An Abstract Interaction Concept for Designing Interaction Behaviour of Service Compositions[*]

Teduh Dirgahayu[1], Dick Quartel[2] and Marten van Sinderen[3]

[1] Centre for Telematics and Information Technology (CTIT), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{t.dirgahayu, m.j.vansinderen}@utwente.nl

[2] Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
dick.quartel@telin.nl

Abstract. In a service composition, interaction behaviour specifies an information exchange protocol that must be complied with in order to guarantee interoperability between services. In order to control its design complexity, interaction behaviour can be designed using a top-down design approach utilising high abstraction levels. However, current interaction design concepts that merely represent interaction mechanisms supported by communication middleware force designers to design interaction behaviour close to an implementation level. Such design concepts cannot be used for designing interaction behaviour at high abstraction levels. Designers need an interaction design concept that is able to model interactions in an abstract way. In this paper we present such a design concept called *abstract interaction*. We show the suitability of our abstract interaction concept for designing interaction behaviour at high abstraction levels by comparing it to BPMN interaction concept in an example.

1 Introduction

To run its business efficiently, an enterprise makes its business processes interact with each other and, if necessary, with the business processes of its partners. Service-oriented computing facilitates the realisation of such interacting business processes by means of service compositions [2][3][9]. Business processes are exposed as services and then linked to make them interact with each other.

An interaction between services can be simple, e.g. sending a message from one service to another, or complex, e.g. a negotiation for some procurement. A complex interaction is composed of a number of simpler interactions performing certain behaviour. We call this behaviour *interaction behaviour*. Interaction

[*] This work is part of the Freeband A-MUSE project (http://www.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025.

behaviour specifies an information exchange protocol that must be complied with in order to guarantee interoperability between services. Choreography and orchestration [2][14] are common ways to specifying and implementing interaction behaviour.

Several design methods have been proposed for designing interaction behaviour [1][4][5][6][7][8][10][19][20][21][22]. We argue that the interaction design concepts adopted in those design methods force designers to design interaction behaviour close to an implementation level. It is because those design concepts are very much similar to interaction mechanisms supported by communication middleware, e.g. message-passing and request/response interactions. Designing close to an implementation level reveals design complexity at once.

A top-down design approach utilising high abstraction levels can be used to give the designers some control over the complexity of an interaction behaviour design. Using such an approach, designers transform user requirements gradually into designs. In this way, the designers reveal the complexity in a controlled manner. To be able to do that, designers need an interaction design concept that can model interactions in an abstract way.

The objective of this paper is to present an interaction design concept for designing interaction behaviour at high abstraction levels. In order to define the design concept, we identify problems with current interaction design concepts and define a set of requirements for the design concept. To show its suitability, we compare the interaction design concept to BPMN interaction concept in an example.

This paper is structured as follows. Section 2 describes problems with current interaction design concepts. Section 3 presents an interaction design concept and its use in the design of interaction behaviour at high abstraction levels. Section 4 compares the interaction design concept to of BPMN to show their suitability for designing interaction behaviour at high abstraction levels. Finally, section 5 concludes this paper and identifies future work.

2 Problems with Current Interaction Design Concepts

Current methods for designing interaction behaviour adopt interaction design concepts represented in different design languages, such as UML [1][8][10][19][20], BPMN [5][6][21], Petri Nets [4][7], or others, e.g. Let's Dance [22] and ISDL [16].

UML supports two kinds of interactions, namely *sending a signal* and *calling an operation* [13]. They represent message-passing and request/response interactions, respectively.

BPMN represents an interaction as a *message flow* showing that a business process sends a message and another business process receives that message [12]. In Let's Dance, an interaction is made up from two communication actions, namely a *message sending action* and a *message receipt action* [22]. Thus, interactions in BPMN and Let's Dance represent message-passing interactions.

Petri Nets [15] do not have any interaction concept. To model an interaction, the design methods in [4][7] use a pair of Petri Net transitions: one transition represents an activity sending a message and another represents an activity receiving that message. An interaction that is modelled this way represents a message-passing interaction.

The interaction design concept in ISDL [17] can be used to model interactions at a high abstraction level and interaction mechanisms at an implementation level. The design method in [16], however, does not give some hints on how to use ISDL interaction design concept for modelling interactions at high abstraction levels.

As mentioned earlier, an interaction design concept representing interaction mechanism forces designers to design interaction behaviour close to an implementation level. Designing close to an implementation level reveals design complexity at once. Examples of such complexity are as follows.

A complex interaction has to appear as a composition of interaction mechanisms. When an interaction behaviour design involves many complex interactions, such compositions will increase design complexity. Furthermore, when complex interactions are related with each other, presenting complex interactions as their compositions potentially makes the relationships between them unclear, i.e. which interaction mechanisms belong to a complex interaction. Some structuring technique has to be applied to make those relationships clear. The application of such a technique adds more complexity into the design.

The participants of an interaction can be primary or supporting participants. A primary participant is a participant that concerns with the result of the interaction. A supporting participant is a participant that facilitates the interaction between primary participants. For example, the primary participants of a payment are a payer and payee. This payment may include a supporting participant, e.g. a bank that provides money-transfer service. To produce a complete interaction behaviour design at or close to an implementation level, designers have to identify all the participants and take into account their behaviour in the design. Considering the behaviour of the supporting participants might add unnecessary complexity at the early phases of a design process.

Some design methods [6][8][10] introduce multiple abstraction levels. However, when dealing with interaction behaviour, those design methods specify the interactions in terms of interaction mechanisms. To be able to design interaction behaviour at high abstraction levels, designers need an interaction design concept that can model interactions in an abstract way.

3 Interaction Design Concept for High Abstraction Levels

In this section we present an interaction design concept for designing interaction behaviour at high abstraction levels.

3.1 Requirements

We define three requirements for the interaction design concept.

R1. *The interaction design concept should be independent from any interaction mechanisms*. This requirement is to prevent the design concept from forcing designers to design interaction behaviour close to an implementation level.

R2. *The interaction design concept should be able to model a complex interaction abstracting from the interaction's behaviour*. This requirement is to allow designers to include a complex interaction into a design without cluttering the design with the details about the interaction's behaviour. Such details shall be decided and included into designs at lower abstraction levels. As a result, a design at a high abstraction level would be less complex and easy to understand.

R3. *The interaction design concept should be realisable using existing interaction mechanisms*. An interaction behaviour design produced by designers is eventually realised by developers by mapping the design onto existing interaction mechanisms. This requirement is to allow the design concept to expressively model interaction mechanisms. An expressive model avoids misinterpretation between the designers and developers.

3.2 Abstract Interaction Concept

To fulfil requirement R1, we define an *interaction* as an activity which is shared by multiple participants to establish some common results. An interaction represents a negotiation between participants in order to establish the results. An interaction either occurs for all participants or does not occur. If the interaction occurs, all participants can refer to the interaction results. If the interaction does not occur, none of the participants can refer to any (partial or temporal) result of the interaction.

The participation of each participant is represented by an *interaction contribution*, which defines the constraints it has on the interaction results. An interaction can only occur if the constraints of all participants are satisfied. In this case, common results are established. The results are the same for all participants; but possibly a participant may not be interested in the complete results.

To fulfil requirement R2, we define the notion of *abstract interaction* by specialising the definition of interaction. An abstract interaction is an interaction at a higher abstraction level that represents a composition of interactions performing certain interaction behaviour at a lower abstraction level. An abstract interaction is concerned only with (i) the results of the interaction behaviour and (ii) the constraints that should be satisfied by the results. An abstract interaction, hence, may abstract from supporting participants, the results and constraints of the composed interactions, and the relation between the composed interactions.

In a top-down design process, an abstract interaction is meant to be refined into an interaction behaviour design. This design consists of a composition of interactions that are more concrete than the abstract interaction they refine. The design may also introduce supporting participants. An abstract interaction does not impose a certain interaction behaviour design. The interaction behaviour design, however, must conform to or be consistent with the abstract interaction it refines. The interaction behaviour design must establish the results specified by the abstract interaction without violating the constraints of the abstract interaction.

To fulfil requirement R3, we define that an abstract interaction may specify the direction in which its information flows. In an interaction mechanism, such a direction can be identified and gives an indication of the roles of the participants. For example, in a message-passing interaction, the information flows from a sender to a receiver. A participant from which the information originates plays the role of the sender. A participant to which the information sinks plays the role of the receiver. The ability to specify such a direction makes abstract interaction expressive enough to model interaction mechanisms.

3.3 Modelling using Abstract Interaction Concept

To support the design of interaction behaviour using abstract interaction, we borrow behavioural design concepts defined in ISDL [16]. We design the interaction behaviour for the following scenario.

A buyer service interacts with a seller service in the purchase of an article. The buyer wants to buy a notebook for a maximal price of 900 euro and wants the notebook to be delivered to Enschede within seven days. The seller is willing to sell any article listed in its catalogue with a minimal price that depends on the particular article. The seller can deliver a purchased article within two and five days if its target delivery location is in Europe. The purchase interaction consists of three phases: selection, payment, and delivery. In the selection phase, the buyer selects a notebook whose price is not higher than 900 euro from the seller's catalogue. In the payment phase, the buyer pays the seller for the selected notebook. Finally, in the delivery phase, the seller delivers the purchased notebook to the buyer.

At a high abstraction level, we design the purchase interaction between the buyer and seller as a single abstract interaction (see **Fig. 1**). Services are represented as rounded rectangles. An interaction is represented as segmented ellipses linked with a line. A segmented ellipse represents the interaction contribution of a service. Interaction results are represented as *information attributes*. An information attribute has an information type and will be assigned a value when the interaction occurs. Information attributes and constraints are written in boxes attached to their corresponding interaction contributions. If this interaction occurs, it results in the purchase of a notebook that is delivered to Enschede at some price and delivery days that meet the constraints defined by the participants. This design abstracts from how the purchase is performed.

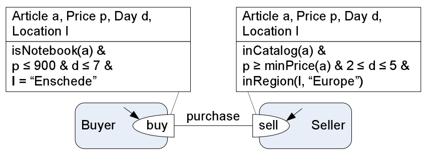
Article a, Price p, Day d, Location l
isNotebook(a) & p ≤ 900 & d ≤ 7 & l = "Enschede"
Article a, Price p, Day d, Location l
inCatalog(a) & p ≥ minPrice(a) & 2 ≤ d ≤ 5 & inRegion(l, "Europe")
Buyer
buy
purchase
sell
Seller

Fig. 1. A *purchase* interaction as an abstract interaction

At a lower abstraction level, we refine the design to show the behaviour of the *purchase* interaction (see **Fig. 2**). The purchase interaction is decomposed into three interactions: *selection*, *payment*, and *delivery* representing the phases within the *purchase* interaction. The interaction behaviour design also defines the relations between those interactions. The relations are represented as the arrows between the interaction contributions.

To be realisable, the interactions in **Fig. 2** have to be further refined into their interaction behaviour because they cannot be straightforwardly mapped onto existing interaction mechanisms. We show their refinement in section 4.2.

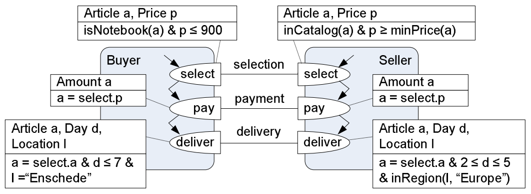
Article a, Price p
isNotebook(a) & p ≤ 900
Article a, Price p
inCatalog(a) & p ≥ minPrice(a)
Buyer
Seller
selection
select
select
Amount a
a = select.p
payment
pay
pay
Amount a
a = select.p
Article a, Day d, Location l
a = select.a & d ≤ 7 & l ="Enschede"
delivery
deliver
deliver
Article a, Day d, Location l
a = select.a & 2 ≤ d ≤ 5 & inRegion(l, "Europe")

Fig. 2. The behaviour of the *purchase* interaction

3.4 Modelling Interaction Mechanisms

Abstract interactions at an implementation level should be realisable. At this level, an abstract interaction should expressively model its target interaction mechanism. We illustrate how to represent a message-passing interaction as an abstract interaction.

Fig. 3 models the behaviour of a message-passing interaction between a sender and receiver. The sender gives a message "Hello" to communication middleware through a *send* interaction. The middleware then passes the message to the receiver through a *receive* interaction.

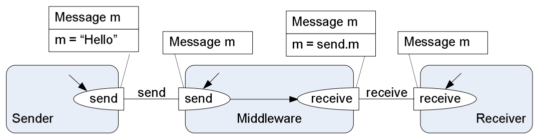

Fig. 3. The behaviour of a message-passing interaction

The middleware plays the role of a supporting participant. An abstract interaction should be able to abstract the interaction behaviour from the middleware participation while maintaining the model expressiveness. For indicating the direction in which the message flows, we use an arrow to link the interaction contributions (see **Fig. 4**).

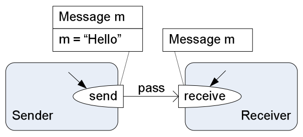

Fig. 4. Message-passing interaction abstracting from the middleware

4 Comparison with BPMN

In this section, we show the suitability of our abstract interaction concept by comparing it to BPMN interaction concept. We choose BPMN because of two reasons. First, BPMN interaction design concept represents a message-passing interaction. Hence BPMN can be considered as a representative of other design languages whose interaction design concepts also represent message-passing interaction. Second, BPMN supports abstraction levels by providing the notation of *abstract processes* and *(collapsed) sub-processes*. Therefore, we can compare the use of the concepts at multiple abstraction levels.

We use the purchase scenario as described in Section 3.3. We add the following user requirements. To facilitate payment, the buyer and seller agree to use a money-transfer service provided by a bank. To facilitate delivery, the seller makes use of a delivery service provided by a courier.

4.1 Designs in BPMN

At a high abstraction level, we model the purchase scenario as interacting abstract processes (see **Fig. 5**). The model shows the message exchange between participants. All participants, i.e. the primary and supporting participants, and all message flows have to appear in the design.

We cannot abstract closely-related message flows into a single message flow because such an abstraction is not supported by the semantics of message flow. Abstracting the design from the supporting participants, i.e. the bank and the courier, will remove the message flows numbered with 5, 6, 8, 9, 11, 12, and 14. This would leave the design incomplete and unclear. Questions may arise, e.g. after receiving an invoice (no. 4), should the buyer pay the invoice before notifying the seller about the payment (no. 7)?

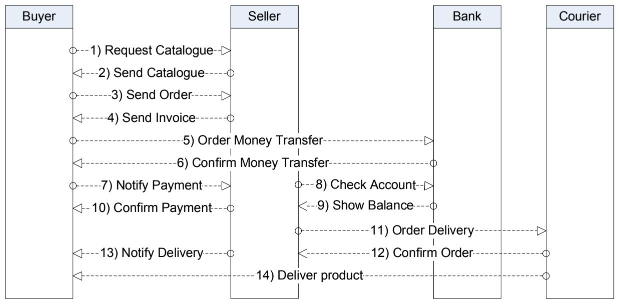

Fig. 5. The purchase scenario as interacting abstract processes

To model the phases in the purchase scenario, we refine the design by adding the phases as collapsed sub-processes within the participants' processes. The collapsed sub-processes are *selection*, *payment* and *delivery* (see **Fig. 6**). Since the interactions are already at implementation level, we do not refine them.

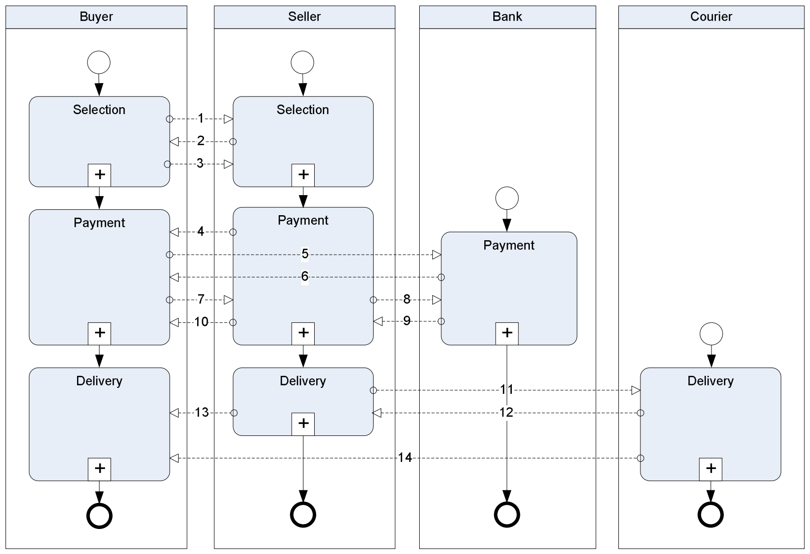

Fig. 6. Phases are represented as collapsed sub-processes

To model the complete private business processes of the participants, we further refine the design by expanding the sub-processes with activities (see **Fig. 7**). We do not refine the interaction.

4.2 Designs using Abstract Interaction Concept

At a high abstraction level, we represent the purchase scenario as a single *purchase* interaction between the primary participants, i.e. the buyer and the seller (see **Fig. 1**). The interaction models the results intended from the scenario. The design abstracts from the supporting participants, i.e. the bank and courier.

To model the phases, we refine the *purchase* interaction into three interactions: *selection*, *payment*, and *delivery* (see **Fig. 2**). Information attributes and constraints are refined and distributed over the interactions.

To include the participation of the bank and the courier, we further refine the design by introducing the bank in the *payment* interaction and the courier in the *delivery* interaction (see **Fig. 8**). For brevity, we omit the information attributes and constraints.

We further refine the design to model the behaviour of each interaction (see **Fig. 9**). The refinement results in the choreography between the participants. We structure the behaviour of the buyer and the seller to indicate the phases. Refinement should be further done until all the interactions become realisable.

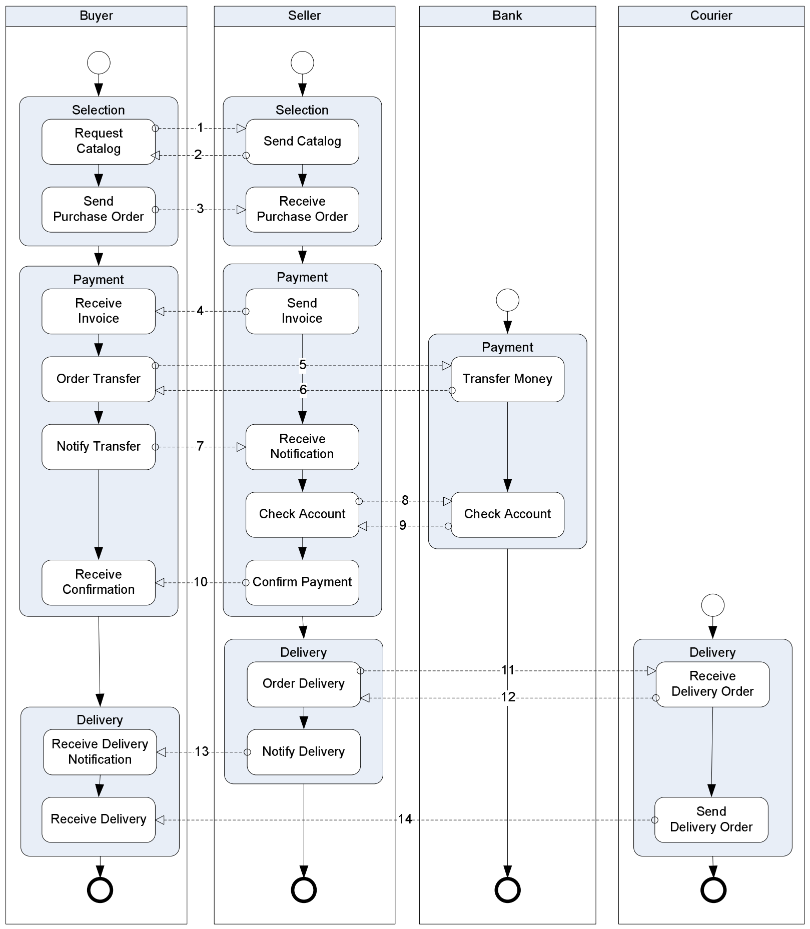

Fig. 7. The purchase scenario in BPMN

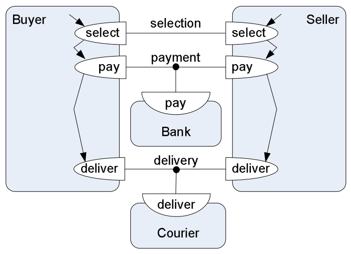

Fig. 8. The participation of the *bank* and the *courier*

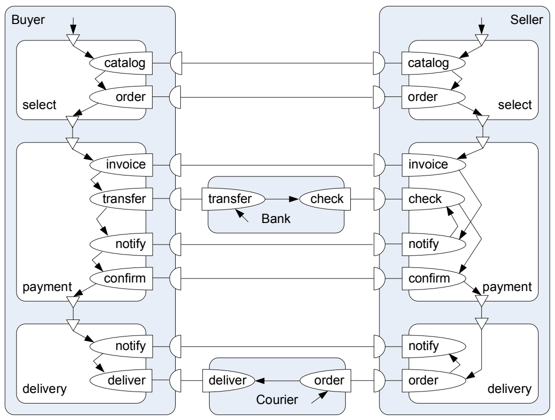

Fig. 9. The behaviour of the *purchase* interaction

4.3 Discussion

Ultimately an interaction is performed to establish some results. The results are more essential than the way they are established. Therefore, we define that an interaction design concept is suitable for designs at high abstraction levels if it can represent an interaction and its results abstracting from the way the results are established.

Abstraction levels in BPMN can only be applied within the behaviour of individual business processes participating in interaction behaviour. BPMN cannot raise the interaction behaviour to a higher abstraction level. BPMN cannot model interactions and its results without specifying how the results should be established. We conclude that BPMN interaction design concept is not suitable for designing interaction behaviour at high abstraction levels.

Our abstract interaction concept is defined with an intention to model interaction behaviour designs at high abstraction levels. As evidence, we have shown its suitability in the design of the example. We start the design by modelling the results that are expected from the scenario. Identification and inclusion of the supporting participants and detailed interactions are deferred until they matter to the design. For instance, at a lower abstraction level, we want to show the phases in the scenario. We model the phases as interactions (see **Fig. 2**) without defining yet the interaction behaviour of the phases. We claim that the abstract interaction concept is suitable for designing interaction behaviour at high abstraction levels.

5 Conclusions

We have presented an interaction design concept called *abstract interaction* for designing interaction behaviour of service compositions at high abstraction levels. An abstract interaction is able to represent interaction behaviour as a single interaction at a high abstraction level. An abstract interaction is concerned with the results of the interaction behaviour and the constraints which should be satisfied by the results, abstracting from the behaviour itself. We have shown the suitability of the abstract interaction concept in the design of interaction behaviour at high abstraction levels.

An abstract interaction may have several possible refinements at a lower abstraction level and, hence, multiple realisations. Thus, the abstract interaction concept can be used to extend the approach defined in the Model-Driven Architecture (MDA) [11]. An abstract interaction may have not only multiple realisations at different technology platforms, but also multiple realisations with different interaction behaviour. For example, the design in **Fig. 2** can be refined into different interaction behaviour in which the payment interaction is done using credit card.

The abstract interaction concept supports as many abstraction levels as needed by designers. In some cases, designers prefer to have a limited set of abstraction levels; each of which has a pre-defined purpose. Design methods defining such a limited set of abstraction levels can be developed as guidelines in designing interaction behaviour using the interaction concept. For example, a framework in [18] defines three generic abstraction levels. At a high abstraction level, a service is modelled as a single interaction between a service user and provider. At a lower abstraction level, this interaction is refined into choreography of multiple interactions. At another lower abstraction level, the service provider may be refined into an orchestration of other service compositions. The abstract interaction concept fits within this limited set of abstraction levels.

The abstract interaction concept is applicable at different abstraction levels. This capability allows designers to apply the same refinement patterns and conformance assessment method consistently. The interaction concept does not require designers to master different design concepts and tools for different abstraction levels.

In future work, we will identify patterns of interaction refinement. Such patterns may serve as guidelines for designers in refining an abstract interaction into an interaction behaviour design. We will also develop rules to support conformance assessment for those patterns. The patterns and rules should include time attributes of an interaction. Time attributes are useful for specifying the time moment and the duration an interaction may occur.

6 References

[1] Baresi L, Heckel R, Thöne S, Varró D, (2003) Modeling and validation of service-oriented architectures: application vs. style. Proc. 9th European Software Engineering Conf.: 68-77

[2] Benatallah B, Dijkman RM., Dumas M, Maamar Z, (2005) Service Composition: Concepts, Techniques, Tools and Trends. Service-Oriented Software Engineering: Challenges and Practices. Idea Group, Inc.: 48-66

[3] Curbera F, Khalaf R, Mukhi N, Tai S, Weerawarana S, (2003) The Next Step in Web Services. Communications of the ACM 46(10): 24-28

[4] Dijkman R, Dumas M, (2004) Service-Oriented Design: A Multi-Viewpoint Approach. International Journal of Cooperative Information Systems 13(4): 337-368

[5] Dijkman RM, (2006) Choreography-Based Design of Business Collaborations. BETA Working Paper WP-188, Eindhoven University of Technology
[6] Emig C, Weisser J, Abeck S, (2006) Development of SOA-Based Software Systems - an Evolutionary Programming Approach. Proc. Advanced Intl. Conf. on Telecommunications and Intl. Conf. on Internet and Web Applications and Services: 182-187
[7] Hamadi R, Benatallah B, (2003) A Petri Net-Based Model for Web Service Composition. Proc. 14th Australasian Database Conf.: 191-200
[8] Kramler G, Kapsammer E, Retschitzegger W, Kappel G, (2006) Towards Using UML 2 for Modelling Web Service Collaboration Protocols. Interoperability of Enterprise Software and Applications, Springer: 227-238
[9] Leymann F, Roller D, Schmidt M-T, (2002) Web Services and Business Process Management. IBM Systems Journal 41(2): 198-211
[10] Millard DE, Howard Y, Jam E-R, Chennupati S, Davis HC, Gilbert L, Wills GB, (2006) FREMA Method for describing Web Services in a Service-Oriented Architecture. Technical Report ECSTR-IAM06-002, University of Southampton
[11] OMG, (2001) Model Driven Architecture (MDA). ormsoc/2001-07-01
[12] OMG, (2006) Business Process Modeling Notation (BPMN) Specification. dtc/06-02-01
[13] OMG, (2007) Unified Modeling Language: Superstructure version 2.1.1. formal/ 2007-02-03
[14] Peltz C, (2003) Web Services Orchestration and Choreography. IEEE Computer 36(8): 46-52
[15] Peterson JL, (1981) Petri Net Theory and the Modeling of Systems. Prentice-Hall
[16] Quartel D, Dijkman R, van Sinderen M, (2004) Methodological support for service-oriented design with ISDL. Proc. 2nd Intl. Conf. on Service Oriented Computing: 1-10
[17] Quartel D, Ferreira Pires L, van Sinderen M, (2002) On Architectural Support for Behaviour Refinement in Distributed Systems Design. Journal of Integrated Design and Process Science 6(1): 1-30
[18] Quartel DAC, Steen MWA, Pokraev S, van Sinderen MJ, (2007) COSMO: A Conceptual Framework for Service Modelling and Refinement. Information Systems Frontiers 9: 225-244
[19] Skogan D, Grønmo R, Solheim I, (2004) Web service composition in UML. Proc. 8th IEEE Intl. Enterprise Distributed Object Computing Conf.: 47-57
[20] Thöne S, Depke R, Engels G, (2003) Process-Oriented, Flexible Composition of Web Services with UML. LNCS 2784: 390-401
[21] White SA, (2005) Using BPMN to Model a BPEL Process. BPTrends 3(3): 1-18
[22] Zaha JM, Dumas M, ter Hofstede A, Barros A, Decker G, (2006) Service Interaction Modeling: Bridging Global and Local View. Proc. 10th IEEE Intl. EDOC Conf.: 45-55