A Method for Conceptual Modeling of Semantically Integrated Use-Case Scenarios

Remigijus Gustas and Prima Gustiene

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

Abstract. A use-case is specified as a set of possible scenarios of interactions. Scenarios can be decomposed into workflows on different granularity levels. Use-cases are fundamentally a text-based documentation form written in natural language text. The textual description of a complicated scenario can be ambiguous, incomplete and inconsistent. In this paper we demonstrate a conceptual modeling method for representing use-case descriptions by using a graphical language. Simple interaction loops are viewed as fundamental elements for composition of scenarios. Each interaction loop is analyzed separately and therefore it provides a natural way of decomposition. Modeling of overlaying interaction flows between organizational and technical components enables separation of crosscutting concerns in system engineering without requirement to specify a complete solution. The goal of this paper is to demonstrate the advantages of conceptual modeling approach, which allows to introduce evolutionary extensions and to construct use-case scenarios with a comprehensible structure.

Keywords: Interaction dependencies, separation of concerns, basic pattern of a transaction, interaction loops, scenarios.

1 Introduction

Use-cases are a way to capture requirements. It is the form of requirements engineering [1]. According to Cockburn [2] a use-case is a description of the possible sequences of interactions between the system and its external actors. Each sequence of interaction events can be viewed as a scenario. Every use-case contains a set of possible scenarios related to the goal of a primary actor. Historically use-cases descriptions are written as text-based documentation. Natural language is verbose and flexible to use in conversations, but it is really problematic when it comes to system modeling as natural language is notoriously ambiguous in its meaning [3]. Especially, complex scenarios may result in ambiguous, incomplete and inconsistent textual descriptions.

According to Jacobson [4], use-cases can be seen as different stakeholders concerns, which are important to understand while exploring and collecting the requirements for the system. Very often a certain concern is spread across multiple components. It means that the realization of functional requirements, which are usually specified as use-cases can cut across multiple components. Inability to keep crosscutting concerns separate causes problems for system designers when the requirements are modified.

T. Skersys et al. (Eds.): I3E 2011, IFIP AICT 353, pp. 138–152, 2011.

The designer must identify all related components and to find out how these components are affected by introduced changes. Especially, modifying the requirements, which are related to a big number of diagrams, is quite problematic. Poor understanding of crosscutting concerns makes it difficult to make even simple evolutionary extensions of information system (IS) specifications.

The scope of scenario can vary. It may include all business events, or it may include just some events, which are of interest to one specific actor. Scenarios can be used to define workflows on different granularity levels. Our studies indicate that simple interaction loops [5] can be viewed as fundamental elements for the composition of scenarios. Each interaction loop can be analyzed separately as it is required by the principle of separation of concerns. In such a way, interaction loops provide a natural way of decomposition. Two related loops can be used for analyzing integrity between static and dynamic aspects of some scenario. They are also useful as a concern composition mechanism. A scenario is an excellent means for describing the order of interactions. It can be conceptualized as a combination of several loops between a set of loosely coupled actors.

Use-cases can be described on different levels of abstraction and they can be combined into scenarios in various ways. In the object-oriented modeling approaches, a scenario is typically specified by a textual narrative description. A textual description of a complicated scenario can be incomplete and inconsistent with other representations. One of the goals of this paper is to demonstrate how use-case scenarios can be expressed by using Semantically Integrated Conceptual Modeling (SICM) method [6]. This approach puts into a foreground the modeling of interactions [7] among actors [8]. Interaction dependencies are used to preserve the modularity of concerns and to integrate behavioral effects with structural changes in various classes of objects.

Use-cases can be viewed as slices, which are analogous to overlays. Such overlays can be stacked on top of each other. In this way, overlays can be understood as a concern separation technique [4]. Use-case slices define the behavior that extends the element structure. So, it looks like use-case composition mechanism and concern separation principles are quite obvious. However, introducing evolutionary changes, which are related to a big number of diagrams, is quite problematic in object-oriented approach using Unified Modeling Language (UML). It is common to all system analysis and design methods to separate disparate views [9], because a human mind allows focusing on a particular type of diagram at a time. Designers are typically dealing with one type of diagram, which defines behavioral, interactive or structural aspects of a system in isolation. Therefore, it is difficult to take into account semantic dependencies between the static and dynamic aspects related to a particular concern in a very early modeling phase. That is why most conventional IS design methods are not so useful for the detection of inconsistency or incompleteness in various use-cases specifications. In this paper, we will demonstrate how interaction dependencies can be used to construct unambiguous graphical descriptions of scenarios with sequential, iterative, synchronized and alternative behavior. Most of information system methodologies are quite weak in representing the alternative scenarios and the consequences if commitments between actors are broken.

2 Use-Case Descriptions

Use-case diagrams can be produced in two steps. The first task of the system analyst or project team member is writing use-case descriptions by using a natural language text. The second task is to translate the use-case descriptions into use-case diagrams. A use-case description should contain all needed information for building other UML diagrams. Unambiguity of use-case descriptions is important to validate the semantic integrity of the diagrams. The problem is that the narrative text, which defines flows of events of different use-cases, can be ambiguous, incomplete and inconsistent. There are three types of event flows, which are documented for a use-case: 1) Normal flow of events, 2) Subflows, and 3) Alternate flows.

A use-case can be understood as a transaction. Any transaction can be analyzed as a simple workflow loop [5], which captures service value exchange between two or more parties. Both requests and responses are viewed as necessary business events. According to Ferrario and Guarino [10], service interactions are not objects or products, they are events. Service responses cannot be delivered without initiating service requests. A response can be viewed in a number of ways. It can be represented by a promise to deliver a desirable result to service requester or it can be viewed as statement, which brings a desired value flow [11] to service requester. Any workflow loop indicates that service provider receives service request and transforms it into service response. Service requester, request, service provider and response are minimal set of elements for defining any type of a service interaction loop. It is illustrated graphically in figure 1.

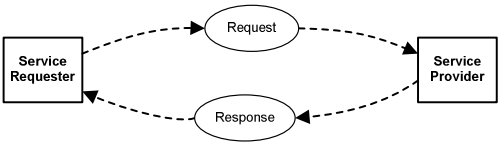

Fig. 1. Basic interaction loop

The presented elementary interaction loop is a basic element, which can be used in a very early conceptual modeling phase for separation of concerns in information system analysis and design. Two loosely coupled actors will be represented by the following expression [12]:

If Request(Service Requester ▪▪▪▪▶Service Provider)
then Response(Service Provider ▪▪▪▪▶ Service Requester).

We will demonstrate the conceptualization of normal flow of events and alternate flows for the slightly modified case study example, which was analyzed by Jacobson and Ng [4]. In this example, the Reserve Room use-case is extended by the Handle Waiting List use-case. Both use-cases are represented in figure 2.

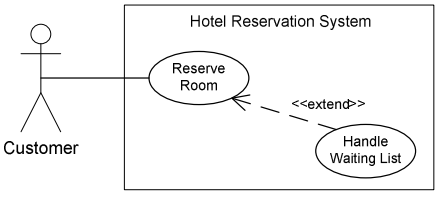

Fig. 2. Example of use-case diagram

The Reserve Room use-case scenario can be graphically defined by two interaction loops between Customer and Hotel Reservation System. The primary interaction loop is composed of the underlying interaction loop. Both loops of the Reserve Room use-case scenario are graphically defined in figure 3.

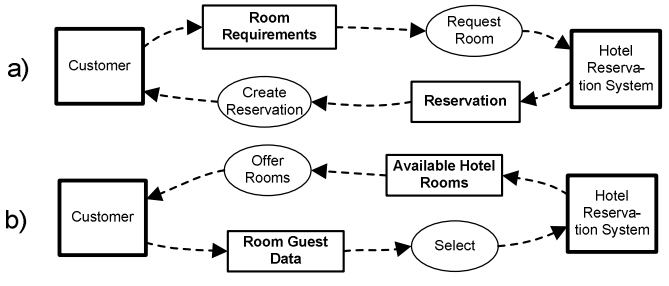

Fig. 3. Elementary interaction loops of the Reserve Room use-case

The presented service requesters and service providers are actors. Actors are active concepts, which can only be motivated by a set of interaction dependencies that keep the enterprise system viable. Interaction dependency R(A┅►B) indicates that actor A is an agent, which can perform action R on one or more recipients B. Interaction dependencies in the diagrams are graphically indicated by broken arrows. Broken arrows denote moving flows between actors such as information, decisions or materials. Actors are represented by square rectangles and actions are represented by ellipses. All actions are used to show the legal ways in which actors interact with each other. The first interaction loop (it is graphically represented in figure 3,a) cannot be executed without triggering the underlying interaction loop on the more specific granularity level (figure 3,b). The second interaction loop is initiated by the Hotel Reservation System.

Use-case scenarios are conceptualized by identifying the flow of interaction events. Each event makes about the same amount of progress towards completion of a use-case. The equivalent narrative text of the Reserve Room use-case normal flow of events (see figure 3) can be described as follows:

1) The Customer **requests room** by entering the specific room requirements (including the desired period of stay),

2) The Hotel Reservation System **offers** various available **rooms** with different rates,

3) The Customer **selects** the available room and enters the required information about an expected room guest. This step can be repeated several times. It means that more than one room can be selected,

4) The Hotel Reservation System **creates reservation** with the details of all selected hotel rooms, displays reservation information to the Customer and consumes the selected types of available room.

Both graphical and textual descriptions in essence define the Reserve Room use-case as the flow of interaction events. Events are identified by using two kinds of guidelines as far as the syntactic and semantic structure of sentence is concerned. The syntactic guideline has something to do with the form of sentence. Each individual event should be expressed as subject-verb-object and, optionally, preposition-indirect object [13]. Such constructions are useful in identifying actors, operations and classes. The second set of guidelines is related to the semantic roles various concepts play in the sentences. Normally, each event should define the requester or performer of the action. The use of request–response type of sentences results in user-centered requirements documentation that can be useful for user training and testing. If the use-case description is too complex, it should be decomposed into a set of more simple flows. Subflows or alternate flows can be used for this purpose.

3 The Basic Constructs of SICM Approach

Event flows can be defined in terms of essential interactions between organizational or technical components. Technical components correspond to enterprise subsystems such as machines, software and hardware. Organizational components can be humans, organizations, their divisions or roles, which denote the groups of people. Interaction dependencies among actors are important for the separation of crosscutting concerns. By following interaction dependencies, it is possible to explore various ways in which enterprise system components are used. Event flows can be analyzed as a set of workflow loops [5]. A workflow loop in SICM method [14] is considered as a basic element of scenario, which describes interplay between service requesters and service providers. In its simplest form, any workflow loop is viewed as a response to request that creates promise or provides a value to service requester.

Interaction dependencies are extensively used in a foreground of enterprise engineering methods [7]. These methods are rooted in the interaction pattern analysis and philosophy of language. The purpose of introducing them was initially motivated by the idea of creation computer-based tools for conducting conversations. The goal of this paper is different. We are going to demonstrate how to apply the interaction dependencies in combination with the set of semantic dependencies, which can be used for the graphical description of use-case scenarios. The sequences of interaction events are crucial for analyzing scenarios, which are expressed in terms of requests and responses between actors. For example, Create Reservation action can be viewed as a promise in connection to Request Room action.

Behavioral and structural aspects of interactions can be analyzed in terms of their reclassification, creation or termination effects. When two subsystems interact one may affect the state of each other [15]. Structural changes of objects are defined in terms of object properties [16]. 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 objects. Otherwise, this action is not purposeful. Property changes may trigger object transitions from one class to another. The behavioral effects of communication actions are expressed by using transition links (——►) between various classes of objects. Reclassification of object can be defined in terms of communication action that is terminating an object in one class and creating it in another class. Sometimes, objects may pass several classes, and then they are terminated. Graphical notation of the reclassification construct is graphically represented in figure 4.

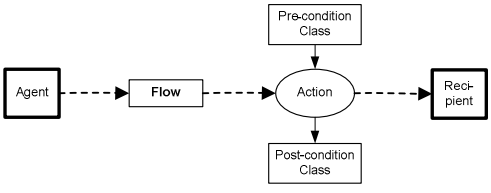

Fig. 4. Graphical notation of reclassification

Unbroken arrows indicate control flow of creation and termination effects. Object classes represent a persistent or transient set of objects. Fundamentally two kinds of changes are possible during any reclassification: termination and creation of an object. A creation is denoted by outgoing transition arrow to a post-condition class. A termination action is represented by a transition dependency directed from a pre-condition object class. Before an object can be terminated, it must be created. A pre-condition class in the termination construct is understood as final. For instance, when the Hotel Reservation Request is created, it can be reclassified to the Hotel Reservation by using Create Reservation action.

Structural changes of objects are manifested via static and dynamic properties. Dynamic properties are represented as actions, which are connected to classes by creation and termination links. Static properties can be represented by the mandatory attributes. The mandatory attributes are linked to classes by the single-valued or by multi-valued attribute dependencies. One significant difference of the presented modeling approach is that the association ends of static relations are nameless. The justification of this way of modeling can be found in some other papers [12], [17]. 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 various static dependencies is represented in figure 5.

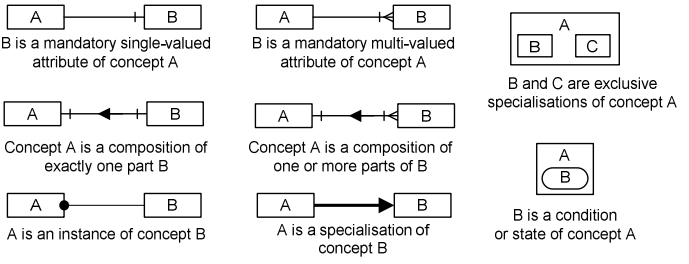

Fig. 5. Notation of static dependencies between concepts

The diagram presented in figure 3 does not provide any semantic details of control flows between communication actions. It shows only the necessary events in a Room Reservation scenario. The actions such as Request Room and Create Reservation should also specify the acceptable ways for structural changes to occur in different classes of objects. In general, communication actions can be sequential, iterative, alternative or synchronized with the secondary workflow loops. Triggering conditions of the secondary interaction loops may depend on the objects, which are created or terminated in the overlaying interaction loops. Pre-condition and post-condition classes are crucial to understand the dynamic aspects of interactions. A created object in one loop can be consumed in another. The creation and termination of objects allows constructing scenarios, which are enclosing optional or mandatory workflows.

Overlapping classes can be used to synchronize interaction loops together. For instance, Create Reservation action terminates Hotel Reservation Request object, which was created as a result of Request Room action, and creates Hotel Reservation object. These creation and termination effects are graphically described in figure 6.

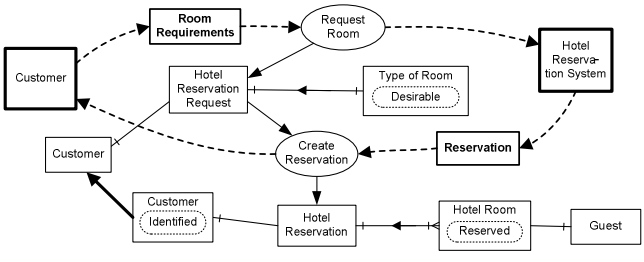

Fig. 6. Overlaying description of the Reserve Room use-case scenario

The corresponding Hotel Reservation and Hotel Reservation Request objects are composed of parts, which must be terminated and created at the same time. In this way, creation and termination effects define constraints on various types of objects in sending and receiving interaction flows between actors. Inheritance, composition and mandatory attribute dependencies can be used for reasoning about the consequences of object creation and termination effects. According to the conceptual modeling rules [6], the creation of Hotel Reservation is causing creation of at least one Hotel

Room[Reserved] and expected Guest. It should be noted that the semantic power of UML object flow and sequence diagrams combined together is not sufficient for capturing the equivalent effects.

4 Composition of Interaction Loops

Elementary interaction loops can be viewed as fundamental elements for defining scenarios [17]. Graphical representation of scenario is an excellent means for describing the order of interactions. More specific underlying interaction loops can be analyzed in the context of the overlaying loops on higher granularity levels as it is required by the principle of separation of concerns. It means that the scope of scenario can vary. A scenario may include just some events, which are of the interest to one specific actor. However, it can naturally be linked to all other overlaying business events. The natural language descriptions of such complex use-case scenarios require a lot of supplementary information, which specifies pre-conditions, post-conditions and other special constrains for inserting subflows into main flow of events. In this section, we demonstrate how to construct the graphical descriptions of scenarios with a more comprehensible structure.

The Reserve Room use-case scenario can be graphically defined by a number of interaction loops between Customer and Hotel Reservation System. The primary interaction loop (see figure 6) can be viewed as an overlay of secondary loop on more specific granularity level. Composition of two interaction loops of Reserve Room use-case scenario is presented in figure 7.

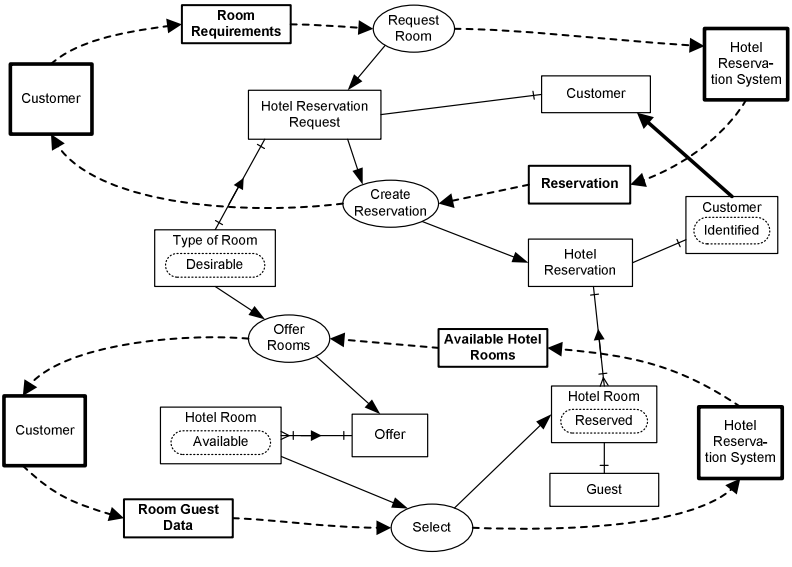

Fig. 7. Integrated conceptual representation of the Reserve Room scenario

The primary loop of the Reserve Room use-case is synchronized with the underlying interaction loop. It defines the functionality of offering and selecting available hotel rooms. The underlying interaction loop is as follows:

If Offer Rooms(Hotel Reservation System ▪▪▪▪▶Customer)
then Select(Customer ▪▪▪▪▶ Hotel Reservation System).

According to the presented control flow, the Select room action cannot be triggered prior to Offer Rooms action. Select action can only be performed in parallel with the Create Reservation action, because the creation of Hotel Reservation is synchronized with the creation of its compositional part Hotel Room[Reserved]. The termination of Hotel Reservation Request is synchronized with the reclassification of Type of Room[Desirable] to Offer, which is composed of at least one Hotel Room [Available].

A simple interaction loop between service requester and provider in the SICM method is viewed as the basic element of any communication process [5]. Interaction loops may be composed together into more complex interaction webs by using creation and termination links. If the object transition effects cannot be conceptualized by using pre-condition or post-condition classes, then the communication action is not purposeful. Interaction dependencies without purposeful actions make no sense and should be eliminated. The modeling of interactions and object transition effects together is critical for reaching semantic integrity among static and dynamic aspects of IS specifications. In the presented example, object creation, termination and reclassification effects show very important semantic details of unambiguous scenario in which two interaction loops are composed together. The more specific workflow loop is underlying the primary interaction loop. The underlying loop is required for the selection of desirable room type and for providing necessary data about guest. Underlying loops can be mandatory or optional. By following the interaction dependencies between actors, designers are able to understand the creation and termination effect in various classes of objects. In this way, the transition links are used to capture the dynamic dependencies between interaction loops.

5 Bridging from SICM Constructs to the Basic Pattern of a Transaction

Interaction dependencies are successfully used in the area of enterprise engineering [7]. Our intention is to apply the interaction dependencies in combination with the conventional semantic relations, which are used in the area of system analysis and design. Interaction loops can be expressed by interplay of coordination or production events, which appear to occur in a particular pattern. The idea behind a conversation for action schema [18] can be explained as turn-taking. Requester (R) initiates a request (R: Request) action and then is waiting for a particular promise (P: Promise) or a service provision (P: State) action from Performer (P). *Request*, *Promise* and *Acceptance* are typical coordination actions, which are triggered by the corresponding types of basic events. Coordination events are always related to some specific production

event. Both coordination and production events can be combined together into scenarios, which represent an expected sequence of interactions between requester and performer. We will show how creation, termination or reclassification constructs of the SICM method can be used to define the new facts, which result from the main types of events of the basic transaction pattern [7]. Four basic events and related reclassification effects are represented in figure 8.

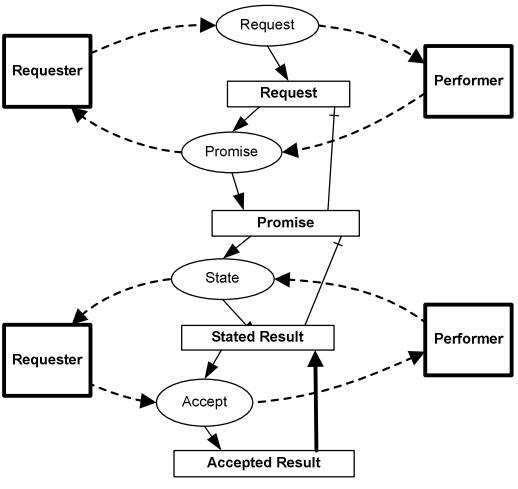

Fig. 8. The basic pattern of a transaction

New facts resulting from four basic events are instantiated by such classes of objects as *Request*, *Promise*, *Stated Result* and *Accepted Result*. Two interaction loops between Requester and Performer of the basic transaction pattern are composed together. A promise is created in the first interaction loop. It can be consumed in the next interaction loop. Created or terminated objects and their properties are interpreted as facts, which represent requests, promises and statements about delivered or accepted results. For instance, the Create Reservation action in figure 6 can be interpreted as a Hotel's promise to Provide Hotel Room. Request Room and Create Reservation are typical coordination actions, which can be viewed as triggering events for a corresponding production action.

Two interaction loops, which are illustrated in figure 9, represent one example of the basic pattern of a transaction. It is obvious from the presented example that the Provide Hotel Room business event is viewed as a production event. It creates effects, which can be associated with the transition P: State in the conversation schema (see figure 8). Production event creates the new fact of Stated Result. For example, an Assigned Hotel Room is supposed to create a value for a Room Guest.

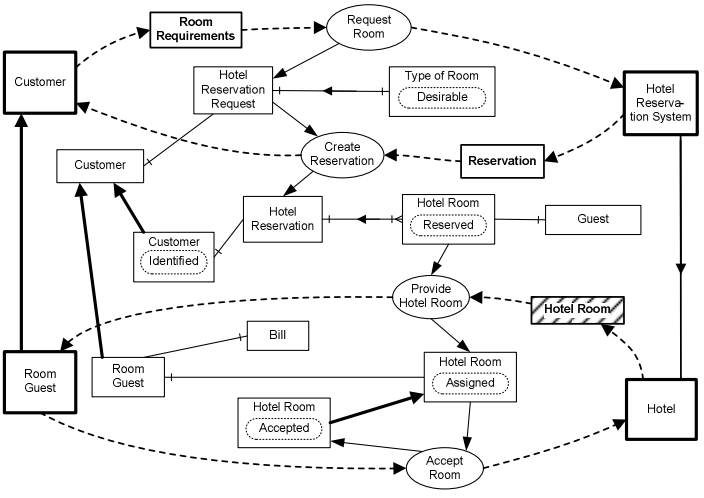

Fig. 9. Example of the basic transaction pattern

It is often the case in practice that the promise or acceptance actions are missing, because they are performed tacitly. For instance, the Create Reservation and Accept Room actions are missing in the following service interaction loop:

If Request Room(Room Guest ▪▪▪▶Hotel)
then Provide Hotel Room(Hotel ▪▪▪▶ Room Guest).

The pattern, which is illustrated in figure 8, defines the case when service requester and performer are consenting to each other's communication actions. For a communication action to be successfully performed an agent initiates the interaction flow and a recipient agrees to accept it. An enterprise system can be analyzed as the composition of the autonomous interacting components, which may not necessarily consent with each other. Actors can be involved in various interaction loops, because they want to get rid of problems or to achieve their goals. Goals, problems and opportunities [14] may help to understand why different actors act, react or not act at all. For instance, an agent may be not interested to initiate any interaction, or a recipient may refuse to accept the interaction flow. There are many other alternative business events [7], which may take be superimposed on the basic transaction pattern.

6 Alternative Interaction Loops

Alternative interaction loops should be introduced to handle possible breakdowns in the basic interaction pattern. These alternatives are represented by such reclassification actions as Reject and Decline in the standard pattern of a transaction. The alternative actions are necessary for actors involved in the business process to deal with unexpected situations. For instance, a performer may fail to deliver a desired result on time. A performer may experience difficulties in satisfying a request. For example, Hotel Reservation System may Reject Request, because the request requirements

were simply incorrect or incomplete. Instead of promising, the performer may respond by rejecting request. Requester may also express disappointment in stated result and decline it. Decline is represented by the termination of Stated Result and creation of Declined Result object. For instance, the Hotel Guest may decline the assigned hotel room, which was assigned by the Provide Hotel Room action. The basic transaction pattern can be supplemented with two dissent patterns, which are represented in figure 10. This extended schema is known as the standard pattern [7].

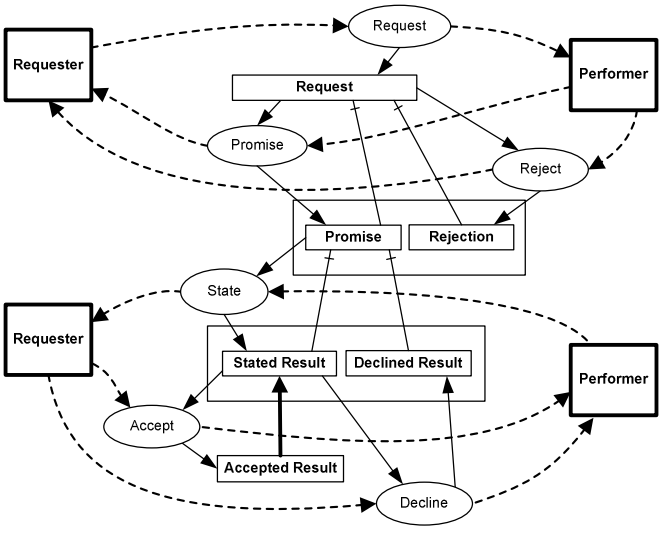

Fig. 10. The standard transaction pattern

Alternative actions can be represented by different reclassification, creation or termination events with the same object. For example, Request can be either reclassified to Promise or to Rejection. It means that Promise and Reject actions are exclusive. The creation of Promise or creation of Rejection object can only be performed once. The alternative actions must be introduced to handle the breakdowns in the main interaction pattern. For instance, the normal Reserve Room use-case scenario can be accomplished if and only if one or more desirable types of rooms are available for the required period of stay. This flow of events would fail when there are no available rooms, which can be offered. The alternative flow is inserted when the normal flow of events fails. The Handle Waiting List use-case represents such alternative, which can be described as follows:

1) If desirable type of room is not available (failure to offer at least one available room), then the Hotel Reservation System **offers waiting list** possibility.

2) If customer **rejects waiting**, then the Hotel Reservation System declines hotel reservation request by Reject Request action.

3) If customer **accepts waiting**, then the system puts customer on a waiting list and preserves information about his Hotel Reservation Request.

Two different alternatives for handling a Hotel Reservation Request are represented in figure 11 by Reject Request and Handle Waiting List actions.

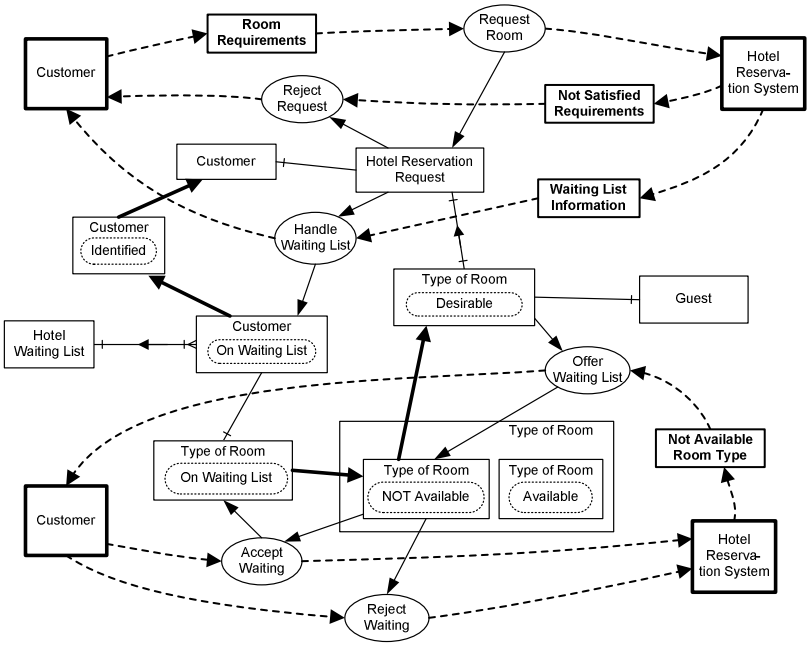

Fig. 11. Two alternatives of handling Hotel Reservation Request

Create Reservation action is an event of the expected scenario. It can be performed successfully on a condition that Hotel Reservation is created. Possibility of failure to compose the Hotel Reservation of at least one Hotel Room[Reserved] would cause a breakdown in the basic transaction pattern, which requires the definition of at least one alternative event. The first alternative is represented by Handle Waiting List action, which defines the reclassification effects of Hotel Reservation Request object. The second alternative is the termination of Hotel Reservation Request by Reject Request action. This option may be caused by a failure of the Handle Waiting List action. Please note that Customer[On Waiting List] object can be created just in case the Customer agrees to Accept Waiting in the underlying interaction loop.

In practice, it is also common that either requester or performer is willing to completely revoke some events. For example, the requester may withdraw his own request. There are four cancellation patterns [7], which may lead to partial or complete rollback of a transaction. Every cancellation action can be performed if the corresponding fact exists. For instance, the Withdraw Request action can be triggered, if a request was created by the Request action. In our example, Withdraw Request action is missing. Nevertheless, it is reasonable and should be introduced. The possibility to superimpose four cancellation patters on the standard pattern is not the only advantage of the presented modeling approach. The SICM method has sufficient expressive

power to cover the other special cases, which are not matching the standard pattern and four cancellation patterns. For instance, it is unclear how the methodology for design and engineering of organizations [7] would cope with the Handle Waiting List alternative, which is represented in figure 11. This option is also excessive in comparison with all legal transitions, which are defined by the conversation for action schema [18].

7 Concluding Remarks

The goal of this paper was to demonstrate the advantages of conceptual modeling approach, which allows introducing evolutionary extensions of use-case scenarios. We have demonstrated how use-case narrative descriptions can be replaced by graphical representations. Integrated conceptual modeling method was used to visualize event flows in terms of underlying, sequential and alternative interaction loops, which are fundamental elements for composition of use-cases scenarios. Elementary interaction loops are important for system architects to construct scenarios, which have an understandable structure. The networks of interaction loops may span across several organizations or partnerships. Each interaction loop can be analyzed separately as it is required by the principle of separation of concerns. In such a way, interaction loops provide a natural way of decomposition of use-case scenarios.

Introducing underlying interaction loops allows system designers to meet evolving needs of stakeholders and to avoid scenario breakdowns, which can be viewed as hidden requirements defects. The breakdowns in the main scenario can be eliminated by introducing the alternative actions, which are necessary to deal with failures. The presented way of interaction loop composition suggests a flexible way for managing the complexity of conceptual representations. We have demonstrated by examples some basic principles of a non-traditional conceptual modeling approach, which allows designers to visualize and to analyze semantic integrity between conceptual representations of use-case scenarios. The advantage of such conceptual representations is that interaction loops can be gradually enhanced or effectively replaced on demand.

Semantic integrity problems in the early system development stages are one source of errors, because use-case descriptions touch several classes. It is very difficult to achieve semantic integrity between the static and dynamic aspects of complex scenarios, because the conventional conceptual modeling methods are developed for analyzing business processes and business data in isolation. Most graphical modeling techniques are not flexible for the visualization of interplay among behavioral, interactive aspects and structural changes of objects. It was also demonstrated how sequential, iterative, parallel and alternative behavior is captured by identifying the transition dependencies between classes in various interaction loops. Separation of concerns is achieved by decomposing complex scenarios into simple underlying loops, which can be analyzed in isolation. The presented graphical descriptions of scenarios are easier to understand, extend and maintain.

References

1. Arlow, J., Neustadt, I.: UML 2 and the Unified Process. Practical object-oriented analysis and design. Pearson Education, Inc. (2009)
2. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Boston (2001)
3. Miles, R., Hamilton, K.: Learning UML 2.0. O'Reilly, Sebastopol (2006)
4. Jacobson, I., Ng, P.-W.: Aspect-Oriented Software Development with Use Cases. Pearson, New Jersey (2005)
5. Denning, P.J., Medina-Mora, R.: Completing the Loops. Interfaces 25(3), 42–57 (1995)
6. Gustas, R.: Modeling Approach for Integration and Evolution of Information System Conceptualizations. International Journal of Information Systems Modeling and Design 2(1), 45–73 (2011)
7. Dietz, J.: Enterprise Ontology: Theory and Methodology, p. 256. Springer, Berlin (2006)
8. Wagner, G.: The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. Information Systems 28(5) (2003)
9. Zachman, J.A.: A Framework for Information Systems Architecture. IBM System Journal 26(3), 276–292 (1987)
10. Ferrario, R., Guarino, N.: Towards an Ontological Foundation for Services Science. In: Domingue, J., Fensel, D., Traverso, P. (eds.) FIS 2008. LNCS, vol. 5468, pp. 152–169. Springer, Heidelberg (2009)
11. Gordijn, J., Akkermans, H., van Vliet, H.: Business Modelling is not Process Modelling. In: Conceptual Modeling for E-business and the Web, pp. 40–51. Springer, Berlin (2000)
12. Gustas, R.: Conceptual Modeling and Integration of Static and Dynamic Aspects of Service Architectures. In: International Workshop on Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Sciences, pp. 17–32. Springer, Hammamet (2010)
13. Dennis, A., Wixom, B., Tegarden, D.: System Analysis and Design with UML. An Object-Oriented Approach, 3rd edn. John Wiley & Sons (2010)
14. Gustas, R., Gustiené, P.: Pragmatic - Driven Approach for Service-Oriented Analysis and Design. In: Johannesson, P., Söderström, E. (eds.) Information Systems Engineering: From Data Analysis to Process Networks, pp. 97–128. IGI Global, New York (2008)
15. Evermann, J., Wand, Y.: Ontology Based Object-Oriented Domain Modeling: Representing Behavior. Journal of Database Management 20(1), 48–77 (2009)
16. Bunge, M.A.: Treatise on Basic Philosophy: A World of Systems Ontology II, vol. 4. D. Reidel, Dordrecht (1979)
17. Gustas, R.: Overlaying Conceptualizations for Managing Complexity of Scenario Specifications. In: IFIP WG8.1 Working Conference on Exploring Modeling Methods for System Analysis and Design, London, UK (2011)
18. Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Ablex, Norwood (1986)