An approach to relate business and application services using ISDL[1]

Dick Quartel, Remco Dijkman and Marten van Sinderen
Centre for Telematics and Information Technology
{D.A.C.Quartel, R.M.Dijkman, M.J.vanSinderen}@utwente.nl

Abstract

This paper presents a service-oriented design approach that allows one to relate services modelled at different levels of granularity during a design process, such as business and application services. To relate these service models we claim that a 'concept gap' and an 'abstraction gap' need to be bridged. The concept gap represents the difference between the conceptual models used to construct service models by different stakeholders involved in the design process. The abstraction gap represents the difference in abstraction level at which service models are defined. Two techniques are presented that bridge these gaps. Both techniques are based on the Interaction System Design Language (ISDL). The paper illustrates the use of both techniques through an example.

1. Introduction

Following object-orientation and component-orientation, service-orientation is the current paradigm in developing enterprise applications. Informally, the service-oriented paradigm is characterized by the explicit identification and description of the externally observable behaviour, or service, of a software application. Applications can then be integrated, based on the description of their externally observable behaviour, without the need for knowledge of their internal functioning.

Interestingly, the same paradigm could also be applied to objects and components, which are however at a finer level of granularity than software applications, and to businesses and business units, which are at a coarser level of granularity. Therefore, the service-oriented paradigm could be ideal to bridge the gap between services at different levels of granularity, such as business and application services. However, we claim that before this can be done, two important issues must be addressed.

Firstly, stakeholders that focus on business services may use another conceptual model than stakeholders that focus on application services. We consider a conceptual model as the set of concepts that are applied to represent, or model, some system. For example, stakeholders in application services may use 'message-passing' concepts to represent interactions between service providers, while stakeholders in business services may use concepts such as 'negotiation' or 'customer contact'. To relate business services to application services this 'concept gap' must be bridged.

Secondly, business services are generally considered at higher abstraction levels than application services, both with respect to the level of granularity and with respect to the level of detail at which their interactions are described. For example, a business service may describe the interaction 'register client', while several application services may be involved in this business service, describing interactions like 'enter client's name', 'store client's address in database', etc. To relate business services to application services this 'abstraction gap' must also be bridged.

This paper aims to relate business services to application services. To achieve this aim, it describes an approach to service-oriented design that explicitly addresses the two 'gaps' outlined above. It presents a technique, called conformance assessment, to assess that some composition of services at a lower abstraction level correctly implements a service at a higher abstraction level. Also, it presents a technique to relate the different conceptual models used by different stakeholders in a service-oriented design process. We use the concepts and conformance assessment technique from the Interaction Systems Design Language (ISDL) [18, 19] as a basis for bridging the gaps. We focus on behaviour concepts and conformance assessment of behaviour.

The paper is further structured as follows. Section 2 describes the principles of our service-oriented design approach. Section 3 presents three representative

[1] 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.

conceptual models used by different stakeholders. Section 4 introduces ISDL, including a technique for conformance assessment. Section 5 describes a technique to relate different conceptual models via ISDL. Section 6 illustrates the use of both techniques. And section 7 presents our conclusions and future research.

2. Service-oriented design

The purpose of our service-oriented design approach is to systematically design application support for business processes. Several design milestones are distinguished, each associated with its own conceptual model.

2.1 Design approach and milestones

A design process consists of a number of design steps that can be ordered in various ways, such as top-down, bottom-up, and iterative. Figure 1 illustrates the basic characteristics of a service-oriented design step.

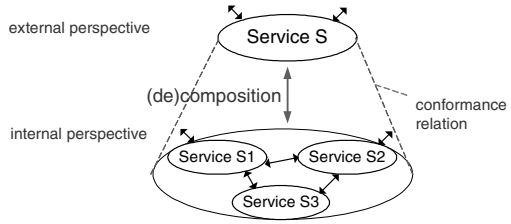


Figure 1. External and internal perspectives

The basic design step distinguishes between an *external* and *internal system perspective*. Here, a system represents some entity that provides a service, e.g., a business, application or software component. The external system perspective corresponds to the perspective of the system users. These users are only interested in the functionality, or behaviour, provided by the system as a whole. The system is considered as a black box, and the externally observable behaviour of the system is called the system's *service*. This service is defined by the interactions between the system and its environment (the service users) that the system is capable of supporting, including the relationships between these interactions. The internal system perspective corresponds to the perspective of the system designers. This perspective shows how the system is internally structured as a composition of parts. These parts have to interact to fulfil the purpose of the system as a whole. By considering each part as a system, the external and internal perspectives can be applied again to the system parts. This results in a process of repeated or recursive decomposition, yielding several levels of decomposition, also called levels of abstraction.

We assume that for each design step both the behaviour of the service and the behaviour of its design are defined completely. This allows one to assess the conformance between the service specification and its design, since the external behaviour of the design should correspond to the service behaviour of the external perspective (as illustrated by the dashed lines in Figure 1).

To structure the design process, we distinguish the following design milestones, which are the result of one or more design steps, representing a design or specification that satisfies certain design objectives:

- *business process model*, which defines the activities to be performed by one or more enterprises, and their relationships;
- *application service*, which decomposes the business process into the application and its environment, and defines how the application supports the business process through interactions with the environment;
- *application design*, which designs the application service in terms of a composition of sub-services to be provided by application building blocks; and
- *application implementation*, which implements the design using a specific service computing technology or platform.

A more elaborate discussion of our service-oriented design approach can be found in [17].

2.2 Conceptual models and their relations

Since different stakeholders are involved in different milestones, different conceptual models may be used for these milestones. We distinguish three conceptual models for the milestones from the previous subsection, as illustrated in Figure 2. The application design concepts address both the application service and the application design milestone.

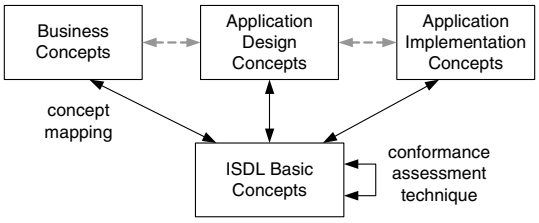


Figure 2. Concept mapping via ISDL

To assess conformance between designs that are constructed from different conceptual models, we need conformance assessment techniques that can be applied to different conceptual models. Depending on the number of conceptual models and the complexity of their relations, assessing conformance between conceptual models in pairs may be inefficient. Therefore, we propose the

approach illustrated in Figure 2 [10]. In this approach different conceptual models are related indirectly via mappings onto a single, basic conceptual model; in this way requiring the use of only a single conformance assessment technique and associated tools. We propose the concepts from ISDL as a basic conceptual model for behaviour modelling. ISDL and its conformance assessment technique are explained in section 4. Mappings from the conceptual models to ISDL are described in section 5.

Designers need modelling languages to express the concepts from which they construct their models. Languages are related to conceptual models via a concept mapping. Figure 3 illustrates the relation between conceptual models and languages. A language consists of two parts: language concepts and a notation. Language concepts define the abstract syntax and the semantics of a language. A notation defines the textual or graphical concrete syntax of a language. A notation is related to language concepts via an interpretation mapping. Language concepts are related to a notation via a representation mapping.

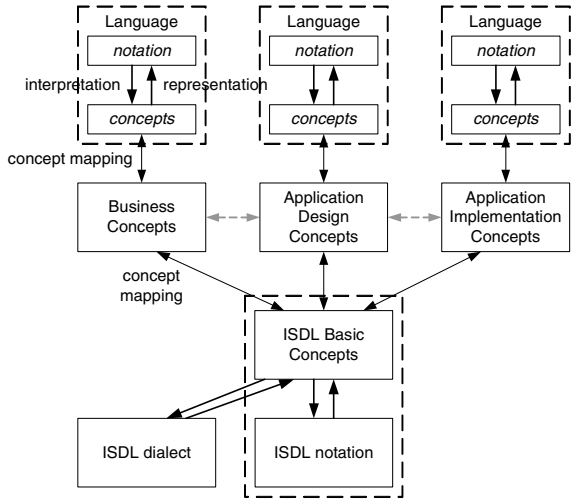


Figure 3. Concepts and Languages

Two different approaches can be followed in choosing a modelling language to express some conceptual model.

The first approach is to use an existing modelling language, e.g., because one is familiar with it or tool support is available. In this case one has to define a mapping between that language and the concepts that it represents.

The second approach is to use the modelling language associated with the basic conceptual model, in our case ISDL. Using this approach we may want to introduce additional notational elements to represent the concepts from the conceptual model in an intuitive and convenient way. In this way we introduce a dialect of ISDL. An example of an ISDL dialect for business process modelling is Amber [12]. Added notational elements are specializations of existing elements (similar to UML stereotypes) or shorthands for compositions of notational elements. In contrast to the first approach, a profile needs to extend the representation and interpretation mappings, but no concept mapping is needed.

Consequently, a choice between both approaches may depend on the possibility and effort needed to relate a language to concepts on one hand, and to extend the representation and interpretation mappings on the other hand.

3. Conceptual modelling

This section presents the conceptual models for modelling business processes and applications. These conceptual models are based on work described in [14]. In addition, a conceptual model of BPEL is presented as an example of a conceptual model at the application implementation level.

3.1 Business concepts

Figure 4 depicts a conceptual model supporting business process modelling. Concepts and their relationships are represented using a class diagram.

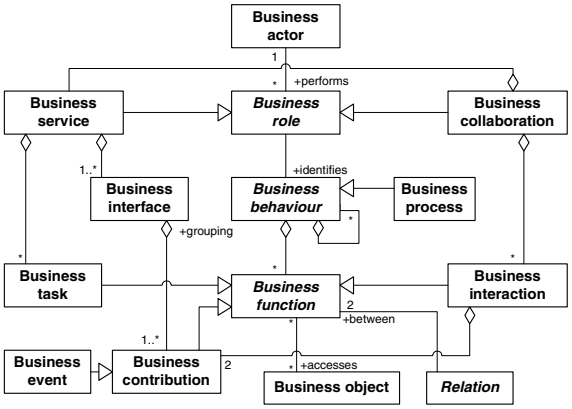


Figure 4. Business concepts

A *business actor* represents some entity, such as a person or organization, that can perform some business role. A *business role* identifies some business behaviour. A *business behaviour* consists of business functions, which represent units of behaviour, i.e., pieces of functionality or activities. Three types of business functions are distinguished. A *business task* is performed

by a single business role. A *business interaction* is performed by two or more business roles in cooperation. And a *business contribution* represents the participation of a single business role in this cooperation. A specific type of contribution is a *business event*, representing that the environment can trigger the business behaviour.

A *business object* represents some entity that is manipulated by a business function, e.g., some information or product. *Relations* between business functions determine the possible orders in which they can be performed. Due to space limitations we do not elaborate on the relation concept, and therefore declare it as abstract. [1] defines relations commonly used to specify the possible orders in which activities can be performed.

To support the service-oriented paradigm, the concept of *business service* is identified explicitly as a type of business role. A business service provides one or more *business interfaces*, each consisting of one or more business contributions, allowing interaction with the business environment. Business functionality is provided to the environment through these interactions. A *business collaboration* represents a composition of interacting business services (roles).

3.2 Application design concepts

Figure 5 depicts a conceptual model supporting application modelling. Where possible, its structure is aligned with the business conceptual model.

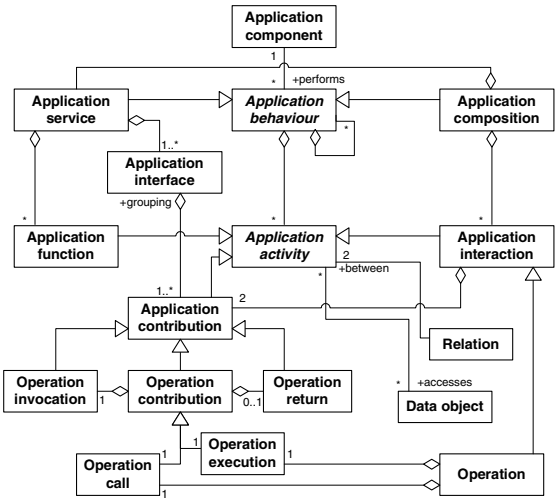


Figure 5. Application concepts

An *application component* represents some entity that performs application behaviour, e.g., a software component, application or information system. An *application behaviour* consists of application activities that represent units of application behaviour. Three types of application activities are distinguished. An *application function* is performed by a single application behaviour. An *application interaction* is performed by two or more application behaviours in cooperation. And an *application contribution* represents the participation of a single behaviour in this cooperation. A type of interaction commonly used is an *operation*, which consists of two parts: an *operation call* (at the client side) and an *operation execution* (at the server side). Both parts consist of an *invocation*, and possibly a *return*.

An *application service* provides one or more *application interfaces*, each consisting of one or more application contributions, allowing interaction with the environment. Application functionality is provided to the environment through these interactions. An *application composition* represents a composition of interacting application services.

3.3 Application implementation concepts

Figure 6 depicts the high-level structure of the conceptual model underlying BPEL4WS [3]. This language is used to implement business processes by coordinating the collaboration between web-services, possibly from different enterprises.

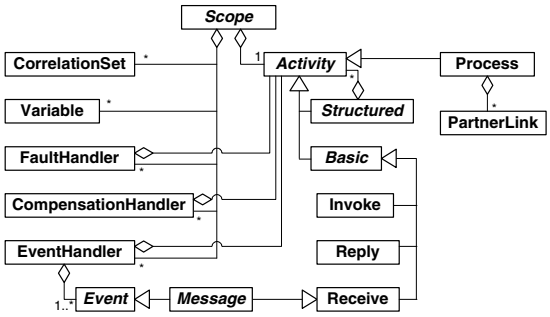


Figure 6. BPEL concepts

An *activity* represents some unit of behaviour that is performed by a business process. A *structured* activity is composed of other activities and defines their possible ordering. Examples of structured activities are sequence, switch and flow. A *basic* activity represents an elementary unit of behaviour. Examples of basic activities are *invoke*, *reply* and *receive*, which are used to call web services.

An activity is defined within a *scope*. This scope may associate with the activity (i) *correlation sets*, which are used to correlate messages of the conversation in which the activity is involved, (ii) *variables*, which are used to hold data and messages, (iii) *fault handlers*, which handle error situations, (iv) *compensation handlers*, which allow

one to reverse the effects of the associated activity, and (v) *event handlers*, which allow the concurrent handling of *events*. A specific type of event is a *message* event, which is similar to a receive activity.

A *process* represents some business process. We consider a process as a type of activity, i.e., the top-level activity. In addition, it defines *partner links*, representing the interfaces and roles of the business partners involved in the business process.

4. ISDL

The Interaction Systems Design Language (ISDL) is aimed at modelling distributed systems at higher abstraction levels. In particular, we use ISDL for business process and distributed application design [13, 18, 19].

We have chosen to use ISDL, because it provides a small, but expressive set of basic and generic concepts for behaviour modelling, aimed at modelling the behaviour of systems from varying domains and at successive abstraction levels. The semantics of ISDL has been defined formally, and a method for conformance assessment has been defined. Furthermore, an integrated editor and simulator is available, and tools supporting conformance assessment and model-to-model (code) transformations are being developed.

4.1 Concepts and notation

Figure 7 depicts part of the behaviour conceptual model of ISDL, including the entity concept. Figure 8 shows how these concepts are represented.

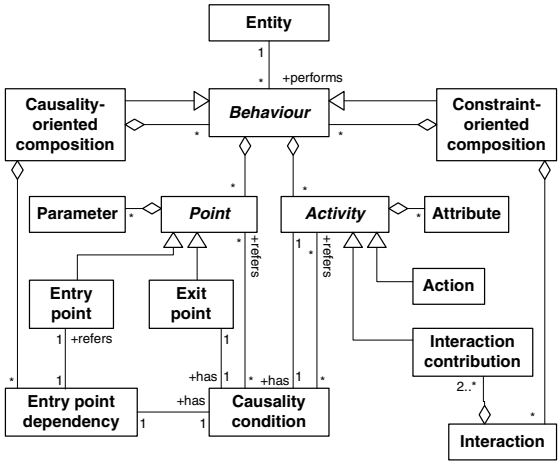


Figure 7. ISDL concepts

The *entity* concept represents a system (part) that can perform some behaviour. A *behaviour* is essentially a set of causally related activities. An *activity* represents some unit of behaviour that is atomic, i.e., cannot be split at the abstraction level at which it is defined. Furthermore, an activity either happens, in which case reference can be made to its result, or does not happen at all, in which case no reference can be made to any result, not even to partial results. We distinguish three types of activities. An *action* is performed by a single behaviour (entity). An *interaction* is performed by two or more behaviours in cooperation. An interaction is expressed as two or more connected *interaction contributions*, which represent the participation of the involved behaviours.

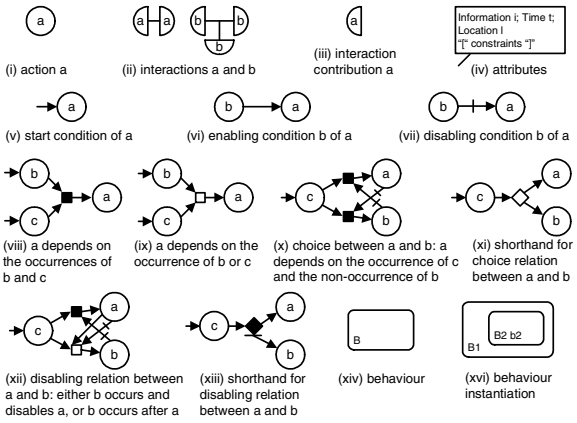


Figure 8. ISDL language elements

An activity can have *attributes* to represent the relevant characteristics of the occurrence of the real-world activity being modelled. Predefined attributes are the information, time and location attribute (see Figure 8 (iv)), representing the activity result (e.g., some information or product), the time of occurrence at which the result is available, and the location where the result is available, respectively. Constraints can be defined on the possible attribute values. These constraints also specify the relation between attribute values established in causally dependent activities. ISDL does not prescribe a language for defining attribute types and constraints, but provides bindings to existing languages that can be used for that purpose. Currently, bindings to Z, Java and Q exist.

Relations between activities are modelled by *causality conditions*. Each activity has a causality condition, which defines how this activity causally depends on other activities. An activity is enabled, i.e., allowed to occur, if its causality condition is satisfied. Three types of basic causality conditions are identified as illustrated in Figure 8: (v) the start condition represents that activity `a` is enabled from the beginning of some behaviour and independent of any other activity, (vi) enabling condition `b` represents that activity `b` must have occurred before `a`

can occur, and (vii) disabling condition ¬b represents that activity b must not have occurred before nor simultaneously with a to enable the occurrence of b. These elementary conditions can be combined using the *and*- and *or*-operator to represent more complex conditions. Figure 8 depicts some simple examples.

Containment of one behaviour by another (the composite), is represented by behaviour instantiation. A behaviour instantiation represents that some behaviour instance is created in the context of the behaviour that contains the instantiation.

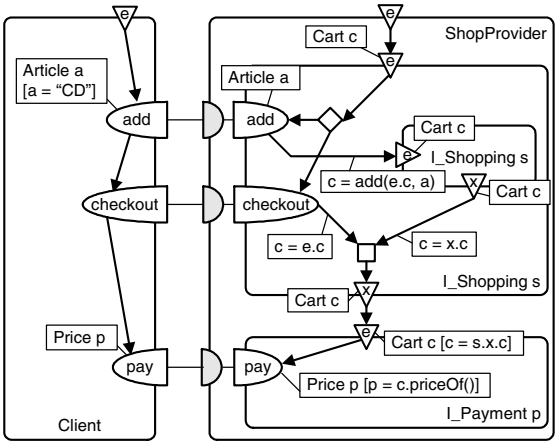


Figure 9. Behaviour composition in ISDL

Behaviours in a composite behaviour can be related using: (i) *constraint-oriented composition*: interactions that relate the interaction contributions of the component behaviours; and/or (ii) *causality-oriented composition*: entry and exit points that represent a causality condition entering a behaviour or a causality condition exiting a behaviour, respectively. The condition that an entry point represents is associated to it via an entry point dependency. Entry and exit points are represented by triangles that point into or out of a behaviour, respectively. 'Attributes' of points are called *parameters*. Interaction contributions of a component behaviour can contribute to interactions of their composite behaviour. This is represented by drawing a line between the interaction contributions of the component and interaction contributions of the composite. Figure 9 depicts a composite behaviour in ISDL. It shows two behaviours that are related by interactions. The ShopProvider behaviour is a composite of two interface behaviours. For example, interface I_Shopping allows a client to add articles to its shopping cart and to check out. The interface behaviours contribute to the interaction contributions of the provider behaviour (represented by the circle segments in gray), and are related by an enabling condition that exits one behaviour and enters the other. Normally, we represent a behaviour and its instantiation separately (so in Figure 9 there would e.g. be a behaviour I_Shopping and an instantiation s). For brevity, we represent them as one.

4.2 Conformance assessment

A method has been defined for ISDL to assess the conformance of an abstract behaviour to a concrete behaviour that refines the abstract behaviour. For a detailed explanation of this method we refer to [18]. Section 6 presents an example

Figure 10 illustrates our approach to conformance assessment. The (concrete) service design adds design information to the (abstract) service specification, e.g., the interactions between the constituent sub-services, or the refinement of an abstract activity into smaller, more concrete activities. To assess conformance, we abstract from the added design information. After abstracting from this information, the obtained abstraction should be equivalent to the original service specification. The particular notion of equivalence being applied, determines the type of service refinements (decompositions) that are considered correct. We assume that the occurrence of each abstract activity corresponds to the occurrence of one or more concrete activities. This assumption makes it possible to compare the abstract behaviour with the concrete behaviour. Concrete activities that correspond to abstract activities are called *reference activities*, since they are considered reference points in the concrete behaviour for assessing conformance. Concrete activities that are not reference activities are called *inserted activities*, since they are inserted during behaviour refinement.

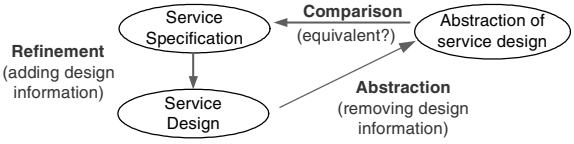


Figure 10. Conformance assessment

Two elementary types of behaviour refinement are distinguished: activity refinement and causality refinement.

Activity refinement allows one to model in more detail a real-world activity that is represented by a single abstract activity. This activity is decomposed into a concrete activity structure, which consists of multiple related, more concrete (sub-)activities. The concrete activity structure makes its result available through the occurrence and associated attributes of one or more of its final activities, which are the reference activities that correspond to the original abstract activity. A concrete

activity structure can make its result available through the occurrence of (i) a single final activity, (ii) a conjunction of multiple, independent final activities, (iii) a disjunction of multiple, alternative final activities, or (iv) a combination of these options.

Causality refinement allows one to model the relations between abstract activities in more detail through adding inserted activities. Abstract activities are not further detailed, and therefore correspond to a single reference activity. Causality refinement should obey the following conformance criteria: (i) an indirect relation between reference activities defined via an inserted activity in the concrete behaviour must be equivalent to the relation defined directly between the corresponding reference activities in the abstract behaviour; and (ii) similarly, an indirect relation between attributes must be equivalent to the direct relation.

Rules have been defined to obtain the abstraction of a behaviour that has been obtained through activity or causality refinement. These rules are explained in [18].

5. ISDL profiles

This section describes the mapping of the conceptual models from section 3 onto ISDL, indicating that ISDL is at least expressive enough to represent those conceptual models. A complete definition of the mappings falls outside the scope of this paper.

5.1 Business and application concepts

The three types of business functions and application activities distinguished in Figure 4 and Figure 5, respectively, can be mapped onto actions, interactions and interaction contributions. For example, Figure 11 shows two alternative ISDL models of a business task, depending on whether (i) one is only interested in the task result, or (ii) also in the period during which it is executed.

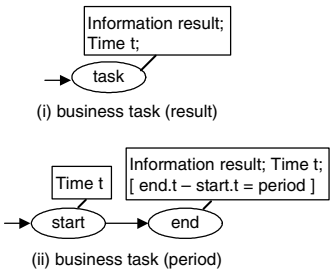


Figure 11. Business task

Relations between business functions and application activities can be modelled using causality conditions in ISDL. Business and data objects can be modelled using activity attributes. Business behaviours (roles) and application behaviours are mapped onto ISDL behaviours. Business collaborations and application compositions are modelled as interacting ISDL behaviours using constraint-oriented composition. Business and application services are represented by ISDL behaviours having interaction contributions and, possibly, actions. These interaction contributions may be grouped into business or application interfaces by introducing a sub-behaviour for each group of interaction contributions. These sub-behaviours are composed using causality-oriented composition, such that relations between interfaces are represented by entry and exit points. An example is given in Figure 9.

5.2 BPEL concepts

BPEL activities can be modelled in ISDL using compositions of actions, interaction contributions, causality conditions and behaviours. The scope of an activity can be represented by a composite behaviour in ISDL that contains the activity. This behaviour may instantiate other behaviours representing fault, compensation and event handlers. Figure 12 illustrates the containment relationship between these behaviours, abstracting from the causal relations between them.

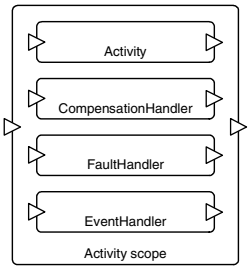


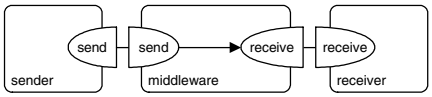
Figure 12. Activity scope in ISDL

A fault handler behaviour is enabled by an error activity representing the occurrence of some error situation. The ISDL disabling and choice relation can be used to model the disruption of the scoped activity by the error activity. A compensation handler behaviour is enabled by the occurrence of the scoped activity, such that it can refer to the activity result. In addition, this behaviour depends on some other activity that initiates the compensation and reverses the established result. An event handler behaviour consists of interaction contributions, representing events that can be triggered by the environment. BPEL variables are represented by activity attributes and entry/exit point parameters.

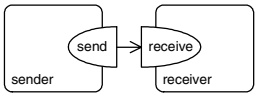
Correlation between messages is modelled in ISDL using constraint-oriented composition of interaction contributions and causal relations between these

contributions. Since interaction contributions are uniquely identified correlation is implicit. Only in case a behaviour can have multiple instances of the same interaction contribution explicit correlation may be required, which can be modelled using attributes.

Figure 13 depicts an ISDL model of one-way message passing. Because ISDL adopts a synchronous interaction model, the activity of sending and receiving the message is modelled separately, making the role of middleware explicit. A shorthand is introduced to represent message passing directly in ISDL.



(i) one-way message passing, modelling the role of middleware explicitly



(ii) shorthand for one-way message passing, abstracting from the role of middleware

Figure 13. One-way message passing

A BPEL process is modelled by the top-level behaviour of a composite behaviour in ISDL. Partner links are modelled through constraint-oriented composition of the business partner behaviours.

6. Example

This section presents parts of the design of a web-shop application. The aim of this example is to illustrate how business and application services can be related in our service-oriented design approach.

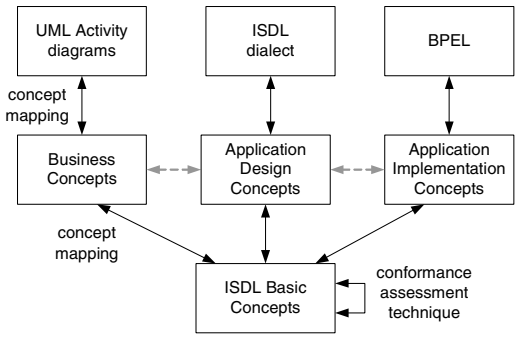


Figure 14. Applied conceptual models

Figure 14 depicts the modelling languages that are used to model these services. The web shop business process is modelled by a UML Activity diagram. A dialect of ISDL that comprises composite concepts to represent frequently occuring constructs of basic concepts is used to model the web shop application service and its design. And BPEL is used to model part of the implementation.

6.1 Business process

Figure 15 depicts a UML activity diagram of a web-shop business process. Activity select represents the selection and addition of articles to the client's shopping cart, followed by a request from the client to checkout. The web shop coordinates the payment and ordering of the articles. In case of success, the checkout request is accepted, and logistics is responsible for receiving, packing and delivering the articles. The client may cancel the order until the articles are packed. A checkout is rejected in case payment or ordering fails. In the latter case and in case of cancellation, the client gets a refund.

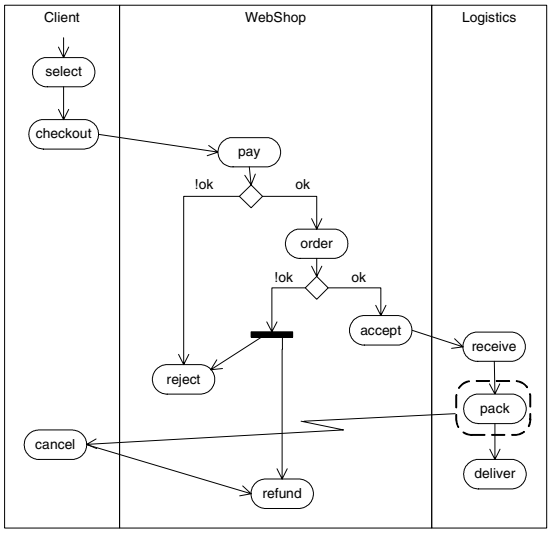


Figure 15. Web-shop business process

This example shows a typical use of activity diagrams, identifying business roles (and actors), business tasks and flow relations. Interactions between business roles are not considered, although some tasks require the involvement of two business roles. In general, such tasks are assigned to the role that is 'most responsible' or 'takes the initiative'. Furthermore, interactions in activity diagrams would be limited to message passing, which does not allow one to model interactions at a high abstraction level representing more complex negotiations.

Figure 16 depicts an alternative ISDL model that represents the interactions between the business roles explicitly. For example, activity checkout is an interaction

between a client and the web shop, in which the contents of the shopping cart is established. ISDL allows one to model the constraints each role has on this contents, without modelling how the negotiation on these constraints takes place, e.g., through a complex pattern of message passing. In this example, we added the simple constraint that the client wants to shop for a maximum amount of money and the web shop only accepts orders for some minimum amount. Interactions inform and ready are introduced to represent that the web shop has to inform logistics about the new order and, vice versa, that the order has been packed. A filled diamond represents an and-split (concurrency) in ISDL.

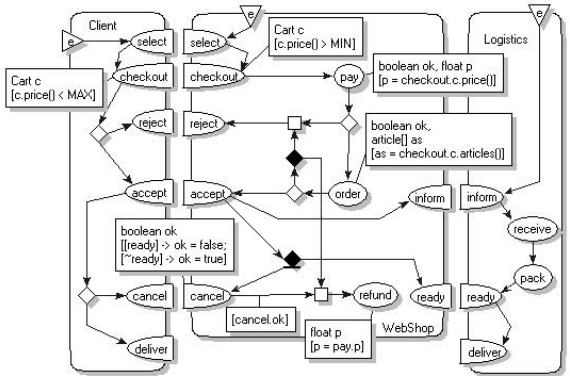


Figure 16. Interactions between business roles

We assume here that the web shop needs to be supported by some application. Therefore, the WebShop behaviour defines the required application service. In this respect, the model of Figure 15 may be considered a model of the integrated application service, including its relationship to certain business tasks in the application environment. This model integrates the contributions of the involved actors by modelling all activities as actions, in this way abstracting from the individual responsibility of each actor in the business process. Figure 17 depicts an ISDL representation of this integrated behaviour, abstracting from interactions inform and ready. The behaviour has been structured using causality-oriented composition, such that each sub-behaviour corresponds to a swimlane in the activity diagram. Since causality-oriented composition is a pure syntactic operation in ISDL, and therefore no interactions are modelled between sub-behaviours, the integrated model does not define the individual behaviour of actors. This allows one to focus on what the web shop business process should do, and not on how this can be done, or by whom.

The application service of Figure 16 can be obtained from the integrated model by considering in which tasks the application is either completely or partially involved. Activities for which the application is completely responsible are modelled as actions inside the web shop behaviour. Activities for which the application is partially responsible are modelled as interactions between the application and the other actors involved.

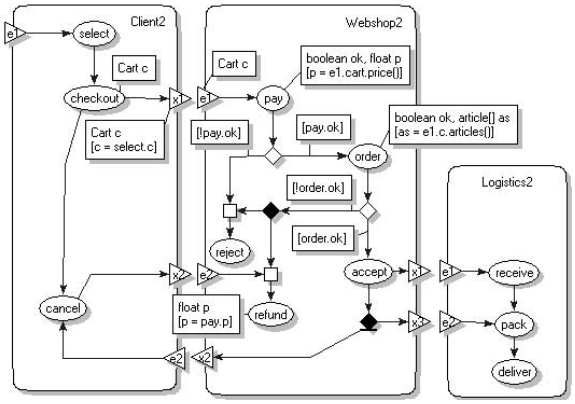


Figure 17. Integrated web-shop behaviour

6.2 Application design

Figure 18 depicts a design for the web shop application service. Three application components are distinguished, each providing an application service that defines application contributions, and possibly internal application functions. The Payment service allows one to check the status of a client's account, credit and debit the account, and send a notification about the account. The Ordering service allows one to order articles, using a two-phase commit pattern. The Coordinator service is responsible for client interaction and coordinating the payment, ordering and delivery of the articles.

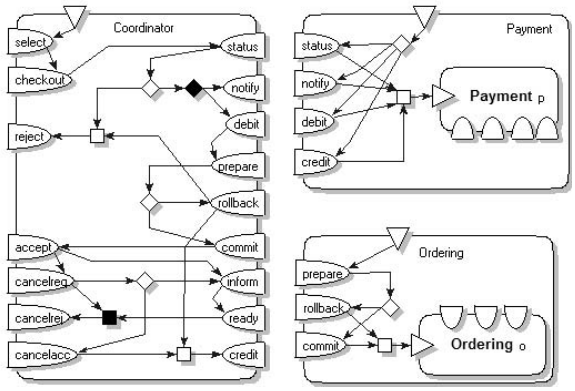


Figure 18. Application design

Figure 19 depicts the composite (or integrated) behaviour of the design, which is obtained by integrating the interactions between the application components.

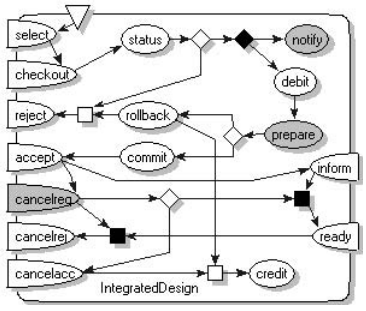


Figure 19. Integrated application design

This behaviour should conform to the application service defined for the web shop in Figure 16. This can be assessed using the method of section 4.2. Activities in gray are inserted activities obtained through causality refinement. Figure 20 depicts the behaviour that results after abstracting from the inserted actions.

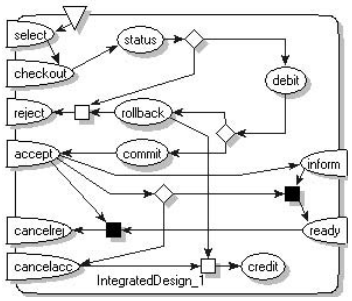


Figure 20. Abstraction from inserted actions

Three instances of activity refinement are used:

(i) status and debit refine abstract activity pay, with debit being the (single) final activity;

(ii) rollback and commit refine abstract activity order, forming a disjunction of final activities; and

(iii) cancelrej(ect) and cancelacc(ept) refine abstract activity cancel, forming also a disjunction of final activities.

Figure 21 depicts the behaviour that results after replacing each final activity structure by the abstract activity it refines.

Without proof, we state that applying the abstraction rules to these refinements renders an abstract behaviour that is equivalent, and thus conformant, to the WebShop behaviour in Figure 16. This behaviour can be obtained from the behaviour in Figure 21, after renaming action credit to refund and removing the enabling relation between actions inform and ready, while this relation is implied by behaviour Logistics in Figure 16.

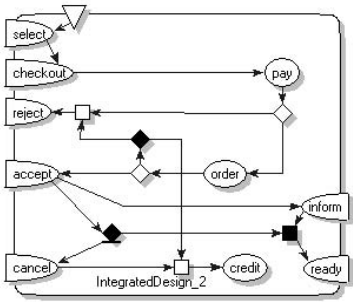


Figure 21. Abstraction from final actions

6.3 Application implementation

As an example, we consider the implementation of interaction debit from Figure 18. In BPEL, this interaction can be implemented using an invoke activity at the Coordinator side, and a receive and reply activity at the Payment side, as shown in Figure 22. Although this involves two BPEL processes, one at the Coordinator side and one at the Payment side, we represent them in one figure.

```
<process name="DebitCoordinator"/>
  ...
  <invoke name="debit_invoke" operation="debit">
    <catch faultName="fault">
      ...
    </catch>
  </invoke>
  ...
</process>

<process name="DebitPayment"/>
  ...
  <sequence>
    <receive name="debit_receive"
             operation="debit"/>
    <switch>
      <case condition="...">
        <reply name="debit_reply"
               operation="debit"/>
      </case>
      <otherwise>
        <reply name="fault_send"
               operation="debit"
               faultName="fault"/>
      </otherwise>
    </switch>
  </sequence>
  ...
</process>
```

Figure 22. BPEL implementation of interaction debit

Figure 23 depicts an ISDL model of this implementation. Interaction contributions debit_invokereq and debit_invokersp represent the sending of the invoke request and the receipt of the reply message, respectively.

Interaction contributions fault_send and fault_receive represent the sending and receiving of a fault message in case the debit operation fails. At the Coordinator side, fault_receive may disable the occurrences of debit_invokereq/rsp, but may not occur after debit_invokersp has occurred. For brevity, activities and fault handlers are not defined in separate behaviours as would be the case when applying the rules from section 4.1 systematically.

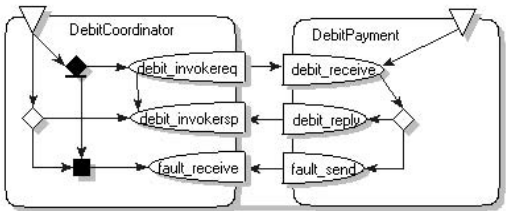


Figure 23. Implementation of interaction debit

Activities debit_invokereq/rsp, debit_receive and fault_receive are inserted activities obtained through causality refinement. Activities debit_reply and fault_send are alternative final activities of abstract activity debit, such that a positive debit_reply corresponds to a successful debit, and a negative debit_reply or fault_send corresponds to an unsuccessful debit. Assuming the BPEL semantics is captured properly, Figure 23 would be a correct implementation of interaction debit.

7. Conclusions

We have described a service-oriented design approach to relate business and application models that are produced at successive abstraction levels and are possibly constructed from different conceptual models. This approach consists of two ISDL-based (meta-)modelling techniques. First, we have explained how ISDL profiles can be used to relate different conceptual models used by different stakeholders, including the modelling languages the stakeholders use to express them. More work is needed on the precise and complete definition of these profiles. Second, a technique is provided to assess the conformance of ISDL models. The combination of this technique with ISDL profiles allows one to relate any two models produced in the design process. Besides using ISDL as underlying conceptual model, we believe ISDL is also suited to be used directly as a modelling language, possibly by introducing one or more dialects.

Various research groups have proposed languages for service-oriented design [4, 5, 7, 16]. [16] also supports a form of conformance verification. Our work extends this work, because we describe how different modelling languages in the design process can be related and because we consider modelling at higher levels of abstraction. Our work complements the work on design processes for service-oriented design [2, 8], because we take a more precise (formal) approach to modelling and conformance verification. Finally, design languages have been proposed to graphically represent (XML-based) service descriptions (see e.g. [6, 15]). Our work contributes to this area, because we also consider higher abstraction levels. We refer to [9] for a more detailed overview of related work.

Currently, our work focuses on the development of techniques and associated tools to support our service-oriented design approach, based on ISDL. An integrated editor and simulator is available for use [13]. The simulator allows a designer to "step through" an ISDL behaviour, by allowing one to simulate in each step the execution of an action or interaction for which the causality condition is satisfied. A prototype tool that calculates the abstraction rules for causality and activity refinement exists. Further, we are working on tools to relate and transform meta-models to facilitate the implementation of ISDL profiles. An ISDL profile for BPEL, including some preliminary results on an ISDL-to-BPEL/WSDL model transformation have been presented in [11]. In addition, we plan to develop profiles for UML activity and state diagrams.

8. References

[1] W. van der Aalst, et al. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.

[2] G. Alonso, et al. *Web Services: Concepts, Architectures and Applications*. Springer, 2003.

[3] BEA Systems, Microsoft, IBM, and SAP. *Business process execution language for web services (BPEL4WS) version 1.1*. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, May 2003.

[4] B. Benattallah, et al. Conceptual modeling of web service conversations. In *Proc. of the 15th Int. Conf. on Advanced Information Systems (CAiSE)*, Klagenfurt, Austria, 2003. Springer.

[5] B. Benatallah, Q. Sheng, and M. Dumas. The Self-Serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, Jan/Feb. 2003.

[6] BPMI. *Business process modeling language (BPML) version 1.0*. http://www.bpmi.org/bpml-spec.esp, Nov. 2002.

[7] T. Bultan, et al. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. of the Int. Conf. on the World Wide Web (WWW)*, Budapest, Hungary, May 2003.

[8] C. Bussler. *B2B integration - concepts and architecture*. Springer, 2003.

[9] R. Dijkman and M. Dumas. Service-oriented design: a multi-viewpoint approach. In: *International Journal of Cooperative Information Systems* 13(4), pp. 337-368, 2004.

[10] R. Dijkman, et al. An Approach to Relate Viewpoints and Modeling Languages. In *Proceedings of the 7th IEEE*

Enterprise Distributed Object Computing (EDOC) Conference, Brisbane, Australia, pp. 14-27, 2003.
[11] T. Dirgahayu. Model-Driven Engineering of Web Service Compositions: A Transformation from ISDL to BPEL, M.Sc. thesis, University of Twente, The Netherlands, July, 2005.
[12] H. Eertink, et al. A business process design language. In *Proc. of the World Congress on Formal Methods*, 1999.
[13] *ISDL home*. http://isdl.ctit.utwente.nl/, n.d.
[14] H. Jonkers, e.a. *Concepts for Modelling Enterprise Architectures*. Int. Journal of Cooperative Information Systems, vol. 13, no. 3, Sept. 2004, pp. 257-287.
[15] K. Mantell. *From UML to BPEL*. http://www-106.ibm.com/developerworks/webservices/library/ws-uml2bpel/, 2003.
[16] M. Mecella, et al. Modeling e-service orchestration through Petri nets. In *Proc. of the 3rd Intl. Workshop on Tech-nologies for E-Services (TES)*, pp. 38–47. Springer, Sept. 2002.
[17] D. Quartel, et al. Methodological support for service-oriented design with ISDL. In *Proc. of the 2nd Int. Conf. on Service Oriented Computing*, New York City, NY, USA, 2004.
[18] D. Quartel, et al. On architectural support for behavior refinement in distributed systems design. *Journal of Integrated Design and Process Science*, 6(1), March 2002.
[19] D. Quartel, et al. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems*, 29:413–436, 1997.