mathit{InputPort, OutputPort, ConnectionOperator}), \mathit{Geometry}),\end{aligned}$$

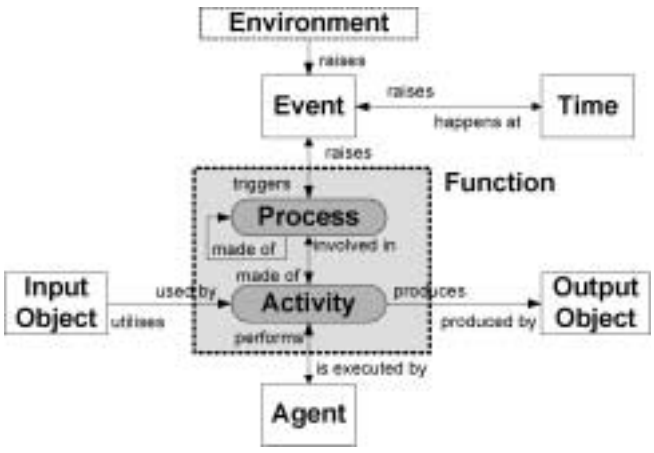

Fig. 1. The enterprise meta-model according to the UEML core.

Flow(*IOFlow, ResourceFlow, ControlFlow*(*TriggerFlow, ConstraintFlow*)),
Activity(*InputPort, OutputPort*)), *Relations*(*Precedence relation, HasIP, HasOP*,
Contains, IOcarried, Object_carried, Resource_carried)}.

It must be pointed out that this version of UEML specifies only two essential constructs for Enterprise process-oriented modelling, namely, *Activity* and *Flow.* Meanwhile here are no *Process* and *Function* constructs in the UEML 1.0 core. We can only assume that those two Enterprise modelling constructs are hidden under the construct *Activity*. The construct *Event* is also omitted in the UEML 1.0 core, though it was presented in the earlier version of the UEML.

It is neither clear which constructs support decision-making mechanism in the UEML 1.0 core. Though, it could be assumed that such decision-making mechanism should be modelled because a control flow (*ControlFlow)* – the output of decision-making – is declared as the construct of the UEML 1.0 core.

Both UEML versions are comprised of only two types of modelling constructs – objects (or entities) and relationships between objects (semantic relationships). Thus only the static structure of the Enterprise is taken into consideration while dynamic aspects of the Enterprise are still left behind. The sequence of transactions among Enterprise constructs is neither included.

3. One more Aspect of Enterprise Modelling

Enterprise modelling usually involves the definite set of aspects: function, behaviour, information, resource and organization (Vernadat, 2001; GERAM, 1999). In addition, it is possible to distinguish one more aspect of an Enterprise modelling, defined as *the management point of view*. From this point of view the major Enterprise modelling constructs are identified. In the management control systems literature similar aspect is called *the management control perspective* (Anthony, 2003; Drury, 2001; Merchant, 1997). The management control perspective focuses on the organisation management issues; meanwhile the particularity of *the management point of view* is the refinement of information processing constructs and their interactions in the Enterprise.

Some business systems are able to choose their own behaviour. Business processes in such systems are guided by the *decision-making mechanism*. That is why the modelling of *management process* and *management information flows* must be taken into account as mandatory aspects of Enterprise modelling. The scope of management process modelling is the internal structure of management information (Enterprise knowledge, data, objectives) and the management information processing as well.

It is claimed in Systems and Control Theory that a system can be controlled effectively only if some *feedback loops* (also called *control loops*) are implemented. Consequently, the components of the control loop should be included into Enterprise meta-model.

It should be pointed out that the term *Control flow* (in the sense of *workflow*) is associated with the concept *Activity* in the UEML 1.0 (Vernadat, 2001). However, the earlier

version of UEML core (UEML, 1999) includes separate modelling constructs *Function* and *Process*, and that makes this UEML core closer to the Enterprise modelling from the management point of view.

Further, the Control Theory defines the typical structure of a *System* – a real world System with internal "mechanism" of control. *A System* involves the following mandatory (complex) constructs: a real world *Process*, a *Control System* and a *Feedback Loop* which creates *an Information flow* (*Control flow*) between a *Process* and a *Control System* (Gupta and Sinha, 1996). *A Control System* performs a definite set of activities (*Functions*, related to a definite *criterion)* aimed to control *a Process*. Any *Function* takes (makes measurements of) *a Process state attributes*, calculates a *Process control attributes* and in that way influences the state of a *Process.*

Before we go further, let us define that any item (structural unit) of *a System* is named *an object. An object* could be conceptualised as *an entity* or *a class* (of the UML), or in some other way in accordance with particular modelling methodology.

Therefore, any *System* is a set of interdependent objects that interact regularly in order to perform a task. A *System* can be conceptualised in accordance with the above stated considerations (principles) from *the* management *point of view*:

a. A model *of a System* includes the following subsets of constructs: *a Process, a Function, a Control System*, and *a Feedback Loop*;
b. *A Process* consists of a partially ordered set of steps – *sub-processes* (or *stages*);
c. Any *Function* comprises two constructs (Fig. 2) – *a Control System* and a *Feedback Loop*;
d. *A Control System* comprises the following mandatory constructs: *Data Processing* and *Decision Making*; these two constructs are interrelated by *Data Flows* between them;
e. *A Feedback Loop* comprises two types of constructs: a construct for transferring data flow (*Process state attributes*) directed from *Process* to *Control System*, and a construct for transferring data flow (*Process control attributes*) from *Control System* to *Process*;
f. *A Feedback Loop* creates an interaction between *a Process* and *a Control System: a Control System* transforms a set of *Process state attributes* to a set of *Process*

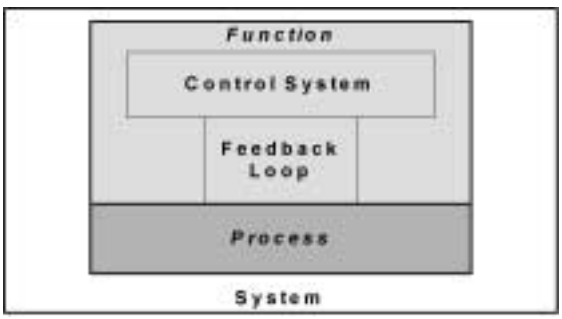

Fig. 2. Interaction of *Function* and *Process* in a *System*.

control attributes, and hereby influences *a Process* itself;

g. Any management *Function* is defined as a mandatory sequence of steps – *interactions* between the structural elements of *a Control System* and *Feedback Loop*. *An interaction* of any two constructs of *a Function* is an information transferring process.

Consequently, every single management *Function* in the Enterprise model is comprised of:

a) a definite set of mandatory structural elements that are defined as instances of meta-level elements *Control System* and *Feedback Loop*,
b) a definite set of relations, defined as ordered sequences of interactions between structural elements of the *Function*.

In summary, a management *Function* is one of the complex constructs of an Enterprise model, comprising objects of a *Control System* and *Feedback Loop*, and an ordered sequence of *interactions* between these objects.

There are few Enterprise models that contain some enumerated elements of *a System*, but not the complete set of interacting constructs: *Process*, *Function*, *Control System*, *Feedback Loop*. One of such models is Value Chain model that declares *Business Enterprise* as a mandatory interaction of the primary activities (Enterprise processes) and secondary activities (management functions) (Turban, 1999). The process management model of the Framework for Managing Process Improvement (DoD, 1994) should also be mentioned, as it makes a distinction between the concepts "*Function*" and "*Process*". *A process* here is defined as a unit of workflow through an Enterprise, and a *function* is a specified type of work applied to a product or service running within a process. *Function* sets the rules and controls the resources assigned to the *process* (activity).

The analysis of the Enterprise modelling from *the management point of view* gives some new aspects for the Enterprise modelling itself:

– The matter under investigation is a content of information, information processing and decision-making activities in the organizational system.
– It is aimed at the enhancement of the Enterprise model that can be used as a source of domain knowledge for business process analysis and IS development.
– A set of EMM core constructs and their types of relation should be revised *from the management point of view* (EMM core constructs are the mandatory elements of the meta-model in order to support computer-aided modelling of the Enterprise). These constructs should include a definite set of structural elements and relations for the modelling of management functions.

4. The Interaction of Function and Process

The interaction of Enterprise meta-model core elements *Process* and *Function* is formally assumed as a *Control Process*. It is defined as a *Feedback Loop* between *Process* $P(j)$ and *Function* $F(i)$. The analysis of the *Function-Process* interaction is a background of the formalized model of the organizational system (an Enterprise model) described

in (Gudas, 1991). Fig. 3 presents the basic components of the formalized model of the *Function-Process* interaction.

From the management point of view *Process* $P(j)$ is defined by two sets of attributes: a set of *Process state attributes* and a set of *Process control attributes*. A set of *Process state attributes* includes the *Process input* (material flow) *attributes*, *Process output* (material flow) *attributes* and the *attributes* of the particular *Process* $P(j)$.

A management *Function* consists of the predefined sequence of mandatory steps of information transformation (*Interpretation, Information Processing, Realization*); these steps compose a management cycle (a feedback loop). A definite set of attributes (a set of information items) is formed and transmitted during each management step. A management *Function F(i)* is initiated by some *Event* – a fact or a message associated with some internal or external (environmental) object. This definition of *Function* is close to the definition of function presented in (ENV 40003, 1990). This paper presents more detailed content of *Function F(i)* since it defines a sequence of definite types of interacting information activities (*Interpretation, Information Processing, Realization*) directed to control *Process* $P(j)$ (Fig. 4).

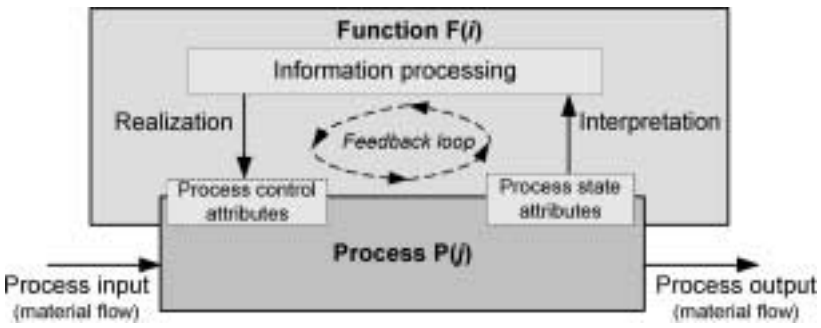

Fig. 3. The formalized model of the *Function-Process* interaction.

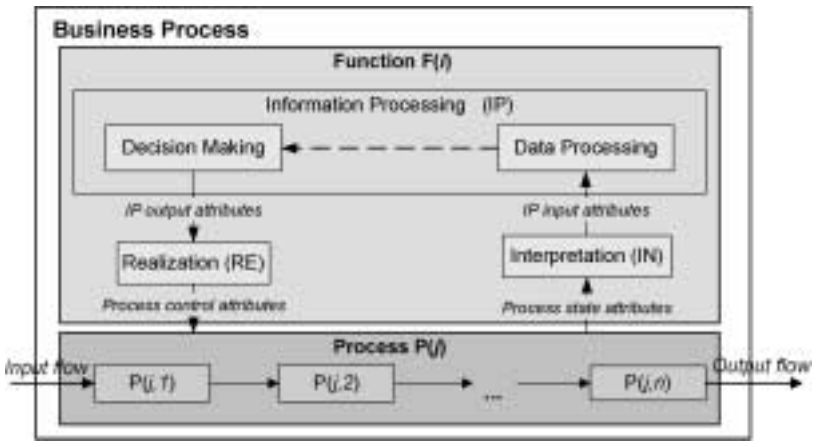

Fig. 4. The structured model of the *Function-Process* interaction.

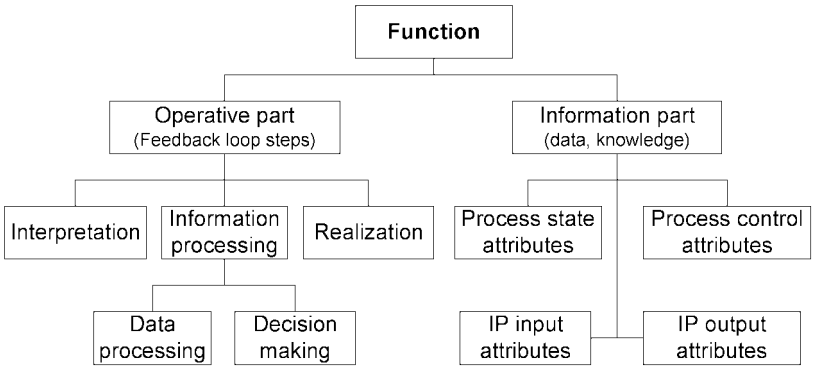

Fig. 5. Structure of the construct *Function*.

Fig. 4 presents the structured model of the *Function-Process* interaction. The concept *Process* is assumed as "a black box". The internal structure of *Process* is not important, only the information about the state of *Process* is considered from the point of view of some definite management *Function*. The concept *Process* (dark grey box in Fig. 3) is characterized by a set of *Process state attributes* (this set comprises subsets of *Input flow attributes*, *Output flow attributes* and *Process attributes*) and it is influenced by the output of management *Function* – a set of *Process control attributes*.

All other constructs of *Function-Process* interaction in the structured model (except constructs *Process*, *Input flow* and *Output flow*) are assumed to be the components of the construct *Function* – an Enterprise management function (Fig. 5).

It is assumed, that *Process* and management *Function* are activated by some *Event*. A definite set of *state attributes* of an activated *Process* is the information flow defined as an input of (one or more) specific management *Function* that is activated by some particular *Event*.

It should be pointed out, that the set of attributes of management *Function* is closely related to the description of function presented by CIMOSA (ENV 40003, 1990). The CIMOSA specification of function includes the structural part (the list of sub-functions is used), the functional part (goals, limitations, functional description, necessary equipment, input, output) and the part of an attitude (goals, limitations, procedural rules, events, end state).

5. The Enterprise Meta-Model

The Enterprise meta-model (EMM) is illustrated in Fig. 6. The Enterprise meta-model development process is supervised from the management point of view which is defined in Section 4. The Enterprise meta-model integrates common constructs of the Enterprise modelling standards and frameworks and new Enterprise modelling constructs developed from the management point of view.

The Enterprise meta-model is developed for the model-driven information systems engineering. It is supposed to be a formal structure for domain knowledge evaluation

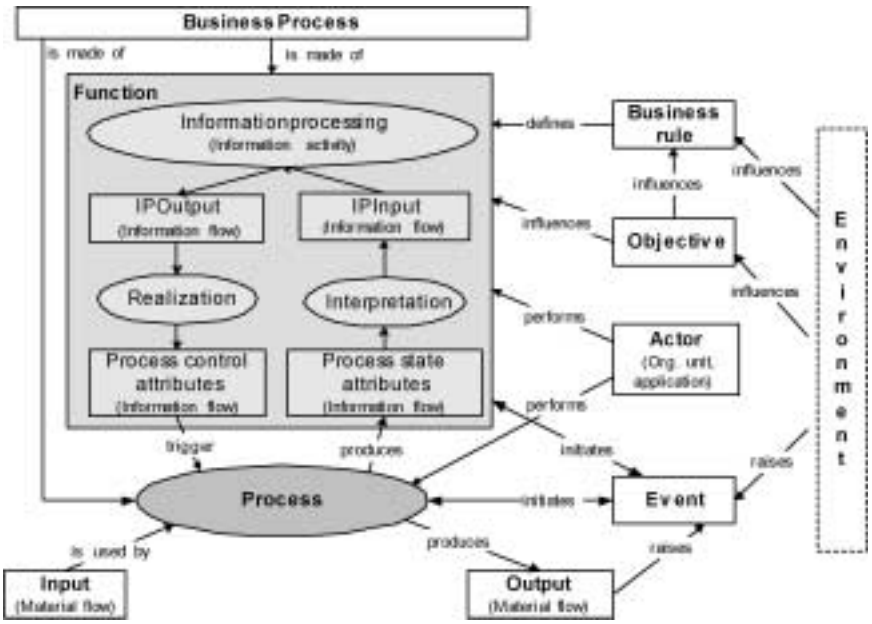

Fig. 6. The principle schema of the Enterprise meta-model (EMM).

during the process of Enterprise model development. This EMM-based Enterprise model should be used as the source of domain knowledge in all stages and steps of the IS development life cycle.

The EMM includes basic constructs for the Enterprise modelling identified from the management point of view: *Business Process, Function, Process, Actor, Business Rule, Event, Objective and Environment*.

Formalized description of the EMM is as follows:

$$EMM = \{\textit{EMM\_Object}(\textit{Business Process}(\textit{Function}(\textit{Process state attributes, Interpretation, IP input, Information processing, IP output, Realization, Process control attributes}), \textit{Process, Input, Output}), \textit{Business rules, Objectives, Actor, Event}), \textit{Relation}(\textit{Made of, Define, Influence, Performs, Initiates, Raises, Produces, Is used by, Trigger})\}.$$

The EMM core constructs are shown in Fig. 7 (UML notation). For the comparison purposes the composition of the UEML core (UEML, 1999) is shown in Fig. 8.

The components of the *Operative part* of the construct *Function* are information activities *Interpretation* (*IN*), *Realization* (*RE*) and *Information Processing* (*IP*). The *IN*, *IP* and *RE* are the mandatory steps of a management *Function*. *Interpretation* transforms *Process state attributes* (the input of the *IN*) in accordance with the requirements of the next step of the management cycle *Information Processing*. The result of *IN* is a control flow *IP input*. Control flow *IP input* is an input of the next step in the management cycle,

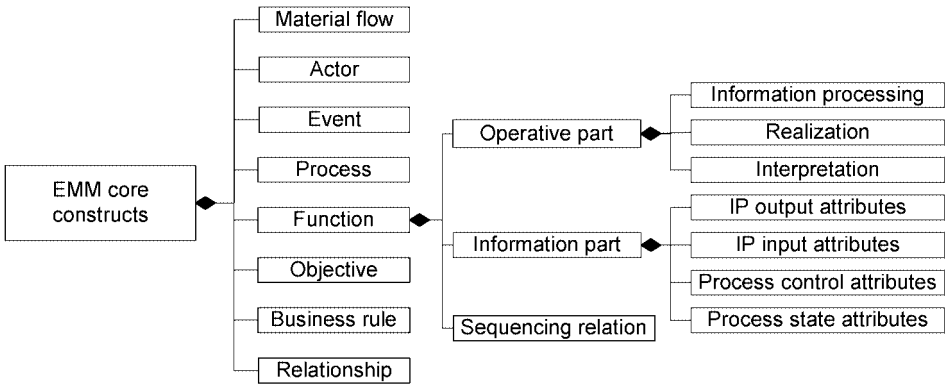

Fig. 7. Composition of the EMM core.

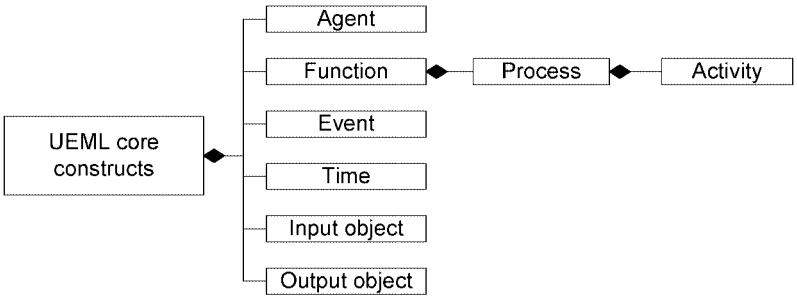

Fig. 8. Composition of the UEML core.

namely, *Information Processing*. The management step *Information Processing* includes data processing and decision-making procedures. The output of the information activity *IP* is a control flow *IP output. Realization* (*RE*) is the last step in the management cycle. The *RE* is the information process concerning the implementation of decision *IP output*, developed by the management step *IP*. The result of *RE* is *Process control attributes* – a set of attributes, aimed to influence the state of *Process*.

The constructs *Event, Actor, Business rule* and *Objective* have their definite roles at *Function – Process* interaction. Construct *Event* represents a change in the system's state and initiates or triggers *Process* or/and some *Function*. Construct *Actor* is an active resource (human, application, machine with control device) used to support the execution of *Process* or/and *Function.* Construct *Business rule* defines a set of conditions, constraints and calculations to be associated with particular *Function* (its information activities).

Construct *Objective* defines a set of Enterprise business goals. The content of the *Objective* influences the definition of *Business Rules* hence the execution of *Function* as well.

The construct *Environment* represents the environment of the Enterprise. This construct refers to the outside objects or entities that can influence Enterprise *Objectives* and raises definite *Events.*

6. The Architecture of Knowledge-based CASE System

Fig. 9 depicts the architecture of the CASE system enhanced by the *Knowledge Base*. The Knowledge Base of the CASE system consists of two parts: Enterprise meta-model (EMM) and Enterprise model (EM). Enterprise meta-model is the model of generic level; Enterprise model includes the partial and particular level models in accordance with GERAM (GERAM, 1999).

The Knowledge Base of the CASE system is supposed to be the third active source of Enterprise knowledge (next to Analyst and User) for Information Systems engineering. Enterprise meta-model (EMM) in this enhanced CASE system is a source of pre-defined knowledge, and is used to control the process of *Business Domain Knowledge Acquisition* and *Analysis*. It is also used to control the construction of an Enterprise model (EM) for particular business domain.

Knowledge-based IS development supposes that all stages of IS development life cycle are supported by the CASE system's Knowledge Base. The Knowledge Base of the CASE system in conjunction with appropriate algorithms assures the consistency among the IS analysis and design models, gives new possibilities for verification and validation of IS development life cycle steps. Moreover, it can be used to simulate and improve business processes within the Enterprise.

The Knowledge Base of the CASE system can be also used to verify business domain knowledge, which was captured by analysts and used to construct particular EM – it is done by verifying constructed EM against the pre-defined knowledge structure of the EMM. This verification technique allows the analyst to identify logical gaps in the business process models and formal gaps that can occur in the acquired management function models.

A logical gap is a semantic discontinuity between the elements of the problem domain model (for instance, workflow model). The logical gaps appear when problem domain knowledge is acquired incompletely.

A formal gap shows the absence of some formally predefined mandatory component of the particular model. Gaps of this type can be identified during the process of veri-

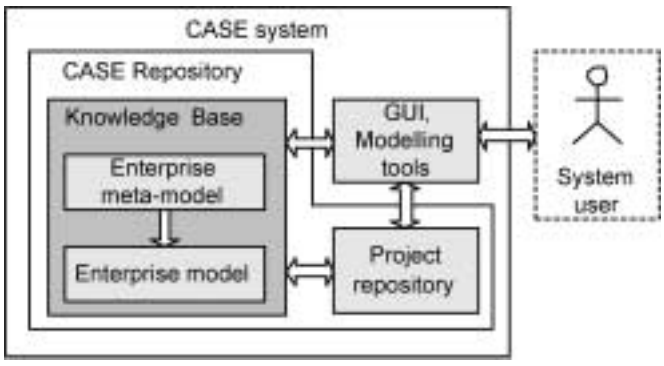

Fig. 9. Architecture of the CASE system with Knowledge Base.

fication of the particular model against predefined formal constraints represented by the EMM.

Enterprise meta-model presents an architecture that improves the integration process of business modelling and IS design. Therefore, the Knowledge Base of the CASE system can be used to identify discontinuities among business process models and IS design models.

7. The Class Diagram of the Enterprise Meta-Model

The Knowledge Base of CASE system (Fig. 9) includes the Enterprise meta-model (EMM) as its major part. The EMM is used as a "normalized" knowledge structure to control the process of construction of an Enterprise model (EM) for particular business domain.

The UML class diagram of the Enterprise meta-model is shown in Fig. 10. The classes of the EMM class diagram correspond to the certain constructs of the Enterprise meta-model depicted in Fig. 6.

We will further discuss constructs of the Enterprise meta-model (Fig. 10). The constructs of the EMM are presented in the same formal way as a set of UEML core constructs in (Vernadat, 2001).

Event. An event depicts a change in the system state. It represents a solicited or unsolicited fact that triggers a process or a function.

A construct *Event* can be formally defined as follows:

$$Event =< EventID,\ Time, \{Expression\}, \{relatedEMelement\} >,$$

where *EventID* is the event identifier; *Time* is clock time or time interval when the event occurs; *Expression* is a boolean expression that sets to true when the event becomes ac-

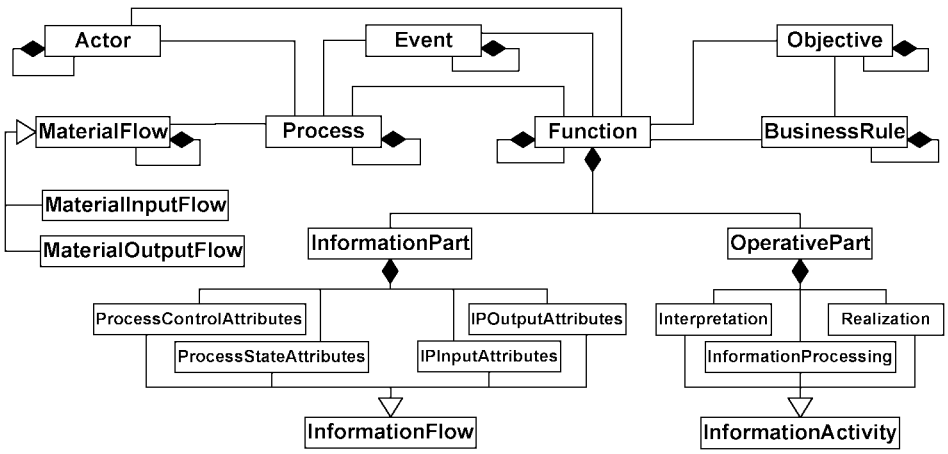

Fig. 10. The Class diagram of the Enterprise meta-model (UML notation).

tive; figure brackets denote a set of elements (therefore, {*Expression*} means that there is a set of expressions related to the event); {*relatedEMelement*} is a set of Enterprise elements (defined in the EMM) associated with the event.

Process. Process is a partially ordered set of steps, which can be executed to achieve some desired material end-result. Process consumes material resources (it is an input of the process) and produces some material output – production. Processes are triggered by one or more event occurrences.

Construct *Process* can be formally defined as follows:

$$Process =< ProcID, \{ProcStep\}, \{Expression\}, \{relatedEMelement\} >,$$

where *ProcID* is the process identifier; {*ProcStep*} is a set of process steps; {*Expression*} is a set of triggering events that need to be true to start the process; {*relatedEMelement*} represents a set of Enterprise model elements related to the process.

Function. Let us describe the function in more detailed way because it is a complex construct. Construct *Function* can be formally defined as follows:

$$Function =< FuncID, \{InformationPart\}, \{OperativePart\}, \{relatedEMelement\} >,$$

where *FuncID* is the management function identifier; {*InformationPart*} is a set of mandatory attribute types important to the *Function*; {*Operative Part*} is a set of mandatory information processing method types; sets {*InformationPart*}, {*OperativePart*}, {*relatedEMelement*} are represented as classes in the class diagram depicted in Fig. 10; {*relatedEMelement*} is a set of Enterprise model objects related to the *Function.*

InformationPart can be formally defined as a set of mandatory attribute types:

$$InformationPart = < ProcessStateAttributes, IPInputAttributes, IPOutputAttributes, ProcessControlAttributes > .$$

ProcessStateAttributes, IPInputAttributes, IPOutputAttributes, ProcessControlAttributes are represented as classes in the class diagram depicted in Fig. 10; these classes are generalized as a higher-level class *InformationFlow*.

Operative Part can be formally defined as a mandatory set of information processing method types:

$$Operative\ Part =< Interpretation, InformationProcessing, Realization > .$$

The classes *Interpretation, InformationProcessing and Realization* are generalized as a higher-level class *InformationActivity*.

This hierarchy of components of the class *Function* is defined in accordance with the formal definition of management function (Figs. 3 and 4).

Interpretation is a step of information feedback loop directed from the *Process* to the *InformationProcessing* of the *Function*. *Interpretation* is an information activity, and it is

aimed to transform and transfer *ProcessStateAttributes.* Construct *Interpretation* can be formally defined as follows:

$$IN = < IntID, \{InputAttrib\}, \{IntRule\}, \{OutputAttrib\}, \{preCond\}, \{postCond\}, \{relatedEMelement\} >,$$

where *IntID* is the identifier of this information activity; {*InputAttrib*} is a set of process state attributes (*ProcessStateAttributes*) related to the identified interaction of the *Function* and the *Process*; {*IntRule*} is a set of rules for identification, capturing and interpretation of the process state attributes; {*OutputAttrib*} is a set of interpreted attributes that are required by the information processing activities (*InformationProcessing*) of the *Function* (this set corresponds to the *IPInputAttributes* in Fig. 10); {*preCond*} is a set of pre-conditions that have to be satisfied to enable execution of this particular information activity; $\{postCond\}$ is a set of post-conditions (ending statuses) of this particular information activity; {*relatedEMelement*} is a set of identifiers of the *Function* and related information activities of that *Function* as well as identifier of *a Process*, which is managed by that *Function*.

InformationProcessing is an information activity of the function and it is aimed to transform systematized data about the controlled process to management decisions. Construct *InformationProcessing* can be formally defined as follows:

$$IP = < InProcID, \{InputAttrib\}, \{InProcRule\}, \{OutputAttrib\}, \{preCond\}, \{postCond\}, \{relatedEMelement\} >,$$

where *InProcID* is the identifier of this information activity; {*InputAttrib*} is a set of attributes that represent systematized information about the *Process*; $\{InProcRule\}$ is a set of rules for data processing and decision-making; {*OutputAttrib*} is a set of attributes that represent management decisions of the *Function* (this set corresponds to the *IPOutputAttributes* in Fig. 10); {*preCond*} is a set of pre-conditions to be satisfied to enable execution of this particular information activity; $\{postCond\}$ is a set of post-conditions (ending statuses) of this particular information activity; {relatedEMelement} is a set of identifiers of a management *Function* and related information activities of that *Function*.

Realization is a step of information feedback loop directed from the *InformationProcessing* to the *Process. Realization* is an information activity and it is aimed to transform *InformationProcessing* output (management decisions of the *Function*) to process control attributes (*ProcessControlAttributes*). Construct *Realization* can be formally defined as follows:

$$RE = < ReID, \{InputAttrib\}, \{ReRule\}, \{OutputAttrib\}, \{preCond\}, \{postCond\}, \{relatedEMelement\} >,$$

where *ReID* is the identifier of this information activity; {*InputAttrib*} is an input of the Realization and represents management decisions of the *Function*; {*ReRule*} is a set of

rules for realization of management decisions, in other words for transformation of *InformationProcessing* output (*IPOutputAttributes*) into a set of process control attributes (*ProcessControlAttributes*) aimed to influence the *Process.* {*OutputAttrib*} is an output of the *Realization* and represents the *Process* control attributes (*ProcessControlAttributes*); {*preCond*} is a set of pre-conditions to be satisfied to enable execution of the *Realization*; {*postCond*} is a set of post-conditions (ending statuses) of this particular information activity; {*relatedEMelement*} is a set of identifiers of the *Function* and related information activities of that *Function* as well as identifier of *Process*, which is managed by that *Function*.

Business Rule. A business rule, as a construct of the Enterprise model, is considered as some condition, constraint or calculation related to some Enterprise element, which is defined in the Enterprise meta-model. Construct *BusinessRule* can be formally defined as follows:

$$BusinessRule = < BRuleID,\ BRuleBody, \{preCond\}, \{postCond\}, \{relatedEMelement\} >,$$

where *BRuleID* is the identifier of the business rule, *BRuleBody* is a formal expression of a rule; {*preCond*} is a set of pre-conditions to be satisfied to enable business rule execution; {*postCond*} is a set of post-conditions (ending statuses) of the business rule; {*relatedEMelement*} is as set of Enterprise model elements related to the *BusinessRule.*

Actor. An actor denotes an element of organisation structure provided with responsibility on identified management functions and/or processes. Construct *Actor* can be formally defined as follows:

$$Actor =< ActorID, \{Responsibility\} >, \{relatedEMelement\} >,$$

where *ActorID* is an identifier of the actor; {*Responsibility*} is a set of responsibilities assigned to the actor (department, division, role or some other organization unit); {*relatedEMelement*} is a set of Enterprise model elements related to the *Actor*.

Objective. An objective is a statement of preference about possible and achievable future situations that influences the choices within some behaviour. Construct *Objective* can be formally defined as:

$$Objective =< ObjID,\ ObjBody, \{relatedEMelement\} >,$$

where *ObjID* is an identifier of the objective, *ObjBody* is a formal or informal statement defining a content of the *Objective*; {*relatedEMelement*} is a set of Enterprise model elements related to the *Objective.*

It should be noted that this minimal set of constructs complies with the basic principles stated in the Sections 4 and 5 with the emphasis on the implementation of the *Function – Process* interaction principles from *the management point of view*.

8. Conclusions

The problems of the Knowledge-based Information Systems engineering have been discussed in this paper. The Enterprise modelling is considered as the major source of knowledge in Information Systems engineering. The Enterprise model formalizes the structure and behaviour of organizational system in order to understand business Enterprise, specify requirements and improve the process of Information Systems design and implementation.

The Enterprise modelling is analysed from the new perspective, namely *the management point of view*. The peculiarity of this approach to Enterprise modelling is the identification of two different types of Enterprise activities, defined as *Function* and *Process.* The absence of the constructs *Function* and *Process* in other Enterprise modelling approaches makes it impossible to define the feedback loop and, consequently, refine the information flow of Enterprise management.

The definition of the *Function* in the Enterprise model presented in the paper differs from the definition of this particular construct used in other Enterprise modelling approaches. The definition of the construct *Function* is formalized, because it is based on the principles found in Systems and Control Theory and therefore has a strong theoretical background.

The formalized analysis and modelling of the *Function – Process interaction* refines a set of new constructs of the Enterprise modelling. As a result the Enterprise meta-model (EMM) has been constructed and discussed in this paper. The EMM is intended to be a formal structure aimed to integrate the domain knowledge for the IS engineering needs.

Knowledge Base of the enhanced CASE system should include the Enterprise meta-model (EMM). The EMM is used as the "normalized" knowledge architecture to control the process of construction of an Enterprise model (EM) for the particular business domain. The usage of such Enterprise model facilitates the automation of the whole IS development process. Some work in this area has already been done (Skersys and Gudas, 2003; Lopata and Gudas, 2003). The architecture of the Knowledge-based CASE system for the enhanced IS engineering is presented in this paper.

References

Anthony, R., and V. Govindarajan (2003). *Management Control Systems*. 11th ed. McGraw-Hill/Irwin, Boston.

Drury, C. (2001). *Management Accounting for Business Decisions*, 2nd ed. Thomson Learning College.

ENV 12 204 (1996). *Advanced Manufacturing Technology Systems Architecture – Constructs for Enterprise Modelling*. CEN TC 310/WG1.

ENV 40 003 (1990). *Computer Integrated Manufacturing Systems Architecture – Framework for Enterprise Modelling*. CEN/CENELEC.

DoD (1994). *Department of Defence Technical Architecture Framework for Information Management.* Version 3.0. Defence Information Systems Agency, Center of Standards.

GERAM (1999). *GERAM*: *Generalised Enterprise Reference Architecture and Methodology.* Version 1.6.3. IFIP–IFAC Task Force on Architectures for Enterprise Integration.
`http://www.cit.gu.edu.au/~bernus/taskforce/geram/versions/geram1-6-3/v1.6.3.html.`

Gudas, S. (1991). Organisational system as a hierarchy of information processes. In *Applications of Artificial Intelligence in Engineering VI* (*AIENG 91*). Computational Mechanics Publications, Southampton, Boston. pp. 1037–1050.

Gupta, M.M., and N.K. Sinha (1996). *Intelligent Control Systems*: *Theory and Applications.* The Institute of Electrical and Electronic Engineers, Inc., New York.

Lopata, A. (2002). Enterprise model based identification of information resources. In *Proceedings "Informacinės technologijos 2002"*. Technologija, Kaunas. pp. 377–381 (in Lithuanian).

Lopata, A., and S. Gudas (2003). Enterprise model based generation of the use case model. *Informacijos mokslai*, **26**, 134–140 (in Lithuanian).

Merchant, K. (1997). *Modern Management Control Systems*, 1st ed. Prentice Hall, Upper Saddle River, New Jersey.

Persson, A. (2001). Enterprise modelling in practice: situation factors and their influence on adopting a participative approach. Department of Computer and Systems Sciences, Stockholm University/Royal Institute of Technology. Report Series No.01-020– pp.334.

Popkin (2002). Popkin Software. In *Building an Enterprise Architecture*: *The Popkin Process*. Version 1.0. `http://www.popkin.com.`

Schekkerman, J. (2003). *How to Survive in the Jungle of Enterprise Architecture Frameworks*. Trafford, Canada.

Skersys, T., and S. Gudas (2003). Enterprise model based generation of class model. *Informacijos mokslai*, **26**, 199–205 (in Lithuanian).

Stephen, J., and S. Kendall (2004). *MDA Distilled*: *Principles of Model-driven Architecture*. Addison-Wesley Pub. Co.

Totland, T. (1997). Enterprise modelling as a means to support human sense-making and communication in organizations. Norwegian University of Science and Technology (NTNU), Trondheim IDI-raport 1997:8.

Turban, E., E. McLean, J. Wetherbe (1999). *Information Technology for Strategic Advantage.* 2rd ed. John Wiley & Sons.Inc.

Universal Enterprise Modelling Language (1999). IFAC-IFIP Task Force, UEML Group. `http://www.cit.gu.edu.au/ ∼bernus/taskforce/archive/UEML-TF-IG.ppt.`

Vernadat, F. (2001). UEML: towards a unified enterprise modelling language. In *Proceedings of International Conference on Industrial Systems Design, Analysis and Management* (*MOSIM'01, 2001*), Troyes, France.

S. Gudas is a doctor of computer sciences, associate professor of Information Systems Department of Kaunas University of Technology and Kaunas Faculty of Humanities of Vilnius University. His research interests include computer-aided information systems engineering methods and tools, enterprise modelling for information systems engineering.

A. Lopata is a doctor of computer sciences, lecturer of Information Systems Department of Kaunas University of Technology. His main research interest includes enterprise modelling, user requirements acquisition, analysis and specification stage of information system development life cycle.

T. Skersys received degree of master of informatics science in 2001. He is a doctoral student since 2001 and junior research assistant since 2003 at Kaunas University of Technology Information Systems Department (ISD). His research interests include computer-aided methods for information systems engineering, nterprise modelling and business rules-extended IS development.

Žinių apie probleminę sritį integravimas informacijos sistemų inžinerijoje

Saulius GUDAS, Audrius LOPATA, Tomas SKERSYS

Straipsnyje analizuojamas žiniomis grindžiamos informacijos sistemų (IS) inžinerijos būdas. Pasiūlytas ir teoriškai pagrįstas organizacijos veiklos meta-modelis, kuris atskiria ir detaliai aprašo organizacijoje vykstančius technologinius procesus, valdymo funkcijas ir šių elementų tarpusavio sąveiką. Tai siūlomo organizacijos veiklos modeliavimo būdo, skirto žiniomis grindžiamai IS inžinerijai, ypatumas. Pasiūlytojo veiklos meta-modelio pagrindu gali būti sudaryta CASE sistemos žinių bazė. Pateikta tokios žinių baze papildytos CASE sistemos principinė schema.