Exploiting Semantics of Inter-Process Dependencies to Instantiate Predefined Integration Patterns

Georg Grossmann **Michael Schrefl** **Markus Stumptner**

University of South Australia, Advanced Computing Research Centre, Mawson Lakes, SA 5095, Adelaide, Australia. E-mail: {`cisgg, cismis, mst`}`@cs.unisa.edu.au`.

Abstract

The integration of behaviour has long been a major goal for research into information systems integration. Its greater complexity compared to the well understood integration of structure means that it must be considered from a highly abstract (conceptual) level so the person responsible for the integration can focus on the integration rather than on implementation details. This paper extends previous work which defined ways to identify correspondences between processes and offered integration patterns arising from these correspondences. It investigates the integration of heterogeneous activity specifications and addresses the problem of defining activity specifications in the integrated process. As a result, the user can instantiate the integration patterns automatically and need not to deal with parameter bindings.

1 Introduction

In an age where the more or less automated interaction of business information systems is common reality, the development of advanced methods for supporting the integration process that enables this interaction between systems retains and in fact increases its significance to business operations, whether these systems represent a linkup between multiple organisations (e.g., communication between services in an e-business environment) or a linkup with legacy systems (e.g., a small branch-specific database interacting with the company ERP system). In this context, support for the integration of behaviour rather than just data structures continues to represent a long-term research goal. Building on the integration of structure which has been extensively investigated by past work in database research and which is a current topic in many research papers dealing with XML based data (Eyal & Milo 2001, Rodriguez-Gianolli & Mylopoulos 2001), the integration of heterogeneous behaviour has still not gained as much attention, mostly due to the inherent difficulty of handling the dynamic aspects of a domain.

This paper builds on earlier work (Grossmann et al. 2004, 2005) which dealt with the integration of heterogeneous object behaviour on the schema level. The object behaviour is represented by public accessible business processes and are integrated by a global business process that combines and/or extends their functionality where the global business process models the behaviour of the integrated object type. Our approach requires the input of two business processes in form of UML Activity Diagrams and creates the global business process semi-automatically. The integration is based on a correspondence model which is linked to a set of activity integration patterns that create or transform parts of processes. Depending on the applied integration patterns, two different strategies can be followed: (1) Two *local business processes* are integrated by a single *global business process* and replaced by it, e.g., a small bank is taken over by a larger competitor and its accounting system is integrated into the existing system. (2) Two local business processes are integrated loosely and build a global business process. Supply chain management is a typical example for this scenario where different systems exchange data to achieve a common goal. We will focus on the first scenario.

The main contribution of this paper is the definition of activity specifications in the global business process and the binding of global and local parameters. In previous work, the description of integration patterns has focused on control flow and ignored activity interfaces. However, in practice, activity specifications are a key part of the behaviour description and need to be considered for parameter binding during instantiation. This work shows how activity specifications are defined in the integrated activity diagram and how their parameters are bound. The paper is structured as follows: Section 2 gives an overview of the integration approach. Section 3 discusses parameter correspondences which will lead to different instantiation options in combination with integration patterns. The last section deals with related and future work.

2 Integration approach

Integration of object types consists of structure- and behaviour based integration. First one focuses on the integration of object types and their attributes whereas latter one targets the methods implemented by the object types and the order of their execution. This paper proposes an approach to behaviour based integration where two pre-existing behaviour of object types are integrated. The behaviour descriptions of two object types o_1 and o_2 is represented by two UML activity diagrams B_1 and B_2, respectively, and integrated by a single diagram B which defines the behaviour of object type o. Object type o is the result of the structure based integration of o_1 and o_2 which is usually performed prior to the integration of behaviour. The approach presented here allows a consistent integration of two object type behaviours

---

This research was partially supported by the CIEAM CRC Project SI-302 "Improved OPAL Monitoring and Management System".

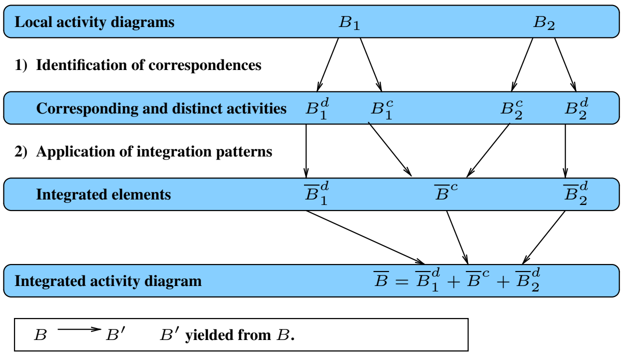
Local activity diagrams
B_1
B_2
1) Identification of correspondences
Corresponding and distinct activities
B_1^d
B_1^c
B_2^c
B_2^d
2) Application of integration patterns
Integrated elements
$\overline{B}_1^d$
$\overline{B}^c$
$\overline{B}_2^d$
Integrated activity diagram
$\overline{B} = \overline{B}_1^d + \overline{B}^c + \overline{B}_2^d$
$B \longrightarrow B'$ B' yielded from B.

Figure 1: Integration approach for two processes.

in a semi-automated way. Consistency means that the behaviour of the input object types is observable in the behaviour of the integrated object type (Schrefl & Stumptner 2002). In case more than two business processes need to be integrated, successive pairwise integration is applied.

The integration approach consists of two steps, *identification of correspondences* and *application of integration patterns*. The first step identifies correspondences between business processes and their activities where the correspondences form the basis for a semantically meaningful integration. The identification may need human interaction in case where correspondences cannot be identified uniquely but tool support is possible by checking conditions on object property values. The correspondence model supports relationships between a group of activities which is explained in the technical report of this paper (Grossmann et al. 2007). The second step involves the application of predefined integration patterns which create the integrated activity diagram successively. For each correspondence identified in the first step, one integration pattern is applied where the input of the pattern are the activities participating in the correspondence and output is an integrated construct which becomes part of the integrated business process. The concept of the integration approach is depicted in Figure 1.

2.1 Identification of correspondences

At the beginning of the integration process, correspondences are identified. They are defined as *semantic relationships* $s(x, y)$ where s is a specific semantic relationship type which can be set between two business processes. Relationships are set first on the level of business processes, i.e., x and y represent two activity diagrams, and second, on the level of activities, i.e., x and y represent a single activity node or a group of activity nodes belonging to two different activity diagrams. The identification of business process relationship in the first step is easier than targeting activity relationships immediately because they correspond to the relationship of their object types which can be identified without analysing the object type's behaviour. Further, business process relationships determine possible activity relationships as found out in previous work (Grossmann et al. 2004). Semantic relationships supported by the integration approach are explained in Table 1.

Each activity diagram is defined by a set B_i of elements as specified in UML. The identification of correspondences separates sets B_1 and B_2 in two subsets of corresponding and distinct elements as shown in Figure 1. B_1^c consists of elements which correspond to elements in B_2^c, B_1^d and B_2^d consist of elements which do not correspond to another element.

2.2 Integration patterns

In a second step integration patterns are applied to the activities that correspond to each other. In previous work, we have identified a set of integration patterns which specify in a declarative way how two activities can be integrated, see Figure 2. Integration patterns distinguish between *local* and *integrated* activities where local activities belong to the local (or input) activity diagrams. As mentioned in Section 1, two integration strategies can be followed, (1) an integrated business process replaces local business processes, or (2) an integrated business process is created out of loosely coupled business processes. We have identified integration patterns for both strategies but concentrate on patterns supporting the first strategy here. Patterns are divided in two categories, one category holds patterns for executing both corresponding activities and the other category holds patterns for executing one activity. The first category contains four different ways in which two activities can be executed as shown in the first row of Figure 2. Additionally, the integrated activity may *aggregate* or *process* output data of both activities as shown in the second and third row. *Aggregate* means that data of both activities are combined, e.g., to compute the sum. *Process* means that a result is computed from both values, e.g., the maximum is determined. The difference to *aggregate* is that the resulting value may not be of the same type as the input values. The second category contains patterns which execute only one activity. The first two represent *static selection*, i.e., the choice which activity will be executed is made prior to the integration. The last three patterns of the category cover *dynamic selection*, i.e., the choice of activity execution is made during run-time. In the *user selection* pattern, user interaction is necessary to choose an activity and in *dynamic binding* the activity is chosen according to a separate guard condition. The last pattern, *conditional binding*, defines conditions on the properties of the integrated instance from which a decision for a single activity can be derived.

2.3 Integration choices

Each semantic relationship explained in Table 1 is assigned to at least one integration pattern. If alternative integration patterns are applicable during integration then the system integrator has an *integration choice*. Previous work contains a complete mapping of all semantic relationships to integration patterns (Grossmann et al. 2004). The main contribution of this paper is the automatic instantiation of applied integration patterns.

3 Instantiation of integration patterns

The instantiation of integration patterns has two requirements, (1) activity specifications of the integrated business process must be defined so they can be invoked, and (2) parameters of these specifications must be bound to local parameters in order to trigger local behaviour automatically.

Activity specifications are defined by integrating the activity specifications of the local activities. The integration is based on the identification of parameter correspondences in context of business process and activity correspondences explained in the next section. Afterwards, local and integrated parameters are bound which allows an automatic instantiation of the activity integration patterns. We first explain which parameter correspondences may appear in the context of process and activity correspondences. Independently from parameter correspondences, we de-

Name	**Definition**	**Example**
identity-extension (identity$^{\mathrm{E}}$)	The extensional level of the identity relationship captures the notion of *sameness* of two processes p_1, p_2 under the *closed world assumption* (Maier 1983). It holds between p_1, p_2 if their *real-world extensions* are identical at all point in time. The *real-world extension* of a business process p is the set of all real world objects modelled as instances of p. Activities of identity$^{\mathrm{E}}$-related processes may be *identity-related* (both model the same functionality of their instances) or *distinct* (does not correspond to any activity in the other process).	Two on-line street maps MapForce and PathFinder provide a map for city Melbourne. Activities MapForce.changeStreetName and PathFinder.changeRoadName as well as MapForce.findStreet and PathFinder.findRoad model identical functionality.
identity-intension (identity$^{\mathrm{I}}$)	The intensional level captures the notion of *sameness* of p_1, p_2 under the *open world assumption.* It holds between p_1, p_2 if their *real-world intensions* are identical. The *intension* of a business process p is the set of all real-word objects that exist at a point of time and could belong to p, had they actually been modelled as instances of p. Activities of identity$^{\mathrm{I}}$-related processes may be identity-related or distinct.	Two train information systems AdelTrans and MelTrans are identical implementations but installed in cities Adelaide and Melbourne, respectively, and manage the local trains, i.e., their instances do not model the same real-world objects.
role	Two processes which model the behaviour of the same real world object in a different situation or context are *role*-related, i.e., it holds between p_1, p_2 if every instance i_1 of p_1 that is equivalent to an instance i_2 of p_2 is role-related to i_2. Activities of role-related business processes may be distinct, identity-, role-, or counterpart-related. Two activities are role-related if they model the same functionality but the functionality depends on the role of the business processes.	Mobile phone yPhone is sold by different vendors on on-line platforms Orange and Lemon. Activities Orange.getNumberSold and Lemon.getNumberSold return the number of yPhone sold at Orange and Lemon and are role-related. Activities Orange.getPrice and Lemon.getPrice are counterpart-related (see counterpart-relationship below).
history	Two processes which represent the behaviour of the same real-world object at different real-world times are *history*-related, i.e., it holds between p_1, p_2 if every instance i_1 of p_1 that is equivalent to an instance i_2 of p_2 is history-related to i_2. Activities of history-related processes may be distinct, identity-, or history-related. Two activities are history-related if they model the same functionality but their functionality depends on the points of time at which the business processes model their instances.	Two on-line systems of the same company for recruiting people A-WantsYou and for its employees A-StaffAccess are history-related. Activities A-WantsYou.submitCV and A-StaffAccess.updateCV are history-related whereas A-WantsYou.getBirthday and A-StaffAccess.getBirthday are identity-related.
counterpart	Two processes not representing the behaviour of the same real-world object can be regarded as *counterpart*-related if they share common properties but represent alternate situations in the real-world. It holds between p_1, p_2 if every instance i_1 of p_1 that shares some common properties with some instances i_2 of p_2 is counterpart-related to i_2. Activities of counterpart-related business processes may be distinct, identity-, or counterpart-related. Two activities are counterpart-related if the model the same functionality but the functionality depends on the counterpart-relationship.	Two on-line booking systems of airlines AirStar and FlyOz offer flights for the same route and are counterpart-related. Activities AirStar.sendOffer and FlyOz.sendOffer are counterpart-related.
category	Two business process instances not representing the behaviour of the same real-word object can be regarded as *category*-related if they share some common properties which belong to a category. It holds between p_1, p_2 if every instance of p_1 is category-related to every instance of p_2. Activities of category-related processes may be category-related or distinct. Two activities are category-related if their functionality can be perceived to belong to a common category.	The behaviour of two monitoring systems of a nuclear reactor WatchRadio and a gas plant WatchGas are category-related. Activities WatchRadio.startReactor and WatchGas.initiatePlant are category-related as well.

Table 1: Semantic relationships between two business processes p_1, p_2 and their activities.

scribe different options how integrated activities can be instantiated. At the end of this section we assign to each correspondence one instantiation option.

3.1 Parameter correspondences

One prerequisite of parameter integration is the identification of parameter correspondences between two activity specifications, similar to the identification of process and activity correspondences in Section 2. Parameter correspondences are defined by the same semantic relationships explained in Table 1. Previously we found out that activity correspondences may differ from the correspondence of their processes, e.g., two role-related business processes may contain two counterpart-related activities. The same holds for parameter correspondences. Parameters may hold different correspondences, e.g., two counterpart-related activities of two role-related business processes may hold counterpart-related parameters. The activity correspondence is a good candidate for the default correspondence of parameters because they indicate the semantic relationship of their functionality and parameters are necessarily part of this functionality.

Activity corresp.	**Business process correspondence**					
	I-E	**I-I**	**R**	**H**	**P**	**C**
I	I, D	D	I, D	I, D	I, D	-
R	-	-	R, I, D	-	-	-
H	-	-	-	H, I, D	-	-
P	-	-	P, I, D	-	P, I, D	-
C	-	-	-	-	-	C, D

Table 2: Possible parameter correspondences in context of business process and activity correspondences. I-E=identity$^{\mathrm{E}}$, I-I=identity$^{\mathrm{I}}$, I=identity, R=role, H=history, P=counterpart, C=category relationship, and D=distinct.

However, the correspondence may also differ from the activity correspondence, e.g., two counterpart-related activities of two role-related business processes may contain two identity-related parameters. Table 2 contains an overview of possible parameter correspondences in context of process and activity correspondences.

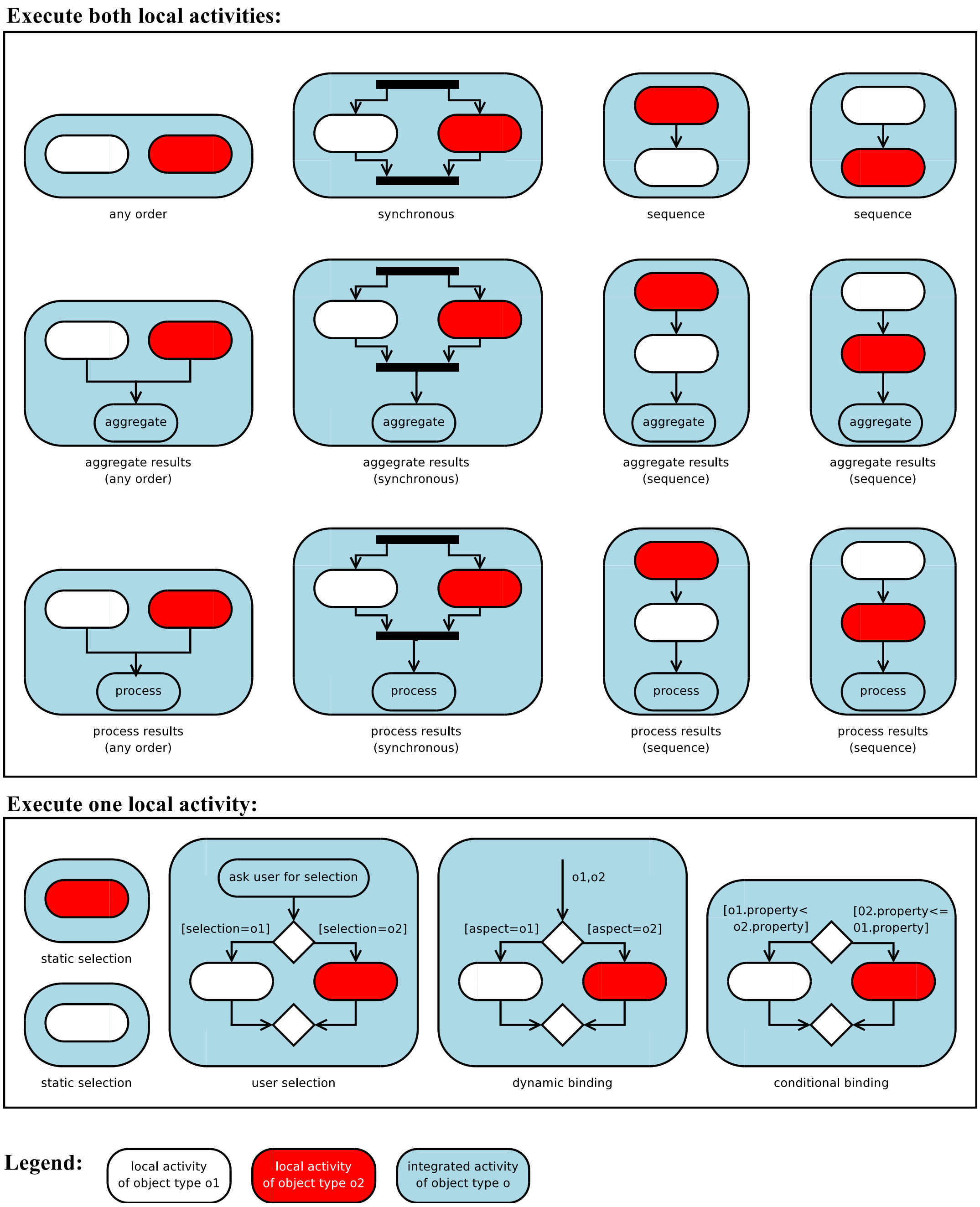

Figure 2: Integration patterns for two local activities.

3.2 Instantiation options

Activity specifications can be integrated and bound in different ways. In this section we investigate the integration independently from parameter correspondences and target the integration of a set of *local parameters* by a single *integrated parameter* and the binding of them.

An integrated parameter is defined by a set of parameter properties listed in Table 3. The properties include the *name* of the parameter, the *activity* it belongs to, the parameter *type* (value- or object type), possible states in case of an object-type parameter, the direction (input or output), a default value in case of a value-type, and a domain which defines possible values of a value-type parameter. All properties except *type*, *states*, and *domain* can be defined straight forward as explained in Table 3. If local parameters are of object-types, the integrated parameter either (a) adopts the object-type of one of the local parameters or (b) defines an object-type which integrates the object-types of the local parameters in a super-type. In (a), the set of states of the integrated parameter is equal to the set of the local parameter from which the object-type was chosen. In (b), the set of states of the super-type holds one state which comprises all states of the local parameters. For this, a specialisation function $h^{+}_{(B_i,B)}(s_n) = s$ must exist which derives a state of the super-type from the states of the local parameters. B_i represents the activity diagram of the object-type o_i of a local parameter p_i and B the activity diagram of the super-type o of o_i. The function transforms a state $s_n \in S_{p_i}$ to state s where S_{p_i} is the set of states defined in local parameter p_i and s is the state defined in the integrated parameter p of type o which integrates p_i. In other words, if an object of type o_i is in a state s_n then the same object regarded as object of type o is in state s. In case of value-types, the integrated parameter either adopts the value-type of one local parameter or defines a value-type different from the local ones. The restricted domain in form of additional pre- and post-conditions is defined according to Table 3.

Property	Definition
(1) Name	Arbitrary unique name.
(2) Activity	Activity which integrates the corresponding local activities.
(3) Type	Object-type parameters: (1) equal to one local parameter or (2) super-type of both. Value-type parameters: (1) equal to one local parameter or (2) different value-type.
(4) States	(1) equal to one local parameter or (2) derived from specialisation funct.
(5) Direction	Same direction as local parameters.
(6) Default value	(1) Equal to one local parameter or (2) not defined.
(7) Read-only	Equal to local parameters.
(8) Domain	(1) Union of local parameters, (2) intersection of local parameters, (3) equal to one local parameter, or (4) no restriction.

Table 3: Definition of integrated parameter properties.

3.3 Instantiation choices

Certain parameter correspondences offer multiple ways to integrate the parameters. An example of each correspondence can be found in (Grossmann et al. 2007).

Identity-related parameters: Identity-related parameters in context of identity-related activities are integrated into a single parameter. The types of the local parameters must be identical and are adopted by the integrated parameter. Parameter binding differs depending on the functionality of the local activities. Activities have either an *updating* functionality, e.g., updating a relationship or a property value, or a *read-only* functionality, e.g., selecting a related object. If the local activities perform an update then both must be executed to keep the states of local objects consistent. This means that the integrated input parameter must be bound to the input parameter of both local activities. On the other side, if the functionality is read-only then only one activity need to be executed and so the integrated parameter is bound to only one local activity. Identity-related input parameters in the context of role-, history-, and counterpart-related activities are also integrated into a single parameter and bound to input parameters of both activities because both might be executed. A transformation function between integrated and local parameter solves conflicts, e.g., type conflicts.

Role-related parameters: Role-related parameters may appear only in role-related activities. The activities might be integrated in different ways where either one or both activities are executed. If only one activity will be executed then the role-related parameters of the chosen activity are adopted unchanged and the parameters are bound directly. If both activities are executed then the role-related parameters are integrated by a single integrated parameter of the same or different type as the local parameters. The integrated parameter is bound to the local parameters with a transformation function which converts value types in case of type difference and aggregates data (*sum*, *max*, or *min*) provided by local parameters.

History-related parameters: History-related parameters may appear only in history-related activities which are always integrated in a sequence. History-related input parameters are adopted unchanged in the integrated activity because the data which they provide model the same real-world object but at different points in time. Therefore both must be provided separately. The adopted parameters are bound directly to the local parameters. History-related output parameters are bound to a single joined output parameter whose value reflects the difference.

Counterpart-related parameters: The counterpart-relationship may only appear between output parameters in counterpart-related activities. A single parameter integrates counterpart-related parameters and provides values which are computed by a function that aggregates the values of the counterpart-related parameters, e.g., the maximum or minimum.

Category-related parameters: Category-related parameters may appear only in category-related activities. These activities are integrated by *dynamic binding* which means that during run-time one local activity (corresponding to one of the local process instances i_1, i_2) is executed. We call the local instances (i_1 or i_2) the *local aspect* of the integrated execution. Before run-time it is unknown if a local activity of i_1 is going to be executed or one of i_2. Therefore, category-related parameters are adopted unchanged and the integrated value depends on the choice of i_1 or i_2. This is achieved creating a guard-condition of the form " `self.aspect=o`" on the data flows that bind the parameters. `self.aspect` returns the local aspect of the integrated instance and `o` is the object type of i_1 or i_2.

Distinct parameters: Distinct parameters in the context of identity ^{I}- and counterpart-related business processes are integrated in the same way as counterpart-related parameters. They are adopted unchanged and bound directly with guard condition `self.aspect=o` as explained before. Distinct parameters in context of identity ^{E}-, role-, history-, and category-related processes are adopted unchanged in the integrated activity and bound directly.

So far, activity specifications of corresponding activities have been integrated which lead to set $\overline{B}^c$ as shown in Figure 1. Distinct activities are adopted unchanged and result in sets $\overline{B}_1^d$ and $\overline{B}_2^d$. The final step of the integration is the combination of integrated activities contained in sets $\overline{B}^c$, $\overline{B}_1^d$, and $\overline{B}_2^d$ to an integrated business process. Two integrated activities are connected with a control flow if output state of the object type is equal to the input state of the other activity. A guard condition is attached on each control flow to check local aspects of the integrated execution. Guards on input and output control flows of distinct activities check for one local aspect and guards on control flows between two activities in $\overline{B}^c$ allow instances of both local aspects to pass. For the special treatment of initial and final nodes see (Grossmann et al. 2007).

4 Related work

The Model Driven Architecture (MDA) has been proposed by the Object Management Group (OMG) to enhance the efficiency and quality of software development (Koehler et al. 2005, Kleppe et al. 2003). MDA separates application logic from the underlying implementation by distinguishing between Platform Independent Models (PIMs) and Platform Specific Models (PSMs). The models are characterised by different levels of abstraction. MDA proposes a development life cycle where in the ideal case three models are created. Starting point is the development of a PIM model. In a next step a PSM model is generated from the PIM model by a transformation tool. The last step generates executable code from a PIM model. Between the models are transformation tools which map models from a higher to a lower abstraction level, if possible, in an automated way.

Our aim is first, to provide an extensible integration framework for business processes on the PIM level and second, to map the integrated business process onto PSM level. The extensibility of the framework is supported by a meta model architecture which enables the development of arbitrary integration patterns forming integration patterns by following specific consistency criteria (Schrefl & Stumptner 2002). Similar to Casati et al. (Casati et al. 2000) who propose a rule- and pattern-based approach for designing business processes, we propose an integration approach based on correspondences and integration patterns. In business process management our approach can be applied for the design of cross-organisational workflows (Schulz & Orlowska 2004, Chebbi et al. 2006). Past research has focused mainly on defining rules how public workflows can be derived from private workflows and has not paid so much attention on the integration of non matching public workflows.

This paper made a step closer to the second aim, the mapping of an integrated business process to PSM level. It investigated the integration of parameters and data flows on the level of activity specifications which belong to a lower abstraction level of business process description. Therefore our approach can create an integrated business process which is closer related to PSMs. The goal to map the result of an integration to specification on the platform specific model is part of future work. Related research has investigated the mapping from PIMs to PSMs where object-oriented models were successfully mapped to executable code (Albert et al. 2006). Alternatively to executable code, it is possible to generate an intermediate model which can be interpreted by an execution engine, e.g., Web service standards like the Business Process Execution Language (BPEL). Web services can be regarded as models on the PSM level. Although platform independent, they rely on the service oriented architecture (SOA) which must be provided. Leymann pointed out the close relation of workflows and Web services and emphasises workflows as an easy to use technology for service composition (Leymann 2006). There exist already approaches that map workflows to Web service execution languages directly (Lassen & van der Aalst 2006).

5 Conclusion and future work

We have presented an extension of earlier work which focuses on the parameter binding of predefined integration patterns. The binding is automated and allows an immediate instantiation of the integration patterns. In some cases, user interaction is necessary in a pre-integration phase to identify correspondences of parameters.

There are some issues which we are going to address in future work. First, we have not considered exceptions in this paper. Exceptions need to be investigated further in context of integration patterns, e.g., an offer might never be returned to the integrated business process. Second, all participants were treated equally. This simplifies the examples but does not reflect many real world scenarios. There might be a dominant participant, e.g., a large company, which has more influence on the partnership than others and which decides on integration patterns. For example, if there is a *value type unit* conflict between two parameters then the dominant participant insists on using its unit.

Furthermore, we are investigating domain independent integration patterns at the moment which can be applied in different domains, e.g., in financial sector as well as in health care sector. Another aspect is the identification of domain properties which are driving the integration. For example, in the financial sector the *counterpart-relationship* might be common for comparing prices whereas in the health care sector the *componentOf* appears more often where the treatment of a patient consists of a series of hospital visits.

References

Albert, M., Munoz, J., Pelechano, V. & Pastor, Ó. (2006), Model to Text Transformation in Practice: Generating Code from Rich Associations Specifications, *in* ‘Proc. of ER Workshops 2006’, LNCS 4231, Springer, pp. 63–72.

Casati, F., Castano, S., Fugini, M., Mirbel, I. & Pernici, B. (2000), ‘Using Patterns to Design Rules in Workflows’, *IEEE TSE* **26**(8), 760–785.

Chebbi, I., Dustdar, S. & Tata, S. (2006), ‘The view-based approach to dynamic inter-organizational workflow cooperation’, *DKE* **56**, 139–173.

Eyal, A. & Milo, T. (2001), ‘Integrating and customizing heterogeneous e-commerce applications’, *The VLDB Journal* **10**, 16–38.

Grossmann, G., Ren, Y., Schrefl, M. & Stumptner, M. (2005), Behavior Based Integration of Composite Business Processes, *in* ‘Proc. of BPM’, LNCS 3649, pp. 186–204.

Grossmann, G., Schrefl, M. & Stumptner, M. (2004), Classification of business process correspondences and associated integration operators, *in* ‘Proc. of eCOMO 2004’, LNCS 3289, pp. 653–666.

Grossmann, G., Schrefl, M. & Stumptner, M. (2007), Exploiting Semantics of Inter-Process Dependencies to Instantiate Predefined Integration Patterns, Technical report, UniSA, ACRC. http://www.mrgg.at/documents/TR-UNISA-CIS-KSE-07-03.pdf.

Kleppe, A., Warmer, J. & Bast, W. (2003), *MDA explained - The Model Driven Architecture: Practice and Promise*, Addison-Wesley.

Koehler, J., Hauser, R., Sendall, S. & Wahler, M. (2005), ‘Declarative techniques for model-driven business process integration’, *IBM Systems Journal* **44**(1), 47–65.

Lassen, K. & van der Aalst, W. (2006), WorkflowNet2BPEL4WS: A Tool for Translating Unstructured Workflow Processes to Readable BPEL, *in* ‘Proc. of OTM 2006’, LNCS 4275, pp. 127–144.

Leymann, F. (2006), Workflow-Based Coordination and Cooperation in a Service World, *in* ‘Proc. of OTM 2006’, LNCS 4275, pp. 2–16.

Maier, D. (1983), *The Theory of Relational Databases*, Computer Science Press.

Rodriguez-Gianolli, P. & Mylopoulos, J. (2001), A Semantic Approach to XML-based Data Integration, *in* ‘Proc. of ER’, pp. 117–132.

Schrefl, M. & Stumptner, M. (2002), ‘Behavior-consistent Specialization of Object Life Cycles’, *ACM TOSEM* **11**(1), 92–148.

Schulz, K. A. & Orlowska, M. E. (2004), ‘Facilitating Cross-Organisational Workflows with a Workflow View Approach’, *DKE* **51**(1), 109–147.