Conceptual Modeling Patterns of Business Processes

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

Prima Gustiené
Department of Information Systems
Karlstad University
Karlstad, Sweden
Prima.Gustiene@kau.se

***Abstract* — System modeling patterns are similar to workflow patterns, which were established with the purpose of delineating the requirements that arise during business process modeling on a recurring basis. Traditionally, only dynamic aspects are used for the specification of modeling patterns leaving aside the static aspects of business processes. The paper presents the conceptual modeling patterns where integrity of totally different aspects can be analyzed. The advantage of such a modeling approach is that it enables visualization and integration of different modeling dimensions of information system specifications using a single diagram. Many graphical representations do not allow such visualization and integration of static and dynamic aspects. We also represent graphically interpretation of the conversation for action schema by constructs of our semantically integrated conceptual modeling method.**

Keywords-Modeling patterns; service-oriented constructs; static and dynamic aspects; sequence, iteration, synchronization, selection and enclosing patterns, universal interaction pattern.

I. INTRODUCTION

Analysis patterns are groups of concepts that represent a common construction in business modeling [7]. They are similar to workflow patterns that were originally established with the aim to define and visualize the fundamental requirements that arise during business process modeling on a recurring basis [19]. Workflow patterns are usually defined by using Business Process Modeling Notation, Unified Modeling Language (UML) Activity Diagram [16], or a Colored Petri-Net model [15]. All these notations are able to express process behavior but do not take into account the static aspects of business processes. They do not explicitly show what happens with the objects, which represent data, when some activity takes place. Integration of static and dynamic aspects is important for the control of semantic integrity among interactive, behavioral and structural aspects of a system [9]. Semantic integrity is critical to maintain the holistic representation of system specifications. To capture the holistic structure of the problem domain, it is necessary to understand how various components are interrelated. Analysis patterns presented in this paper are constructed using the principles of service orientation and they are called conceptual modeling patterns. These patterns are important for two major reasons. Firstly, they can be used for demonstration of the interplay among fundamental constructs that are used in system analysis and design process. Secondly, patterns are important for the evaluation of the expressive power of semantic modeling languages [18]. Comprehension and visual recognition of these patterns is necessary for building more specific pattern variations and composing them in different ways. Each modeling pattern language can be formally described using a set of modeling constructs and semantic rules.

Service-oriented modeling method [9] presented in this paper is based on the ontological principles [2] of the concept of service [6], and on a common understanding of the general structure of service, which is not influenced by any implementation decisions. The most fascinating idea about a service concept is that it can be applied equally well to organizational as well as technical settings. It means that the conceptual representations of service define computation independent aspects of business processes. Business processes can be seen as service compositions, which are used to specify service architecture. Service architecture can be applied for the specification of business processes in terms of organizational or technical services. Our assumption is that service-oriented representations can be communicated among business experts and system designers more effectively. Using service-oriented modeling, information systems can be structurally visualized as evolving conceptualizations of service architectures.

The concept of service in the area of information systems is mostly bound to the term of service-oriented architecture. According to Hagg and Cummings [12], Service-Oriented Architecture (SOA) is a software architectural perspective, where service is the same as component in component-based system development methodologies. SOA represents a set of guidelines and design principles, such as loose coupling, encapsulation, reuse and composability [5] [22], in which business processes can be effectively reorganized to support the business strategy [17]. From a business management perspective, SOA can provide the possibility to reach business flexibility. It enables business processes to be analyzed in terms of services. Conflicting views on the concept of service is one of the obstacles to the attempts to

develop a new science of services [3] and new academic programs focusing on services [1]. This discipline takes a broader perspective of services as opposed to technical descriptions [20].

We use the concept of service as in the sense of service science. It *"can be understood as an action or a set of actions that are performed for some value"* [21]. In the context of enterprise modeling, it is necessary to have a broader understanding and interpretation of the service concept as the definition of service goes well beyond activities that are realized using software applications. The definition of service provided by Sheth [20] emphasizes a provider - client interaction that creates and captures value. It emphasizes a value exchange between two or more parties and a transformation received by a customer [3]. The concept of service facilitates a change of business data from one valid and consistent state to another. In the public sector it sometimes denotes organizational actions. According to Ferrario and Guarino [6], services are not transferable, because they are events, not objects. The main purpose of service orientation is to capture business-relevant functionality. Taking into account the nature of the service concept, which is based on interaction between different actors to create and capture value, a service-oriented way of thinking could be applied for a computation-neutral analysis and design of business processes as well as for creation of conceptual modeling patterns.

This paper is organized as follows. In the next section, static and dynamic aspects of service interaction are described. Five different modelling modeling patterns of an integrated method are presented in the third section. In the in the fourth section, we describe a conversation for action schema and its interpretation in terms of a semantically integrated conceptual modeling method. Finally, concluding remarks are presented. This is an extended version of paper [1], which was published in BUSTECH 2014.

II. SERVICE AS AN INTERACTION

A service cannot be defined without specifying the interaction, the result of which creates value to the actors [8] involved. Service is first of all a dynamic act of doing something to somebody. It means that there are more elements necessary to construct a concept of service than just the process of 'doing'. As there are always some actors involved in such process, it signifies that it is a communication act or an interaction between human, organizational or technical components. One is asking for something and another actor provides it. The purposeful action always takes place in a service. It prescribes responsibilities for the actors involved [10]. Every business process action is goal-driven and it should always result in some value to an actor. To get the result, which provides value on demand, four key elements are necessary: service requester, service request, service performer and service response. Interrelations among these elements construct an interaction loop, which is necessary to represent service structure. Without one of these four elements, the concept of service loses its meaning. Service performers receive service requests and transform them into responses that are sent to the service requesters. Service can be characterized by an interaction loop that can be defined by a number of flows in two opposite directions. This idea is represented graphically by an elementary service interaction loop, which is delineated in Figure 1.

The main principle of service-oriented method is based on designing services as interactions among different enterprise actors. Service architecture can be represented by a composition of interaction loops. Actors in interaction loops can be seen as active elements. These elements can be organizational or technical subsystems. Organizational subsystems can be individuals, companies, divisions or roles, which denote groups of people. Technical subsystems can be represented as software or hardware components. Any coordination flow between actors [4] must be motivated by the resulting value flow. In such a way, any enterprise system can be represented and analyzed as a set of interacting loosely connected subsystems that form service architecture.

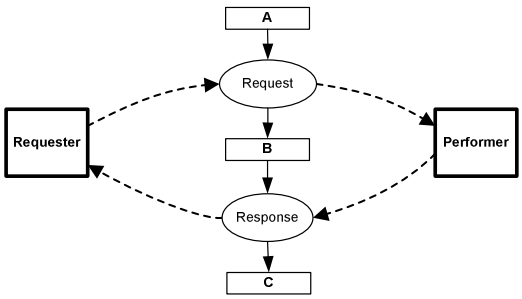

Figure 1. An elementary service interaction loop

The dynamic aspect of service includes not just interaction (▪▪▪▪▶) between actors, but also the resulting behavior among passive classes of objects when service actions are initiated. The transitions between passive classes of objects are resulting from interactions between active concepts. The internal behavior or so called objective perspective defines the dynamic aspect, which is expressed by object transitions between various classes of objects. Concepts A, B, and C define the structural aspects of data. These concepts constitute pre- and post-condition classes, which will be explained later. In such way, service modeling enables integration of business process and business data (see Figure 1).

There are two basic events for semantic modeling of service construct: creation and termination of objects [9]. These two events are used for the definition of a reclassification event, which is considered as a generic modeling construct. A creation event is denoted by an outgoing transition arrow to a post-condition class. A termination event is represented by a transition dependency directed from a pre-condition object class. Before an object is terminated, it must be created. Since a future class makes no sense for a termination event, it is not included in a

specification of action. Pre-condition class in a termination action can be understood as final during an object's life time. Reclassification of an object can be defined in terms of a communication action that is terminating an object in one class and creating it at the same time in another class. Sometimes, objects pass several classes, and then they are removed. A graphical notation of the reclassification action is presented in Figure 2.

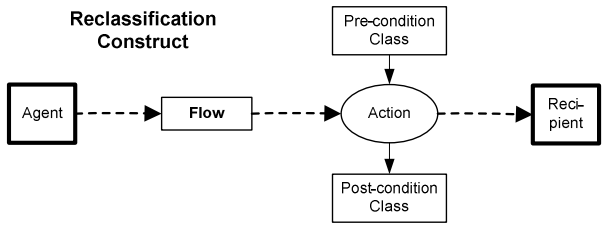

Figure 2. Graphical representation of a reclassification action

Fundamentally, three kinds of changes are possible during any transition (—►). An action is either terminating or creating an object, or it can perform termination and creation at the same time. Pre-condition and post-condition classes typically define constraints on objects, which restrict the sending and receiving of communication flows between technical or business components. A reclassification action in a computerized system can be implemented either as a sequence of one or more object creation and termination operations. Request and response flows, together with created and terminated object classes, are crucial to understand the semantic aspects of service interactions. A pre-condition object class and the input flow should be sufficient for determining a post-condition object class.

The attribute dependencies are stemming from the traditional data models. Semantics of static dependencies in object-oriented approaches are defined by multiplicities. They represent a minimum and maximum number of objects in one class that can be associated to objects in another class. We use only mandatory static dependencies from at least one side of association. A graphical notation of the attribute dependencies and their cardinalities is represented in Figure 3.

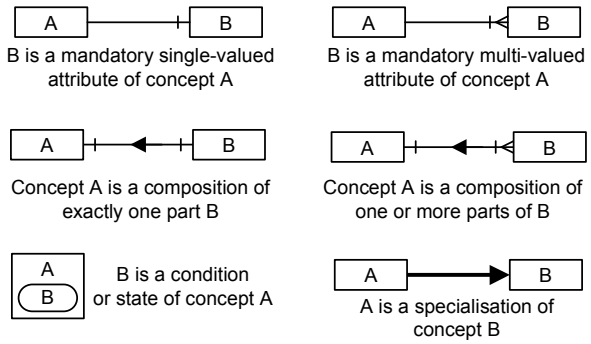

Figure 3. Graphical notation of the attribute dependencies

This notation corresponds to a classical way for representing associations between two entities [13]. One significant difference of this notation in service-oriented modeling method [9] from the traditional approaches is that the association ends are nameless. Dependencies are never used to represent association names or mappings between two sets of objects in two opposite directions. Any two concepts (in the same way as any two actors) can be linked by the attribute, inheritance or composition dependencies [9].

III. CONCEPTUAL MODELING PATTERNS

Constructs based on service orientation were used for the design of five modeling patterns. A single diagram type helps to focus on modeling integration of static and dynamic aspects. Various combinations of dependencies are able to express the main workflow control patterns such as sequence, iteration, selection, synchronization and enclosing of transaction. Ignoring the static aspects of data in the pattern modeling research creates fundamental difficulties. If just dynamic aspects are taken into consideration, then the quantity of patterns increases and their usage for business process modeling becomes more complex. Comprehensibility and visual recognition of the fundamental patterns is necessary in constructing more specific pattern variations by composing them in various ways.

Similar attributes are inherited by more specific classes according to the inheritance link (—►). Inheritance arrow denotes a specialization and generalization. Inheritance is always pointing out to a more general concept. In the diagram in Figure 4, it is possible to see two subclasses Reservation[Bill Sent] and Reservation[Paid], which are characterized by two different sets of dependencies.

We may distinguish between complete or incomplete as well as total and partial inheritance situations [24]. All these cases can be expressed by using the exclusive specialization and mutual inheritance link. Mutual inheritance dependency (◄==►) can be used for representing classes that are viewed as synonyms. It is defined as follows:
A◄==►B if and only if A—►B and B—►A.

Classification dependency (●—) specifies objects or subsystems as the instances of concepts. Classification is often referred to as instantiation, which is reverse of classification. It should be noted that classification dependency in the object-oriented approaches is a more restricted relation. It can be only defined between an object and a class. A class cannot play a role of meta-object, which is instantiated in another class.

Any class A can be viewed as an exclusive generalization of concepts B and C. A concept can be specialized by using a notion of state. For instance, *Payment* is specialized by *Payment*[*Confirmed*] as a result of confirm payment action. Various states of *Reservation* concept such as *Bill Sent* and *Paid* are also represented in Figure 4.

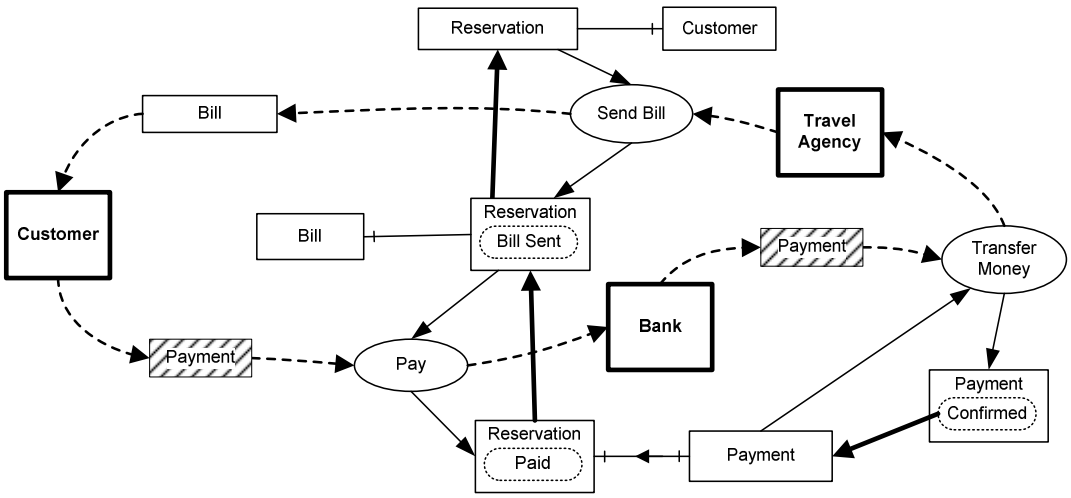

Figure 4. Example of static and dynamic dependencies

This example shows an interaction flows among three actors: Travel Agency, Customer and Bank. Interaction dependency (A ▪▪▪▶B) indicates that one actor (A) depends on another actor (B). Rectangles with shaded background are used for denotation of resource flows and light rectangles indicate information flows. Three communication actions *Send Bill*, *Pay*, and *Transfer Money*, which are triggered in a sequence, are used to express the business process of payment. *Pay* action can be executed only if the *Send Bill* action has been completed. It uses a *Reservation*[*Bill Sent*] object and produces a *Reservation*[*Paid*] object. When *Customer* receives the bill, he initiates the *Pay* action, which creates a new object *Reservation*[*Paid*] and links it to the specific *Payment*. A *Reservation*[*Paid*] consists of the compositional object of *Payment*. *Transfer Money* action can be executed only if the process of payment has been confirmed. So, according to this example, every action creates new object links that are associated with the post-condition object class. Since the post-condition class of Payment Confirmed is linked with the pre-condition class Payment by the inheritance link, the initial object is not terminated. Removal of objects in more general classes with their own attributes should occur if they are not preserved by the created objects. For instance, the missing inheritance arrow from the post-condition class would justify termination of a post-condition class object. Note that a Reservation object is required to be created in advance by another service, which is not presented in this example.

Service architecture can be composed of various interaction loops. The semantics of such composition is defined by using two or more constructs of the basic action (Figure 2). The composition of these three types of constructs can be used for the conceptualization of a continuous or finite lifecycle for one or more objects in the service interaction loop. A lifecycle of an object is typically represented by an initial, intermediate and final class. A creation event corresponds to a starting point and removal action – to the end point in an object's lifecycle. The most critical issue in the modeling of the interaction details is the semantic integrity of static and dynamic aspects. It is not sufficient to represent what type of objects are created and terminated. Service-oriented models must clearly represent attributes that must be either removed or preserved in any creation, termination and reclassification action. This is crucial to ensure the consistency of integrity constraints.

A. Sequence pattern

A pattern of sequence is a special case of an elementary interaction loop, which was presented in Figure 1. It consists of a request and response. A service request creates an object of type B, which in the second communication action is reclassified to the object of type C. These two actions are performed in a sequence and are represented in Figure 5.

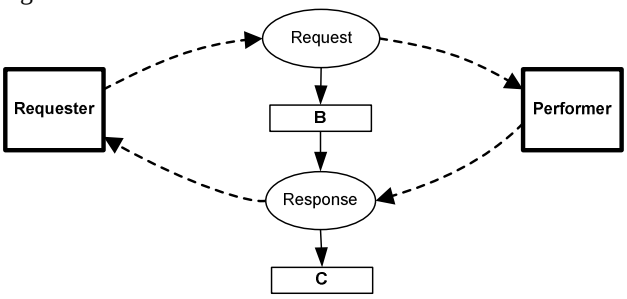

Figure 5. Sequence pattern

This pattern is used for representation the succession of events. For example, customer may order the goods by

creating a purchase order. If the goods are available, a vendor accepts the purchase order. The example of sequence pattern is represented in Figure 6.

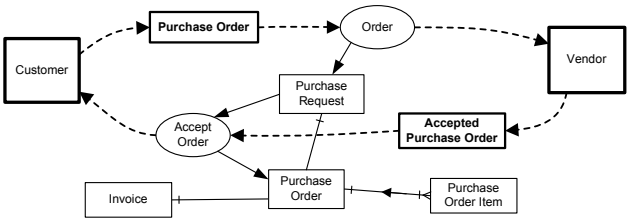

Figure 6. Example of a sequence pattern

A newly created purchase order is defined by three properties: Invoice, a set of Purchase Order Items and Purchase Request, which are necessary for delivering the order. It is not specified what will happen after that. Either the customer may withdraw the order, or it must be delivered.

B. Iteration pattern

Iteration pattern is a special case of a sequence pattern. It consists of one creation action and one removal action. The first action creates an object, which is subsequently removed. The iteration pattern is represented in Figure 7.

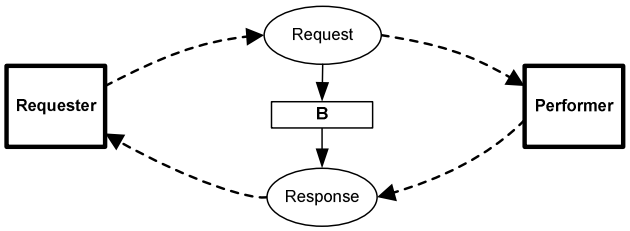

Figure 7. Iteration pattern

This pattern can be used for the representation of events that are repeated a number of times. For example, customer may order goods, which are not available. In this case, the vendor rejects the purchase order by removing it from existence. The message about the rejected purchase order is sent to the customer. The example of iteration pattern is represented in Figure 8.

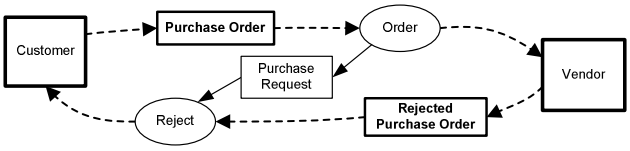

Figure 8. Example of iteration pattern

As we can see, when the Purchase Request is removed, the Customer may initiate a new the Purchase Order again. This interaction loop can be repeated a number of times. The diagrams that are represented in Figure 6 and Figure 8 can be superimposed into the single diagram. In this way, we can see what kind of alternative actions are available to the actors involved.

C. Synchronization pattern

A synchronization pattern is used when some activities must be performed concurrently. This pattern combines two parallel paths of activities. Both paths must be completed before the next process can take place. The primary interaction loop is composed of a more specific loop on a lower level of granularity. In this case, a service interaction loop on the lower layer of decomposition is viewed as an underlying interaction loop. The execution of the underlying loop must be synchronized with the primary interaction loop. The synchronization pattern is presented in Figure 9.

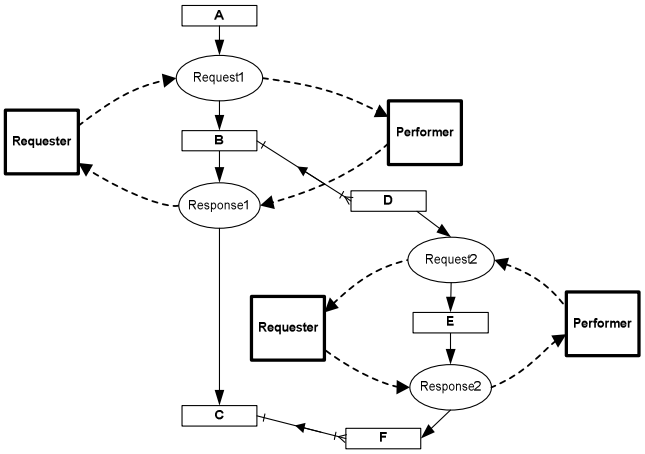

Figure 9. Synchronization pattern

This pattern illustrates that the action of Request1 creates a compositional object B, which consist of parts D. At least one part D must be created. Then object B is reclassified to C, object D must be also reclassified to E and then to F. If a compositional object is created, then the parts are created as well. If a compositional object is removed, then the parts are terminated at the same time. That is the reason why the action is propagated from a whole to a part according to the rule of class composition. The propagation of actions is a useful modeling quality. It allows a natural modeling of concurrency. Synchronization pattern is similar to concurrent activities (fork and merge of control) in an activity diagram [16].

The graphical example of synchronization is illustrated in Figure 10. In this example, the object reclassification effects represent the important semantic details of an unambiguous scenario in which three interaction loops are combined. Create Reservation action propagates to parts on the lower level of abstraction. Termination of *Hotel Reservation Request* requires termination of *Hotel Room*[*Desirable*]. Creation of *Hotel Reservation* requires

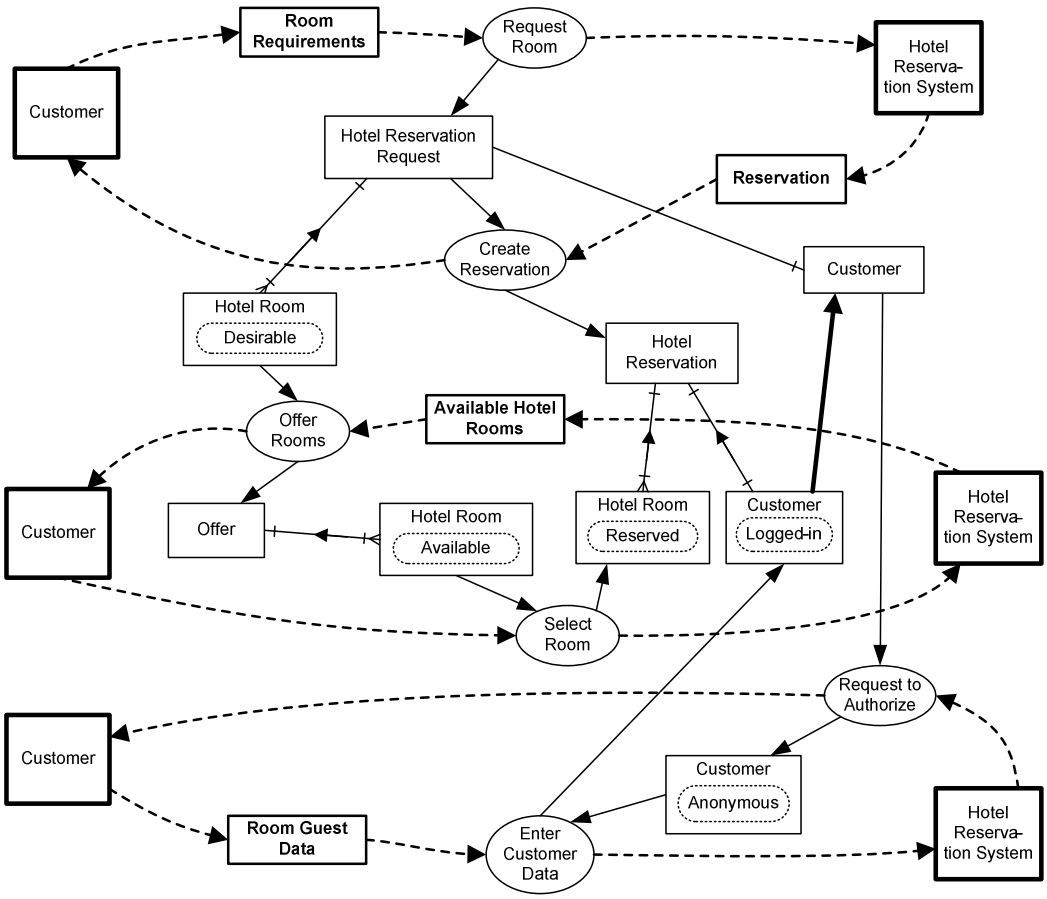

Figure 10. Example of a synchronization pattern

creation of one or more *Hotel Room*[*Reserved*]. According to the presented diagram, the underlying interaction loop action *Select Room* can be reiterated more than once, because *Hotel Reservation* is defined as the composition of one or more *Hotel Room*[*Reserved*].

The underlying interaction loop describes a Customer's response to the Hotel Reservation System's request. If a customer expects to receive a Reservation flow from the Hotel Reservation System, it is necessary for him to get a reply in the underlying loop from the technical component. The request and reply of the second underlying loop is specified as follows:
If Offer Rooms (Hotel Reservation System ▪▪▪▶ Customer), then Select Room(Customer ▪▪▪▶ Hotel Reservation System).

The actions of the underlying loop are synchronized with the primary interaction loop. According to the presented description, Create Reservation is a reclassification action, which is composed of the Offer Rooms and Select Rooms actions on the lower granularity level. The Select Room action cannot be triggered prior to the Offer Rooms action. It can be performed several times for each *Hotel Room*[*Available*]. *Hotel Reservation* is a compositional object. When it is created, such parts as *Hotel Room*[*Reserved*] and *Customer*[*Logged-in*] must be created as well. The first underlying loop is necessary for offering available rooms and selecting of a desirable room. Creation of *Customer*[*Logged-in*] object requires to initiate Request to Authorize and Enter Customer Data actions that are represented by the second underlying loop.

This modeling pattern is similar to a synchronization, which can be defined by fork and merge of control in UML activity diagram.

D. Selection pattern

The *Selection* pattern can be expressed using a composition of two different sequences between the same two actors. It represents two alternative outcomes of a service request that can be selected by service performer. Two possible ways of replying by performer are mutually exclusive. Only one type of response is expected by a service requester. If the first alternative is rejected, then the performer is trying to invoke the second alternative. The selection pattern was previously published and it can be found in [11]. It is similar to branches in UML [16]. The selection pattern is represented graphically in Figure 11.

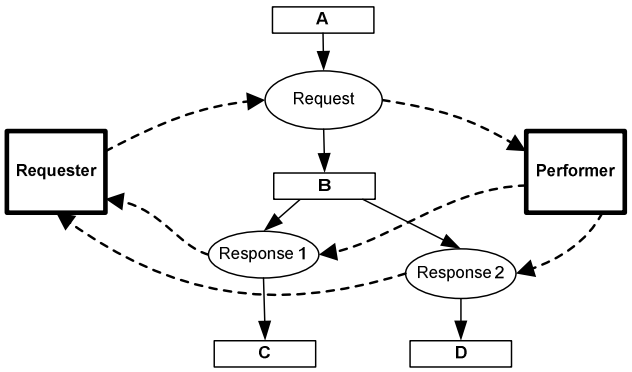

Figure 11. Selection pattern

Response 1 and Response 2 are two exclusive actions of a service performer. If Response 1 is initiated, then a pre-condition class object B is removed and a post-condition class C is created. If Response 1 has failed, then Response 2 is triggered, which reclassifies object B to D. The example of selection pattern is represented in Figure 12.

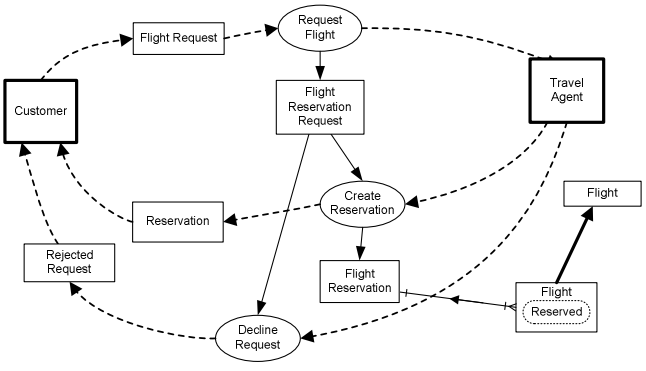

Figure 12. Example of a selection pattern

The selection pattern in the presented example can be explained as follows. The *Flight Reservation Request* is created and then it is reclassified into *Flight Reservation* in the *Create Reservation* action from the *Travel Agent*. If *Travel Agent* cannot create a *Flight Reservation*, then the alternative action of Decline Request is taking place. In this case, the *Flight Reservation Request* is terminated and a flow of Rejected Request is sent to the *Customer*. This action allows the *Customer* to reiterate the search again.

This pattern reminds us alternatives in UML, which are typically described as branches in activity diagram.

E. Enclosing pattern

An *enclosing pattern* is defined by a primary and a secondary interaction loop between requester and performer. In carrying out the work, a performer may play the role of requester in the secondary interaction loop by initiating further interactions. In this way, a network of loosely coupled actors with various roles comes into interplay to fulfill the original service request. Organizational systems may be composed of several interaction loops, which are delegated to more specific components. Enclosing pattern is similar to the enclosing of a transaction [4]. An enclosing pattern is represented graphically in Figure 13.

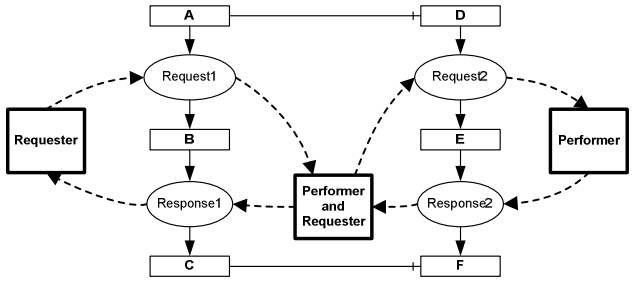

Figure 13. Enclosing pattern

The primary interaction loop consists of Request1 and Response1 actions. For the creation of object B in the primary loop, it is necessary to create its property E in the secondary loop. The reclassification of object B to C requires the removal of E and creation of F. So, the enclosing loop cannot be completed if the secondary loop is not finalized. The example of the enclosing pattern is represented in Figure 14.

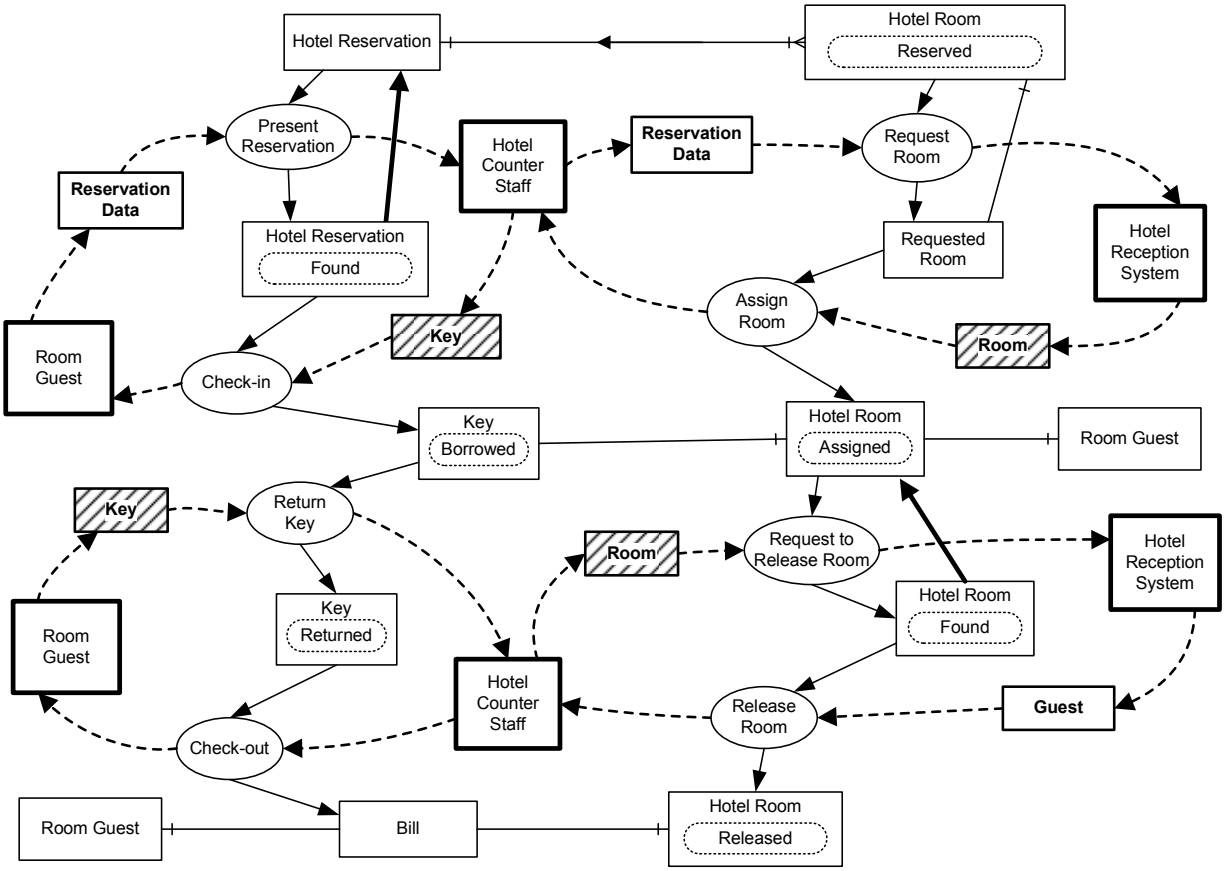

Figure 14. Two examples of enclosing pattern

If a *Room Guest* wants to *Check-in*, he needs to *Present Reservation* to the *Hotel Counter Staff*. If the hotel room is ready, the *Hotel Reception System* assigns this room to the hotel guest and produces the key, which is given to *Hotel Counter Staff*. The *Assign Room* action is executed by the *Hotel Reception System*, which is playing the role of software component. The *Check-in* action is performed by *Hotel Counter Staff*, which is playing the role of the organizational component. There is one enclosing and one enclosed interaction loop, which is represented in Figure 14. The primary interaction loop between Room Guest and Hotel Counter Staff encloses the secondary interaction loop between the Hotel Counter Staff and the Hotel Reception System. So, the *Assign Room* action is considered as a part of the *Check-in* action.

The business process, which is represented in Figure 14, consists of four interaction loops. The first primary loop is an organizational process. The secondary loop corresponds to a computerized process, which creates a *Hotel Room*[*Assigned*] object, and connects it with the *Room Guest* and *Key*[*Borrowed*] objects. The second primary loop is necessary for returning a key and checking-out a guest. It corresponds to an organizational process. The enclosed loop, which is initiated by a *Hotel Counter Staff*, corresponds to a computerized process. It is necessary for finding and releasing an assigned room.

IV. THE EXTENDED UNIVERSAL PATTERN OF INTERACTION

Interaction dependencies are extensively used in the context of enterprise engineering methods [4]. These methods are rooted in the interaction pattern analysis and the philosophy of language. The underlying idea of interaction pattern analysis can be explained by a well-known conversation for action schema [23]. The purpose for introducing this schema was initially motivated by the idea of creating computer-based tools for conducting conversations. Our intention is to apply the interaction dependencies as they are defined by the semantically integrated conceptual modeling approach [9] in combination with conventional semantic relations, which are used in the area of system analysis and design. Interaction loops can be expressed by the interplay of coordination or production events, which appear to occur in a particular pattern. This pattern is represented in Figure 15.

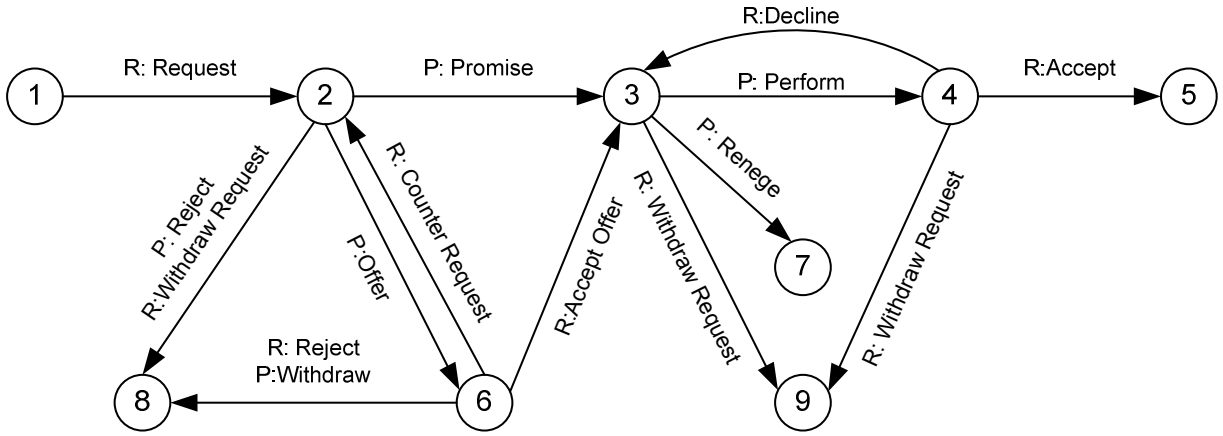

Figure 15. Conversation for action schema (Winograd & Flores, 1986)

The idea behind a conversation for action schema can be explained as turn-taking. Any service interaction pattern can be characterized by the same four types of main events, which compose a basic transaction pattern:

a) Request,
b) Promise,
c) Perform and
d) Accept.

The Requester (R) initiates a request (R:Request) action and then waits for a particular promise (P:Promise). 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, which is represented by P:Perform. 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 semantically integrated conceptual modeling method can be used to define the new facts, which result from the main types of events of the basic transaction pattern.

Various interaction alternatives between two actors can also be defined by interaction dependencies, which may produce different, similar or equivalent behavioural effects. A provider may experience difficulties in satisfying a request. Instead of promising, the service provider may respond by rejecting the request. For example, the hotel reservation system may reject a request of a customer, because it is simply incorrect or incomplete. The Requester may also express disappointment in the result and decline it. Decline is represented by the termination of Result and the creation of a Declined Result object. For instance, the hotel guest may decline the assigned hotel room, which was assigned by the provide hotel room action. In this case, the basic transaction pattern can be complemented by two dissent patterns. This extended schema is known as the standard pattern [4].

In practice, it is also common that either requester or performer is willing to completely revoke some events. For example, a requester may withdraw his own request, a performer may withdraw his promise, a performer may cancel his own stated result or a requester may cancel his own acceptance. These four cancellation patterns may lead to partial or complete rollback of a transaction. These four options, which are known as cancellation patterns, should be integrated into a universal interaction pattern. A provider may also create new Offer on a basis of created Request, which can transformed into a counter request or it can be accepted by requester. All these possible outcomes are represented in the extended universal pattern, which is shown in Figure 16.

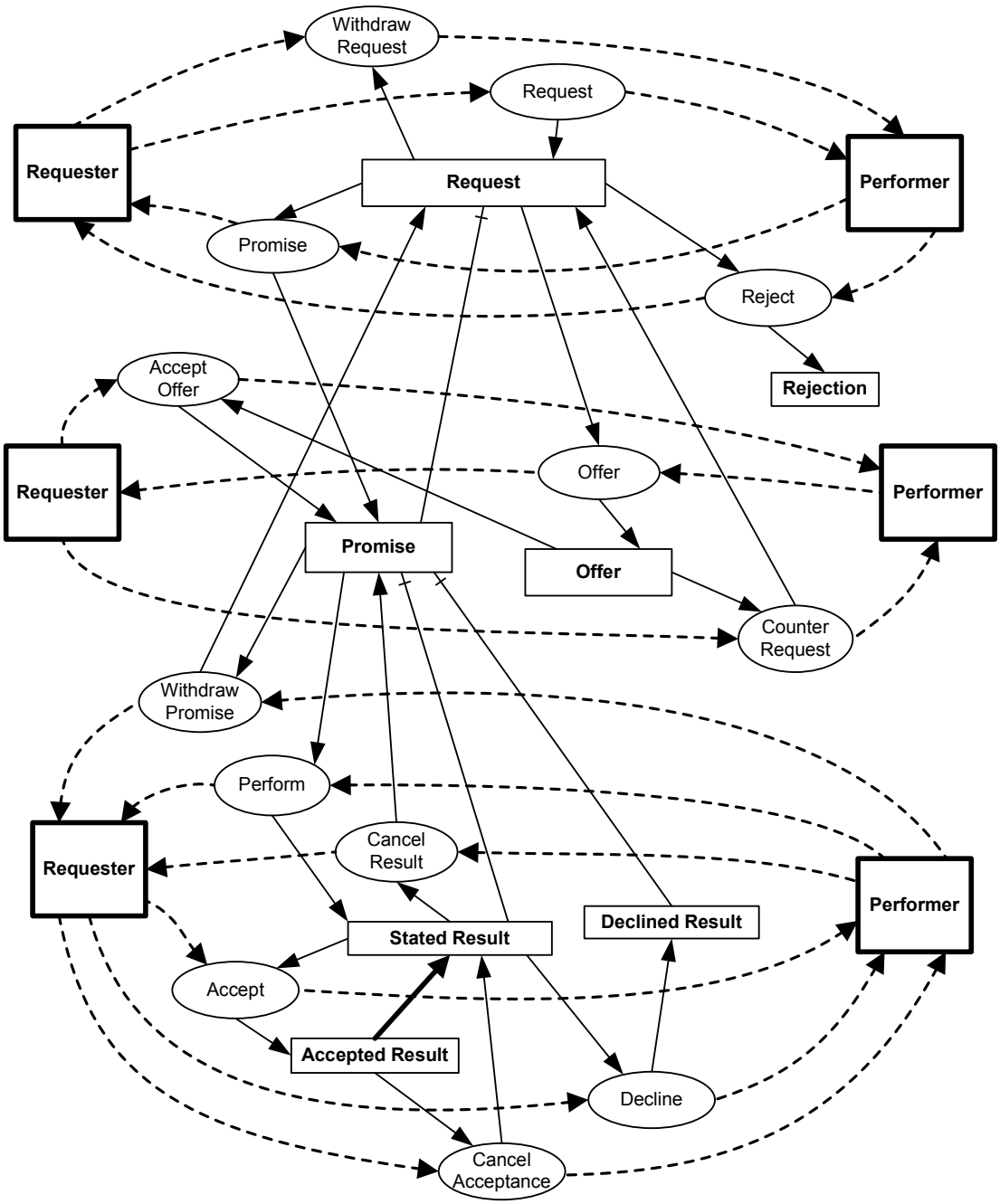

Figure 16. The extended universal interaction pattern

The presented diagram includes the standard transaction pattern and four cancellation patterns, which were analysed by Dietz [4]. It also includes an offer and counter request actions, which are taken from the conversation for action schema [23]. Every cancellation action can be performed if the corresponding fact exists. For instance, the Withdraw Request action can be triggered, if a request object was created by the Request action. Request cancellation event may occur when the customer finds a better or cheaper room in another hotel. A Withdraw Promise action may take place if a Promise for some reason cannot be fulfilled by Performer. For instance, a Hotel Room was damaged as a consequence of some unexpected event. The requester may agree or disagree to accept the consequences of the Withdraw Promise action. Please note that Withdraw Promise action terminates the Promise object and preserves

the Request object. So, the Requester will be forced to cope with four possible alternatives of communication actions such as Promise, Reject, Offer or Withdraw Request. These four alternatives are clearly visible in our new universal interaction pattern.

The third cancellation event is represented by the option Cancel Result. It can be initiated by Performer to avoid the Decline action by requester. The requester typically allows cancelling the result, because after this action the Promise is not terminated. The forth cancellation event may take place when the whole transaction was completed, but the service requester discovers some hidden problem and he regrets acceptance. For instance, the customer may try to Cancel Acceptance of the Hotel Room for the reason that wireless Internet access fails to work properly. The possibility to superimpose four cancellation patters on the standard pattern is not the only advantage of the presented modelling approach. It has sufficient expressive power to cover other special cases, which do not match the universal pattern [4].

V. CONCLUDING REMARKS

The goal of this paper was to demonstrate how the suggested service-oriented constructs can be used for the creation of five different modeling patterns. Traditionally, modeling patterns are constructed taking into account just dynamic aspects of business processes. The advantage of the suggested modeling constructs is that they allow integration of both static and dynamic aspects. One of the main contributions of this paper is the presentation of the extended universal interaction pattern.

The separation of static and dynamic details of the presented patterns creates fundamental difficulties for two major reasons:

1) Since the static aspects must somehow be compensated by using dynamic constructs, the number of patterns becomes bigger than is really necessary. Sometimes, the pattern differences are difficult to understand and they are visually unrecognizable by business experts.
2) If static aspects are not taken into account, then patterns will become more complicated to use them for the purpose of blending enterprise and software engineering.

Interaction dependencies, which define the interplay of coordination or production events, are lying in the foreground of the presented semantically integrated conceptual modeling method. It was demonstrated how interaction dependencies can be analyzed in interplay with the traditional semantic relations in the area of system analysis and design. However, a more systematic comparison with the well-established conceptual modeling languages is necessary. In our future work, we also intend to apply and to validate the method by more realistic trials in industry. The communication for action schema and the extended universal interaction pattern are not fully integrated. So, we need to do more research, which leads to complete integration of these two schemes.

The semantics of service architecture can be defined by using one or more interaction loops. Each interaction loop is composed of creation, termination or reclassification actions. By matching the interaction dependencies from requesters to providers, one can explore opportunities that are available to different actors. The static dependencies define complementary semantic details, which are important for reasoning about service interactions. The examples of corresponding behavior are presented in this paper as well. The novelty of such a way of modeling is that it enables integration of static and dynamic aspects, which are important to maintain a holistic representation of information system specifications. Service-oriented way of modeling is computation-neutral. Diagrams follow the basic conceptualization principle in representing only computationally neutral aspects that are not influenced by any implementation solutions. Since computation-neutral representations are easier to comprehend for business experts as well as system designers, they facilitate understanding and can be used for bridging a communication gap among different types of stakeholders.

REFERENCES

[1] R. Gustas and P. Gustiene, “Three Conceptual Modeling Patterns of Semantically Integrated Method,” *The 4-th International Conference on Business Intelligence and Technology, BUSTECH 2014*, IARIA, pp 19-24.

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

[3] H. Chesbrough and J. Spohrer, "A Research Manifesto for Services Science," *Communications and ACM, 49*(7), 2006, pp. 35-40.

[4] J. Dietz, Enterprise Ontology: Theory and Methodology, Springer, Berlin, 2006.

[5] T. Erl, Service -Oriented Architecture: Concepts, Technology, and Design. New Jersey: Pearson, 2005.

[6] R. Ferrario and N. Guarino, "Towards an Ontological Foundation for Service Science," *Future Internet-FIS2008:The First Internet Symposium, FIS 2008 Vienna, Austria. Revised Selected Papers*, Berlin: Springer, 2008, pp. 152-169.

[7] M. Fowler, *Analysis Patterns: Reusable Object Models*. Menlo Park: Addison-Westley, 1997.

[8] J. Gordijn, E. Yu, and B. van der Raadt, "e-Service Design Using i* and e3 value Modeling," *IEEE Software, 23*(3) 2006, pp. 26-33.

[9] R. Gustas and P. Gustiene, "Conceptual Modeling Method for Separation of Concerns and Integration of Structure and Behavior," *International Journal of Information System Modeling and Design*, vol. 3 (1), New York: IGI Global, 2012, pp. 48-77.

[10] S. Alter, "Service System Fundamentals: Work System, Value Chain, and Life Cycle," *IBM Systems Journal, 47*(1), 2008, pp. 71-85.

[11] P. Gustiené, Development of a New Service-Oriented Modeling Method for Information Systems Analysis and Design, PhD Thesis, Karlstad University Studies, 2010:19, 2010.

[12] S. Hagg and M. Cummings, Managing Information Systems for the Information Age. New York: McGraw-Hill, 2008.

[13] J. A. Hoffer, J. F. George and J.S. Valacich, *Modern Systems Analysis and Design.* New Jersey: Pearson, 2004.

[14] I. Jacobson and P. W. Ng, *Aspect-Oriented Software Development with Use Cases*. New Jersey: Pearson, 2005.

[15] K. Jensen, Coloured Perti Nets. Basic Concepts, Analysis Methods and Practical Use. *Monographs in Theoretical Computer Science, 1, 1997*.

[16] OMG, *Unified Modeling Language Superstructure, version 2.2.* Retrieved March 7, 2014, from www.omg.org/spec/UML/2.2/.

[17] M. P. Papazoglou and W. J. van den Heuvel, "Service-Oriented Design and Development Methodology," *Journal of Web Engineering and Technology, 2*(4), 2006, pp. 412-442.

[18] A. A. Rad, M. Benyoucef and C. E. Kuziemsky, "An Evaluation Framework for Business Process Modelling Languages in Healthcare," *Journal of Theoretical and Applied Electronic Commerce Research, 4*(2), 2009, pp. 1-19.

[19] N. Russell, A. H. M. Hofstede, W. M. P. Aalst and N. Mulyar, "Workflow Control-Flow Patterns: A Revised View," (BPM Centre Report BPR-06-22). Retrieved March 5, 2014 from www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf.

[20] A. Sheth, K. Verma and K. Gomadam, "Semantics to Energize the Full Service Spectrum," *Communications of the ACM, 49*(7), 2006, pp. 55-61.

[21] P. Spohrer, P. Maglio, J. Bailey and D. Gruhl, "Steps Towards a Science of Service Systems," *IEEE Computer 40*(1), 2007, pp. 71-77.

[22] O. Zimmerman, P. Krogdahl and C. Gee, "Elements of Service-Oriented Analysis and Design," Retrieved March 6, 2014 from wsdl2code.googlecode.com/svn/trunk/06-CD/02-Literatur/Zimmermann%20et%20al.%202004.pdf.

[23] T. Winograd and R. Flores, Understanding Computers and Cognition: A New Foundation for Design, Ablex Norwood, NJ, 1986.

[24] M. Blaha and J. Rumbaugh, Object-Oriented Modeling and Design with UML, Pearson Prentice Hall, 2005.