Designing Interaction Behaviour in Service-Oriented Enterprise Application Integration

Teduh Dirgahayu
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
+31 53 4894226
t.dirgahayu@ewi.utwente.nl

Dick Quartel
Telematica Instituut
P.O. Box 589, 7500 AN Enschede
The Netherlands
+31 53 4850451
dick.quartel@telin.nl

Marten van Sinderen
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
+31 53 4893677
m.j.vansinderen@utwente.nl

ABSTRACT

In this paper we present an approach for designing interaction behaviour in service-oriented enterprise application integration. The approach enables business analysts to actively participate in the design of an integration solution. In this way, we expect that the solution meets its integration goal and business requirements. The approach consists of four steps: (i) represent the existing services to be integrated in platform-independent models; (ii) derive the models of the goals and business requirements of the services; (iii) check whether an abstract interaction representing the integration goal may occur between the services; and (iv) if so, (recursively) refine the interaction into a realisable design. The approach is characterised by an early check on the possibility of an integration solution, clear expressions of the integration goal and business requirements, and explicit use of the descriptions of the existing services as bottom-up knowledge during refinement. To support the approach, we present a set of patterns of interaction refinement as guidelines in refining abstract interactions.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *languages, methodologies.* H.1.1 [**Models and Principles**]: Systems and Information Theory – *general system theory.*

General Terms

Design, Languages.

Keywords

Interaction behaviour, interaction design, service-oriented computing, enterprise application integration.

1. INTRODUCTION

Service-oriented computing emerges as a promising paradigm to support enterprise application integration (EAI) [4][8]. In this paradigm, applications are represented as software services that expose their external behaviour without revealing their internal functions and structures. An integration solution is then specified in terms of interactions between such services.

An interaction can be simple, e.g. sending a request for a product catalogue from a retailer to a supplier, or complex, e.g. a negotiation for a product's price through an auction. A complex interaction is composed of a number of simpler interactions performing certain behaviour. We call such behaviour *interaction behaviour*. When designing the behaviour of an integration solution, designers have to take into account (i) the behaviour of the existing services and (ii) the interaction behaviour between those services. In many cases, the behaviour of the existing services should be kept unmodified. The design of the behaviour of an integration solution is hence the design of interaction behaviour between those services.

In designing an integration solution, we should have an integration goal and the descriptions of the existing services to be integrated. The goal refers to the effect that is intended to be established by the integration [7]. Related to the goal are the business requirements of the services, which indicate what the services require to achieve the goal. The goal and business requirements are typically defined in a platform-independent manner at a high abstraction level by business analysts. On the other hand, the fact that service descriptions are technology-specific, e.g. in IDL or WSDL, usually leads to the definition of the integration solution at an implementation level by application developers. Different domains, i.e. business and technology, and the gap between a high abstraction level and an implementation level may result in an integration solution that does not meet the integration goal and business requirements [9].

The objective of this paper is to present an approach for designing interaction behaviour in service-oriented EAI. The purpose of this approach is to enable business analysts to actively participate in the design of an integration solution. With such participation, we expect that the integration solution meets its goal and business requirements. To handle the gap between a high abstraction level and an implementation level, the approach uses a design method utilising multiple abstraction levels.

This paper is further structured as follows. Section 2 presents our integration approach. Section 3 identifies patterns of interaction refinement that can be useful in designing interaction behaviour at multiple abstraction levels. Section 4 illustrates the application of the approach with an example. Section 5 discusses related work. Finally, Section 6 concludes this paper and identifies future work.

2. APPROACH

In this section we propose an approach for designing interaction behaviour in service-oriented EAI. We first describe design concepts and a design method that are used by the approach.

2.1 Interaction Design Method

We define an *interaction* as a shared activity which involves multiple participants to establish some common results or desired effects. 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, a common result is established (same for all, but possibly a participant may not be interested in the complete 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.

To support multiple abstraction levels, we define the notion of *abstract interaction* to represent a composition of interactions as a single interaction at a higher abstraction level. An abstract interaction concerns with (i) the results of the composition and (ii) the constraints which should be satisfied by the results. In this way, an abstract interaction represents the goal of the composition abstracting from the way this goal is achieved.

In a top-down design process, an abstract interaction is meant to be refined into a composition of (less) abstract interactions at a lower abstraction level or to be mapped onto interaction mechanisms supported by communication middleware. An abstract interaction does not impose a certain interaction behaviour design or certain interaction mechanisms. An interaction behaviour design, however, should conform to the abstract interaction it refines. The interaction behaviour design should establish the results specified by the abstract interaction without violating the constraints.

To support modelling with abstract interactions, we use behavioural and information design concepts defined in the COSMO framework [13], e.g. behaviour, activity, causality condition, class, and constraint. Interaction results are represented as *information attributes*. An information attribute has an information type and will be assigned a value when the interaction occurs.

Figure 1 graphically represents an abstract interaction named *purchase* between a *buyer* and a *seller* service. 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. Information attributes and constraints are written in boxes attached to their corresponding interaction contributions. In the figure, the *buyer* wants to buy a notebook for a maximal price of 900 euro. The *seller* wants to sell any article listed in its catalogue with a minimal price that depends on the particular article. If the interaction occurs, it results in the purchase of a notebook at some price that meets both constraints.

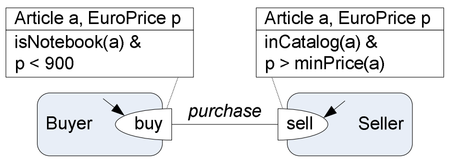

Figure 1. A *purchase* interaction between *buyer* and *seller*

The purchase of a notebook is the goal of this interaction. To achieve this goal, the participants specify their business requirements as interaction constraints. The *buyer* requires that the *seller* has a notebook whose price is less than 900 euro. The *seller* requires that the *buyer* selects an article from its catalogue and agrees for a purchase price that is higher than the article's minimum price. Expressing the goal aspect of an interaction as constraints on the interaction results has been discussed in [13].

In an interaction involving two participants, each interaction contribution should specify the same set of information attributes. In an interaction involving more than two participants, a participant may be interested only in some part of the results. Thus each interaction contribution does not necessarily specify the same set of information attributes. In this case, an information attribute should be specified in at least two interaction contributions. An interaction is possible or may occur if the intersection of the constraints of each information attribute produces a non-empty set. The *purchase* interaction in Figure 1 may occur if the *seller* has a notebook in its catalogue with a minimum price less than 900 euro.

We explain the design method with an example. Suppose that we have a *purchase* interaction as in Figure 1. Since the interaction cannot be realised with any available interaction mechanism, we apply the design method to refine the interaction into a set of (less) abstract interactions at a lower abstraction level, i.e. *selection*, *payment*, and *delivery*, which are to be performed in sequence. Figure 2 shows the refinement result. The *buyer* and the *seller* are refined into *buyer'* and *seller'* respectively, because it is the responsibility of the services to determine the order of those interactions. Information attributes and constraints may also need to be refined. If an interaction at this abstraction level still cannot be mapped onto available interaction mechanisms, the design method is (recursively) applied to that interaction. Different patterns of interaction refinement are possible. We list them in Section 3.

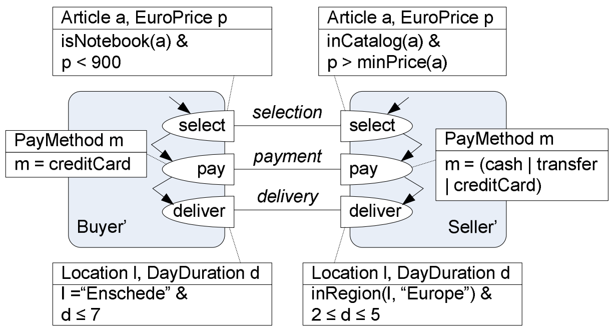

Figure 2. Refinement of the *purchase* interaction (Pattern 1)

2.2 Integration Approach

Our integration approach assumes that the existing services to be integrated have been identified. The approach consists of the following steps.

1. *Represent the services in platform-independent models.* The models are useful for understanding the behaviour of the services, including the way to interact with the services. The models are also useful for identifying (i) information required and produced by the services and (ii) constraints that must be satisfied to interact with the services. Platform-independent models allow business analysts to participate in the design of an integration solution.
2. *Derive the models of the goals and business requirements of the services.* This is done by abstracting the platform-independent models from the way to interact with the services. This step results in models at a higher abstraction level. Each model has only one interaction contribution that clearly represents the goal and requirements of each service. This step includes abstraction of information attributes and constraints. We refer to [13] for information and constraints abstraction. The models allow business analysts to focus on the information attributes and constraints specified by each service.
3. *Check whether an interaction between the services is possible.* First, the models are linked to form an abstract interaction representing the integration goal. Then, the check is applied on the information attributes and constraints of the interaction. If the interaction is not possible, we conclude that it is impossible to integrate the services. The integration solution should be redesigned from the start by identifying existing services that should involve in the integration. Alternatively, the existing services should be modified to make the interaction possible.
4. *If the interaction is possible, (recursively) refine the abstract interaction.* The business analysts should bear in mind that the purpose of the refinement is to eventually enable interactions between models obtained in Step 1. These models of service descriptions constrain the refinement. Patterns of interaction refinement (presented in Section 3) can be used as guidelines in refining an interaction. When the design resulted from the refinement is detailed enough to be realised, application developers take over and realise the design in some implementation platform.

The integration approach offers three main benefits. First, at a high abstraction level, business analysts can early check whether an integration solution may exist (Step 3). Doing so avoids trial-and-error attempts at an implementation level that may end up with a conclusion saying that an integration solution is impossible. Hence, the approach potentially saves time and efforts.

Second, the approach clearly expresses the integration goal and business requirements (Step 2 and 3). Clear expressions of the integration goal and business requirements allow an assessment on whether an integration solution meets its goal and business requirements.

Third, the approach explicitly uses the descriptions of the existing services as bottom-up knowledge during refinement of abstract interactions (Step 4). Such bottom-up knowledge constrains the refinement to ensure that the integration solution can be mapped onto existing services. Without such knowledge, refinement may result in a design of an integration solution that cannot be mapped onto existing services or that requires modification of the existing services.

3. REFINEMENT PATTERNS

In this section we identify some basic patterns of interaction refinement. Each pattern is presented with an example. The information attributes and constraints of an abstract interaction may need to be refined and distributed over a set of (less) abstract interactions. Due to space limitation, we do not show information attributes and constraints in figures. In refinement, multiple patterns can be applied at the same time.

In order to assess conformance, we adopt a method for assessing conformance of designs that use the concepts of the COSMO framework. The method is based on calculating the abstraction of a design (i.e., abstracting from the design details that have been added) and comparing this abstraction to the original design. For this method, abstraction rules have been defined [11][12][13]

We assume that the occurrence of an abstract interaction corresponds to the occurrence of a number of (less) abstract interactions at a lower abstraction level. A (less) abstract interaction that corresponds to the abstract interaction is called a *reference activity*. The results specified by the abstract interaction can be referred only after the occurrence of the reference activity. A (less) abstract interaction that is not a reference activity is called an *inserted activity* [13]. For each pattern, we indicate which interactions are reference and inserted activities. Detailed conformance assessment is beyond the scope of this paper.

Pattern 1: Decomposition into a set of interactions

An abstract interaction between two services is refined by decomposing it into a set of related (less) abstract interactions. Figure 2 shows a model resulted from the application of this pattern on the abstract interaction in Figure 1. Different relationships between the (less) abstract interactions are possible, e.g. choice and concurrency. This pattern can be found in [2][13].

The *delivery* interaction in Figure 2 is the reference activity corresponding to the abstract *purchase* interaction in Figure 1. The *select* and *payment* interactions are inserted activities.

Pattern 2: Introduction of peer services

An abstract interaction is refined by introducing a number of peer services into the interaction. It results in a (less) abstract interaction whose participants are the original services and the newly-introduced peer services. A participant is not necessarily interested (and therefore does not participate) in all information attribute.

Figure 3 illustrates this pattern. A *purchase* interaction is basically done between a *buyer* and a *seller*. In purchasing expensive and high-risk products e.g. shares of a company, the *buyer* may want to introduce its financial *advisor* in the interaction. As a result, the refined *purchase'* interaction has three participants: *buyer'*, *seller'*, and *advisor*. They are all involved in the same interaction, but possibly they are not involved in all information attributes. Since the abstract *purchase* interaction is refined into a single interaction, the *purchase'* interaction is the reference activity.

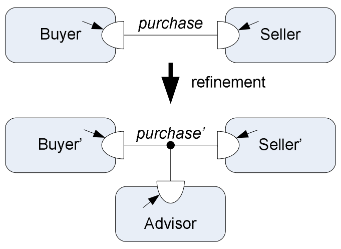

Figure 3. Introduction of peer services (Pattern 2)

Pattern 3: Introduction of an intermediary service

An abstract interaction is refined by introducing an intermediary service that defines the behaviour of the abstract interaction. Each original service then interacts only with the intermediary service. The intermediary service defines the relationships between those interactions.

In contrast to Pattern 2, this pattern eliminates direct interactions between original services. This pattern is useful for refining an abstract interaction involving more than two participants into a realisable design because most available interaction mechanisms support two participants only. This pattern can be found in [1].

Figure 4 illustrates this pattern. A *payment* interaction between a *buyer* and a *seller* is refined by introducing a *bank* that provides a money transfer service. The *buyer* and *seller* are refined into *buyer'* and *seller'* respectively. To transfer a sum of money as payment, the *buyer'* does a *send* interaction with the *bank*. Then the *bank* does a *notification* interaction with the *seller'* to notify that a sum of money has been added to the *seller*'s account. There is no direct interaction between *buyer'* and *seller'*. The *notification* interaction is the reference activity. The *send* interaction is an inserted activity.

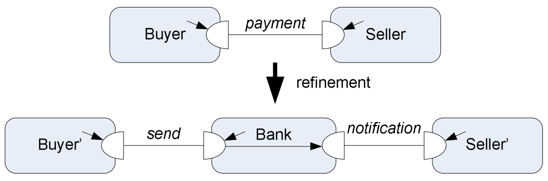

Figure 4. Introduction of an intermediary service (Pattern 3)

Pattern 4: Distribution over pairs of services

An abstract interaction involving more than two participants is refined into a set of (less) abstract interactions that are distributed over pairs of participants. As Pattern 3, this pattern is useful for refining an abstract interaction into a realisable design. This pattern can be found in [2].

Figure 5 illustrates this pattern. A *delivery* interaction has three participants: *seller*, *buyer*, and *courier*. The interaction is refined into a number of (less) abstract interactions, i.e. the *delivery* notification interaction between *seller'* and *buyer'*, the *product handing* interaction between *seller'* and *courier'*, and the *product delivery* interaction between *courier'* and *buyer'*. The *product delivery* interaction is the reference activity. The *delivery notification* and *product handing* interactions are inserted activities.

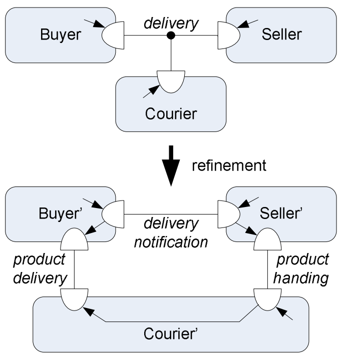

Figure 5. Distribution over pairs of services (Pattern 4)

4. EXAMPLE

To illustrate the application of our integration approach, we apply the approach on an integration case based on an integration scenario presented in the SWS challenge[1]. In this case, we integrate the ordering application of a customer called *Blue* with an order management (OM) application of a manufacturer called *Moon*. The behaviour of these services is given by the scenario.

Step 1. We model in a platform-independent manner the applications as services. Figure 6 shows the model of *Blue*'s ordering application as a service. The service uses two interaction contributions, namely *sendPO* and *receivePOC*, to interact with its business partner. They represent the sending of a purchase order (PO) and the receipt of the purchase order confirmation (POC), respectively. A PO consists of a customer's name (*cust*) and a list of items to be ordered (*items*). A POC consists of the PO (*cust* and *items*) it corresponds to and the order status (*status*).

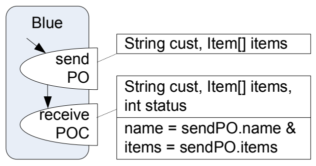
Blue
send PO
receive POC
String cust, Item[] items
String cust, Item[] items, int status
name = sendPO.name & items = sendPO.items

Figure 6. *Blue*'s ordering system

Figure 7 shows the model of *Moon*'s OM application as a service. To place an order, first an order must be created (through *create order* interaction contribution), then the items to be ordered are added one-by-one (*add item*), and finally the order are closed (*close order*). The service then sends back a confirmation for each ordered item (*confirm item*). The service uses a customer ID (*cID*) to create an order and each order is given an order ID (*oID*).

Step 2. We derive the goals and business requirements of the services by abstracting the obtained models. In *Blue*'s service, the information attribute includes the customer's name, the list of items to be ordered, and the summary of the order status. In *Moon*'s OM service, the information attributes includes the customer ID, a list of items to be ordered, and a list of status. The information attributes abstract from the order ID because order ID is meaningful for *Moon*'s OM service only. It is generated and consumed by *Moon*'s OM service.

[1] http://sws-challenge.org/

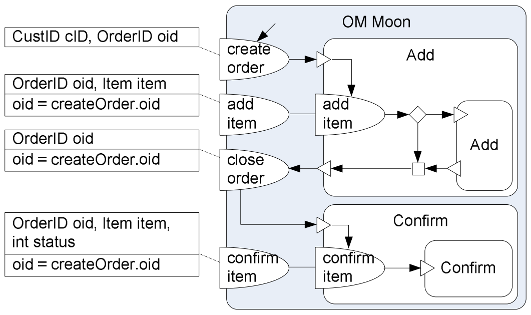

Figure 7. *Moon*'s order management system

Step 3. We link the abstract services to form an abstract interaction as shown in Figure 8. We then check whether the interaction is possible. After checking, we conclude that the interaction is not possible because *Moon*'s OM service requires information about customer ID, while *Blue*'s system does not supply it. No integration solution may exist.

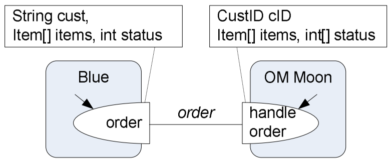

Figure 8. Linking abstract models

We do not want to modify the existing services. Further identification of existing services that are required indicates that *Moon*'s customer relationship management (CRM) application should involve in the integration solution. *Moon*'s CRM service returns the customer ID of a given customer name. Figure 9 shows a new abstract interaction for the integration solution. After checking the information attributes and constraints of the interaction, we conclude that the interaction is now possible.

Step 4. We refine the abstract interaction. We have two options: to apply Pattern 3 or Pattern 4. Considering the models obtained from Step 1, we apply Pattern 3 and produce a design at a lower abstraction level as shown in Figure 10. Information attributes and constraints are omitted for brevity. We introduce an intermediary service called *Mediator* between the original services. *Mediator* service is responsible for receiving a PO sent by *Blue*'s service, searching in *Moon*'s CRM service for the customer ID of the customer name indicated in the PO, placing an order in *Moon*'s OM service, and then sending back a POC to *Blue*'s service. Observe that we apply also Pattern 1 to the interaction between *Blue* and *Mediator*.

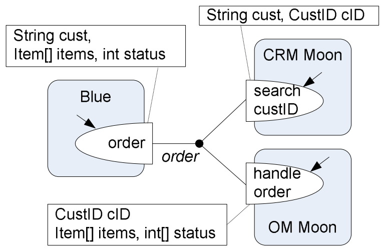

Figure 9. *Moon*'s CRM service is now included

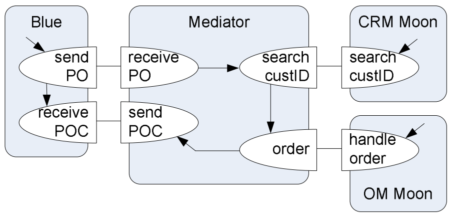

Figure 10. An introduction of *Mediator* as an intermediary

Further refinement (using Pattern 1) on the interaction between *Mediator* and *Moon*'s OM service results in a design shown in Figure 11. The refinement is intended to enable *Mediator* service to interact with concrete *Moon*'s OM service shown in Figure 7. For this, the *Mediator*'s behaviour should match with the interaction contributions defined in *Moon*'s OM service.

5. RELATED WORK

The existence of an integration solution depends on many interoperability issues. Issues that are related to this paper are *data mismatches* and *behaviour mismatches* [9]. Several approaches have been proposed to solve these issues without modifying existing services, e.g. approaches listed in [6][10][14]. However, the absence of information cannot be solved without modifying existing services to provide the required information or to relax interaction constraints. In our approach, the absence of information is represented by the intersection of the constraints of the interaction's information attributes producing an empty set.

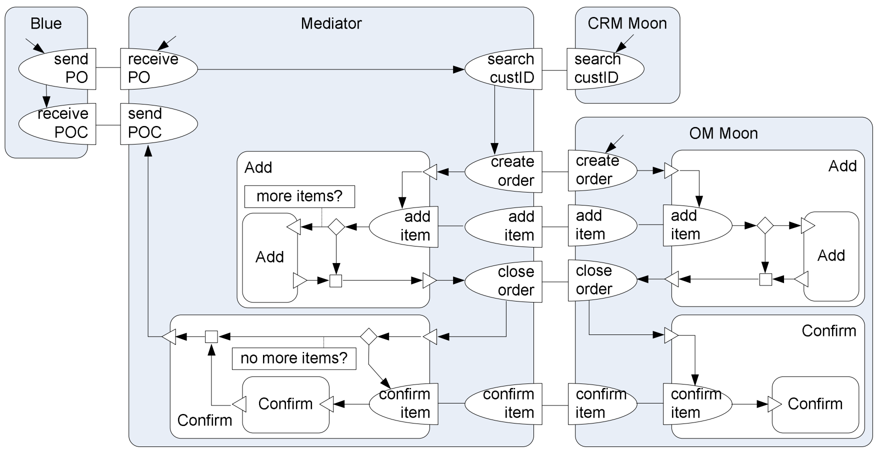

Figure 11. The integration solution

Therefore, we consider that our approach's check on the existence of an integration solution is fundamental. Approaches for solving any data mismatches or behaviour mismatches can be applied only if the check indicates that an integration solution may exist.

Another approach to service-oriented EAI is presented in [9]. The approach defines two abstraction levels, namely business services and information technology (IT) services; and consists of three steps: (i) lifting IT service descriptions to business service descriptions, (ii) solving the integration problem at business services level, and (iii) deriving an IT integration solution from the business integration solution. Our integration approach extends this approach in three ways. First, our approach allows business analysts to check whether services can be integrated before designing an integration solution. Second, our approach captures goals and business requirements in the designs. Third, our approach allows business analysts to define as many abstraction levels as necessary.

An approach for designing an integration solution at multiple abstraction levels can also be found in [5]. At the very first step, the approach considers an integration solution as a broker or intermediary service. Hence, the approach limits itself to modelling integration solutions that are based on a 'hub-and-spoke' architecture [4]. Our integration approach can be used to design an integration solution that is based on a 'hub-and-spoke' (Pattern 3) or a 'point-to-point' architecture (Pattern 4).

EAI can be seen as a way to enable interorganisational workflows [17]. Approaches presented in [3][18] first define a common workflow to be shared by several business partners and then map pieces of the workflow onto those partners. The mapping produces a set of interfaces; each of which should be implemented by the corresponding partner. In service-oriented computing, these interfaces define the partners' service descriptions. These approaches (and also the approach in [5] that we discussed earlier) are purely top-down approaches that do not consider functionality that might already be available at the business partners. Our approach combines a top-down approach with the use of bottom-up knowledge in order to ensure that an integration solution can be mapped onto existing services.

6. CONCLUSION

We have presented an approach for designing interaction behaviour in service-oriented EAI. The purpose of this approach is to enable active participation of business analysts in the design of an integration solution. We expect that active participation of business analysts leads to integration solutions that better meet integration goals and business requirements. The approach uses an interaction design method that supports multiple abstraction levels. The approach can be characterised by its benefits, i.e. (i) an early check whether an integration solution may exist, (ii) clear expressions of the integration goal and business requirements, and (iii) explicit use of the descriptions of the existing services as bottom-up knowledge in refinement. To support the approach, we identify a set of patterns of interaction refinement.

As in [15][16], our approach basically considers interactions as first-class entities. It would be easier to handle a complex interaction in an abstract way and refine the interaction later when some details become essential for its design. By considering an interaction as a first-class entity, an interaction can be a starting point for refinement. In this way, we expect that the refinement results in matched sets of interaction contributions in the participants. If interactions are not considered as first-class entities, interaction refinement is done only as a consequence of the refinement of its interaction contributions. Such refinement potentially results in a set of interaction contributions in one participant that do not match with a set of interaction contributions in another participant. Furthermore, it offers only two patterns of interaction refinement, i.e. Patterns 1 and 4.

The integration approach combines a top-down design approach with bottom-up knowledge. A top-down design approach gradually transforms the integration goal and business requirements into designs that are detailed enough to be realised. In this way, we expect that the integration solution meets its goal and business requirements. Bottom-up knowledge constrains the refinement of an abstract interaction. By considering such knowledge during a design process, we expect that the integration solution can be mapped onto existing services.

In this paper, we have identified some patterns of interaction refinement. Our future work will be the development of specific conformance assessment rules for the patterns. In these rules, we will also include time attributes of an interaction. Constraints on time attributes determine when and how long an interaction may occur. Furthermore, we will apply our approach to more cases in order to evaluate its usability.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] Almeida, J.P., Dijkman R., Ferreira Pires, L., Quartel, D., van Sinderen, M. Model-Driven Design, Refinement and Transformation of Abstract Interaction. *International Journal of Cooperative Information Systems, 15, 4* (2006), 599-632.

[2] de Farias, C.R.G. *Architectural Design of Groupware Systems: a Component-Based Approach*. PhD. Thesis. University of Twente, Enschede, 2002.

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

[4] Erasala, N., Yen, D.C., and Rajkumar, T.M. Enterprise Application Integration in the electronic commerce world. *Computer Standards and Interface, 25* (2002), 69-82.

[5] Johannesson, P. and Perjons, E. Design principles for process modelling in enterprise application integration. *Information Systems, 26* (2001), 165-184.

[6] Klusch, M., Sycara, K. Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Omicini, A. et al. (eds.). *Coordination of Internet Agent*. Springer, 2001, 197-224.

[7] Lamsweerde, A. Goal-Oriented Requirement Engineering: A Guided Tour. In *Proc. of the 5th IEEE Intl. Symp. on Requirement Engineering (RE'01)*, (Toronto, Canada, Aug.

27-31, 2001). IEEE Computer Society, Los Alamitos, CA, 2001, 249-263.

[8] Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H., and Elmagarmid, A.K. Business-to-business interactions: issues and enabling technologies. *VLDB Journal, 12* (2003), 59-85.

[9] Pokraev, S., Quartel, D.A.C., Steen, M.W.A., Wombacher, A., and Reichert, M. Business Level Service-Oriented Enterprise Application Integration. In *Proc. of the 3rd Intl. Conf. on Interoperability for Enterprise Software and Applications (I-ESA 2007)* (Funchal, Portugal, Mar. 28-30, 2007). Springer Verlag, Berlin, 2007, 507-518.

[10] Pokraev, S., Reichert, M. Mediation Patterns for Message Exchange Protocols. In *Proc. of CAiSE'06 Workshops/Open INTEROP Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP)* (Luxembourg, June 5-9, 2006). Presses Universitaires de Namur, 2006, 659-663.

[11] Quartel, D., Ferreira Pires, L., van Sinderen, M. On Architectural Support for Behaviour Refinement in Distributed Systems Design. *Transaction of the SPDS, 6, 1* (2002), 1-30.

[12] Quartel, D.A.C., Ferreira Pires, L., van Sinderen, M.J., Franken, H.M., Vissers, C.A. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems, 29* (1997), 413-436.

[13] Quartel, D.A.C., Steen, M.W.A., Pokraev, S., and van Sinderen, M.J. COSMO: A conceptual framework for service modelling and refinement. *Information Systems Frontiers, 9* (Jul. 2007), 225-244.

[14] Rahm, E., Bernstein, P.A. A survey of approaches to automatic schema matching. *VLDB Journal, 10* (2001), 334-350.

[15] Shaw, M. 1996. Procedure Calls Are the Assembly Language of Software Interconnection: Connectors Deserve First-Class Status. In *Selected Papers From the Workshop on Studies of Software Design* (May 17 - 18, 1993). LNCS, vol. 1078. Springer-Verlag, London, 17-32.

[16] Shaw, M., DeLine, R., Zelesnik, G. Abstractions and Implementations for Architectural Connections. In *Proc. of the 3rd Intl. Conf. on Configurable Distributed Systems (ICCDS '96)*, (1996), 2-10.

[17] van der Aalst, W.M.P. Inheritance of Interorganizational Workflows to Enable Business-to-Business E-Commerce. *Electronic Commerce Research, 2 (2002)*, 195-231.

[18] van der Aalst, W.M.P., Weske, M. The P2P Approach to Interorganizational Workflow. In *Proc. of the 13th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'01)*, (Interlaken, Switzerland, June 4-8, 2001). LNCS, vol. 2068. Springer, Berlin, 140-156.