Interactions, Transitions and Inference Rules in Semantically Integrated Conceptual Modelling

Remigijus Gustas and Prima Gustiené
Department of Information Systems, Karlstad University, Sweden
{Remigijus.Gustas, Prima.Gustiene}@kau.se

Keywords: Service Interactions, Value Exchanges, Inference Rules, Object Transitions, Integration of Static and Dynamic Aspects.

Abstract: To obtain value from the graphical representations that are used by different stakeholders during the system development process, they must be integrated. This is important to achieve a holistic understanding about system specification. Integration can be reached via modelling process. Currently, most of information system modelling methods present different modelling aspects in disparate modelling dimensions and therefore it is difficult to achieve semantic integrity of various diagrams. In this paper, we present semantically integrated conceptual modelling method for information system analysis and design. The foundation of this modelling method is based on interactions. This way of modelling provides possibility of integration of business processes and business data. The inference rules of interactions help in reasoning about the decomposition of concepts. In this method, decomposition of the system is graphically described as classification, inheritance or composition of organizational and technical system components.

1 INTRODUCTION

Conceptual modelling is a fundamental activity in requirements engineering (Nuseibeh and Easterbrook, 2000). It is the act of abstracting a model from a problem domain (Lankhorst, 2005). One of the main problems in conceptual modelling of Information Systems (IS) is that conventional modelling methods define different aspects of a system using different types of diagrams. Integration principles of such diagrams are not clear. The lack of a conceptual modelling method that helps to detect semantic integrity of IS specifications is a big information systems development problem. Semantically Integrated Conceptual Modelling (SICM) method challenges the existing integration problems among interactive, behavioural and structural aspects (Gustas and Gustiene, 2012) of IS. To capture the holistic structure of a system, it is necessary to understand how various components are related.

To obtain value from graphical representations that are used in an organisation by different stakeholders, these representations must be integrated. Integrated enterprise models might help business and information technology experts to communicate in order to assess and trace the impact of organizational changes. Integration can be reached via modelling process. Modelling helps system developers to visualize, specify, construct and document different aspects of the system. Modelling is the only way to control system development process. Various aspects of the system may have many modelling projections, which are typically described by using different types of diagrams. These diagrams are critical to distinguish between disparate dimensions of enterprise architecture (Zachman, 1987). The Zachman Framework (1987) can be viewed as taxonomy for understanding different types of diagrams. This framework defines separate dimensions of business application and data architecture, such as Why, What, How, Who, Where and When. Inability to detect inconsistency among different architecture views and dimensions is one of the fundamental problems in information system methodologies.

Most conventional conceptual modelling languages are plagued by the semantic mismatch between static and dynamic constructs of meta-models. To achieve semantic integration in such a case is very difficult. Unified Modelling Language (UML) (OMG, 2009) uses various types of diagrams to represent behavioural, structural and interaction aspects of the system. Every modelling approach

Gustas R. and Gustiene P.
Interactions, Transitions and Inference Rules in Semantically Integrated Conceptual Modelling.
DOI: 10.5220/0006221700110021
In *Proceedings of the Sixth International Symposium on Business Modeling and Software Design (BMSD 2016)*, pages 11-21
ISBN: 978-989-758-190-8

that covers more than one type of requirements and is represented by the collection of different diagrams must contain the systematic method for the detection of inter-model inconsistency. The static aspects describe characteristics of objects, which are invariant in time. The dynamic aspects describe interactive and behavioural characteristics of objects over time. These aspects are complimentary and they cannot be analysed in isolation.

Inter-model consistency and completeness of system specifications is hard to achieve for non-integrated model collections (Glinz, 2000). Modelling techniques that are realized as collection of models are difficult to comprehend for business experts. There are often semantic discontinuity and overlapping in various specifications, because static and dynamic constructs do not fit perfectly. A number of rules are defined for UML (Evermann and Wand, 2009) that are not supported by available CASE tools. Thus, working with the collections of non-integrated models causes difficulties to realize semantic quality of system specifications, which are represented on various levels of abstraction. By modelling isolated IS views and dimensions creates difficulties for business experts, who determine the organizational strategies. Consequently, this isolation increases semantic problems of communication between business experts and IT-system designers.

The SICM method provides several advantages (Gustas, 2010). Since the method is based on a single diagram type, the integrity rules can be introduced directly into one model. Particular views of specific diagram types, which define structural, behavioural or interactive aspects, can be generated by producing projections of one integrated model. In this paper, we demonstrate how the SICM method can be applied for integration of behavioural and structural aspects of conceptual representations. Given the central role of service concept in this study, it provides us with a possibility to model the most essential parts of the system, which is composed of organizational or technical components. This way of modelling is more comprehensible not just for IS designers, but also for business modelling experts, who are mostly interested in computation-neutral analysis of organizations. The presented SICM method shares many similarities with ontological foundation of service process (Ferrario and Guarino, 2008). Nevertheless, the internal behaviour of service is analysed by using the basic principles of an ontological framework, which is developed by Bunge (Bunge, 1979).

This paper is organized as follows. In the next section, some deficiencies of conceptual modelling approaches are described. How value exchanges are decomposed into different parts is discussed in the third section. In the fourth section, various types of conceptual dependencies and their inference rules are described. And finally, we present the conclusions of this work.

2 DEFICIENCIES OF CONCEPTUAL MODELLING APPROACHES

Conceptual modelling still lacks the methods that provide a possibility to model different problem domains in an integrated way. Integrated graphical representation of business process and business data is very relevant for reasoning about enterprise redesign decisions. As all steps in SICM method uses the same model and the same way of modelling that is based on service interaction flows. It enables enterprise architects to gradually decompose a system and to move smoothly from system analysis to design without being required to represent a complete solution. UML (OMG, 2009) was developed with the ultimate goal to unify the best features of the graphical modelling languages and create a de facto industry standard for system development. However, the semantic integration principles of different UML diagram types are not sufficiently clear. UML models have several weaknesses, which can be summarized as follows: value flow exchanges between actors cannot be explicitly captured; system decomposition principles are ambiguous; it is unclear how to integrate interactive, structural and behavioural aspects together in a single view.

Data flow modelling and clear system decomposition principles were applied in structured analysis and design methods (Gane and Sarson, 1979). UML also supports various types of associations between classes, actors, or between software or hardware components. However, these methods are not suitable for modelling the direct communication among actors that define actor interactions outside the technical system boundary. It is unclear how to visualize the rich context of actor interactions, which are important components in any system. If we have no method how to explicitly capture actors and their interactions, then this important part of specification, which may be viewed as a tacit knowledge, will be hidden from enterprise architects.

One of the benefits of enterprise modelling is the ability to analyse business processes for reaching agreement among various stakeholders on how and by whom the processes are carried out. The industrial versions of information system modelling methods that are intended for business process modelling do not explicitly use the concept of value flow. Value models, which include resource exchange activities among actors, can be viewed as design guidance. The declarative nature of value flows is very useful from the system analysis point of view for the simple reason that flows have very little to do with the dependencies between business activities. Each value flow between actors, that can play the role of service requester and service provider, can be further refined in terms of more specific coordinating interactions among organizational components. The way of modelling, which is based on service flows, is more comprehensible and thus more suitable to discuss changes of process architectures with business developers, enterprise architects, system designers and users. Business process modelling does not deal with the notion of value flow, which demonstrates value exchange among actors involved (Gordijn et al., 2000). Traditionally, information system methodologies are quite weak in representing the alternative value flow exchange scenarios, which usually represent the broken commitments.

Bunge (1979) provides one of the most general ontological definitions of a system. In this paper, his definition serves as the theoretical basis for understanding the notions of organization and enterprise ontology (Dietz, 2001). Bunge's ontological principles are fundamental for the justification of various conceptual modelling constructs in our semantically integrated modelling method (Gustas and Gustiene, 2012). These principles are as follows: enterprise system can be decomposed into subsystems, which are viewed as interacting components; every subsystem can be loosely coupled with interactions to other subsystems; when subsystems interact, they cause certain things to change and changes are manifested via properties.

Any subsystem can be viewed as an object, but not every object is a subsystem. According to Bunge, only interacting objects can be viewed as subsystems. It is quite beneficial to specify service interactions and to keep track of crosscutting concerns (Jacobson and Ng, 2005) between different subsystems in order to justify their usefulness. However, a basic underlying principle in UML is to provide separate models for different aspects. It is not totally clear how these aspects can be merged back into one model. Subsystems in UML cannot be realized as composite classes. UML does not provide any superimposition principles of static and dynamic aspects. There is very little research done on how the structural aspects and state dependent behaviour of objects should be combined with use case models. Classes and their associated state machines are regarded as the realization of use cases. Use case diagrams are typically not augmented with specification of state related behaviour (Glinz, 2000).

System decomposition should be strictly partitioned. Every component partitions a system into parts, which can be loosely coupled with other components without detailed knowledge of their internal structure. Object transitions and structural aspects have to be related to one separate service, which consists of organizational or technical components. The limitation of conventional system modelling methods results in two side effects, better known as tangling and scattering in aspect-oriented software development (Jacobson and Ng, 2005). The treatment of these deficiencies requires the modification of UML foundation. Introducing fundamental changes into UML syntax and semantics with the purpose of semantic integration of collections of models is a complex research activity. However, such attempts would allow using UML to provide computation-neutral type of diagrams, which are more suitable to reason about enterprise architectures. It is recognized that UML support for such task is vague, because semantic integration principles of different diagram types are still lacking (Harel and Rumpe, 2004).

3 VALUE FLOW EXCHANGES AND TRANSITIONS

Semantically integrated conceptual modelling paradigm is based on more rigorous interpretation of human work. A new conception helps us to develop the method of enterprise engineering that allows practitioners to see the sources of breakdowns, the connections to systems design and to guide the redesign of work processes towards greater productivity and customer satisfaction. Business process models of organizations are quite good for viewing moving material and information flows, but they provide no mechanism for ensuring that the service requester is satisfied. Service requesters deal with work processes to be done, agreements on what will be done, who will to it, and whether they are satisfied with what has been done. The movement of information or material flows is a consequence of this work. Service flow modelling is quite intuitive way of system analysis that is easy to understand for business experts and information system designers. Actions in services are required for exchange of business flows. Actions together with exchange flows can be viewed as fundamental elements for specifying business process scenarios. A scenario is an excellent means of describing the order of service interactions. Scenarios help system designers to express business processes in interplay with elementary service interactions between enterprise system components. In such a way, value flows and service interactions provide a natural way of process decomposition.

The technologies to model coordination processes and tracking events have not been available till now. There are some concepts such as commitment and contract that are present in all business scenarios. Understanding these concepts makes it much easier to design and to change systems under construction. Commitment is a promise or obligation of an actor to perform a specific action. Contract is an agreement between service requester and service provider to exchange one asset into another. Thus, the contract may specify what happens if the commitment is not fulfilled. According to McCarthy (1982), the contract consists of increment and decrement events. If an enterprise increases one resource in an exchange, it has to decrease the value of another resource. The contract includes (1) transfer of economic resources, (2) transfer of exchange rights. Any exchange is a process, in which an enterprise receives economic resource and in return gives other resources. For example, a contract contains commitments to sell goods and to receive payments. The terms of the sales order can specify penalties if goods or payments have not been received on time. The creation and termination of primary business data in these exchanges are important for an enterprise. Artefacts such as credit, debit, account balances are derived from these exchanges.

Interaction dependencies are important to conceptualize business processes as services between various enterprise actors. Since actors can be implemented as organizational or technical system components, these components can interact according to the prescribed service interaction patterns to achieve their goals. In SICM, the general service interaction pattern is represented by two interaction dependencies into opposite directions between two actors: service requester and service provider (Gustas, 2010). The idea of this pattern is similar to a well-known DEMO transaction pattern (Dietz, 2006). The SICM pattern is illustrated graphically in figure 1.

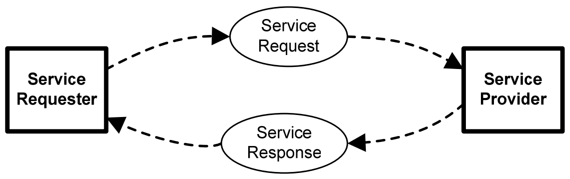

Figure 1: Elementary service interaction loop.

Interaction loop between two actors indicates that they depend on each other by specific actions. Service providers are actors who typically receive service requests, over which they have no direct control. They initiate service responses that are sent to service requesters. These two interacting actors can be used to define more complex interaction activities. Using interaction pattern, as way of modelling, enables system designers to construct the blueprint of interacting components, which can be represented by different actors across organizational and technical system boundaries. Any enterprise system can be defined as a set of interacting and loosely connected components, which are able to perform specific services on request.

Increment and decrement events represent values exchanged in business processes. Value models (Gordijn et al., 2000) clarify why actors are willing to exchange economic resources with each other. Actors, actions and exchange flows are elements that are necessary for demonstrating value exchange. Economic resources are special types of concepts, which represent moving things. Rectangles, with shaded background, are used to represent economic resources and dotted line boxes are used for the representation of exchange flows. Actors are represented by square rectangles and actions are represented by ellipses. Actions that are performed by actors are necessary for transferring economic resources, data or decision flows between actors. Two actors and transfer of value flows into opposite directions is illustrated in figure 2.

This figure illustrates that Deliver and Pay actions may happen at any time. It is not stated, which action should happen first. We just want to show that a customer is exchanging a Payment flow into a Delivery flow. Deliver action is initiated by vendor, because a shipment's moving direction is from Vendor to Customer. On the contrary, the payment is moving from customer to vendor through

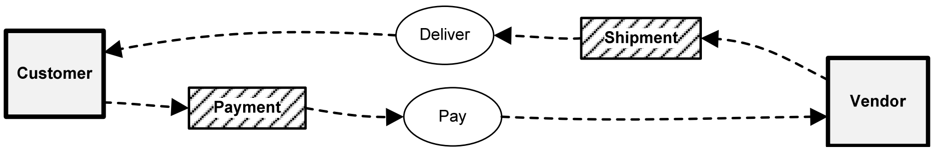

Figure 2: Value exchange.

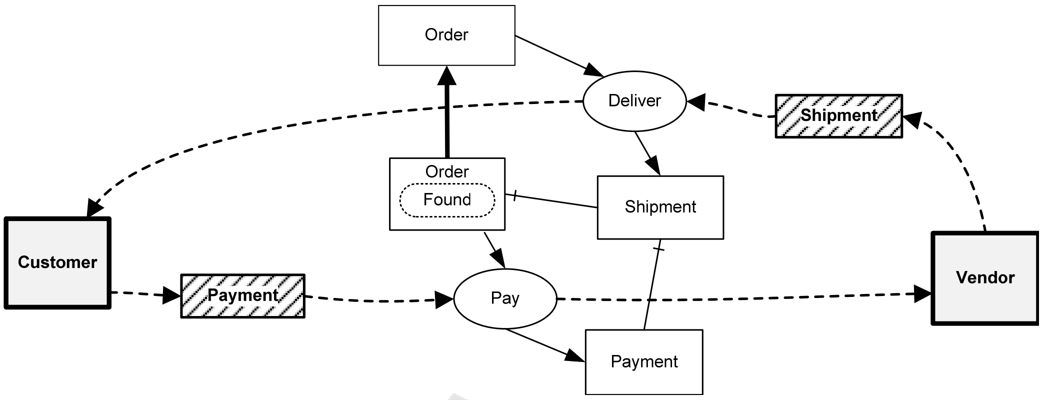

Figure 3: Interaction loop, where Deliver action precedes Pay action.

the action of pay. Action of Pay and Action of Deliver represent increment and decrement events. The process of paying is essentially the exchange of Shipment for Payment from the point of view of both actors. For a Vendor, the pay action is an increment event and deliver action is a decrement event, because it decreases the value of resources under control. For a Customer, it is vice versa. The terms of increment and decrement actions depend on the actor, which is the focus of this model.

Our buying and selling example focuses on the core phenomenon. Most customers pay in advance for shipment, but some customers want to pay, just when they receive the delivered products. If we consider the case of online sales, then customers provide credit card details before the product items are delivered. Some customers receive an invoice later and pay for all their purchases in a certain period. All these cases are covered by the same service interaction pattern, which is illustrated in Figure 2. When shipment is delivered, then the delivery fact is registered in a system by a newly created object with its mandatory properties. The transition arrow (⟶) is pointing to the class, which represents the creation of a new object. In our example, it is an object of Shipment. The fact of money transfer from Customer to Vendor is represented by Payment. If we want to represent that Deliver action precedes Pay Action, then the created Shipment itself or some of it property should be linked by the transition arrow with the Pay Action, which indicates the creation of the next object. The pay action creates the Payment from the Order [Found] object, which is the property of Shipment. It is represented in figure 3.

We may want to ask for payment in advance of shipment. In this case, we show the first action of pay, which is designed to transform the concept of invoice (not shown in our example) to payment. The second action of deliver should be connected through a transition arrow from payment to shipment. In this way, material flow of payment would be exchanged for the shipment flow.

Creation action is represented by a transition into an initial class. Termination action can be represented by transitions from a final class. If termination and creation actions are performed at the same time, then such action is called a reclassification. For instance, the initiation of the order action is typically used to create an Order record in a Vendor database. If customer Order is accepted, then it may be used for triggering the send invoice action. The internal changes are expressed by using transition links between various classes of objects in figure 4.

Creation and termination actions are used together with the object flows. In interaction pattern, a transition arrow to action or transition arrow from action represents a control flow. In such a way, any communication action can be used to superimpose interactions and control flow effects in the same diagram. Order is created by the Order Delivery action and then it is reclassified to Invoice by the send invoice action.

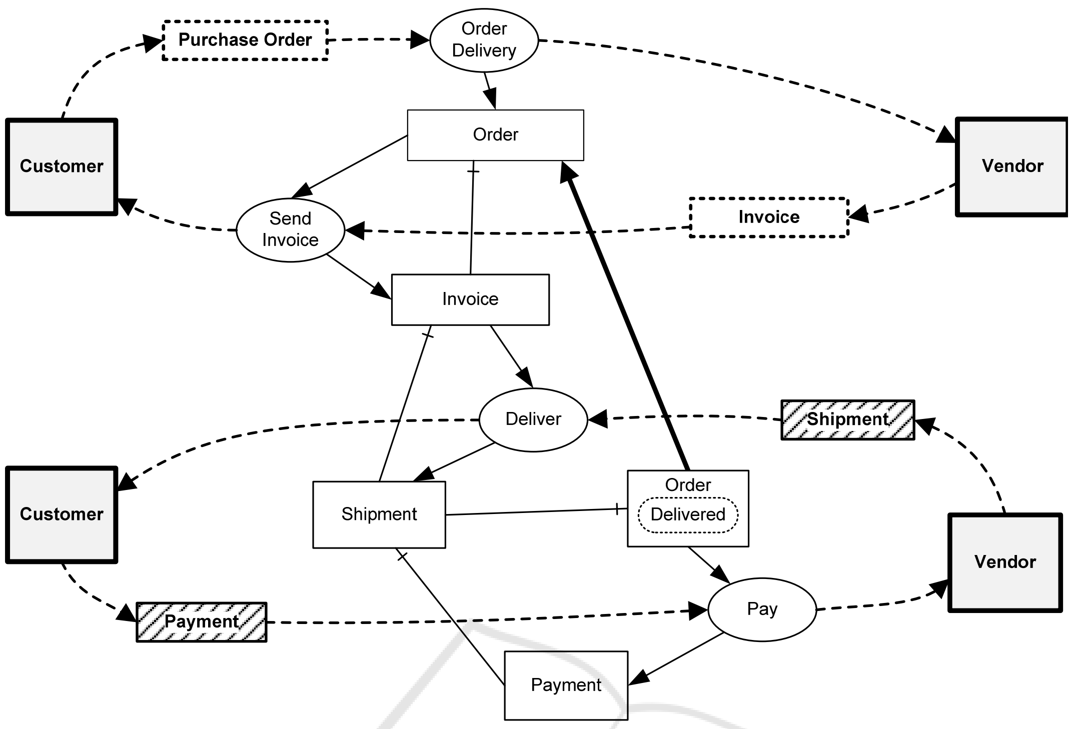

Figure 4: Example of two interaction loops with the creation and reclassification actions.

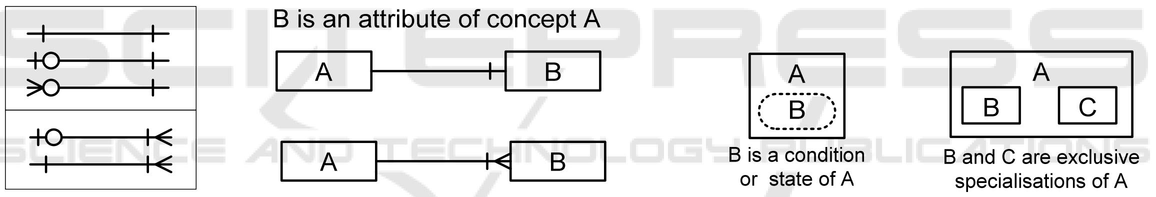
B is an attribute of concept A
A
B
A
B
A
B
B is a condition or state of A
A
B
C
B and C are exclusive specialisations of A

Figure 5: Notation of attribute dependencies.

The reclassification is defined as termination of object in one class and the creation of object in another class. An invoice object is created from the moving invoice flow, which represents data at rest. In the second interaction loop, a vendor delivers shipment to a customer. The delivery action corresponds to the performance act, which produces the result. It is represented by the object of shipment. Finally, the pay action indicates an acceptance of the delivered result. At the same time, it is a second performing action, which represents an exchange of shipment for payment.

Actors represent physical subsystems and structural changes of concepts represent static aspects of a system. This way of modelling allows illustrating actions, which result in changes of the attribute values. All actions are used to show the legal ways in which actors interact with each other. Structural changes of objects can be defined via static properties of objects. They are represented by the mandatory attributes. The mandatory attributes are linked to classes through the single-valued or through multi-valued attribute dependencies. One significant difference of the presented modelling approach is that the association ends of static relations are nameless. The justification of this way of modelling can be found in (Gustas and Gustiene, 2012).

The main reason for introducing nameless attribute dependencies is to improve the stability of conceptualizations. Semantics of static dependencies are defined by cardinalities, which represent a minimum and maximum number of objects in one class (B) that can be associated with the objects in another class (A). Single-valued dependency is defined by the following cardinalities: (0,1;1,1), (0,*;1,1) and (1,1;1,1). Multi-valued dependency denotes either (0,1;1,*) or (1,1;1,*) cardinality. Graphical notation of an attribute dependency between A and B is represented in figure 5.

According to the ontological principles, which are developed by Bunge (Bunge, 1977), the structural changes of objects can be presented via object properties. Properties can be understood as mandatory attribute values. If diagrams are used to communicate unambiguously the semantic details of a conceptualized system, then optional properties should be proscribed (Gemino, 1998). If B is dependent on A, then concept A is viewed as a class and concept B is viewed as a property of A. Any concept can be defined as an exclusive complete generalization of two concepts. Concept can also be characterized by state (Dori, 2002) or condition (Gustas, 2010). Notation of exclusive generalization and notation of state are presented in figure 5 as well.

4 INFERENCE RULES OF INTERACTIONS

A model of a system can be analysed as the composition of organizational and technical components. These components represent various types of actors. Organizational components can be seen as interacting subsystems such as individuals and divisions, which denote groups of people. Technical components can be seen as interacting subsystems such as machines, software and hardware. SICM method distinguishes between two types of concepts: active and passive (Gustas, 2010). Actors can be represented just by active concepts. An instance of any actor is an autonomous subsystem. Its life cycle can only be motivated by a set of interaction dependencies with other actors. Actors are represented by non-overlapping subsystems. Classes of objects, which represent persistent data, are denoted by passive concepts. Mandatory attributes characterize all passive concepts. The objects that are represented by passive concepts can be affected by various interactions. Passive concepts can be related by the static relations such as classification, inheritance and composition.

Classification dependency (●—) specifies objects or subsystems as instances of concepts. Classification is often referred to as instantiation, which is the reverse of classification dependency. Object-oriented approaches treat a classification relation as a more restrictive. It can only be defined between a class and an object. A class cannot play the role of object. In SICM method, each concept can be interpreted again as an instance (Gustas, 2010). For example; MS Outlook ●— E-mail Application, E-mail Application ●— Product Type, Product Type ●— Concept.

Composition (—►—) dependency in SICM method is much stronger form of aggregation, and differs significantly from the object-oriented composition. Composition dependency in SICM method allows just 1 or 1…* cardinalities between wholes and parts. This means that any part cannot be optional. The distinctive and very important features of this type of composition are as follows (Guizzardi, 2007):

a) each part is existentially dependent on a whole, if a whole has a single part, then this part has coincident lifetime with a whole,
b) if a whole has more than one part, then creation of a first part is coincident with the creation of a whole,
c) removal or creation of additional parts can take place any time, but removal of a last part is coincident with the removal of a whole.
d) creation or removal of a whole can be done together with all its parts,
e) part may belong just to one and the same whole.

The definition of composition in general is not so strict. With the help of special modelling techniques, other cases of aggregation can be changed into this strict kind of composition (Gustas, 2010).

Composition hierarchies can be used for detection of inconsistent interaction dependencies between actors. Loosely coupled actors never belong to the same decomposition hierarchy. Interaction dependencies among loosely coupled actors on the lower level of decomposition are propagated into compositional wholes. So, composition links can be used for reasoning about derived interaction dependencies between actors on the higher granularity levels of specification. Interaction dependencies between actors, which are placed on two different composition hierarchies, are characterized by the following inference rules:

1) if Action(X ---►Z), Action(C1) —► C2 and X —►— Y

then Action(Y ---►Z),

2) if Action(Z ---►X), Action(C2) —► C3 and X —►— Y

then Action(Z ---►Y).

Interaction dependency Action(X ---►Z) between two actors X and Y indicates that subsystem denoted by X is able to perform an action on one or more subsystems of Z. Action(X ---►Z) represents base interaction dependency and Action(Y ---►Z), which

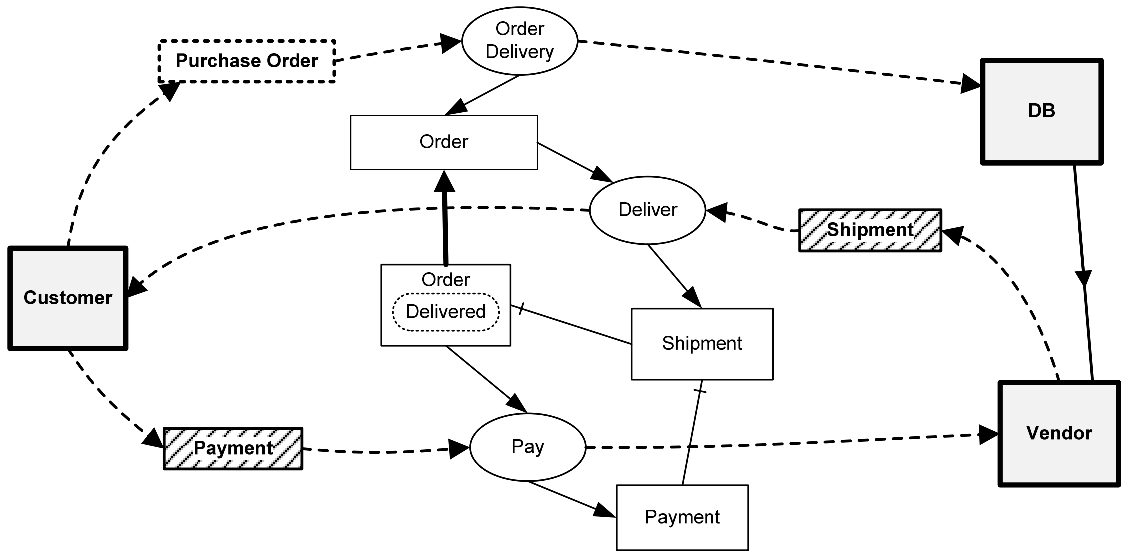

Figure 6: Base interactions between Customer and Vendor.

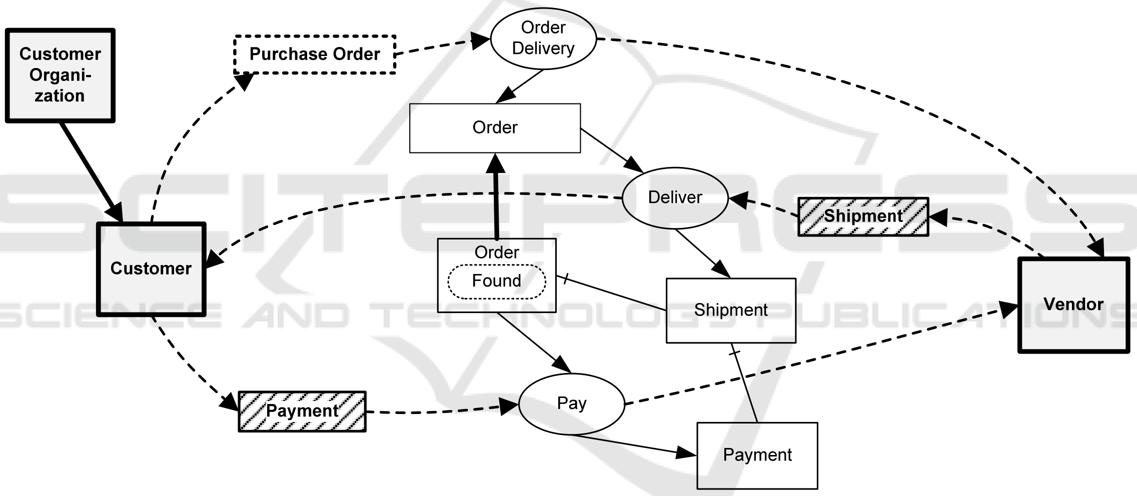

Figure 7: Derived and base interactions between Customer and Vendor.

is shown in the second part of the rule, represents derived interaction dependency. For instance, if Order Delivery (Customer $\dashrightarrow$ DB),

Order Delivery($\perp$) $\rightarrow$ Order, (DB —▶— Vendor) and then Order Delivery(Customer $\dashrightarrow$ Vendor) where $\perp$ represents an empty class. Two subsystems of Organization, DB (Database) and Vendor together with their interaction dependencies are represented in figure 6. This example is based on a well-known situation in Ford Motor Company after a radical change (Hammer, 2000). Ford Motor Company plays the role of an organization, which places a purchase order into a shared database (DB). The same service interaction loop, which was discussed previously, is represented in this diagram as well. The interaction loop between Customer and Vendor represents an exchange of Shipment for Payment. Please note that the derived interactions cannot be in conflict with the specified dependencies in other diagrams. The interaction links, which are presented in figure 6, are consistent with the interaction dependencies of figure 7.

Static and dynamic similarities of active concepts can be shared by more specific concepts according to the following rule:

if X ➡ Y and Y ➡ Z then X ➡ Z.

For instance, if a Company is a specialization of Customer Organization, and Customer Organization is an Organization, then for a Company can be

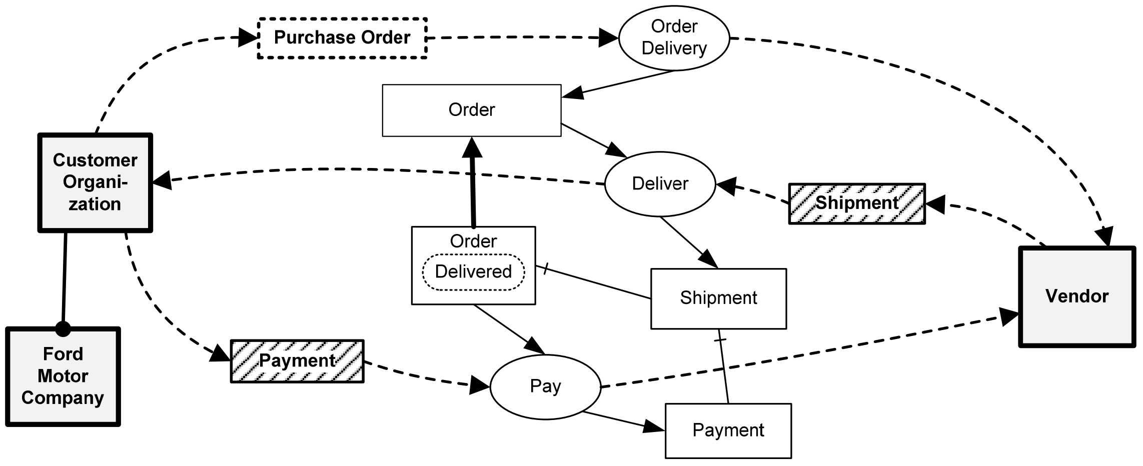

Figure 8: Derived interaction dependencies of Customer Organization.

applied static and dynamic similarities of an Organization.

More specific actors inherit interaction dependencies from the more generic actors. It should be noted that in the object-oriented approaches, inheritance link is defined just for attributes and operations. Inheritance dependency is convenient for sharing service interaction loops of more general actors. Interaction dependencies are inherited according to the following inference rules:

1) If Action(Y ---▶Z), Action(C1) ⟶ C2 and X ➡ Y then Action(X ---▶Z),
2) If Action(Z ---▶Y), Action(C2) ⟶ C3 and X ➡ Y then Action(Z ---▶X).

For example, if a Customer Organization is a Customer then Customer Organization inherits all service interaction links, which are represented for this more general concept. If Order Delivery(Customer ---▶Vendor),

Action(⊥) ⟶ Order and Customer Organization ➡ Customer then Order Delivery(Customer Organization ---▶Vendor). Customer Organization has the opportunity to send a purchase order to a Vendor and Vendor is obliged to deliver Shipment to the Customer Organization. The derived interaction dependencies of Customer Organization are represented in figure 8.

Classification dependencies can be also used for reasoning about derived interaction dependencies between actors. Interaction dependencies are propagated according to classification dependency links. Interaction dependencies between actors are characterized by the following inference rules:

1) if Action(Y ---▶Z), Action(C1) ⟶ C2 and X ●— Y then Action(X ---▶Z),
2) if Action(Z ---▶Y), Action(C1) ⟶ C2 and X ●— Y then Action(Z ---▶X).

For instance,
if Order Delivery(Customer ---▶Vendor),
Order Delivery(⊥) ⟶ Order and
Ford Motor Company ●— Customer
then Order Delivery(Ford
Motor Company ---▶Vendor).

This interaction loop can be replaced by simply switching from Customer to Ford Motor Company. It is represented in figure 9.

The responsibilities of different actors can be analysed using conceptual models of interactions. For instance, the Order Delivery action can be viewed as an opportunity to send a Purchase Order by Ford Motor Company to the Vendor. If Vendor accepts it, then he is responsible to Send Invoice to Ford Motor Company.

Please note that the opportunities, responsibilities, commitments and obligations of these two actors must be consistent with interaction dependencies. Inconsistency can be detected by naming the conflicts between actions or flows. More specific actors must be justified by their intrinsic communication actions, which are defined in terms of the complementary interaction dependencies of these actors. The presented inference rules are useful, but they are insufficient for reasoning about the consistency of interaction dependencies, which can be defined on different levels of specification. To understand the deep structure of service interactions, the behavioural and structural aspects of communication actions must be studied.

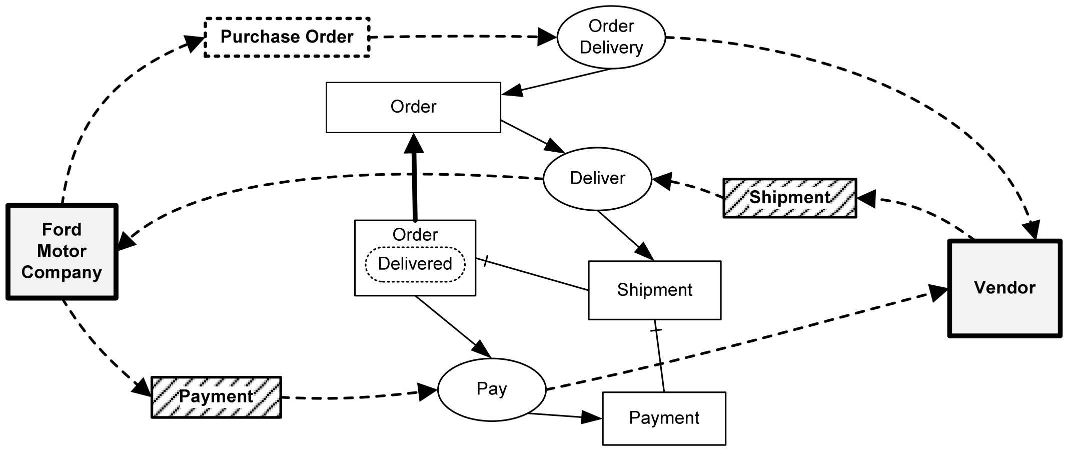

Figure 9: Derived interaction loop of Ford Motor Company.

5 CONCLUSIONS

The main contribution of this paper is presenting an integrated way of modelling. SICM provides us with a holistic method. One of the goals is to demonstrate how interactive, transitional and structural aspects of conceptual modelling can be integrated. Object-oriented modelling method projects static and dynamic aspects using different diagram types. In this case, to reach sematic integration of business processes and business data is very difficult. The semantic integration principles of different UML diagram types are not sufficiently clear. Since different modelling dimensions are highly intertwined, it is crucial to maintain integrity of various diagrams. We have demonstrated the interplay of three different aspects of conceptual models.

Interactions between actors are important to follow value exchange. Increment and decrement events represent economic resources exchanged in various business processes. Value exchanges are represented by two performing actions into opposite directions. The performing actions are triggered for the reason of coordinating actions. In this way, these actions are related to one value exchange. Every communication action is able to produce new facts that can be represented by various classes of objects. Value flow and service interactions provide the natural way of system decomposition. Value models help to clarify why enterprise actors want to exchange business objects with each other.

The ultimate goal of this paper is to overview deficiencies of conceptual modelling approaches and generic integration principles for development of holistic models of information systems. Bunge's ontological principles of decomposition are lying in foregrounded in SICM method. Actors can be seen as organizational or technical system components. Organizational components are denoted by individuals, groups, or company divisions. Technical components can be seen as software or hardware system components. Decomposition of information system is based on semantic relations of classification, composition and inheritance. Similarities of these relations are explained in comparison with object-oriented approaches. Inference rules of the semantic relations are presented in this paper as well. The behavioural and structural dimensions of interactions were analysed in terms of creation, termination and reclassification action.

Conceptual modelling methods, which put emphasis on active concepts, typically focus on analysing interactivity between organizational and technical components. This tradition is quite successful for modelling of external behaviour of a system. In contrast, the object-oriented approach is based on modelling static and dynamic aspects of concepts, which can be represented by various classes of objects. The majority of textbooks in the area of systems analysis and design recommend concentrating first on domain modelling. Most conventional system analysis and design methods do not put into foreground modelling of active concepts. These methods project the structural, interactive and behavioural aspects into totally different types of diagrams that cause difficulties to integrate static and dynamic aspects of enterprise architecture dimensions. Very few emerging approaches to modelling make attempts to illustrate the deep interplay between active and passive structures. We have illustrated with simple examples how to represent integration of various aspects of information systems.

REFERENCES

Blaha, M., and Rumbaugh, J. (2005). Object-Oriented Modelling and Design with UML. London: Pearson.

Bunge, M. A. (1979). *Treatise on Basic Philosophy, vol. 4, Ontology II: A World of Systems*. Dordrecht, Netherlands: Reidel Publishing Company

De Marco, T. (1979). Structured Analysis and System Specification. New York. Prentice Hall.

Dietz, J. L. G. (2001). DEMO: Towards a Discipline of Organisation Engineering, *European Journal of Operational Research (128)*, Elsevier Science, 351-363.

Dietz, J. L. G. (2006). Enterprise Ontology: Theory and Methodology. Berlin: Springer.

Dori, D. (2002). Object-Process Methodology: A Holistic System Paradigm. Berlin: Springer.

Evermann, J. and Wand, Y. (2009). Ontology Based Object-Oriented Domain Modeling: Representing Behavior, *Journal of Database Management, Vol. 20, Issue No 1*, 48-77.

Ferrario, R., and Guarino, N. (2008). Towards an Ontological Foundation for service Science, *First Future Internet Symposium,* FIS 2008, Vienna, Austria, 152-169. Berlin: Springer.

Gane, C., and Sarson, T. (1979). *Structured System Analysis*. New York: Prentice Hall.

Gemino, A. (1998). To be or maybe to be: An empirical comparison of mandatory and optional properties in conceptual modeling, *Proc. Ann. Conf. Admin. Sci. Assoc. of Canada,* Information Systems Division, Saskatoon, 33-44.

Glinz, M. (2000). Problems and Deficiencies of UML as a Requirements Specification Language, *Proc. of the 10-th International Workshop on Software Specification and Design*, San Diego, 11-22.

Guizzardi, G. (2007). Modal Aspects of Object Types and Part-Whole Relations and the *de re/de dicto* distinction, *19th International Conference on Advanced Information Systems Engineering,* Trondheim, Lecture Notes in Computer Science 4495. Springer. .

Gustas, R. (2010). A Look behind Conceptual Modeling Constructs in Information System Analysis and Design, *International Journal of Information System Modeling and Design,* Vol. 1, Issue No 1, 78-107.

Gustas, R., and Gustiene, P. (2012). Conceptual Modeling Method for Separation of Concerns and Integration of Structure and Behavior, *International Journal of Information System Modeling and Design,* vol. 3 (1), 48-77.

Gordijn, J., Akkermans, H., and van Vliet, H. (2000). Business Process Modelling is not Process Modelling, *Conceptual Modelling for E-Business and the Web*, LNCS 1921, 40-51. Berlin: Springer.

Hammer, M. (1990). Reengineering work: Don't Automate, Obliterate, *Harvard Business review*, 104-112.

Harel, D., Rumpe, B. (2004). Meaningful Modeling: What's the Semantics of 'Semantics'?, *IEEE Computer*, 64-72.

Jacobson, I., and NG, P-W. (2005) Aspect-Oriented Software Development with Use Cases. Pennsylvania: Pearson Education.

Lankhortst, M. (2005). Enterprise Architecture at Work: Modelling, Communication, and Analysis. Berlin: Springer.

McCarthy, W.E. (1982). The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review, Vol. LVII, No. 3, 554-578.

Nuseibeh, B., and Easterbrook, S. (2000). Requirements Engineering: A Roadmap. *In Proceedings of the Conference on the Future of Software Engineering: International Conference on Software Engineering* New York: ACM Press, 35-46.

OMG. (2009) *Unified Modeling Language Superstructure, version 2.2.* Retrieved from www.omg.org/spec/UML/2.2

Zachman, J. A. (1987) A Framework for Information System Architecture, *IBM Systems Journal,* Vol. 26, No 3.