Inference Rules of Semantic Dependencies in the Enterprise Modelling

Remigijus GUSTAS
Department of Information Systems, Karlstad University, 651 88 Karlstad, Sweden
Remigijus.Gustas@kau.se

Abstract. Enterprise models should have a capacity to describe consistently business processes across organisational and technical system boundaries. It would help system designers to understand why the technical system components are useful and how they fit into the overall organisational system. The implementation bias of many information system methodologies is a big problem for inconsistency and integrity control. The same implementation oriented foundations are often used in system analysis and design phase, without rethinking these concepts fundamentally. Common repository of most CASE tools does not guarantee the consistency of enterprise architectures for a reason that interplay among static and dynamic dependencies is not available. Enterprise modelling and integration should stick to the basic conceptualisation principle that prescribes analysis of only conceptually relevant aspects. It cannot be influenced by any implementation details. The consistency problems are best detectable and traceable at the conceptual layer. In this study on semantic dependencies, we demonstrate how various fundamental concepts from different classes of models can be interlinked and analysed together. An important result of this study is a set of inference rules. The inference capability is an intrinsic feature of logical approaches, but the conventional methods of system analysis have not yet dealt in sufficient detail with the inference principles.

Keywords. Enterprise Modeling, Integration of static and dynamic dependencies, inference rules

Introduction

The term of Enterprise Architecture (EA) has been used for many years within information system engineering community. It refers to various types of graphical representations that define how business, data, technology and software application structures (Spewak, 1992) are perceived from different points of view. The concept of enterprise in the context of information system development denotes a limited area of activity in organisation (Bubenko, 1993) that is of interest by a planner, owner, designer or builder (Zachman, 1996). In the Nineties, the term of enterprise architecture started to be used even by business managers, especially those involved in business process re-engineering, to refer to the graphical descriptions of organizational processes. Today, enterprise models (yet another name of EA) denote a comprehensive graphical description of the syntactic, semantic and pragmatic relations across organizational and technical system boundaries (Gustas & Gustiene, 2004).

For instance, when managers are talking about the alignment of information technology (IT) systems and applications with respect to business processes, they define graphical enterprise models, which demonstrate how the alignment should be achieved. US Government agencies are required by law to maintain EA, because of congressional legislation (Raines, 1997).

There are many approaches to define components of enterprise architecture. The Zachman framework (Zachman, 1996) is considered as a comprehensive guide of documents or blueprints that comprise any enterprise architecture. Typically, EA is defined in various perspectives such as the "what", "who", "where", "when" and "how". Various actors involved in the enterprise modelling usually concentrate on one of the following layers: the strategic layer (planner view), the conceptual (implementation independent) layer (owner view) and the implementation-oriented layer (designer and builder views). An integrated conceptual representation of the organisational and technical system is necessary to develop a holistic understanding of EA and to plan orderly transitional processes from the current to the target enterprise architecture. There are two basic challenges facing the overall enterprise engineering process: enterprise modelling and integration (Vernadat, 1996). Modelling involves visualisation or externalisation of EA from different points of view such as planner, owner, designer, builder and subcontractor. It involves representation and population of various cells of the Zachman Framework with instances of diagrams that represent the strategic (pragmatic), conceptual (across organisational and technical system boundaries) and the implementation dependent (logical, physical) system development perspectives. Nevertheless, views and perspectives do not make the enterprise architecture. To obtain value from the graphical representations, these documents must be integrated. The developers of enterprise engineering tools usually rely on a common repository. Since the interplay among semantic dependencies is not completely clear, the existence of a common repository does not guarantee the consistency and integrity of enterprise models.

Fragmented representations of EA are difficult to maintain even for very experienced system analysis and design specialists, not just for a reason that they deal with the same semantic details on different levels of abstraction, but because these representations are huge. Many experts have been building information systems for years, but just few of them have actually learned how to keep track of interdependencies among various diagram types. The difficulty resides in the implementation bias of many information system methodologies. The implementation-oriented diagrams are much more complex than conceptual models and thus they are difficult to trace. Sometimes, similar semantic constructs are applied in different perspectives without rethinking them fundamentally. These deficiencies of EA results in a difficulty to integrate views of two subcultures: business management and IT development personnel. The traditional information system analysis and design methods are not working well enough for the achievement of that purpose. For example, the UML (Booch et al., 1999) provides twelve standard diagram types for representation of a technical system solution. Nevertheless, it is very little known how to control consistency and integrity of these diagrams. To our knowledge, the UML foundation is not provided by any inference or reasoning rules. Another weakness is the focus on the technical system (logical) part and very little possibilities to define the organisational (would it be conceptual or strategic) system part. The underlying modelling foundation of the conventional information system modelling approaches is too weak for

development of the semantic reasoning rules on EA. Inference rules at the conceptual layer (not the logical rules) are crucial to enable reasoning about organisational and technical process consistency.

This paper is organised as follows. The second and third section defines the basic elements and dependencies that are used for modelling of EA at the conceptual layer. In the fourth next section, the importance of critical quality aspects is discussed. The inference rules are introduced in the fifth section. The conclusion section outlines the perspective of enterprise modelling in other areas and discusses the future work.

1. Basic Dynamic Dependencies in the Enterprise Modelling Approach

Understanding the strategic dependencies (Yu & Mylopoulos, 1994) is essential for reaching a consensus on how the current or future situation of enterprise architecture looks like. The dependencies between various technical and organisational components involved describe the "who" perspective. A strategic dependency link between two actors (agent and recipient) indicates that one actor depends on another actor. An instance of actor can be an individual, a group of people, an organisation, a machine, software or hardware component. The dependent actors in a diagram can be related by the actor dependency link (⋯▸). The actor dependency is usually seen as a physical, information or a decision flow between two participants. Graphical notation of the actor dependency between an agent and a recipient is presented in figure 1.

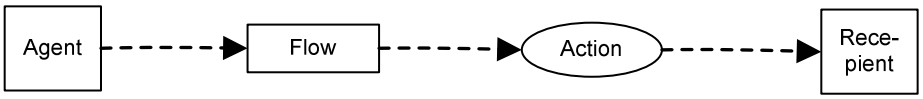

Figure 1. The Action and Flow dependency

A strategic flow dependency link between actors at the conceptual level is defined as a communicative action. Thus, it is considered at the same time to be an action and a communication flow (Goldkuhl, 1995). An agent A initiates a flow F by using action C to achieve his goal (Gustas & Gustiene, 2004). If another actor B is a recipient of the same flow F, then such dependency is defined as (A ⋯▸ B) /C(F). An ellipse notation is used to distinguish actions. The flow dependency represents a communication channel for transferring information or physical flow between agent and recipient. For instance, (Company ⋯▸ Person) /Employ(Contract).

An action is typically changing a state of affairs, i.e. a state of an object that is instantiated in some class. Otherwise, the action is not purposeful. In other words, any action should be interpreted as an object transition. Cohesion of a transition dependency and communication dependency results into a more complex abstraction that defines two perspectives of action: static and behavioural. We will use an extended graphical notation to describe such a twofold nature of action. The extended communication action dependency between two states of the same object is represented Figure 2.

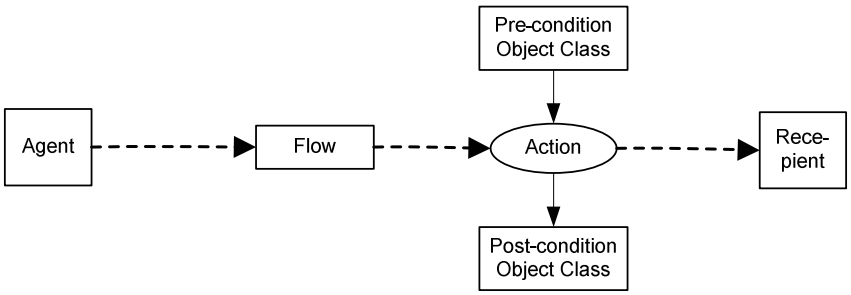

Figure 2. Extended graphical notation of the communication action

Actions are carried out by actors. At the same time, any action can be defined as a transition (→) from the precondition state to the postcondition state. If an object is instantiated in a precondition class D, then, by using action C, the transition to the postcondition class E can be performed. This is represented by (D → E) /C(F). For instance, (Applicant → Employee) /Employ(Contract). Graphical example of the transition dependency is illustrated in figure 3.

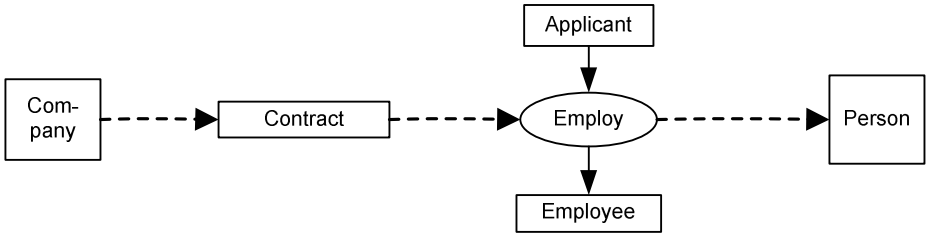

Figure 3. Communication action of Employ

The transition dependency typically defines a semantic relationship between two states of the same object. Such transitions are normally considered as the "how" perspective. Both the transition and communication dependencies describe a very important part of knowledge about business processes. Unfortunately, many communication-based approaches often neglect some behavioural aspects of the state transition and vice versa, many software engineering approaches disregard the dependencies of communication (Action Technologies, 1993), (Winograd & Flores, 1986), (Gustas, 2000).

A simple description of an object's state, for example, in terms of a finite-state machine, would mean definition whether an object exist or not in some state. However, an individual that just exists in a particular state, without relating it to other objects is not very useful. Analysts need to know how and why the associations of object classes are going to be changed and what are the static dependencies among them. Fundamentally, two kinds of changes occur during any transition: declassification of object from the current class and classification of the same object to the next class. Sometimes, it is referred as a reclassification event (Martin & Odell, 1998). A declassification event typically removes an existing object and classification event creates an object. Different classes of objects are

characterised by at least one entirely new attribute. The importance of such noteworthy difference is essential to understand semantics of communication action.

2. Interplay among the Basic Static and Dynamic Dependencies

The static concept dependencies define the "what" perspective. These dependencies are stemming from various semantic models that are introduced in the area of information system analysis and design. For instance, semantics of static dependencies in object-oriented approaches are defined as cardinality constraints, which represent a minimum and maximum number of one set of objects that can be associated to another set of objects. Graphical notation (Hoffer et al., 2004), (Martin & Odell, 1998) of typical associations is represented in Figure 4.

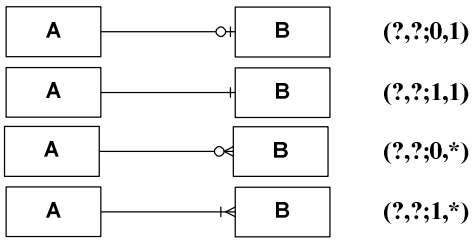

Figure 4. Graphical notation of cardinality constraints

Note: the meaning of '*' is 'many' (i.e. more than one), the meaning of '?' is 'undefined', which corresponds to a constraint less link (0,*) or to a link with unspecified cardinality constraints. Notations that are commonly used at the initial phase of concept modelling have to provide a clear understanding of cardinality constraints in both directions. The common static dependencies that may be specified between any two concepts A and B are as follows:

1. (0,1;1,1) - Injection dependency which will be denoted by A ===> B ,
2. (1,1;1,1) - Bijection dependency (A ⟷ B),
3. (0,* ;1,1) - Total Functional dependency (A ⟶ B),
4. (1,1;1,*) - Surjection dependency (A ⟵⟶ B),
5. (0,1;1,*) - Surjective partial functional dependency (A ==>> B),
6. (1,*;1,*) - Mutual multivalued dependency (A ⟸⟹ B),
7. (0,* ;1,*) - Total multivalued dependency (A ⟶⟶ B),
8. (0,1;0,1) - Partial injection dependency, (A |==> B) ,
9. (0,* ;0,1) - Functional (partial) dependency (A ⟼ B),
10. (0,*;0,*) - Multivalued (partial) dependency (A⟼⟶ B).

In the final phase of enterprise modelling, only first five dependencies, from a total number of ten, are used. These totally applicable links are considered as basic concept attribute links. The remaining five dependencies can be expressed in terms of more primitive ones. It should be noted that not basic dependencies are used without any limitations in the conventional approaches of system analysis and design, except cases No 6, 7 and 10. Many-to-many associations in the design phase are normally converted to a new

aggregated concept, which can be defined as an aggregation or composition of concepts A and B. The other relations, such as No 8 and 9 as a rule are easily transformable to one of the basic dependencies, by using the inheritance link. This process is entitled as eliminating of semantic holes (Gustas, 1994).

Dependencies can be shared by extracting and attaching them to a more general concept. Inheritance (⟶) is as a core link to connect a specific concept to more general one. It enables evolution of the specialisation hierarchy without information loss and is similar to the dynamic specialisation (Leonard, 2004). Therefore, the inheritance dependency in the enterprise modelling is richer than the classical inheritance. Graphical notations of the basic static dependencies are presented in figure 5.

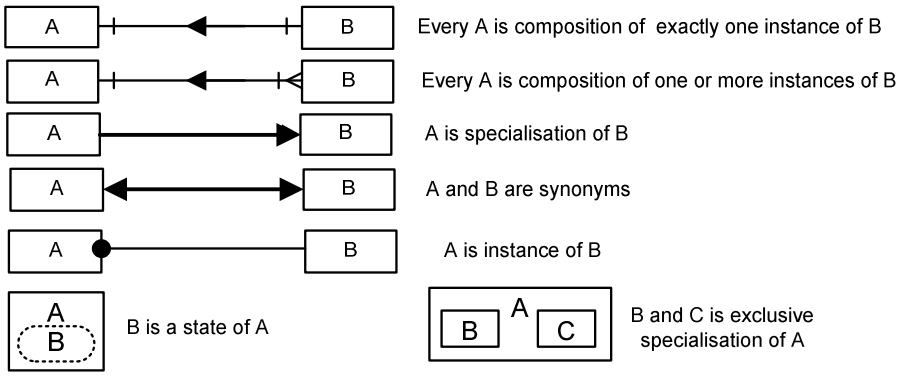

Figure 5. Graphical notation of the basic static dependencies

Composition is a conceptual dependency link (multivalued ⟙ and singlevalued ⟙▶), which is used to relate a whole to other concepts that are viewed as parts. The composition dependency in the enterprise modelling is a more restricted link as compared to other methodologies. It is characterised by the following properties: 1) a part cannot simultaneously belong to more than one whole. If it does belong - then it must be the same whole, 2) a part and a whole are created at the same time, 3) once a part is created, it is terminated at the same time the whole is terminated. This definition is more strict as compared to a composition that is used in the object - oriented approach (Maciaszek, 2001), (Stumpf & Teague, 2005). Not just a part, but also a whole is dependent on a part as well. Therefore, the cardinality is never equal to zero. This difference is significant, since it has an impact on update operations (see inference rules). The creation and removal of objects is always propagated to its parts and vice versa.

The introduced set of dependencies is used together with the transition dependency that was defined in the previous section. The transition dependency can be used to specify semantics of a possible state change. An actual state always persists until the action terminates. Termination of action causes a transition to the next state. The graphical notation of transition dependency Employ (Person [NOT Employed]) ⟶ Person [Employed] is represented in figure 6.

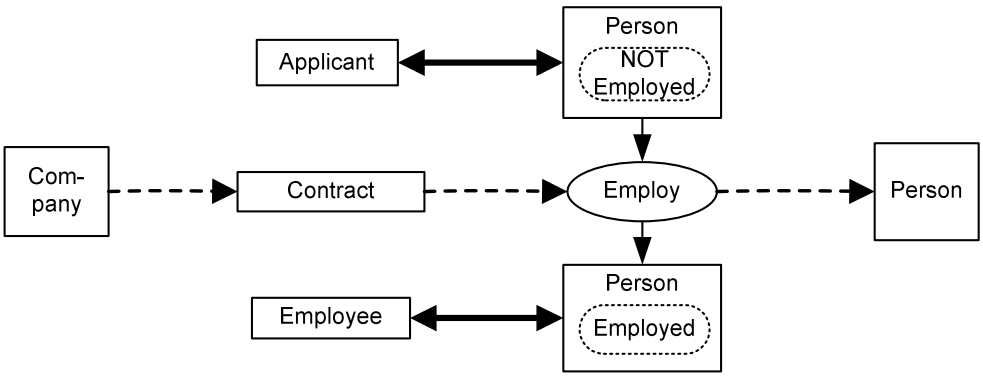

Figure 6. Communication action of Employ with synonymous concepts

Since every transition represents reclassification of an object (from one class to another), the transition dependency in the enterprise modelling is not linking explicitly two states like in Object-Process methodology (Dori, 2002). Instead, the object transition dependency connects two classes of objects. For instance, two states such as 'NOT Employed' and 'Employed' are attached to a concept of Person by using an operation of restriction (Gustas, 1994). It results in two new compound names such as Person [NOT Employed] and Person [Employed] that are representing more specific subclasses of Person.

In the object oriented-approach (Martin & Odell, 1998) fundamentally two kinds of changes may occur: creation and removal of an object in a class. Sometimes, objects are passing several states, and then are destroyed. An action of object removal will be specified as follows: **(PRECONDITION CLASS → ⊥) /ACTION( )**. Here: ⊥ is an empty element that indicates the initial or final state in a life cycle of object. A graphical notation of dependency between a final state and action, that defines the deletion operation of object, can be viewed as a special case of a state-transition dependency, which is not provided by a next state. Removal of an object terminates all its associations. And visa versa, a creation of an object must bring all its pre defined static dependencies into existence. The creation action of an object in a particular class can be specified by the following expression: **(⊥ → POSTCONDITION CLASS) /ACTION( )**.

The basic dependencies are very important to understand the semantics of any communication action. A typical action workflow loop (Action Technologies, 1993) can be defined in terms of two or even three communicative action dependencies between actors. By matching the actor dependencies from agents to recipients, one can explore opportunities that are available to these actors. We shall illustrate the basic semantic dependencies in one action workflow loop between two actors (Person and Company) together with the creation, destruction and transition actions, which are represented in figure 7.

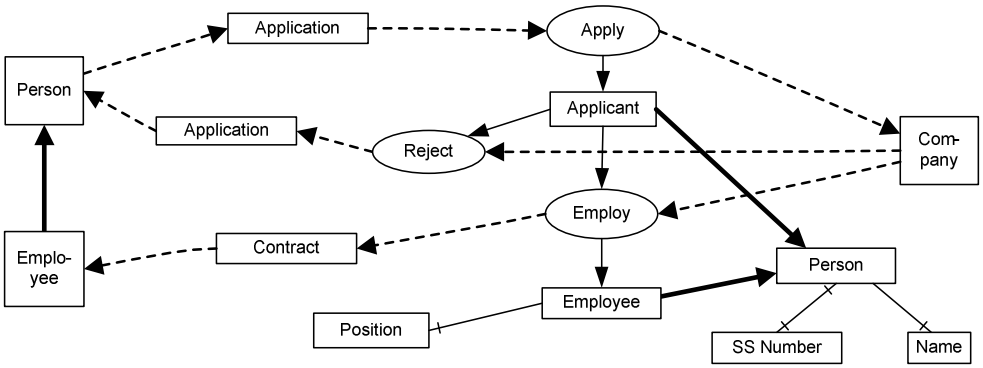

Figure 7. Graphical example of the integrated static and dynamic description

This diagram illustrates that a person is authorised to apply for a job by sending an application to a company. If a company accepts the application, then an object of applicant is created. According to the business definition that prescribes the diagram, a company is obliged either to employ or to reject the application. If company decides to reject, then the object of applicant is destroyed (Reject action is triggered by Company), otherwise it is reclassified to employee (Employ action is triggered). Please note that object of Employee (the same as Applicant) is a specialisation of a Person concept, but it is characterised by the additional attribute of Position. The two other attributes (Name and SS Number) must be instantiated at the time when the applicant is created. Otherwise, the Apply action is rejected. In this diagram, the attribute of Position is essential to characterise the semantic difference between two concepts such as Applicant and Employee. If employee is terminated by the next action, then the association to position will be removed. In such a way, sequences of communicative actions prescribe sequences of the update operations, which should be defined at the implementation dependent layer. The object oriented way of interpretation of enterprise models was demonstrated in the other paper (Gustas & Jakobsson, 2004). Additionally, such diagrams at the conceptual layer may serve as a basis to analyse obligations and authorisations of the actors involved. For instance, if (Person ⋯▸ Company) /Apply (Application) then (Company ⋯▸ Employee)/Employ (Contract) or (Company ⋯▸ Person)/Reject (Application).

3. Semantic Consistency and Integrity

Information system quality (Gustiene, 2003) is essential for understanding of the organisational and technical system fitness. Nevertheless, it is very little known on how the semantic quality problems might be treated in a systematic way. Surely, various diagrammatic descriptions that humans employ for representation of enterprise architectures on different levels of abstraction should be evaluated by certain quality criteria. Lindland, Sindre and Solvberg (Lindland et al., 1994) have proposed a framework for understanding the syntactic, semantic and pragmatic quality. The syntactic quality can be characterised by

correctness of modelling language. A better pragmatic quality of system specification would mean agreement on what is going to be achieved. The pragmatic quality is build upon the semantic quality. Two characteristic features of the semantic quality are validity and completeness (Krogstie, 2003). Validity means that all statements in the model are consistent and relevant to the problem. Completeness means that the enterprise model contains all relevant statements. It is not easy to apply these two criteria in practice, because we do not know how the semantic quality can be controlled. The semantic quality criteria are still poorly analysed in the literature.

The most difficult part of enterprise modelling is arriving at a coherent, complete and consistent semantic description of a new system that is defined across organisational and technical boundaries. The ability to describe essential system solutions in a clear and sufficiently rich way is acknowledged as crucial in many areas including software requirements engineering, object oriented system design (Booch et al., 1999), structured analysis (Yourdon, 1989), conceptual modelling, e-service composition (Hull et al., 2003), semantic web (Daconta et al., 2003), knowledge management and information system modelling (Hoffer et al., 2004). One of the major difficulties using conventional system engineering approaches is that very little support for semantic consistency control is available. Such needs are especially acute for teams, which are operating in a collaborative enterprise modelling environment or they are involved in joint information system development sessions, where different perspectives (e.g. static and dynamic), different points of view and system descriptions on different levels of abstraction are used.

The semantic consistency criterion refers to whether or not conceptual descriptions are compatible. Violation of compatibility gives rise for inconsistency. CASE tools have analysis facilities that typically control some type of inconsistency in a single perspective. For example, when designer is drawing a data flow diagram (Hoffer et al., 2004) and decomposition of process takes place, most CASE tools will automatically place the inflows and outflows on the lower level of abstraction. Deleting such flows would cause the diagrams out of balance. CASE tools that typically support methods for identification of discrepancies during the process of database view integration (Batini et al., 1986) are concentrating just on consistency of the static perspective. None of the information system modelling approaches is dealing with the semantic consistency in few perspectives at the same time. The difficulty lies that interplay between static and dynamic dependency types is not clear.

The usefulness of great number of semantic dependencies in the area of information system analysis and design is another open problem. For instance, many dependencies that are introduced in database theory encounter problems with missing attribute values. These problems result from the fact either that the instance of the attribute is temporally unknown, but applicable, or that its value can never be known, because the attribute is not applicable. For unambiguous specification of system semantics, in the final phase of enterprise modelling, only basic dependencies are used. These dependencies are totally applicable. Not basic dependencies can be eliminated in various ways. This process is entitled as semantic normalisation. It is achieved by defining alternative courses of actions or defining more precise semantics of the static links. Semantic normalisation of dependencies helps analysts to improve the semantic quality of diagrams and to use a full power of inference rules. Normalisation of concept diagrams is performed through appropriate transformations. Some

methods call this process as elimination of semantic holes (Gustas, 1994), strengthening or restricting (Borgida, 1984), (Brachman & Schmolze, 1985). In object-oriented approach, the transformation process is called sharpening the meaning of concepts (Martin & Odell, 1998). The diagrams, which are defined in terms of the basic dependencies, have ability to communicate unambiguously the semantics of enterprise models. This idea is illustrated in figure 8. We are not going any further into this subject for the reason of space limitations.

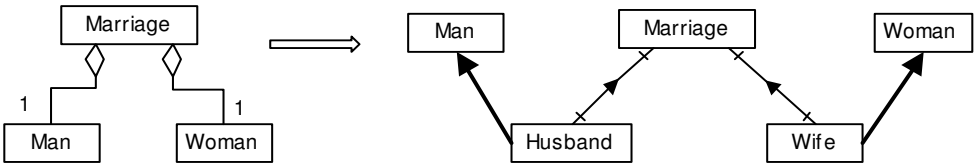

Figure 8. Interpretation of aggregation through the basic dependencies

Aggregation and composition that is used in object oriented system design is a weaker form of the enterprise modelling composition dependency, which was defined in the previous chapter. For example, the concept of Marriage can be defined as an aggregation of Man and Woman. It is interpreted in terms of the basic dependencies as the composition of Husband and Wife that are specialisations of Man and Woman. Composition and specialisation dependencies are very useful semantic relations, because they clearly represent how destruction and creation operations are propagated in the corresponding hierarchies. For instance, if an object of Wife is removed then the object of Marriage and Husband will cease to exist. Despite of that, the same object is not declassified (Leonard, 2004) as an instance of Woman.

Enterprise modelling approach has no implementation bias (Gustas & Gustiene, 2002), (Gustas & Gustiene, 2004) like many other information system methodologies. It follows the basic conceptualisation principle (Griethuisen, 1982). Enterprise modelling and integration is dealing exceptionally with the conceptually relevant aspects and it is not influenced by the possible implementation decisions. Many information system methodologies are not following the basic conceptualisation principle. Consequently, the information system modelling quality suffers significantly. Consistency control at the conceptual layer is critical, because it is a driving force for a change management in a systematic way. Since conceptual models define system semantics, they help designers to understand why the technical system components are useful and how they fit into the overall organisational system. The traditional methods of information system analysis are based on the idea of dividing the technical system representations into three major parts that are known as data architecture, application architecture and technology architecture. Although there is a great power in separation of different technical architectures, there is also a deep fallacy in such orientation. One consequence is the difficulty to apply the automated reasoning rules for inconsistency analysis of the isolated views. Such system development tradition is not taking into account interdependencies that are crucial to glue the static and dynamic aspects of the enterprise models.

4. Inference Rules of Semantic Dependencies

System analysts and designers may concurrently perceive and represent concept dependencies in a number of ways. Similarities and differences are not difficult to identify, if just the basic set of the static and dynamic dependencies is used. Discrepancies among various views can be automatically detected by using a specific set of inference rules. Not all structural differences of enterprise models are interpreted as semantic conflicts. Some dependencies can be viewed as weaker and some of them as stronger. A set of the weaker dependencies can be derived according to the following inference rules:

1) if A ===> B then A ⟶ B ,
2) if A ===> B then A ==>> B ,
3) if A ⤙► B then A ⤙ B ,
4) A ===> B , B ===> A if and only if A ⟷ B ,
5) A ==>> B , B ⟶ A if and only if A ⟛ B ,
6) A ⟶► B , B ⟶► A if and only if A ◄⟷► B.

A generalisation hierarchy consists of interconnected concepts by inheritance links on different levels of abstraction. In the enterprise modelling, not like in object-oriented methods, all kind of the basic dependency links could be inherited (Gustas, 1998) according to the special inference rules:

1) if A ⟶► B , B ⟶► C then A ⟶► C ,
2) if A ⟶► B , B ===> C then A ===> C ,
3) if A ⟶► B , B ⟶ C then A ⟶ C ,
4) if A ⟶► B , B ==>> C then A ==>> C ,
5) if A ⟶► B , B ⤙ C then A ⤙ C ,
6) if A ⟶► B , B →► C then A →► C.

The object – oriented tools take advantage of inheritance just for attributes and operations. The presented six inference rules demonstrate the inheritance capability of the communication and transition dependencies. Such dependencies are going together with the communication action. For instance, if Manager ⟶► Employee, Employee →► Person[NOT Employed] /Terminate(Employee**.**SS number) then Manager →► Person[NOT Employed] /Terminate(Employee**.**SS number). Graphical illustration of the same dependency is given in figure 9.

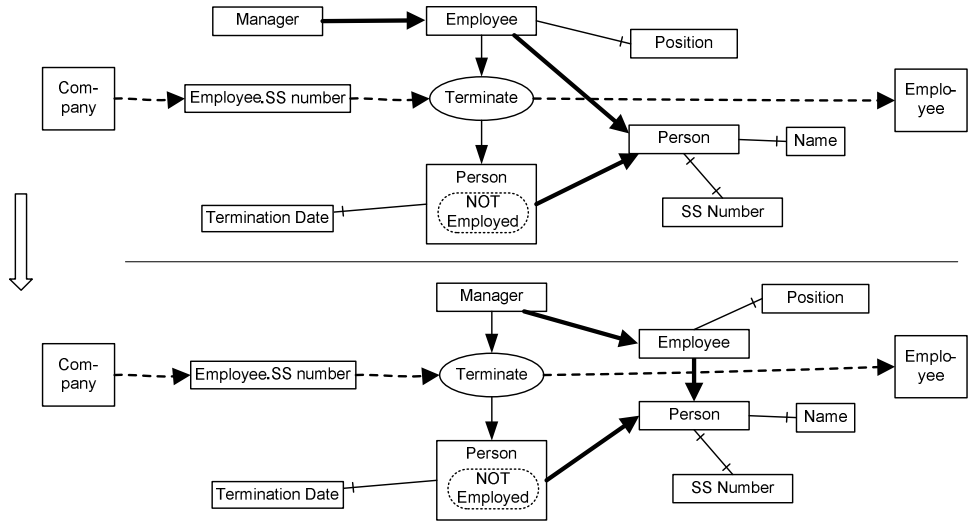

Figure 9. Graphical illustration of the inherited transition dependency

Clearly, the derived transition dependency link is redundant and, therefore, it can be removed by using an appropriate semantic normalisation procedure.

The semantic quality of diagrams can be improved by eliminating ambiguity of concepts. It can be achieved by the following inference rules:

1) **if A ===> B , B ===> C then A ===> B.C , B.C ⟶ C,**
2) **if A ⟶ B , B ⟶ C then A ⟶ B.C , B.C ⟶ C,**
3) **if A ==>> B , B ==>> C then A ==>> B.C, B.C ⟶ C.**

Here **B.C** is the operation of prefixing C in the context of concept B (Gustas, 1994).

The prefixing rules are useful to resolve ambiguity of concepts. Since Address is considered as an attribute of two concepts, it may be perceived in two ways as a company address or as an employee address. The prefixing rules is an important technique for resolving ambiguity of concepts that are used in different contexts. Conceptual representations of ontologies (Daconta et al, 2003) have not yet dealt with the concept of context. Illustration of the prefixing operation is presented in figure 10.

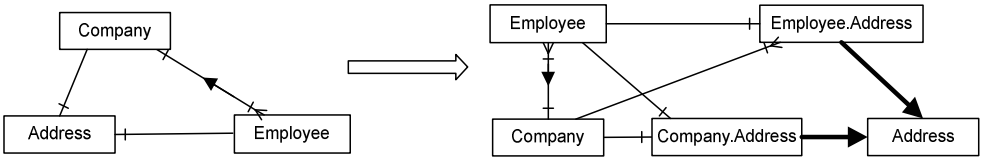

Figure 10. Inference rules with the prefixing operation

The composition dependency is transitive. It is characterised by the following property: **if A ≻— B, B ≻— C then A ≻— C**. Graphical illustration of the inference rule is presented in Figure 11.

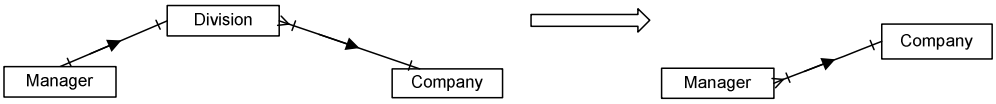

Figure 11. Transitiveness of the composition dependency

A composition hierarchy is quite useful to analyse how various transition dependencies are propagated along the operations.

The basic communication dependencies together with the composition and inheritance links constitute a formal basis for inconsistency analysis and view integration in enterprise modelling. It should be noted that the communication dependency between two actors might be refined in terms of the object transition between two concepts. New concept is always characterised in terms of a different set of the static dependencies, which define the contents of change in the receiving organisational or technical component. The actor communication links are inherited by the more specific concepts and are propagated along the singlevalued composition hierarchy. The inference rules are as follows:

1) **if A ⋯▸ B, B ◂≻— C then A ⋯▸ C ,**
2) **if A ◂≻— B, A ⋯▸ C then B ⋯▸ C ,**
3) **if A ⟶ B , B ⋯▸ C then A ⋯▸ C ,**
4) **if A ⟶ B, C ⋯▸ B then C ⋯▸ A .**

For instance, if Employee ⟶ Person, (Person ⋯▸ Company) /Apply(Application) then (Employee ⋯▸ Company) /Apply(Application). Illustration of how the communication dependencies are propagated along with the composition hierarchy is represented in figure 12.

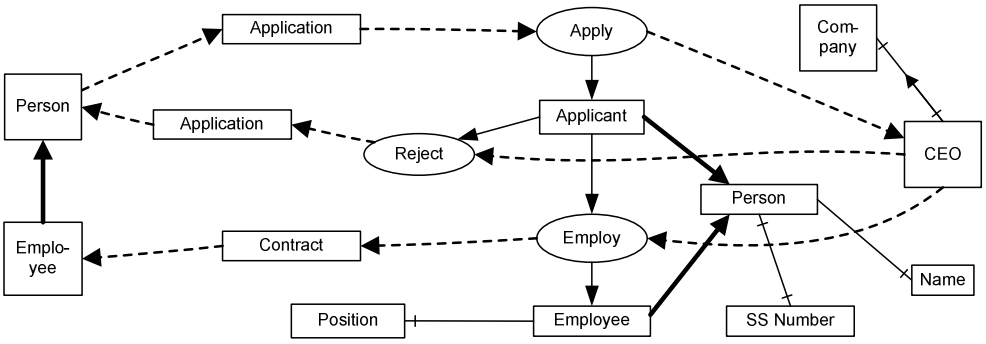

Figure 12. Illustration of propagation and inheritance of the communication links

According to the presented inference rules, the communication actions of Apply, Reject and Employ are relevant for CEO, but at the same time they are propagated to Company (see the graphical representation in Fig.7).

An instantiation dependency (≤) indicates that an object is an instance of a concept. The instantiation is the reverse side of the classification dependency (see the notation in chapter 3). The classification dependency is characterised by the following inference rules:

1) **if A ≤ B, B ⟶ C, then A ≤ C.**
2) **if C ≤ A, B ⟶ A, ~ (C ≤ B) then C ≤ ¬B, ¬B ⟶ A,**
3) **if C ≤ A, B ⟶ A, ~ (C ≤ ¬B) then C ≤ B.**

Here: ¬ is the operation of concept negation (Gustas, 1994), ~ is the logical negation.
For instance, if IBM ≤ Company, Company ⟶ Organisation then
IBM ≤ Organisation, if John Smith ≤ Employee, Manager ⟶ Employee,
~ (John Smith ≤ Manager) then John Smith ≤ ¬Manager, ¬Manager ⟶ Employee (note that ¬Manager concept in a natural language corresponds to 'NOT Manager'). Since precondition and postcondition class object sets are exclusive, the last two inference rules have some important consequences for system model evolution and integration. The rules can be implemented by the CASE tools to generate hidden semantic details in the dynamic part of a diagram. Example of automatic generation of the new concept (NOT Manager) together with the transition and inheritance dependency is graphically represented in figure 13.

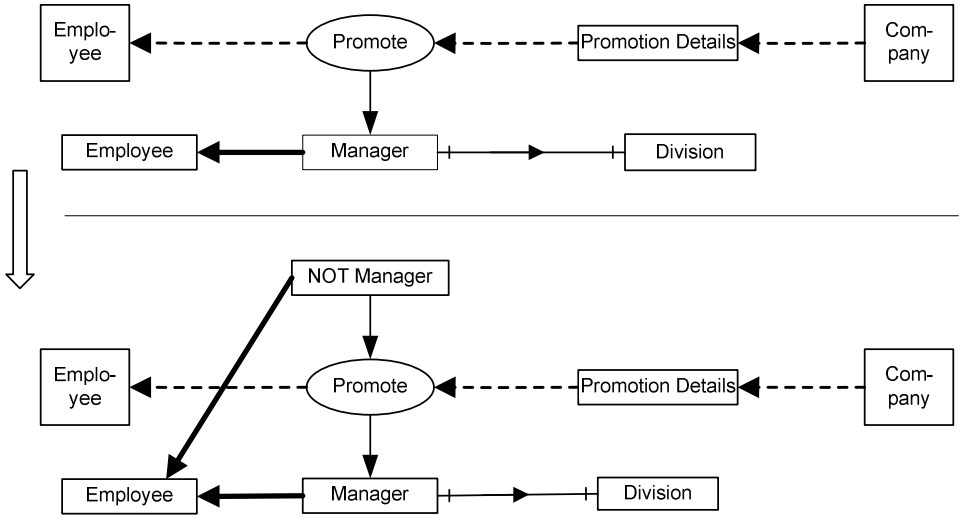

Figure 13. Generation of the negated concept and inheritance dependency

It is not difficult to prove validity of the following rule:

(⊥→ E) /A( ), E ⟶ G if and only if (¬E → E) A( ), ¬E ⟶ G, E ⟶ G.

According to the presented graphical example, for an employee object to be promoted to a Manager (see the Promote action), he must be instantiated first as a NOT Manager. Automatic generation and visualisation of hidden semantic links is useful for conceptual integration of communication actions from semantically similar, but structurally differently defined interaction loops. For instance, inference rules justify integration of two diagrams that are represented in figure 14.

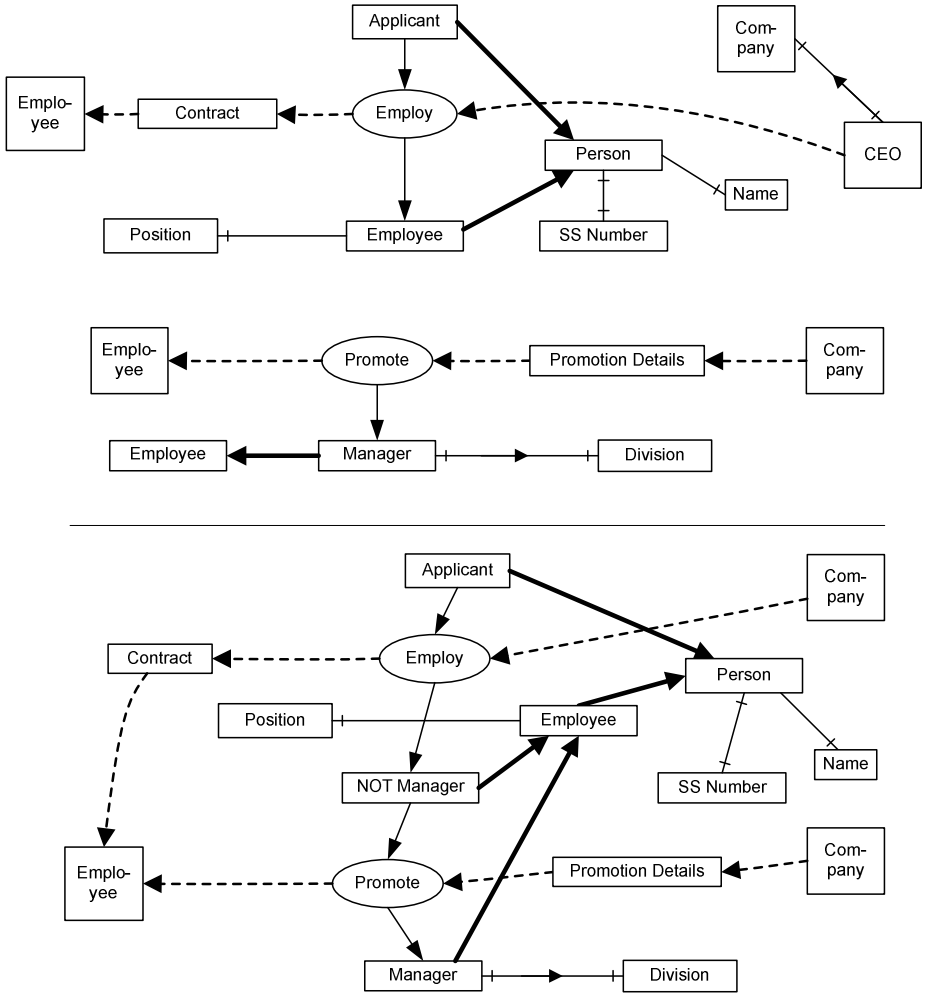

Figure 14. Integration of communication actions from different interaction loops

Instances of concept identifiers are viewed as objects of the same concept class. These objects are defined according to the following rule:

if X ≤ A.B, A ⟷ A.B then X ≤ A[B = X], A[B = X] ⟶ A.

Here: A[B = X] is the operation of concept A restriction by the condition [B = X].

Inference rules of instances might remind inferences in the predicate logic. The logical inferences such as, **if X is A and A has C, then X has C**, can be derived from the previous rule in combination with inference rules of the inheritance dependency. For instance:

if IBM ≤ Company, Company[Name = IBM] ⟶ Company, Company ⟶ Address then Company[Name = IBM] ⟶ Address.

5. Concluding remarks and Outlook

Organizational or technical system components in the enterprise modelling approach are viewed as actors. The organizational system part might consist of humans, departments, companies, etc., and the technical system part is typically composed of software or hardware components. Communication flow dependencies among actors of various types together with the actor composition and generalization links are used to refine enterprise architectures. The inference rules of semantic dependencies are critical for inconsistency detection between more specific and more general levels on various levels of abstraction. Therefore, integrity among dependencies, which are specified by various people such as information technology planners, business owners, system designers, users and builders, can be controlled in a more systematic way. The key issue of enterprise modelling is graphical description and motivation of the true information needs from the point of different actors involved.

A starting point of the enterprise modelling is implementation independent. It is introduced for the purpose of describing information system architectures across technical and organisational boundaries. Currently, we aim at the extended enterprise modelling approach with reasoning rules and engineering process that is similar to what architects use when constructing buildings. The graphical models are useful for visualisation and reasoning about the semantic consistency of information system at the implementation independent layer of abstraction. Since UML is a de facto industry standard, using it is quite reasonable on the implementation dependent layer of abstraction (Gustas & Jakobsson, 2004). Nevertheless, UML is not provided by any automatic reasoning or inference rules. The presented set of inference rules is critical for inconsistency control on the conceptual layer, because it makes possible to use just one diagram type for semantic definition of the entire system.

Various studies of enterprise engineering problems in different companies and in the public sector have demonstrated that a consistent and integrated conceptual representation is necessary to understand orderly transitional processes from the current to the target enterprise architecture. Information system requirements need to be captured, visualised and agreed upon. Just as the complex buildings or machines require explicit representations of their design structures, so does overall enterprise architecture. The static and dynamic dependencies are not analysed in isolation. Therefore, the semantic modelling process can

have some assistance for the integrity and consistency control across various views and perspectives.

Enterprise models can be represented as ontological descriptions that are defined and published using XML artefacts. Graphical descriptions of various services on the Internet are subject for search, change, analysis and integration (OASIS BCM, 2003) across technical and organisational boundaries. Service representations should include not just software components, but definition of interoperation details among various business actors involved. A self-describing nature of services on the Internet and particularly the ability to define requirements for business collaborations in terms of enterprise models would provide significant competitive advantages. Recent developments in the area of semantic web (Berners-Lee et al., 2001) give us indication that enterprise modelling can provide the automated support needed for e-business integration at the strategic and conceptual layer. It has the potential to reduce web-based system development complexity and costs, to increase e-business re-engineering efficiency and to identify new revenue streams. However, before the collaborative enterprise engineering becomes reality, there are a number of challenging problems that need to be solved. The most important fundamental issues include e-service composition and integration.

The term of ontology is used in many different senses. Many researchers of semantic web would use the notion of ontology to call a diagram, which represents a specification of a conceptualisation. The most typical ontology for the semantic web has quite simple static structure and a set of inference rules. Additionally, e-service architecture should define its actors, concepts, actions and business relationships. Enterprise modelling can be applied as an approach for definition of service oriented architectures. The model is sufficiently rich to express the static and dynamic structure of available services. A communication action is fundamental enterprise modelling construct. It defines a linkage between transitions of objects and physical actors, who typically initiate or are affected by object transitions. Therefore, this construct helps to cross a boundary between two completely different types of conceptual representations. Definition of a communication action in such a way is critical for the ontological definition of services, because the semantic diagrams are complemented with a capacity to define the actor interoperation details.

References

[1] Action Technologies, (1993), Action Workflow Analysis Users Guide, Action Technologies.

[2] Batini, C., Lenzerini, M. & Navathe, B. L. (1986), A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, Vol. 18, No. 4, 323-363.

[3] Berners-Lee, T., Hendler, J. & Lassila , O. (2001), The semantic Web, Scientific American, May 2001.

[4] Booch, G., Rumbaugh, J. & Jacobsson, I. (1999), The Unified Modelling Language User Guide, Addison Wesley Longman, Inc., Massachusetts.

[5] Borgida, A. T. (1984), “Generalisation/Specialisation as a Basis for Software Specification. On Conceptual Modelling”, M Brodie, J Mylopoulos, J W Schmidt (eds.) On Conceptual Modelling, Springer-Verlag, New York, pp.87-112.

[6] Brachman, R. & Schmolze, J. G. (1985), An Overview of the KLONE Knowledge Representation System, Cognitive science, 9(2), pp. 171-212, 1985.

[7] Bubenko, J. A. (1993) "Extending the Scope of Information Modelling", Fourth International Workshop on Deductive Approach to Information Systems and Databases, Polytechnical University of Catalonia, 73-97.

[8] Daconta, M. C., Obrst, L. J. & Smith, K.T. (2003), The semantic Web: A Guide to the Future of XML, Web Serwices, and Knowledge Management, Wiley, Indianapolis.
[9] Dori, D. (2002), Object-Process Methodology: A Holistic System Paradigm, Springer, Berlin.
[10] van Griethuisen, J. J. (1982), Concepts and Terminology for the Conceptual Schema and Information Base, Report ISO TC97/SC5/WG5, No 695.
[11] Goldkuhl, G. (1995), Information as Action and Communication, in Dahlbom, B. (ed.), The Infological Equation – Essays in Honor of Börje Langefors, Göteborg University, Sweden, pp. 63-79.
[12] Gustas, R. (1994), Towards Understanding and Formal Definition of Conceptual Constraints. Information Modelling and Knowledge Bases VI, IOS Press, pp. 381–399.
[13] Gustas, R. (1998), "Integrated Approach for Modelling of Semantic and Pragmatic Dependencies of Information Systems", Conceptual Modelling - ER'98, Springer, pp. 121-134.
[14] Gustas, R. (2000), Integrated Approach for Information System Analysis at the Enterprise Level, Enterprise Information Systems, Kluwer Academic Publishers, pp. 81-88.
[15] Gustas, R. and Gustiene, P. (2002), Extending Lyee Methodology using the Enterprise Modelling Approach, Frontiers in Artificial Intelligence and applications, IOS Press, Amsterdam, pp. 273-288.
[16] Gustas, R. & Gustiene, P. (2004) Towards the Enterprise Engineering Approach for Information System Modelling across Organisational and Technical Boundaries, Enterprise Information Systems V, Kluwer Academic Publisher, Netherlands, pp. 204-215.
[17] Gustas, R. & Jakobsson, L. (2004) Enterprise Modelling of Component Oriented Information System Architectures, New Trends in Software Methodologies, Tools and Techniques, IOS Press, pp. 88-102.
[18] Gustiene, P. (2003), On Desirable Qualities of Information System Specifications, 10th International Conference On Concurrent Engineering: Research and Applications, Madeira Island, Portugal, pp. 1279-1288.
[19] Hoffer, J. A., George, J. F. & Valacich J.S. (2004), Modern System Analysis and Design, Pearson Prentice Hall, New Jersey.
[20] Hull, R., Christophides, V. & Su, J. (2003), E-services: A look Behind the Curtain, ACM PODS, San Diego, CA.
[21] Krogstie, J. (2003), Evaluating UML Using a Generic Quality Framework, UML and Unified Process, IDEA Group, ISBN: 1-931777-44-6.
[22] Leonard, M. (2004), IS Engineering Getting out of Classical System Engineering, Enterprise Information Systems V, Kluwer Academic Publishers, pp. 35-45.
[23] Lindland, O. I., Sindre, G. and Solvberg, A. (1994), Understanding Quality in Conceptual Modelling, IEEE Software, (11, 2).
[24] Martin, J., Odell, J. J. (1998), Object-Oriented Methods: A Foundation (UML edition), Prentice-Hall, Englewood Cliffs, New Jersey.
[25] Maciaszek, L. A. (2001), Requirements Analysis and System Design, Addison Wesley.
[26] OASIS BCM (2003), Business-Centric Methodology Specification, Version 0.10. Available: http://www.businesscentricmethodology.com [accessed June 9, 2005]
[27] Raines, F. D. (1997), Memorandum for the Heads of Executive Departments and Agencies. Available: http://www.whitehouse.gov/omb/memoranda/m97-16.html [accessed June 9, 2005]
[28] Spewak, S. H. (1992), Enterprise Architecture Planning: Developing a Blueprint for Data, Applications and Technology, John Wiley & Sons.
[29] Storey, V. C. (1993), Understanding Semantic Relationships, VLDB Journal, F Marianski (ed.), Vol.2, pp. 455-487.
[30] Stumpf, R. & Teague, L. (2005) Object Oriented System Analysis and Design with UML, Pearson Prentice Hall, New Jersey.
[31] Vernadat, F. B. (1996), Enterprise Modeling and Integration: principles and applications, Chapman & Hall.
[32] Winograd, T. & Flores, R. (1986), Understanding Computers and Cognition: A New Foundation for Design, Ablex Norwood, N.J.
[33] Yourdon, E. (1989) Modern Structured Analysis, Prentice-Hall, Englewood Cliffs, N.J.
[34] Yu, E. & Mylopoulos, J. (1994), ”From E-R to 'A-R' - Modelling Strategic Actor Relationships for Business Process Reengineering”, 13th International Conference on the Entity - Relationship Approach, Manchester, U.K.
[35] Zachman, J. A. (1996), “Enterprise Architecture: The Issue of the Century”, Database Programming and Design Magazine.