Integration of Interactive, Behavioral and Structural Aspects of Conceptual Models

Remigijus Gustas

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

Abstract. Conceptual modeling is an essential part of enterprise engineering activity. Unfortunately, enterprise modeling methods are projecting interactive, behavioral and structural dimensions of conceptualizations into totally different diagram types. If static and dynamic aspects are analyzed separately, they are more difficult to visualize and to understand for stakeholders. Moreover, in the traditional approaches, there is a paradigmatic mismatch among different enterprise architecture modeling dimensions. Analysis of interplay among interactions, state changes and object creation/termination effects is necessary for understanding integrity problems of conceptualizations. The goal of this paper is to present a modeling approach for semantic integration of static and dynamic views of conceptual models. The presented modeling method can be used for separation of crosscutting concerns of computation neutral specifications.

Keywords: Semantic integration of static and dynamic views, separation of crosscutting concerns, behavior, interaction, structural changes of objects.

1 Introduction

Unified Modeling Language (UML) (OMG, 2010) was developed with the ultimate goal for unifying the best features of the graphical modeling languages and creating a de facto industry standard for object-oriented software design. Recently, UML started to evolve into a language for business and enterprise modeling. However, the semantic integration principles of different diagram types are not completely clear in UML. One of the goals of this paper is to present a modeling approach for integration of static and dynamic aspects of conceptualizations.

Explicit modeling of interaction flows is crucial in system development. Understanding of interaction flow sequences among enterprise components is important for system architects to move smoothly from system analysis to design, without requirement to represent a complete solution. Interaction modeling (Gustas & Gustiene, 2009) helps to develop a coherent graphical representation of business process and business data. Interaction modeling in terms of data flows was the strength of structured analysis and design methods (Gane & Sarson, 1979), (Yourdon & Constantine, 1979). UML supports various types of associations between classes, actors and use cases, and between objects such as software or hardware components. However, sequence, activity or use case diagrams are not suitable for modeling explicit interaction flows between actors. If actor interactions cannot be explicitly captured, then they are

P. van Bommel et al. (Eds.): PoEM 2010, LNBIP 68, pp. 217–231, 2010.

hidden from business modeling experts. In this case, such important relations cannot be maintained by the conventional CASE tools.

Information system designers often focus on a use case (Jacobson & Ng, 2005) modeling. Use case is a unit of functionality that a system can provide for organizational or technical actors. It can be specified by conceptualizing interactions, activities, and state changes in various classes of objects. A coherent use case represents information system functionality that helps to achieve one goal. Unfortunately, use cases typically define implementation-specific services, which are placed inside technical system boundary. The alignment details of business processes with use cases are often not so easy to visualize and to comprehend.

One of the benefits of service orientation (Gustas & Gustiene, 2008) is to analyze business processes in terms of interaction flows. Declarative nature of service flows is very helpful in system analysis phase, because they have very little to do with dependencies between business activities. The particular strength of interaction dependencies is possibility to capture crosscutting concerns among organizational and technical components. Most of conceptual modeling approaches do not deal with the notion of service flow, which demonstrates value exchange among actors involved in business processes (Gordijn et al., 2000). Information system methodologies are quite weak in integrating data flows with related behavioral effects and representing consequences if commitments in delivering flows are broken. The treatment of such weaknesses would require modification of the UML foundation. Introducing fundamental changes in UML with the purpose of semantic integration of collection of models is a complex research activity. However, such attempts would allow using UML as an enterprise modeling language for developing computation neutral type of diagrams, which are more suitable for reasoning about enterprise architectures. It is recognized that UML support for such task is vague, because the semantic integration principles of different diagram types are lacking (Harel & Rumpe, 2004). In this paper, we present a modeling approach for semantic integration of interactive, structural and behavioral aspects of conceptual models.

2 Conceptual Modeling of Interactive Aspects

Many textbooks in the area of systems analysis and design recommend concentrating first on the structural aspects of domain modeling that are based on specification of classes, attributes and associations. The second step is typically analysis of behavioral aspects of domain, which are expressed in terms of events, state transitions and their related effects. Finally, it is recommended to start modeling of interactive aspects. Interactions are often considered as the third leg of the modeling tripod (Blaha & Rumbaugh, 2005). State modeling can be viewed as reductionist projection of behavior, because both states and interactions are necessary to describe behavior fully. Such way of modeling creates difficulties in the detection of discontinuities and breakdowns in business activities, because the knowledge on the choreography of business interactions is missing.

Many designers see use case diagram as an excellent starting point of system analysis. Parallel, sequential, branching or iterative use case execution can be described by using activity diagrams, but normally this type of specification is not

associated with use case diagrams. Use case diagrams are typically not augmented with specification of state related behavior (Glinz, 2000). Many world-class modelers downplay use case diagrams in the early requirement engineering phases. Instead they focus on writing scenarios (Larman, 2009). Another problem is that use case diagram enforces early implementation-related decisions about a technical system boundary and its environment, which is defined in terms of organizational and technical actors. Note that any conceptual representation should follow the basic conceptualization principle (Griethuisen, 1982) in representing only computation neutral aspects that are not influenced by possible implementation solutions. Violation of this principle results in a higher complexity of diagrams.

We believe that analyzing interplay of interactive, behavioral and structural aspects helps to introduce a new approach to conceptual modeling. One of the goals of this paper is to question the conventional way of system analysis and design. We will demonstrate a different way of reasoning in modeling of static and dynamic relations among concepts. A small case study on a conference review management system will be used as a running example, which is important for the demonstration of modeling constructs and their expressive power. Five interactions of a conference review management system are defined by the sequence diagram, which is presented in figure 1.

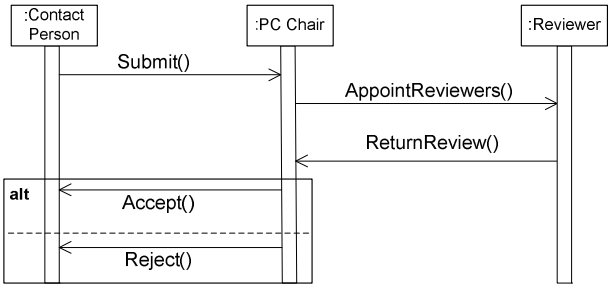

Fig. 1. Main interactions of a conference review management system

This diagram corresponds to our initial description of a conference management system. It is as follows: *One of the authors plays the role of* ***Contact person*** *who* ***submit****s a paper to a conference. The responsibility of a conference program committee (****PC****)* ***chair*** *is to* ***appoint reviewers*** *for every submission. The* ***Reviewer*** *is obliged to* ***return review*** *of the paper to the* ***PC chair*** *on time. Depending on the reviewing outcome, the* ***PC Chair*** *is authorized to* ***accept*** *or* ***reject*** *a submitted paper. If paper is accepted, then revision instructions are sent to the corresponding* ***Contact person****. Otherwise, reviewer comments are included in the rejection letter.*

There are few unconventional features related to the diagram, which is presented in figure 1:

1) The boxes such as :*Contact Person*, :*PC Chair* and :*Reviewer* are roles, not objects in traditional understanding of object-oriented design. Contact Person, PC Chair and Reviewer are organizational components, which are called actors in UML. Actors are typically placed outside technical system boundary in use case diagram.

2) Designers typically decompose higher level business related actions into more detail activities, use cases or bottom level operations, which fit well one of the UML diagram types. Each presented interaction cannot be placed precisely on a single swimlane of activity diagram, since it binds two responsible actors: agent and recipient. The presented actions *Submit*, *AppointReviewers*, *ReturnReview*, *Accept* and *Reject* are not use cases in the traditional understanding. A use case typically represents functionality, which is performed by the system, to yield an observable result to one particular actor (Jacobson & Ng, 2005). For instance, '*Reviewer is obliged to* ***return review*** *of paper to PC chair'* specifies that *Reviewer* is an agent, who is responsible for triggering *Return Review* action. *PC Chair* is a recipient of *Review* flow, which is delivered by using Return Review action. Another difficulty is that the return review action is multiple. It can be triggered by one or more reviewers, which were appointed by *PC chair* for a specific paper. Example of a corresponding lower granularity activity diagram with swimlanes is represented in figure 2.

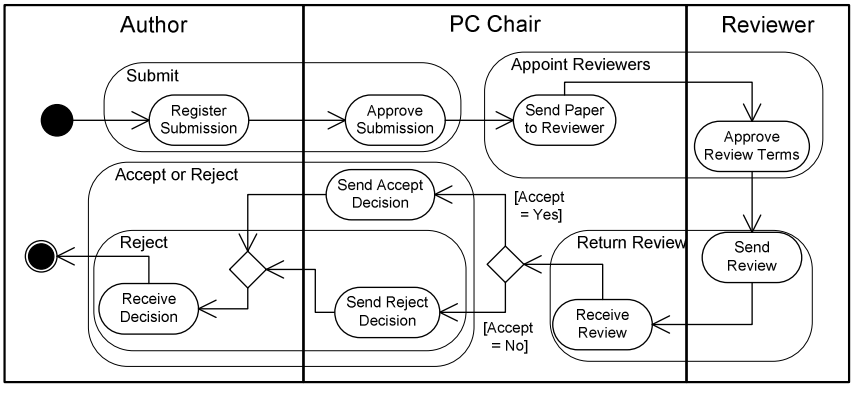

Fig. 2. Lower granularity activities in a conference review management system

As it is illustrated in this activity diagram, one business interaction between two organizational actors is broken down into two or more activities. Such finer granularity activities can be included without any problem into a use case or into activity diagram. The higher granularity business process states such as *Submit*, *Appoint Reviewers*, *Return Review*, *Accept* and *Reject* should be excluded from various UML diagrams for two major reasons:

- All these activities do not fit well a general use case definition (OMG, 2010), because they are communication actions between two actors. Designers are normally interested to focus on a use case functionality, which is performed by the system, to yield an observable result to one particular actor.
- Identification of object-oriented operations is crucial in designing class, state and sequence diagrams. The presented activities cannot be viewed as operations, which are invoked on one type of object.

Although the presented higher granularity business process states do not fit well the constructs of a use case diagram, they represent the communication actions, which are crucial for understanding crosscutting concerns of an enterprise system. The presented actions provide the natural way of decomposition of business process and therefore they must be explicitly captured in an enterprise model.

3 Conceptual Modeling of Behavioral and Structural Aspects

The presented five interactions (see figure 1) can be viewed as triggering events in various state transition diagrams. State diagrams are used to define the behavior of objects. Behavioral aspects can be represented as series of events, which may occur in one or more possible states. State transitions are triggered by events, which specify the permissible ways for changes to occur in different classes of objects. Graphical example of the corresponding state transition diagram is represented in figure 3.

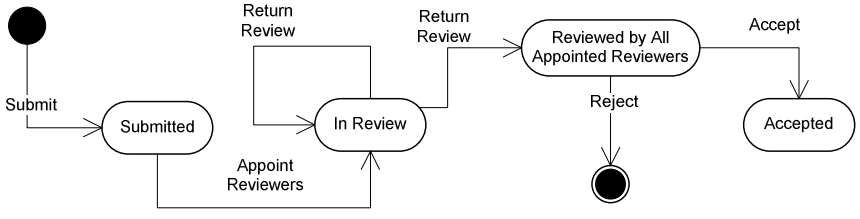

Fig. 3. Events and transitions of a paper object

This diagram represents states and state transitions of a *Paper* class object. State transitions are associated with events, which originate from five interactions in a conference managements system. Communication actions in this diagram are interpreted as triggering events: *Submit*, *AppointReviewers*, *ReturnReview*, *Accept* and *Reject*. The main problem with this diagram is that it is not representing related creation, manipulation or termination effects in various classes of objects.

Classes, associations and attributes are the most fundamental object-oriented constructs, which are used in UML class diagrams. They are stemming from the conventional conceptual modeling approaches, which were developed to capture the static aspects of concepts. Various classes of objects can be identified according to the presented initial description of conference management system such as Contact_Person, Submitted_Paper, Reviewing, Reviewer, Review, Accepted_Paper. Dependencies between concepts can be defined by associations as well as by inheritance, aggregation and composition relations. Associations between concepts are represented in terms of association ends and multiplicities in two opposite directions. The associations, classes and their attributes of a conference review management system are represented by the class diagram in figure 4.

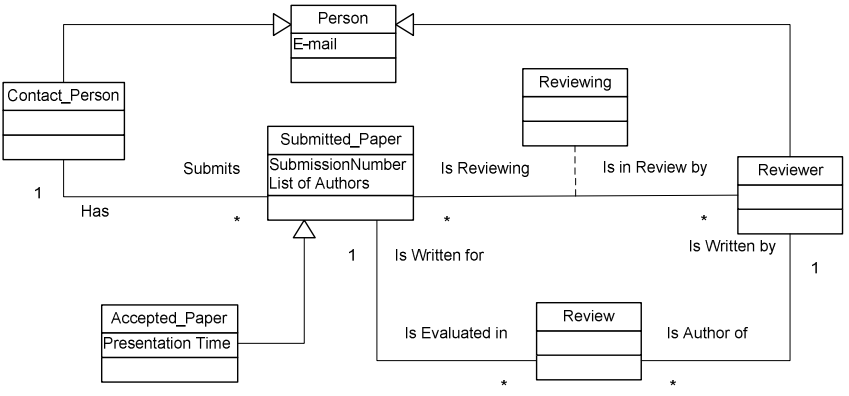

Fig. 4. Main classes and associations of a conference review management system

A state (see the previous diagram) is defined as a collection of object properties, which can be represented by attributes and associations with other classes. Three types of object transition effects can be distinguished in object-oriented design (Martin and Odell, 1998):

1) creation and termination of an object,
2) classification and declassification of an object,
3) connection and disconnection of a link between objects.

These effects are represented by various UML diagrams in a variety of ways. For instance, creation and termination of objects is visualized by the transitions from an initial state and to a final state. Connection and disconnection events can be specified by using sequence diagrams. Classification and declassification effects can be implemented by using sequences of object creation, connection, disconnection and termination operations. Classes, associations and attributes are necessary for understanding the effects of resulting structural changes from the events in a conference management system. One of the main challenges of the next section is to demonstrate the semantic integration of interaction events and related behavioral effects. The remaining part of this paper presents the modeling approach, which combines interactive, behavioral aspects and structural changes of objects in a single computation neutral diagram of a reasonable size.

4 Semantic Integration of Interactive and Behavioral Aspects

Conceptual models of interactions are not difficult to understand for business professionals as well as information system designers. Service interactions are helpful for clarifying why actors are willing to exchange flows with each other. Nevertheless, majority of conceptual modeling methods are not able to capture interaction flows between actors (Gustas & Gustiene, 2009). Actions and flows can be viewed as fundamental elements for defining business scenarios. A scenario is an excellent means for describing the order of interactions. Each interaction can be analyzed separately as it is required by the principle of separation of crosscutting concerns.

Interaction flows are special types of concepts that represent moving things. In our modeling approach, solid rectangles are be used for denotation of material flows and light boxes will indicate data flows. There are no material flows in the presented graphical description of a conference management system. An action with a missing data or material flow is understood as a decision or control. Actions, which will be represented by ellipses, are performed by actors. Actions are necessary for transferring flows between subsystems, which are represented as enterprise actors. Actors are denoted by square rectangles. They represent organizational and technical components of a system. Interaction flows among various actors of a conference management system are illustrated in figure 5.

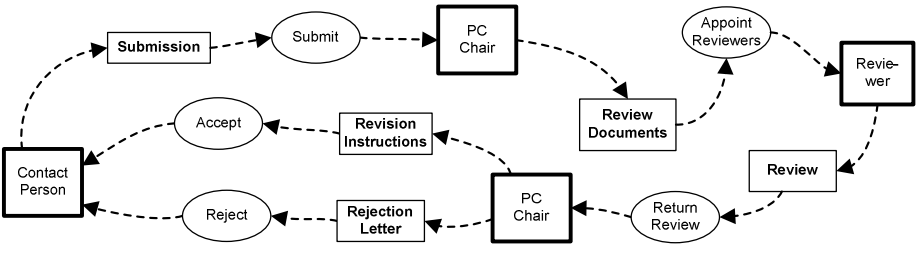

Fig. 5. Main actors and interactions of a conference management system

A contact person has possibility to *submit* a paper. If submission is accepted, the responsibility of a conference PC chair is to trigger the *appoint reviewers* action, which is used to send review documents to reviewers. Reviewer is obliged to deliver review to PC chair by triggering the *return review* action. PC Chair is authorized to *accept* or *reject* a submitted paper by informing a contact person with a special letter.

The main difference of this diagram in comparison with the presented sequence diagram (see figure 1) is that additionally it represents data flows between various actors. There are many other fundamental differences between these two kinds of diagrams, which will be discussed below.

In general, two kinds of interactions between actors can be distinguished (Dietz, 2006) such as production and coordination actions. Service requests are normally coordination actions, which are initiated by service requesters. Coordination actions are necessary to make commitment regarding the corresponding production action, which is supposed to bring a value to service requester. Production acts are normally performed by service providers. For example, the appoint reviewers can be viewed as coordination action, which corresponds to service request. The return review can be considered as a production action, because *Review* flow brings value to PC chair. Service requesters and providers are viewed as subjects or as active concepts (Gustas, 2010), which represent enterprise system components.

The behavioral and structural aspects of interactions can be analyzed in terms of reclassification, creation or termination effects. When two subsystems interact one may affect the state of each other (Evermann & Wand, 2009). Structural changes of objects can be defined in terms of object properties. Interaction dependency R(A┅►B) between two active concepts A and B indicates that A subsystem can perform action R on one or more B subsystems. An action typically manipulates properties of some objects (Gustas & Gustiene, 2009). Otherwise, this action is not

useful. Property changes may trigger objects transitions from one class to another. The internal changes of objects can be expressed by using transition links (—►) between two object classes, which represent passive concepts. Graphical notation of reclassification construct is graphically represented in figure 6.

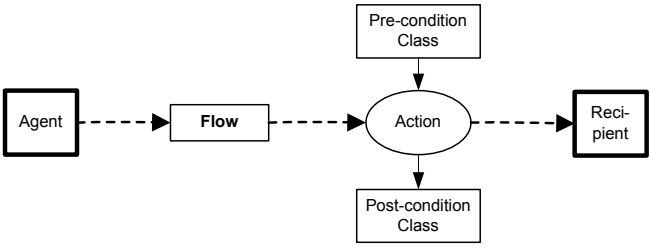

Fig. 6. Construct for representation of reclassification event

Two kinds of fundamental changes occur in reclassification action: removal of an object from a pre-condition class and creation of an object in a post-condition class. Reclassification construct with a missing post-condition class is used for representation of termination of object in a precondition class. A construct without a pre-condition class represents object creation in a post-condition class. For example, *Submit* action can be defined as creation event and *Reject* action can be viewed as termination. *Appoint Reviewers*, *Return Review* and *Accept* are reclassification events (see figure 8). Object creation or reclassification without any properties does not make any sense. So, various types of static and dynamic dependencies between classes are used to define mandatory properties of objects. The lack of noteworthy difference between pre-condition and post-condition class indicates that the specification of a communication action is either incomplete or a communication action is not purposeful. Pre-condition and post-condition classes are typically characterized by two different sets of mandatory attributes, which are sufficient for representing the permissible ways in which changes may occur. Static dependencies such as inheritance, composition, single-valued and multi-valued mandatory attributes are sufficient to visually recognize and comprehend the details of various interaction effects. Graphical notation of concept dependencies is presented in figure 7.

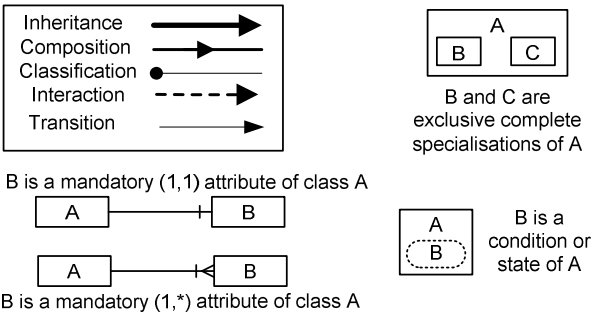

Fig. 7. Graphical notation of dependencies between concepts

One significant difference of our integrated modelling approach from traditional methods is that all dependencies are nameless. Concepts can be specialized by using special conditions or states. Any concept can be also defined as an exclusive complete generalization of other concepts. Actors are also specialized or decomposed by using inheritance, classification and composition dependencies. Inheritance dependency (➧) and composition dependencies (–►–) can be used for reasoning about sharing static and dynamic dependencies between concepts. For example, the diagram, which is represented in Figure 5, can be extended by using the following dependencies:

Contact Person ➧ Author, Reviewer ➧ Person, PC chair ➧ Reviewer,
PC chair –►– Conference, Reviewer –►– Conference.

Composition dependency is a strict form of aggregation, which allows just either 1 or 1..* multiplicities between wholes and parts. Other cases of conventional composition are not legal in the presented modeling approach (Gustas, 2010). Composition and inheritance dependencies can be used for detection of inconsistent interaction dependencies on various levels of abstraction. The presented set of semantic dependencies is sufficient for unambiguous specification of creation, reclassification or termination effects in various classes of objects. These effects are fundamental for integration of behavioural and structural aspects of interactions. Analysis of creation, termination and reclassification effects can be performed by using a special set of rules, which are presented in the next section.

5 Semantic Integration of Behavioral and Structural Aspects of Objects

One of the most general ontological definitions of a system is provided by Bunge (Bunge, 1979). It served as a theoretical basis for understanding the notions of organization and enterprise ontology (Dietz, 2006). Bunge's ontological foundation is important for motivation of our semantic integration principles. They are as follows:

- Enterprise system can be decomposed into subsystems, which are represented as interacting actors,
- Every subsystem can be loosely coupled by interactions to other subsystems,
- When subsystems interact, they cause certain things to change. Changes are manifested via properties.

A transition arrow from and to action represents a control flow, which defines correspondingly termination and creation of various types of objects. A diagram showing object transitions and flows with states has most of the advantages of activity, state, sequence and class diagrams without most of their disadvantages when analyzed in isolation. Each action is used to superimpose interaction and object transition effects in a single diagram. Various combinations of the presented dependencies are fundamental for understanding sequences, alternatives, synchronizations or iterations of object creation, reclassification and termination effects. Behavioural, interactive and structural aspects of previously analyzed diagrams can be integrated into a single conceptual representation, which is presented in figure 8. Note that this conceptualization represents the semantic details of sequence, state transition, class diagrams (see figure 1, 2, 3, 4) including all interactions (see figure 5) with related creation and termination effects.

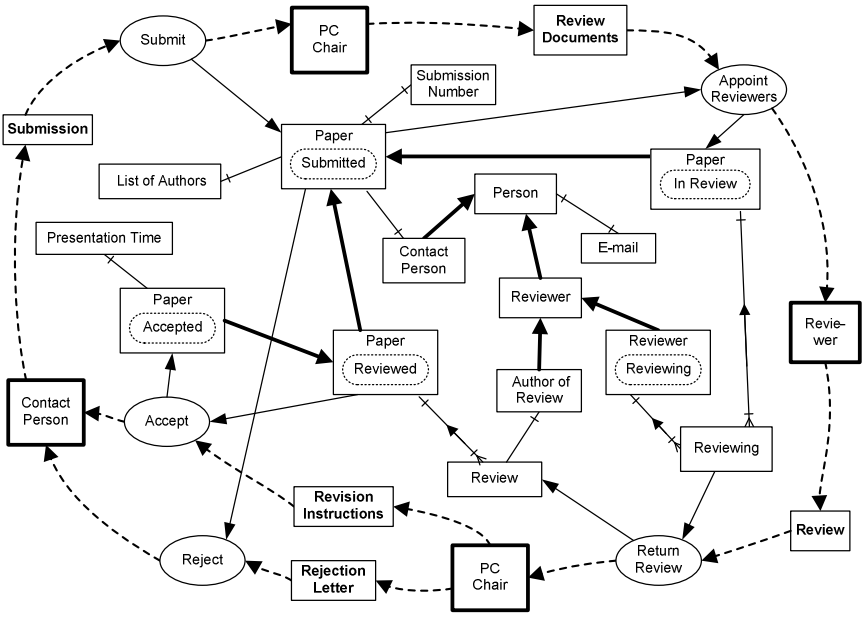

Fig. 8. Interactive, behavioral and structural aspects of a conference management system

1) The first event is *Submit.* According to the presented integrated diagram, the Submit action creates the following effects:

1.1) Creation of the Submitted Paper object, which is characterized by Submission Number and List of Authors. We assume that creation of an object in Contact Person class can be triggered by the Register Person action, which is not included in our initial description (see figure 9). We also assume that designers are interested just in one property of a contact person object. It is represented by the E-mail attribute.

1.2) Creation of the link between a Submitted Paper and one Contact Person object.

2) *Appoint Reviewers* is the second event, which is triggering reclassification of an object from the Submitted Paper into the Paper[In Review] class. Each Paper[In Review] object must be composed of one or more Reviewing objects (reified process). Reviewing is characterized by exactly one Reviewer. We assume that the objects of Reviewer class are created by other communication actions, which are outside of our initial description (a possibility for PC Chair to register a new reviewer is shown figure 9).

3) Changes of the *Return Review* event include creation and termination effects, which can be described as follows:

3.1) Creation of the object in Review class.

3.2) Creation of the mutual property links between the Reviewed Paper and Review object in two opposite directions.

3.3) Creation of the Author of Review property link for the Review object.

3.4) Termination of the corresponding Reviewing object with all its associated properties.

4) Triggering effects of the *Accept* event includes the reclassification of an object from the Reviewed Paper class into Accepted Paper class with an additional property of Presentation Time.

5) Triggering effects of the *Reject* event are defined by the termination of a Submitted Paper object. We assume that, in the case of a paper rejection, all related properties of the submitted paper object are removed. Each set of listed effects must be synchronized. Otherwise, the situation of data inconsistency may arise.

Inheritance dependencies (➡) are useful for reasoning about alternatives of communication action effects. The main rule for understanding of creation and termination effects is as follows:

Rule 1: Termination of an object in an inheritance hierarchy requires termination of all its specializations. For instance, termination of a person object is causing termination of a Reviewer and/or termination of a Contact Person objects (see the diagram in figure 8). Object of the more specific class requires creation of a more generic object. For example, a Paper[Accepted] object cannot be created by *Accept* action prior to Paper[Reviewed] object is created by the *Return Review* action.

Composition dependencies (–►–) are useful for reasoning about synchronization and iteration of object creation, reclassification and termination effects. Four rules (2, 3, 4 and 5) can be applied for understanding existential dependencies effects between wholes and parts. They are as follows:

Rule 2: Creation of object requires creation of all its compositional parts. For instance, the Appoint Reviewers action requires synchronous creation of Paper [in Review] together with at least one associated object of Reviewing. Note that reviewing object represents reified review process, which links one Paper and one Reviewer object in the state of Reviewing.

Rule 3: Termination of object requires termination of all its compositional parts. For example, the Reject action requires termination of a Paper in Submitted as well as in Reviewed states together with all Review objects as compositional parts.

Rule 4: Creation of the first part requires creation of a compositional whole. For example, if the Return Review action creates the first Review of some Paper object, it is necessary to synchronously create the Paper object in Reviewed state.

Rule 5: Termination of the last part requires termination of a compositional whole. For instance, if Return Review action terminates the last Reviewing object, it is necessary to terminate the compositional whole, which is represented by the Paper object in Review state.

Attribute dependencies are useful for reasoning about sequences of object creation, reclassification and termination effects. Rule 6 and 7 are useful for understanding manipulation effects of objects and their properties:

Rule 6: A property, which is viewed as an object on its own cannot be created prior to the creation of the object itself.

Rule 7: Removal of a mandatory object property causes termination of the object.

Any interaction can be used for instantiation or removal of objects and their properties. Some properties can be interpreted as objects on their own. Property can play a

role of object, if it is characterized by its own properties. Such objects must be created prior to be connected as properties of other objects. For example, a Contact Person is a property of Paper [Submitted]. It is also an object, because each Contact Person object is characterized by E-mail, which is inherited from a Person. Instantiation of this object property of Submitted Paper cannot be done prior creation of a Contact Person object itself. The conceptualization, which is presented in figure 9, can be considered as a consistent extension of the previous diagram. It represents possible ways for creation and termination of various kinds of Person objects in a conference management system.

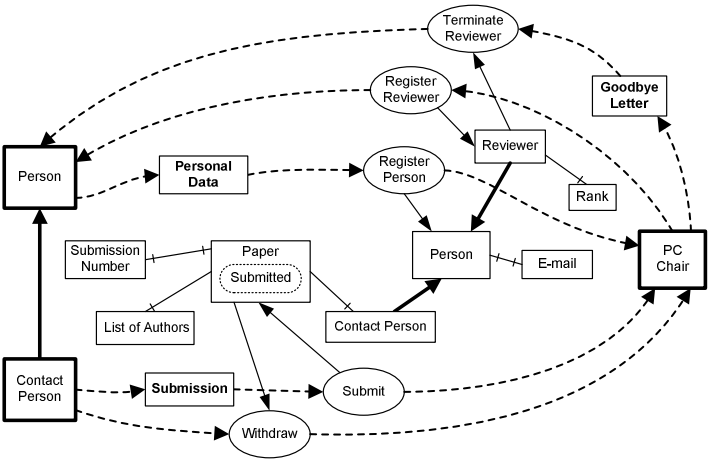

Fig. 9. Complementary interactions in a conference management system

Note that Submit communication action requires first creation of a Person object by using Register Person action. Thus, this action is considered as decomposition (like a use case inclusion) of Submit action. If the Register Person action is missing, then effects of Submit action can be described as follows: 1) Create a Person with E-mail property, 2) Create Paper [Submitted] with such properties as Submission Number, List of Authors and Contact Person, which is reclassified by using exactly one object of a Person class. The Register Person action can be also viewed as an extension of the Register Reviewer action.

Termination of object requires removal of object properties that are defined by the mandatory attributes of a pre-condition class. For example, termination of Reviewer will cause removal of Rank, but it will not cause removal of E-mail property, because an object of Person is not terminated. Sometimes, termination of object may cause removal of it as a property of other objects. Removal of one essential property may be sufficient for the violation of some mandatory dependency links, which would cause termination of other objects. Note that termination of an object (and removal all its properties) does not mean termination of the dependent objects. For example, Contact Person is a property of Paper [Submitted]. Termination of Paper [Submitted] with its contact person property does not mean termination of a Contact Person object, which is characterized by E-mail property. Nevertheless, the removal of a Person or a

Contact Person object would cause termination of the associated Paper[Submitted] objects. In some cases, post-condition class constraints may override termination of a pre-condition class object. One such case is reclassification action with the post-condition class, which is either specialization of the pre-condition class or the pre-condition class is viewed as a mandatory attribute of the post-condition class. Quite often objects are not preserved (see Reviewing in figure 8). They may pass several classes and then are terminated.

6 Concluding Remarks

Conceptual modeling methods, which put into foreground active concepts, typically focus on analyzing interactivity between subjects. The starting point of the enterprise modeling language ArchiMate (Lankhorst et al., 2010) as well as the DEMO approach (Dietz, 2006) is stemming from this modeling tradition. On the contrary, most of the conventional system analysis and design approaches put into the foreground modeling of passive concepts. Only few emerging approaches make attempts to express deep semantics of interplay (Dori, 2002) between active and passive structures of concepts. Analyzing various diagrams in isolation create difficulties in detecting requirement conflicts by business experts, who determine the organizational strategies. Consequently, information system methodologies are not able to bridge a communication gap among business experts and IT-system designers. The presented modeling approach for semantic integration for static and dynamic aspects provides several advantages. It is based on a single diagram type and therefore semantic integrity rules can be introduced directly into the model. Particular views, which define structural, behavioral or interactive aspects, can be generated by producing projections of an integrated model.

The lack of conceptual modeling approach, which can be used for the detection of semantic integrity problems among various types of diagrams, is the cornerstone of frustration for enterprise architects. The basic underlying principle in UML is to provide separate models for different modeling dimensions. According to the ontological principles, which are developed by Bunge (Bunge, 1979), the structural changes of objects are manifested via object properties. Properties in our modeling approach are expressed 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). The decline in cognitive processing performance, that occurs when optional attributes and relationships are used, appears to be substantial (Bodart et al., 2001).

The possibility to conceptualize interactions, behavior and effects of structural changes in a single diagram is not the only benefit of the presented modeling approach. Another important advantage is stability and flexibility of diagrams in dealing with evolutionary changes. Our initial studies demonstrate that separating and merging crosscutting concerns in the presented modeling approach is more efficient in comparison to the conventional modeling techniques. Typically, semantically equivalent changes that are required to be introduced in activity, state transition and sequence diagrams are quite substantial. We should focus on this topic in our future research. Conceptual modeling of service interactions is useful for separation

crosscutting concerns among organizational and technical components. A new modeling approach was demonstrated on small scale example. Let us extend the initial description of a conference management system by introducing the following requirement: '*contact person should have a possibility to* ***withdraw*** *a submitted paper from a conference at any time'*. Such new requirement would cause a very simple extension of the diagram, which is represented in figure 8 (with withdraw communication action between Contact Person and PC chair). Termination of a Paper [Submitted] by Withdraw action is represented in figure 9. Related triggering effects of withdraw action can be visually recognized from the previous conceptualization (see figure 8) by using the rules, which are presented in this paper. There are four sets of effects, which can be identified by analyzing states of a Paper object such as Submitted, In Review, Reviewed and Accepted. Unfortunately, defining semantics of Withdraw action would require significant extensions of UML diagrams. Four new sequence diagrams must be introduced for specification of related effects. The presented state transition diagram (see figure 3) should be extended by four new state transitions with their associated events and effects. The complexity of the activity diagram would also increase dramatically. The problem is that four new variations of withdraw activities must be introduced for evaluation of withdrawal conditions of *Submit*, *Appoint Reviewers*, *Return Review* and *Accept* events.

Traditional information system analysis and design methods are projecting the structural and behavioral aspects of conceptualizations into totally different types of diagrams. The UML individual diagram types are clear, but integrated semantics among models is missing. That is why object-oriented diagrams are difficult to apply for business logics alignment with implementation specific design for making both organizational and technical system parts more effective. The presented modeling approach is capable to capture, in concise form, semantics of structural changes, which are motivated by interaction flows. Similarity of conceptual models before and after adding a complimentary requirement, demonstrates stability of the presented modeling approach.

References

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

Bodart, F., Patel, A., Sim, M., Weber, R.: Should Optional Properties be Used in Conceptual Modelling? A Theory and three Empirical Tests. Information Systems Research 12(4), 384–405 (2001)

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

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

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

Evermann, J., Wand, Y.: Ontology Based Object-Oriented Domain Modeling: Representing Behavior. Journal of Database Management 20(1), 48–77 (2009)

Gane, C., Sarson, T.: Structured System Analysis. Prentice Hall, NJ (1979)

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

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

van Griethuisen, J.J.: Concepts and Terminology for the Conceptual Schema and Information Base, Report ISO TC97/SC5/WG5, No 695 (1982)

Gustas, R., Gustiene, P.: Pragmatic – Driven Approach for Service-Oriented Analysis and Design. In: Information Systems Engineering - from Data Analysis to Process Networks. IGI Global, USA (2008)

Gustas, R., Gustiene, P.: Service-Oriented Foundation and Analysis Patterns for Conceptual Modelling of Information Systems. In: Information System Development: Challenges in Practice, Theory and Education, vol. 1. Springer, Heidelberg (2009)

Gustas, R.: A Look behind Conceptual Modeling Constructs in Information System Analysis and Design. International Journal of Information System Modeling and Design 1(1), 79–108 (2010)

Gordijn, J., Akkermans, H., van Vliet, H.: Business Process Modelling is not Process Modelling. In: Mayr, H.C., Liddle, S.W., Thalheim, B. (eds.) ER Workshops 2000. LNCS, vol. 1921, pp. 40–51. Springer, Heidelberg (2000)

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

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

Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edn. Pearson Education, NJ (2009)

Lankhorst, M.M., Proper, H.A., Jonkers, H.: The Anatomy of the ArchiMate Language. International Journal of Information System Modeling and Design 1(1), 1–32 (2010)

Martin, J., Odell, J.J.: Object-Oriented Methods: A Foundation (UML edn.) Prentice-Hall, Englewood Cliffs (1998)

OMG. Unified Modeling Language Superstructure, version 2.2. (2010), http://www.omg.org/spec/UML/2.2/ (retrieved January 19, 2010)

Yourdon, E., Constantine, L.L.: Structured Design. Prentice Hall, NJ (1979)