

TOWARDS THE ENTERPRISE ENGINEERING APPROACH FOR INFORMATION SYSTEM MODELLING ACROSS ORGANISATIONAL AND TECHNICAL BOUNDARIES

Remigijus Gustas, Prima Gustiené
Information System Department, Karlstad University, Sweden
Email: Remigijus.Gustas@kau.se, Prima.Gustas@kau.se

Keywords: Enterprise engineering, semantics, pragmatics, technical and organisational system components

Abstract: Enterprise Engineering proved to be useful when a generally accepted intentional description of information system is not available. A blueprint of enterprise infrastructure provides a basis for system analysis of the organizational and technical processes. It is sometimes referred as enterprise architecture. The major effort of this paper is the demonstration of how to bridge a gap among various levels (syntactic, semantic and pragmatic) of enterprise engineering. Most information system engineering methodologies are heavily centred on system development issues at the implementation level. Thus, such methodologies are restrictive in a way that a supporting technical system specification can not be motivated or justified in the context of organizational process models. Enterprise models provide a basis for gradual understanding of why and how various technical system components come about. Some undesirable qualities of enterprise engineering are sketched in this paper.

1 INTRODUCTION

Business process improvement is a major concern for today's organisations. The inter-organisational parts of businesses are particularly problematic for change management. Enterprise engineering deals with modelling and integration (Vernadat, 1996) of various organisational and technical parts of business processes. The concept of enterprise in the context of information system development denotes a limited area of activity in the organisation (Bubenko, 1993) that is of interest by a system analysis expert. At the same time, enterprise engineering can be viewed as an extension and generalisation of the system analysis and design phase. Enterprise modelling should take place during the early, middle and late information system development life cycle. The most difficult part of enterprise modelling is to convert fuzzy requirements into a coherent, complete and consistent specification of the desired information system.

Traditional methods of information system analysis and design are based on the idea of a separation of concerns, which are represented by using various static and dynamic diagrams. Although there is a great power of traditional approaches, there is also a

deep fallacy in this orientation. Semantic diagrams of traditional methods usually do not take into account interdependencies of various diagrams as well as the communication dependencies among organisational and technical system components. In industry, this often results in huge financial resources being wasted for IT system development without a significant impact. One of the major reasons of such failures is a communication gap between the management and IT development personnel. Many software maintenance problems arise because every enterprise has to survive a systematic change when new software is introduced.

Various studies of change management problems in different companies and in the public sector have demonstrated that an explicit representation of the organisational and technical system infrastructure is necessary to understand orderly transformations of existing work practices. If the initial software requirements are presented without taking into consideration a context of the organisation, then the requirements engineering approach can not be considered as viable, because it is not able to address the issues of software quality and IT fitness. A blueprint of the enterprise infrastructure provides a basis for the organization's information technology planning. It is referred to as enterprise architecture (Zachman, 1996). Just as the complex buildings or machines require explicit representations of design

structures, so does an invisible enterprise infrastructure. It needs to be captured, visualised and agreed upon (Gustiene & Gustas, 2002). There is no way to study the fitness of the organizational and technical process without taking into account an infrastructure of the existing system prior to the designing a new one.

R. L. Ackoff has identified various causes of misunderstanding (Ackoff, 1989) among information system developers and system users. The condition, under which an information system developer knows what information is needed for the user, is when he has a complete understanding of how an overall system is functioning. It is not so easy to attain such understanding, if developers do not know what organisational and technical system components are and how they are supposed to interact. The trouble is that without complete understanding of organisational architecture, stakeholders have no criteria for determination the irrelevance or ambiguity of the information flows. That is why users frequently are misunderstood or they are provided by ambiguous, incomplete and inconsistent solutions.

Misunderstanding between information system users and designers is a common problem. Precondition of a successful requirement engineering process is a mutual understanding. It can not be achieved without close cooperation of both parties. The problem is that the users do not want to be the software engineering experts. On the other hand, the designers have difficulties to understand the organizational system requirements in which the supporting software system is supposed to operate. According to the enterprise modelling approach (Gustas, 2000), one of the problems in most of conventional software development approaches is that the implementation level requirements can not be represented without taking into consideration the organisational requirements, because they are not able to address some extremely important issues of software quality and IT fitness.

Organisational systems must adopt to fast changing conditions of environment in order to survive. Thus, they are in the process of permanent change. One of the common software development failures is that, by the time the information system application is produced, quite often it is no longer relevant to the existing work practices (Zachman, 1996). Software system problems very often occur for a simple reason that various applications do not fit or they can not support the organisational components as it was anticipated. There are some important attempts to overcome the software component fitness and integration problems by designing software in a non traditional way. The Lyee methodology (Negoro, 2001) is a way for building software on a basis of various logical

structures of layouts (Gustas & Gustiene, 2002) by which information system users are supported in performing their tasks.

Every enterprise has to survive a systematic change at a time when the new software is introduced. Various studies of change management problems in different companies have demonstrated that the explicit representation of the business infrastructure is a necessary condition to understand orderly transformations of their work practices. They need to be guided to achieve a mutual agreement on how the logical design should look like. Such agreement can not be reached without a complete understanding of how overall information system is working.

Many customer organisations have information technology (IT) experts that are able to present various screen, printout and data file layouts to be implemented. Nevertheless, in many cases logical design structures can not be identified without a complete understanding of how an overall information system is working. A long list of software system development failures demonstrates that in many cases such understanding is not available. Furthermore, the customers have great difficulties to capture and to communicate unambiguously (Gustiene & Gustas, 2002) information system requirements by using natural language.

The paper is organised as follows. In the next section three levels of enterprise engineering are shortly introduced. The third, fourth and the fifth section defines the basic elements and dependencies that are used at the pragmatic, semantic and syntactic level. The desirable qualities of enterprise engineering are discussed in the sixth section. The conclusion section outlines the perspective of enterprise engineering approach and the future work.

2 THREE LEVELS OF ENTERPRISE ENGINEERING

Small changes in the organisational infrastructure are sometimes introduced so fast that software system components tend to become obsolete very quickly. One of the major problems of enterprise engineering is that comprehensible analysis methods across organisational and technical system boundaries are not available. If the requirement engineers would like to understand why a technical system component is useful and how it fits into the overall organisational system, then at least three levels of information system models are necessary for maintenance of a systematic change:

- The pragmatic level,
- The semantic level,

- The syntactic level.

These levels can be viewed as three dimensions of requirements engineering (Pohl, 1993). The agreement dimension is dealing with the pragmatic aspects of change analysis, the representation dimension – with the semantic aspects of system analysis and the specification dimension – with the implementation oriented system design aspects.

The most abstract is the strategy-oriented business process analysis level, which sometimes is referred to as pragmatic level. Strategic models are useful for illustration of the actual architectural solutions and general communication infra structure. They are necessary to provide motivation behind new business solutions that can be expressed in qualitative and quantitative terms. The semantic level must have a capacity to describe clearly the static and dynamic structures of business processes across organisation and technical system boundaries. The syntactic level should define implementation-oriented details, which explain the data processing needs of a specific application (Davis & Olson, 1985) or software component.

Most software methodologies are heavily centred on the implementation level. Thus, they are restrictive in a way that a supporting system specification can not be motivated or justified in the context of organizational process models. The consequence is that to apply these approaches in some areas such as electronic commerce is not so simple. One of the problems is that business processes are spanning across organisational and technical system boundaries. These boundaries are not always clear. They are changing over time. If boundaries are not clearly identified and mutually agreed, then the result would be a misunderstanding (Gustiene & Gustas, 2002) between system users and designers.

The pragmatic level concentrates on a strategic description, i.e. it is supposed to give a definition of the "why" - a long term intention or a vision of the enterprise under development. A pragmatic description motivates various enterprise components at the semantic level. Sometimes, desired software components can be easily described before the goals are well understood. Elaboration of the objectives is then done backwards, by asking for the reason for existence of the introduced components. Information system methodologies recognise that it is not enough to concentrate distinctly on one of the levels.

The qualitative criteria of enterprise engineering are essential for understanding various issues of the organisational fitness. Nevertheless, it is rather unusual to see the quantitative aspects taken into account in the traditional approaches of information system analysis and design. The quantitative criteria play an important role in order to assess the pragmatic value of a new organisational solution.

The quantitative terms such as cost, revenue and profit are used in PENG method (Dahlgren et al., 2000). These notions are important to understand the economic value of the new business process. It should be noted that the foundation for system assessment in qualitative terms in PENG is missing.

In the enterprise engineering approach a communication paradigm-based modelling foundation (Gustas, 1997), (Gustas, 2000) is used that has been proved to be useful for representation of the qualitative aspects of business processes across technical and organisational boundaries. Currently, we are aiming at the extended enterprise modelling approach with an engineering process that is similar to what architects use when constructing buildings. The graphical models at various levels of abstraction are used for visualising and reasoning about information system quality. The ultimate goal of the enterprise modelling is to introduce a common basis for integration of various dependencies (Gustiene & Gustas, 2002) that are used in requirements engineering and conceptual modelling (Roland & Prakash, 2001) at the syntactic, semantic (Storey, 1993) and pragmatic level.

One of the most important problems of software engineering is to maintain the existing software. This problem is due to the fact that software development methods have adopted a purely process-driven or purely data-driven approach. During the last years a number of new methods have been developed, but still all these methods suffer from several drawbacks. Most of the methods use a mix of the natural language and semiformal graphical notations for the purposes of software requirement specification. The natural language comments quite often appear ambiguous. It causes semantic problems of misunderstanding (Gustiene & Gustas, 2002) among system users and system developers. Even though these problems were taken into consideration by the more powerful modelling techniques (Yourdon, 1989), (Booch et al, 1999), it did not help to control the integrity and consistency of specifications that are presented on various levels of abstraction.

Another big problem is the implementation bias of many information system modelling techniques. The same concepts have been applied to the design and analysis stages, without rethinking these concepts fundamentally. Modelling at the semantic level should follow the basic conceptualisation principles (Griethuisen, 1982). Only conceptually relevant aspects, both static and dynamic, should be included, thus excluding aspects of physical implementation.

Many methodologies lack a systematic approach to the assessment of software quality and change management. The idea of model-driven approach (Snoeck et al., 1999) is to provide solutions for

solving such problems. Model-driven software development follows the idea, pioneered by Jackson, that the complexity of information system is driven by the complexity of the underlying world (Jackson, 1983). By analysing and modelling a so called enterprise infrastructure, not concentrating on the specification of the desired functionality, information system developers are able to better manage the complexity of a developed system.

Similarity of model-driven approach to most classical development methods is that the enterprise model, which is constructed in the early stage of software development, is an abstraction of what the desired system must do, but not how (Snoeck et al., 1999). Most current software development methods are centred on how a software application needs to be implemented. The separation between the pragmatic, semantic and syntactic - implementation based model leads to a natural division of enterprise engineering products into three different representations. It is more reasonable to conceptualise enterprise architecture and business processes before supporting software system is defined (Checkland, 1981). It seems not useful to model the real world having no functionality in mind. Hence, all three levels of models should contribute to the extended methodology of enterprise-wide engineering technique.

Information system semantics can be defined by using the static and dynamic dependencies of various kinds. Most of the semantic diagrams are based on the entity notations that are provided by several links. Links are established to capture semantic detail about various relationships among concepts. The ability to describe information system in a clear and sufficiently rich way is acknowledged as crucial in many areas. Typically semantic dependencies are defining semantics in different perspectives such as the "what", "who", "where", "when" and "how" (Zachman, 1996). For instance, in the object oriented approach (Martin & Odell, 1998) the "what" perspective is defined by using the class diagram, the "how" perspective can be defined by using the activity diagrams and the "when" perspective can be described by the state-transition diagrams.

3 PRAGMATIC LEVEL

Humans, technical systems or organisations can be thought as actors. Communication dependencies between actors involved describe the "who" perspective. Dependency link between two actors (agent and recipient) indicate that one actor depends on another actor. An agent can be an actor who is able to initiate an action to achieve his goal. A recipient is an actor, who is dependent on the action

performed by an agent. An instance of actor can be an individual, a group of people, an organisation, a machine, a software component, an information system, etc. 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 of a business process. Graphical notation of the actor dependency between an agent and a recipient is presented in Figure 1.

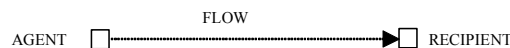


Figure 1: The Flow dependency.

If an actor B is dependent on A on some flow, then such dependency is represented as $A \text{> } B$. A dependency link denotes that a recipient depends on an agent for some information, physical flow or an action. If actor B is dependent on A for a flow F, then this dependency is specified by the following expression: $A \text{> } (F) \text{> } B$. The flow dependency represents a communication channel for transferring of information or physical flow between agent and recipient.

A typical action workflow loop (Action Technologies, 1993) includes two communication flows sent into opposite directions. An agent is an actor who initiates the work flow loop to achieve his goal. A recipient of a communication flow can be viewed as an agent in the next communication action. Actor dependencies in two opposite directions imply that certain contractual relationships are established. For instance: If Person> (Trip Requirements)> Travel Agent then Travel Agent> (Trip Reservation)> Person. Various actor communication dependencies at a strategic level are illustrated in Figure 2.

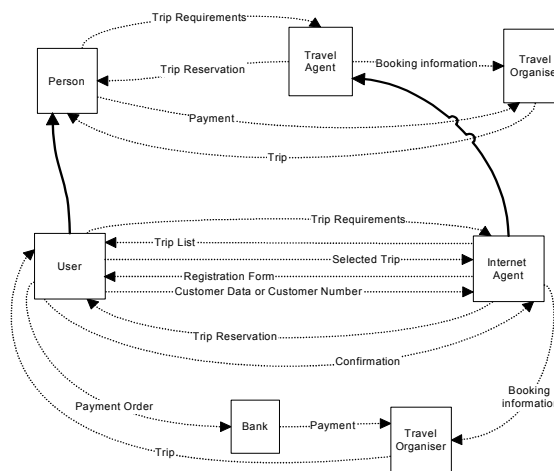


Figure 2: Basic communication flows among actors.

In this example, we can see various information flows among the actors such as User, Internet Agent, Bank and Travel Organiser. The middle part of this diagram illustrates the basic communication flows between User and Internet Agent in the process of making trip reservation by using the Travel Agent's web site. This part represents a simplified process of buying trips on the Internet. A lower part of the diagram demonstrates that the Bank is involved to handle a payment. A Travel Organiser delivers the actual Trip when the payment for the trip is made. A higher part of the diagram demonstrates the core communication loops in the traditional business process. This part represents a non-electronic way of buying trips. It can be overridden by the more specific interactions, should a person take a more specific role of a user or an agent would play a role of Internet Agent. It is not clear at this moment what kind of organisational or technical components in reality can be used to carry out this business process. This diagram represents just the actors involved and strategic dependencies among them. In general, the actor dependency links describe various ways of how agents can affect the recipients. At the semantic level, the actors can be decomposed or specialised by using dependencies of other kinds. For instance, the inheritance link was used to indicate that a User is more specific than Person and a Travel Agent is more general than Internet Agent.

If the communication flow is represented by a material object, then such dependency represents changes of the ownership right for this specific instance of a flow. Before sending it, an AGENT owns the FLOW and later, dependent on whether the flow would be accepted, the ownership right is transferred to a RECIPIENT. Recipient by depending on agent is able to achieve his goals. It should be noted that very often he is not able to achieve them alone. At the same time, a recipient becomes vulnerable to the failure of agent. If agent fails to deliver the flow, then a recipient would be unfavourably affected in the capability to reach the goal (Yu & Mylopoulos, 1994).

Goals of various organisational components stimulate interaction among actors, leading to some further interactions (Warboys et al, 1999). Any actor can achieve a goal by avoiding a problem. The pragmatic dependencies are used to define various intentions of actors. The goal, opportunity and problem dependencies can be used to refer desirable or not desirable situations. The basic pragmatic dependencies of the enterprise modelling approach are represented in Figure 3.

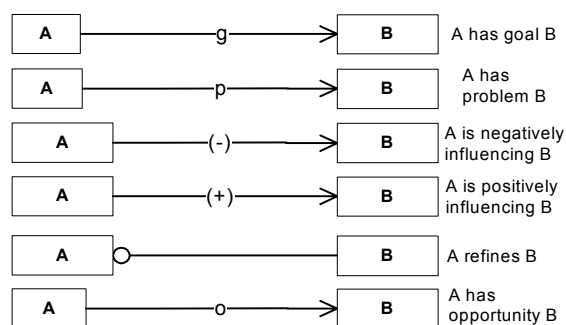


Figure 3: Notation of the pragmatic dependencies

Goals or problems can be decomposed by using a refinement link. It should be noted that the interpretation of some software requirement or situation as a problem, opportunity and goal is relative. The achievement of some goal by one actor can be regarded as a problem for another actor. Negative influence dependency (-) and positive influence dependency (+) are used to indicate influences. Negative influence dependency from situation A to B indicates that A can be regarded as a problem, because it hinders the achievement of goal B. The positive influence dependency from A to B would mean that A can be viewed as an opportunity in the achievement of goal B. A small example of the pragmatic dependencies is illustrated in Figure 4.

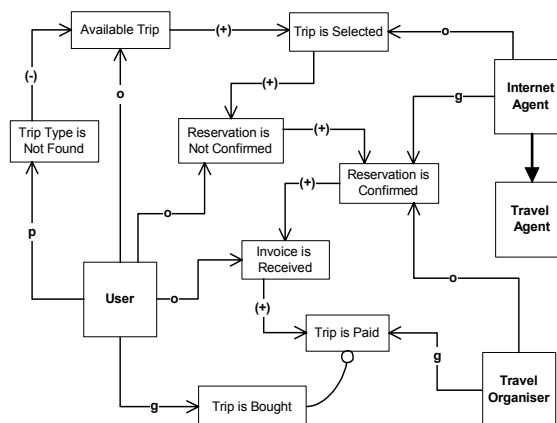


Figure 4: Illustration of the pragmatic dependencies.

The illustrated pragmatic dependencies were defined for the same process of buying trips on the Internet. These dependencies demonstrate how actors can affect each other by using their opportunities or achieving some goals. For instance, if Travel Organiser could reach the goal 'Trip is Paid', then it would automatically cause the achievement of User's goal 'Trip is Bought'. The 'Trip is Paid'

state is positively influenced (+) by 'Invoice is Received', which is viewed as an opportunity for User.

The Internet Agent box at the syntactic level will play a role of software component. We are not going into details of some pragmatic dependencies, because of the space limitations. A discussion on this issue can be found at (Gustas, 1997), (Gustas, 1998).

4 SEMANTIC LEVEL

The semantic dependencies in the enterprise modelling are of two kinds: static and dynamic. Descriptions of organisational *activities* as well as *actors* involved in these activities are based on the dynamic dependencies. So, the dynamic part of the enterprise model can be represented by actions that are using and producing various communication flows and by actors that are responsible for initiation of those actions. The dynamic relations are state dependencies and communication dependencies. The communication dependencies among enterprise actors are relevant for description of the "who" perspective. It is based on the communication action tradition in information system development that is stemming from the Scandinavian approaches. During enterprise engineering process, the actor dependencies at the semantic level are refined into the action and state transition links.

Any flow dependency at the semantic level should be described in a more detail by using a communicative action. Therefore, the actor dependency is considered to be an action and a communication flow (Goldkuhl, 1995). Cohesion of action and communication flow at the semantic modelling level results into a more complex abstraction, which is entitled to as a communicative action (Gustas, 2000). In such a case, the flow dependency link between two actors specifies that the recipient depends on the agent not just only by the specific flow, but by the action as well. Actions will be represented by ellipse. Graphical notations of dynamic constituents in the extended communication flow and action dependency are represented in Figure 5.

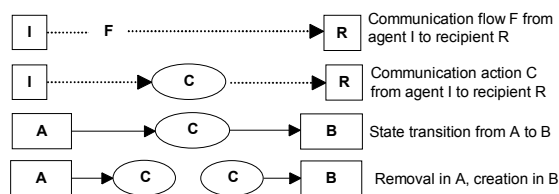


Figure 5: Graphical notation of the dynamic dependencies.

State dependencies define semantic relationships between states of actions. They are normally considered as the "how" perspective. Both state transition and communication dependencies describe a very important part of knowledge about business processes. Unfortunately, many communication approaches often neglect some behavioural aspects of the state transition and vice versa, many software engineering approaches disregard the dependencies of communication.

The static concept dependencies are used for the specification of attributes for various states of processes. They define the "what" perspective. These dependencies are stemming from various semantic models that are introduced in the area of information system analysis and design (Martin & Odell, 1998). Graphical notations of static dependencies are represented in Figure 6.

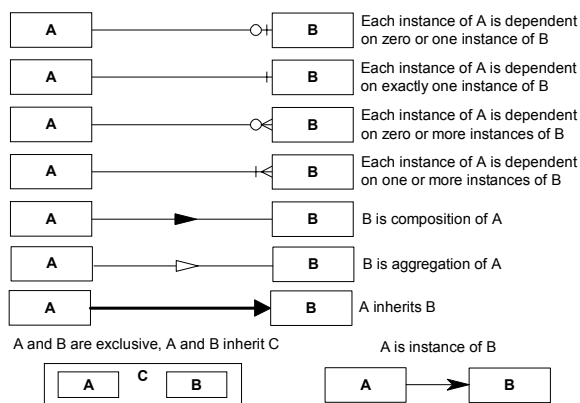


Figure 6: Graphical notation of the static dependencies.

The similarities can be shared between concepts by extracting and attaching them to a more general concept. In this way, all kinds of static and dynamic dependency links can be inherited by several concepts. Composition or aggregation dependency is useful for formation of a new concept as a whole from other concepts that might be viewed as parts.

Dynamic semantic dependencies are used to define relations between different actors, their actions and communication flows. If concept A is connected to B by a communication dependency, then A is an agent and B is a recipient. Depending on whether there is or there is no physical flow component in the action, the communication flows can be information, material or decision. The extended communication loops between User and Internet Agent in the initial part of process of buying trips on Internet are represented in Figure 7.

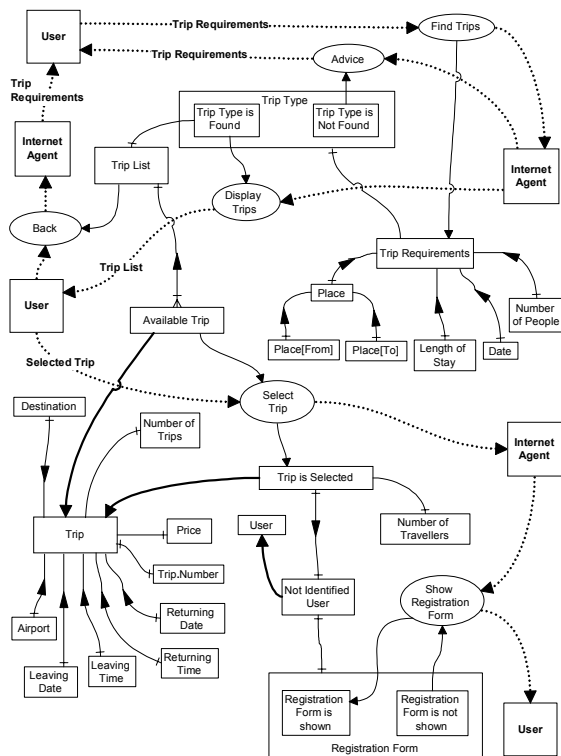


Figure 7: Reservation process at the semantic level.

It should be noted that the presented diagram is incomplete, but its communication dependencies and states are consistent with the previously defined basic constraints (see the diagrams at the pragmatic level). When Find Trips action is executed, then two outcomes are possible. If the desired 'Trip Type is Found', then a 'Trip List' object is created. It consists of 1 or many available trips (see the state 'Available Trip' at the semantic level and the opportunity of user at the pragmatic level). The trip list will be displayed in the next action. If the desired 'Trip Type is Not Found', then User is advised to start the process from the beginning and initial Trip Requirements are displayed. An available trip should satisfy the specific constraints (they are not presented in this paper). If trip is available, then a user can select a trip. If so, then the registration form will be shown to enter or verify user's identity data.

Any communication loop is able to change the static associations between instances. State changes are important to both actors. Without the ability to represent noteworthy state changes, we would have difficulties to understand the rational and effect of every communication loop. Actions express the permissible ways in which state changes may occur. These changes are specified by using the integrated dependency of communication and state transition (Gustas, 2000).

5 SYNTACTIC LEVEL

The enterprise modelling language should be able to represent graphically business processes across the organisational and technical (hardware and software) system boundaries. The outcome of the enterprise modelling effort can be defined in terms of components and their interfaces with the structural definition of the printout, message, screen and file layouts. These layouts can be viewed as syntactic elements of the actor oriented diagrams that define existing or expected communication infrastructures to support various actors involved. Enterprise models suggest a different way of defining the collaboration infrastructures that is based on an intentional way of thinking and on a new constructive way of reasoning.

Modelling primitives of the syntactic level are dependent on the tools, which are used in the software development process. For instance, the enterprise modelling approach that was specially designed for the Lyee methodology (Gustas & Gustiene, 2002) was build on the basis of syntactic elements that are represented in Figure 8.

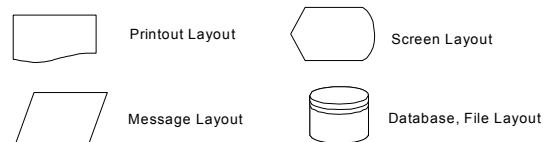


Figure 8: Basic syntactic elements.

These syntactic elements are considered as the implementation perspective or CASE tool dependent. For example, database relations are typical representatives of the syntactic level. File or database layouts can be defined by using the conventional programming languages or traditional database definition languages. Almost the same set of syntactic elements could be used for the object-oriented approach. In this case, the notion of a file layout should be replaced by the notion of class layout (it would contain the operations as well).

Enterprise actors, which at the semantic level are represented by square boxes, at the syntactic level can be thought as humans or technical components. Communication flow dependencies among actors together with the actor composition or generalisation links might describe enterprise infrastructure at the syntactic level. It must be consistent with the previously described the "who" perspective. The technical components can be viewed as hardware (machines, computers, their networks) or software components. The generalisation and aggregation hierarchies of actors in terms of technical components that are used in different communication loops would motivate a relevant hardware or soft-

ware system architecture. Typical instances of the actors in the area of information system development are represented by Figure 9.

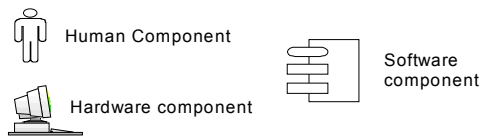


Figure 9: Human and technical system components.

The presented list of the component classes is not exhaustive. Other types of icons such as fax machines, phones, computer networks, etc., can be introduced on demand. These components in combination with the basic layouts can be used as the main building blocks for representation of the technical deployment architecture, software component infrastructure or organisational dependency structure.

From the software component designer perspective, the definition of the software system architecture is not sufficient. The designers need to understand a technical system architecture and organizational infrastructure, where application is going to be installed. Therefore, the enterprise modelling approach focuses not just on the pragmatics and semantics, but also on the software deployment architecture. A small example for illustration of technical deployment details of two software components in the first interaction loop (see semantic level) is represented in Figure 10.

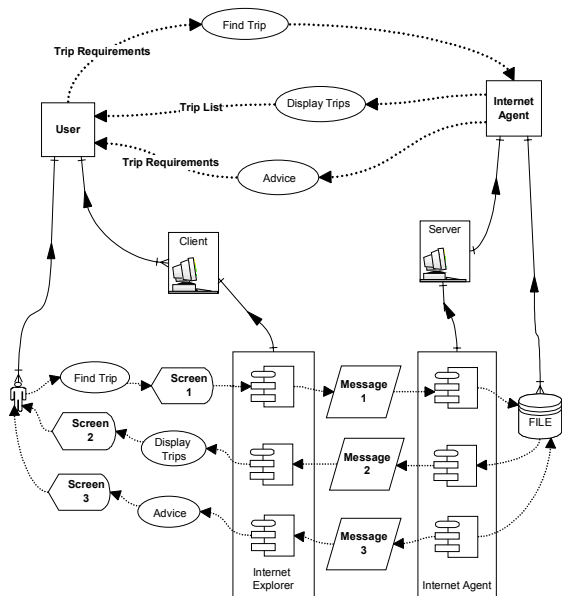


Figure 10: Human and technical system components.

This diagram illustrates that the Internet Agent software component is deployed on the Server and

the Internet Explorer will be used as a client. They are supposed to communicate via screen and message layouts that must have the identical underlying semantic structure as information flows such as Trip Requirements and Trip List. The deployment diagram represents a very important technical system requirements that should be communicated to a software engineer (designer as well as software builder) before a specification of the represented software system components commits.

The communication flow structures with the action names and the generalised state structures define a foundation for a specification of file and screen layouts. For instance, the concepts that are dependent of Trip (see the semantic diagram) constitute the foundation of the file layout that can be represented by the following relation:

TRIP (Trip Number, Destination, Airport, Leaving_Date, Leaving_Time, Returning_Date, Returning_Time, Number_of_Trips, Price).

When Find Trip action is receiving 'Trip Requirements' from a User, a Trip List object may be created, which consists of 1 or many available trips. If the desired 'Trip Type is Found', then the Trip List can be displayed. The flow of 'Trip Requirements' with all possible action names that could affect a course of the illustrated process, constitute the basis for definition of the first screen layout. It will be identified by Screen 1. Representation of the first screen layout at the syntactic level is represented in Figure 11.

Figure 11: Screen layout (Screen 1) of Trip Requirements.

The set of the enterprise modelling dependencies at the semantic level is a subject to the refinement into the implementation dependent structures. It should be noted that the syntactic diagrams are constrained by the dependencies at the semantic and pragmatic levels. The implementation dependent file, message, printout or screen layouts are supposed to define the operating infrastructure of software components. The example of syntactic

diagram that is defined for the Internet Agent (IA) software component is represented in Figure 12.

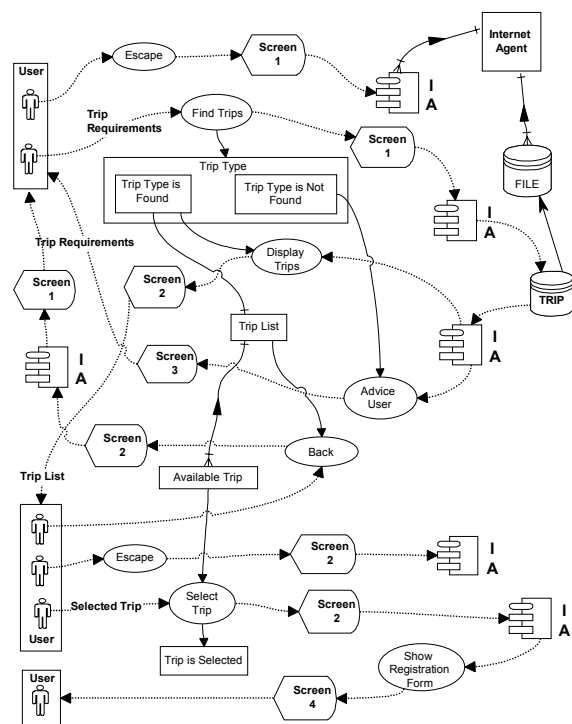


Figure 12: A syntactic diagram of the IA (Internet Agent) software component.

This diagram illustrates how the enterprise model at the semantic level is going to be implemented. Various previously defined semantic dependencies have to be taken into account by a more specific level. It means that the syntactic dependencies must be consistent with the higher levels of abstraction. A special inference rules can be defined in order to validate the semantic consistency. Thus, the enterprise engineering approach can be used as a uniform basis of reasoning about modelling quality at the pragmatic, semantic and syntactic quality.

6 DESIRABLE QUALITIES OF REPRESENTATIONS

Various diagrammatic constructions that humans employ for representation of information system solutions on different levels of abstraction can be evaluated by certain criteria. It means that 'good' diagrams should possess some important qualities. Such qualities are still poorly understood in the area of information systems. Nevertheless, these qualities

are essential when the enterprise engineering product is intended for effective communication of various architectural solutions among system designers and system users.

Lindland, Sindre and Solvberg (Lindland et al., 1994) has proposed a framework for understanding the syntactic, semantic and pragmatic quality. The syntactic quality can be characterised by correctness of modelling language. All constructions of diagrams have to be defined according to the syntax. Two characteristic features of the semantic quality are validity and completeness. Validity means that all statements in the model are relevant to the problem. Completeness means that the model contains all relevant statements. It is not easy to satisfy these two requirements, because we do not know how the semantic quality can be measured. A better pragmatic quality of system specification would mean that all concerned participants understand how the intended system is going to function and all stakeholders agree on what is going to be achieved. Misunderstanding and disagreement on some issues would automatically imply a lower pragmatic quality.

Identification of undesirable characteristics is critical at the semantic level. Semantic representations are defining and constraining the diagrams at the syntactic level. They should also support the diagrams that are defined on the pragmatic level. On the other hand, pragmatic representations should justify enterprise models at the semantic and syntactic level. The undesirable qualities of enterprise engineering at the semantic level are as follows: semantic inconsistency, semantic incoherence, semantic redundancy, semantic incompleteness and ambiguity.

The enterprise models can be used as a uniform basis of reasoning about inconsistency of business process diagrams that are defined on the various levels of abstraction. It is not so easy to attain semantic consistency for the reason of a natural variation between views to the same business activity. Consistency between more specific and more general business process diagram can be controlled on a basis of a special set of inference rules (Gustas, 1997). Inference rules may help to discover inconsistencies between dependencies on different levels of abstraction. According to this approach, a more abstract level is constraining a more specific one. This would imply that a more abstract diagram is a composition and generalisation of a more specific diagram.

Semantic differences of conceptual views are typically introduced by different visions of stakeholders. Inconsistencies between constructs of diagrams that are represented on two neighbouring levels of abstraction indicate that two diagrams are

incompatible. This situation is referred to as inconsistent. Inconsistency demonstrates that the semantic descriptions of enterprise components are contradictory. Consistency issues have to be resolved before the engineering of any software component commits. Introduction of new organisational or technical component might be a reason of the new consistency problems that need careful analysis. Software component fitness to the existing work practices is one of the main issues that can be studied by using enterprise modelling.

Semantic incoherence can be defined as opposed to minimality (Weber, 2001). Minimality means that at some time during the life cycle of subsystem, all state variables are 'used'. Incoherence analysis in the enterprise engineering is questioning the integrity between computerised information system state change and communication flow internal structure. Any parameter of a communication flow is supposed to be used for some purpose. It might be a creation of object or deletion of object in some state. A communication action of removal in a specific state must destroy all association that are relevant for this state. A creation action must establish all relevant associations that are specified in the diagram, otherwise the action is rejected. Some actions can create and destroy associations at the same time. Such internal changes can be followed according to the static dependencies that are defined in various states (Gustas, 1998).

Communication flow parameters can be consumed or emitted by a predefined action. A consumption action is supposed to allocate parameter values into the correspondent state attributes. Every consumed parameter has to be defined in a poststate of action. If a flow parameter is matching different name of the attribute, then an explicit computation rule for the attribute must be defined. Usage of any attribute should be questionable with respect to an overall structure of incoming flow. If an attribute is irrelevant for the action, then it should be removed. Any object can be terminated by a removal action. If so, then no attributes should be specified. Termination process can be represented by an action without a resulting state. In this case, no commitments of actors should be pending in connection to this state. A properly designed termination action should always emit information flow that is resulting from the deleted attributes values. Such flow can be viewed as a confirmation message to the initiator of action.

Semantic completeness of actions in all resulting branches is always relative. It is difficult to specify all imaginable alternatives of business process chains. However, it is rather easy to control a set of all final (desired) states for each actor. Each final state must be consistent with respect to some actor's

goal. Goals define system boundary and therefore may help designers to control a relative pragmatic completeness of a specific business process. A complete business process should consist of coherent actions and all interaction loops in a complete process must terminate. All final states must be desired by at least one of the actors. Besides that, all communication flow parameters should be consumed in action, otherwise they are not useful. If the final states are not desired by any actor or initial states are not problematic to any actor, then this would indicate overspecification or redundancy.

Semantic redundancy in general terms would mean that two different sets of semantic dependencies are used at the same time to express the same meaning. A special set of inference rules (Gustas, 1997) can be used to discover redundancy. If the inferred dependencies are represented in a same diagram, then they would indicate a situation of semantic redundancy. Semantic redundancy creates unnecessary 'noise', which is hindering effective communication among the members of enterprise engineering team. In traditional methods of information system development, structurally different, but semantically equivalent representations (Batini et al, 1986) is a reason of painful problems (Gustiene & Gustas, 2002)

Semantic incompleteness can be opposed to the characteristics of losslessness (Weber, 2001) and cohesion (Amber, 2001). Higher quality of semantic representation should preserve all emerging state dependencies and relevant heritable dependencies. Semantically incomplete representations of business processes can be characterised by the presence of semantic holes (Gustas, 1994). The semantic holes typically appear when a so called not totally applicable dependency is used. A totality of the static dependency would indicate that for any instance of one concept there is always an association to an instance of the dependent concept. Totality of the state transition dependency means that any object, which belongs to a current state, is applicable for the specified transition link. The totality of the actor dependency would mean that any instance, which is classified as an actor, is privileged to affect a dependent actor by initiation of the specified action, if certain preconditions and postconditions hold.

Ambiguity of concepts in the enterprise models indicates a low semantic quality of system specification. Very often it is the reason of incompleteness and poor understanding of what is going on in the business process. Sometimes, experienced system analysts can reason about concept dependencies even when various static and dynamic constraints are not precise. They also play out scenarios with the objects of ambiguous concepts and actively engage in analysis of semantic

dependencies between actions and actors involved. Some conceptual operations (Gustas, 1997) that are introduced in the enterprise modelling can be used as a means for clarification of meaning. These operations are useful to refine semantics of the compound natural language expressions. Compound concepts are introduced by different stakeholders in various stages of information system analysis. Very often ambiguous concepts contain semantic holes, which can be eliminated by using a special set of conceptual operations.

7 CONCLUDING REMARKS

We expect that the enterprise models can be used as a core method to analyse the rationale of the new organisational solution prior a new supporting IT system is introduced. It might help managers and IT experts to define, visualise and to assess various organisational changes by using a fully graphical approach to business process reengineering. This would in turn facilitate justification and motivation of software components that are used to support work of organisational actors involved in various business processes. Enterprise models can be considered as a corporate resource in diagnosing potential problems, these models are crucial to enable reasoning about business process integrity and the purposeful implications of an organisational change.

Analysing systems means creation of the shared set of concepts. Concepts must form a basis of communication among system analysts. To communicate effectively in a process of enterprise engineering, the experts must agree on mutually acceptable concepts that can be shared. A common situation in system engineering is one that refers to the semantically similar concepts. Similar concepts are represented by the same class of objects, yet these concepts do not share the same meaning. Typically, ambiguous concepts are represented by the long natural language expressions. Refinement of ambiguous concepts during the enterprise modelling process might help to sort out some problems of misunderstanding (Gustiene & Gustas, 2002). One simple approach to identify ambiguity problems in the enterprise models is based on the examination of semantic holes. Ideally, the analysts should insist on knowing the complete dependency set of every concept that is used in a semantic diagram.

System analysts in the area of enterprise engineering define first a very general and ambiguous business process scenario at the pragmatic level. Later, they gradually extend it by making a whole lot of assumptions into a well-defined and well-

understood information system representation on the semantic level. On a way to unambiguous business process description, the analysts integrate various parts of specifications and play scenarios with objects of various types in order to validate and justify technical system components at the syntactic level. Semantically clean process description should consist of actions that are defined in terms of unambiguous concepts. The ultimate goal of the described ways for semantic quality improvement is to facilitate a process of enterprise engineering.

Enterprise models can be used for change analysis in a systematic way on a basis of the graphical representations that are defined for both traditional and electronic business. Our expectation is that enterprise modelling might help us to find solutions for the following difficult problems:

1) Ambiguity problem. Systems are spanning across organisational and technical system boundaries. These boundaries are changing over time and not always clear. That is why sometimes requirements specifications are ambiguous.

2) Integration problem. Various models are used for representation of static and behavioural system aspects. A clear way of integration between these models is missing. Enterprise models provide a very comprehensible foundation to understand the interplay between various syntactic, semantic and pragmatic dependencies.

3) Consistency problem. The same reality can be perceived in a number of ways and therefore intentions can be objectified on the various levels of abstraction. Consistent way of dealing with the pragmatic, semantic and syntactic representations of requirements is missing in most information system methodologies.

4) Completeness problem. There are no stopping rules for a process of information system requirement analysis. Pragmatic descriptions might suggest a way of dealing with the over specification at the semantic and at the syntactic level.

5) Change problem. Every new solution can be considered to be a symptom for a new problem. Change management problems can be tackled in a systematic way by using the extended approach.

Software systems should support business processes and software should be regarded as a value-added technology. If so, the technical processes must fit various organisational processes. Change management in the organisational part or in the technical (software) part of the system is a big challenge, because even a simple deviation from the traditional business practice may be considered as a symptom for a new problem. The key issue is determination of the true IT needs and how these needs are integrated into the overall organisational system.

Underlying communication dependencies between various actors play an important role in understanding of various e-business processes on a pragmatic level. A similar view is taken in some business process modelling approaches that are based on the communication paradigm (Winograd & Flores, 1986). A prerequisite of understanding between different groups of people and organisations is that business process components are structured on a basis of viable communication activities. Information system development methods that are able to adopt the communication paradigm can revolutionary change a way in which systems are analysed today. The new principles can not only improve communication between information system developers and users, but also might facilitate a mutual understanding among various stakeholders.

REFERENCES

- Ackoff, R. L. (1989) "From Data to Wisdom: Presidential Address to ISGSR, June 1988", *Journal of Applied System Analysis*, Vol. 16, 3-9.
- Action Technologies. (1993) *Action Workflow Analysis Users Guide*, Action Technologies.
- Ambler (2001) *The Object Primer – The Application Developer's guide To Object Orientation and UML*, Cambridge University Press.
- Booch, G., Rumbaugh, J. & Jacobsson, I. (1999) *The Unified Modelling Language User Guide*, Addison Wesley Longman, Inc., Massachusetts.
- Bubenko, J. A. (1993) "Extending the Scope of Information Modelling", *Fourth International Workshop on the Deductive Approach to Information Systems and Databases*, Polytechnical University of Catalonia, 73-97.
- Checkland, P. B. (1981) *Systems Thinking, System Practice*, Wiley, Chichester.
- Dahlgren, L E, Stigberg, L and Lundgren, G (2000), *Öka Nyttan av IT*, Ekelids förlag (in Swedish).
- Davis, G. B. & Olson, M. (1985) *Management Information Systems*, McGraw Hill, New York.
- Griethuisen, J. J. (1982) *Concepts and Terminology for the Conceptual Schema and Information Base*, Report ISO TC97/SC5/WG5, No 695.
- Gustas, R. (1997) *Semantic and pragmatic dependencies of information systems*, Monograph, Technologija, Kaunas.
- Gustas, R. (1998) "Integrated Approach for Modelling of Semantic and Pragmatic Dependencies of Information Systems", *Conceptual Modelling - ER'98*, Springer, pp. 121-134.
- Gustas, R. (2000) "Integrated Approach for Information System Analysis at the Enterprise Level", *Enterprise Information Systems*, Kluwer Academic Publishers, pp. 81-88.
- 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.
- Gustiene, P. & Gustas, R. (2002) "On a Problem of Ambiguity and Semantic Role Relativity in Conceptual Modelling", *Proceedings of International conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, ISBN 88-85280-62-5, L'Aquila, Italy.
- Goldkuhl, G. (1995) "Information as Action and Communication", *The Infological Equation*, Goteborg University, Sweden, pp. 63-79.
- Jackson, M. A. (1983) *System Development*, Prentice Hall, Englewood Cliffs, N.J.
- Lindland, O. I., Sindre, G. and Solvberg, A. (1994) *Understanding Quality in Conceptual Modelling, IEEE Software*, (11, 2).
- Martin, J. & Odell, J. J. (1998) *Object-Oriented Methods: A Foundation (UML edition)*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Negoro, F. (2001) "Methodology to Define Software in a Deterministic Manner", *Proceedings of ICII2001*, Beijing, China.
- Pohl, K. (1993) "The three Dimensions of Requirements Engineering", *International Conference on Advanced Information System Engineering – CAiSE'93*, Springer Verlag.
- Roland, C. & Prakash, N. (2001) *From Conceptual Modelling to Requirements Engineering*, Annuals of Software Engineering (to be published).
- Snoeck, M., Dedene, G., Verhelst, M. & Depuydt, A. M. (1999) *Object-Oriented Enterprise Modelling with MERODE*, Leuven University Press.
- Storey, V. C. (1993) "Understanding Semantic Relationships", *VLDB Journal*, F Marianski (ed.), Vol.2, pp.455-487.
- Vernadat, F. B. (1996) *Enterprise Modelling and Integration: Principles and Applications*, Chapman & Hall, London.
- Warboys, B., Kawalek, P., Robertson, I. & Greenwood, M. (1999) *Business Information Systems: A Process Approach*, McGraw-Hill Co., London.
- Winograd, T. & Flores, R (1986) *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Norwood, N.J.
- Weber, R. (2001) *Comprehending Decompositions: A Theory and Two Empirical Tests*, unpublished manuscript.
- Yourdon, E. (1989) *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, N.J.
- 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.
- Zachman, J. A. (1996) "Enterprise Architecture: The Issue of the Century", *Database Programming and Design Magazine*.