Verification of model transformations: A case study with BPEL*

Luciano Baresi[1], Karsten Ehrig[2], and Reiko Heckel[2]

[1] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
baresi@elet.polimi.it
[2] Department of Computer Science, University of Leicester, United Kingdom
{karsten,reiko}@mcs.le.ac.uk

Abstract. Model transformations, like refinement or refactoring, have to respect the semantics of the models transformed. In the case of behavioural models this semantics can be specified by transformations, too, describing an abstract interpreter for the language. Both kinds of transformations, if given in a rule-based way, can formally be described as graph transformations.
In this paper, we present executable business processes, their operational semantics and refactoring, as an example of this fact. Using results from graph transformation theory about critical pairs and local confluence, we show that our transformations preserve the semantics of processes. The analysis is performed by means of the graph transformation tool AGG.

1 Introduction

Transformations of models are the key technology of Model-driven Development (MDD), an approach to software development where graphical models (rather than programs) are the focus and primary technical artefact. Model transformations can serve a variety of purposes, including the refinements of models, their mapping to implementations, consistency management, or evolution. In many of these examples, a semantic compatibility between the artefacts before and after the transformation is desired.

With the semiformal nature of most visual models, and the corresponding lack of formal semantics, this relation is often not easy to describe. And if formal semantics exist for both source and target models, they are often given in different semantic domains and using different formal techniques.

It seems that the only general solution to this problem consists in adopting a formalism powerful enough to describe both the intended semantics and the transformation of the models involved, and which provides techniques and tools to demonstrate the relation between them. Since we are dealing with visual

* Work supported in part by the IST-2005-16004 Integrated Project SENSORIA: Software Engineering for Service-Oriented Overlay Computers and by the European Community's Human Potential Programme under contract HPRN-CT-2002-00275, [SegraVis].

models whose abstract syntax is often expressed by means of graphs, we are opting for graph transformation as one such approach.

In this paper we are demonstrating the idea by means of the transformation of executable business processes inspired by BPEL4WS, the Business Process Execution Language for Web Services [13], and presented using the notation of UML activity diagrams. Normally BPEL assumes a centralised approach, where a single entity—usually called *orchestrator*— controls the execution flow and coordinates the interactions with selected services. Centralised execution is easy to describe, but not always adequate if the system is distributed by nature, for example, in the case of inter-organisational business processes, where each party is in charge of a particular fragment of the process.

With this motivation, Baresi et al [3, 4] use transformations to partition a monolithic process into a coordinated set of sub-processes. However, although the distribution is described formally by means of graph transformation rules, no verification of its semantic correctness is given. The present paper addresses this issue and discusses the conceptual and software tools required to

- formalise a notion of semantic compatibility between the distributed and the original process, taking into account that not all features of one may exist in the other;
- verify that this compatibility holds for all processes obtained through application of the proposed transformation rules.

The first aim is achieved using typed attributed graph transformation rules to define the operational semantics of centralised and distributed processes. To this end, a meta model captures the abstract syntax of executable processes, extended by information about their execution state. This meta model, formally represented as a type graph with inheritance, is the basis of a set of operational semantics rules. This approach allows us to specify the operational semantics and transformations in the same formalism, thus simplifying the analysis. The semantic relation is established by associating observations with operational rules, generating the labelled transition systems of processes, and applying the standard notion of bisimilarity modulo a suitable projection of transitions onto common labels.

The proof that the semantic relation holds for all processes obtained from each other by application of transformation rules realising the distribution is based on the idea of *mixed confluence* [8]: If transformation rules are exchangeable with operational rules, the application of a transformation does not reduce the operational semantics. Mixed confluence is shown by critical pair analysis [9] of model transformation vs. semantic rules. This allows us to establish a simulation relation between the given and the derived process. Since graph transformation rules are invertible, the same technique can be used to show bisimilarity.

The paper is organized as follows. After introducing in Section 2 those features of the BPEL language that are relevant to our model, Section 3 describes the BPEL meta model and the transformation from centralised into distributed

processes. Section 4 sketches the operational semantics and Section 5 discusses the correctness of the transformation. Section 7 concludes the paper.

2 Executable Business Processes as Activity Diagrams

BPEL4WS (Business Process Execution Language for Web Services, [13]) based on WSDL (Web Services Description Language, [6]) is an XML-based language designed to enable the coordination and composition —using a workflow-based approach— of a set of Web services. BPEL defines a number of activities to describe the interaction of the process with its partners. Activities can be *basic* or *structured*. Basic activities define the interaction capabilities of BPEL processes: **invoke** activities call operations of external services and (in their synchronous version) wait for the response message to arrive, **receive** activities wait for suitable messages (invocations), **reply** activities answer invocations, etc.

Structured activities can comprise both basic and other structured activities. Examples include **switch** defining branches, **flow** allowing for parallel threads, and **pick** defining branches whose selection is based on the receipt of suitable messages.

In this paper, we consider a subset of the language, ignoring some of the control constructs and features such as asynchronous communication, fault handlers, etc., see [13] for an in-depth presentation. We illustrate the approach by means of the example in Fig. 1, using UML activity diagrams to visualise a process that manages orders received from clients in cooperation between an office and a warehouse. We assume that the **Office** receives the order through a **receive** activity and implicitly validates it. If it is acceptable, the **Warehouse invoke**s shipment, otherwise the **Office** proceeds with a **basic** (local) undo operation. The two orchestrators are distinguished by so-called swimlanes.

3 Transformations

This section demonstrates the use of transformation rules to partition a process into a main process and a set of independent sub-processes. The presentation starts from a simple meta-model introducing the types of nodes and edges required to represent processes. Then, we introduce a sample transformation rule.

The presentation is based on the algebraic double-pushout approach to the transformation of typed and attributed graphs [9]. For a more compact presentation of metamodel and rules we use subtyping as well as negative application conditions.

3.1 Meta-model

The meta-model of Fig. 2, which borrows some concepts from the work proposed in [11], only comprises elements that are used by the example in this paper.

A business process comprises `Elements`, which are distinguished into `Nodes` and `Edges`, linked by means of associations source (`src`) and target (`tar`). Moreover, `Nodes` are further classified into

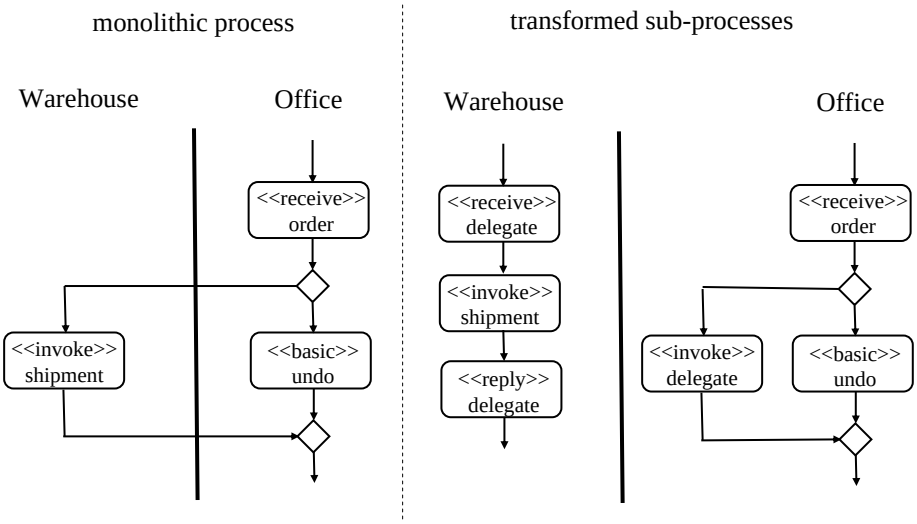

Fig. 1. Example BPEL Processes

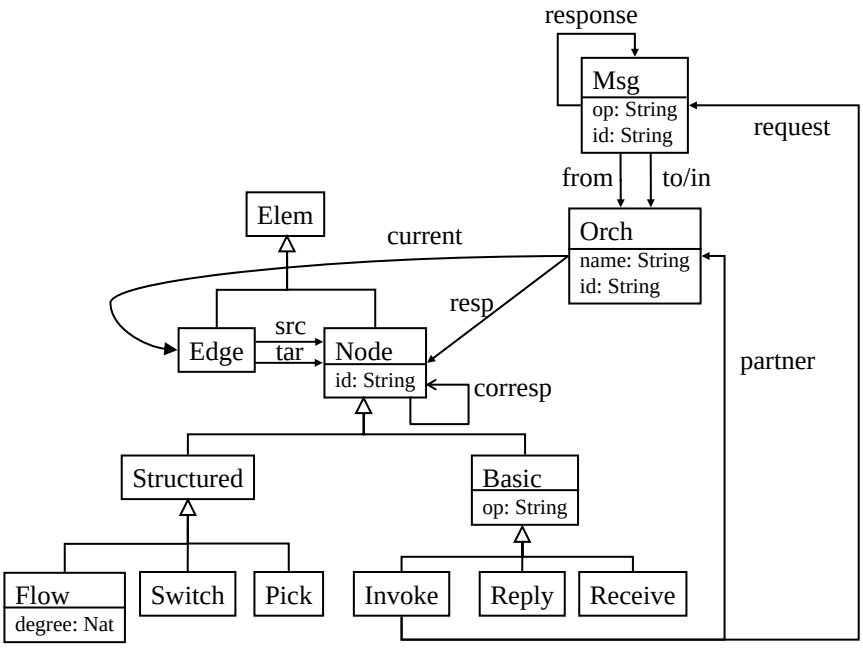

Fig. 2. Our BPEL metamodel

– `Basic` nodes corresponding to `Invoke`, `Reply`, and `Receive` activities [3].

[3] In this paper, we do not consider `Assign` activities since the propagation of data values is not addressed here.

– `Structured` nodes corresponding to the typical constructs of workflow languages, like `Switch`, `Flow`, and `Pick`.

Each structured activity is presented by two `Structured` nodes —related by association `corresponding`— to identify the start and the end of the composed activity. The same association is also used to relate the first and last nodes of a sequence of basic action nodes.

Each `Node` is characterised by the `Orch`estrator that is `resp`onsible for it (that shall execute it). Before starting the application of partitioning rules, the designer must decide how to split the process by assigning the responsibilities for the different nodes to available orchestrators.

Each `Orch`estrator has an element (currently) under execution, which is rendered using association `current` in Fig. 2. By `partner` links we point to `Orch`estrator intended as recipients of `invoke` messages. Messages (`Msg`) specify their sender (`from`), recipient (`to`), whether they are already received but not yet fully processed (`in`), or if they are the (`response`) to an earlier invocation message.

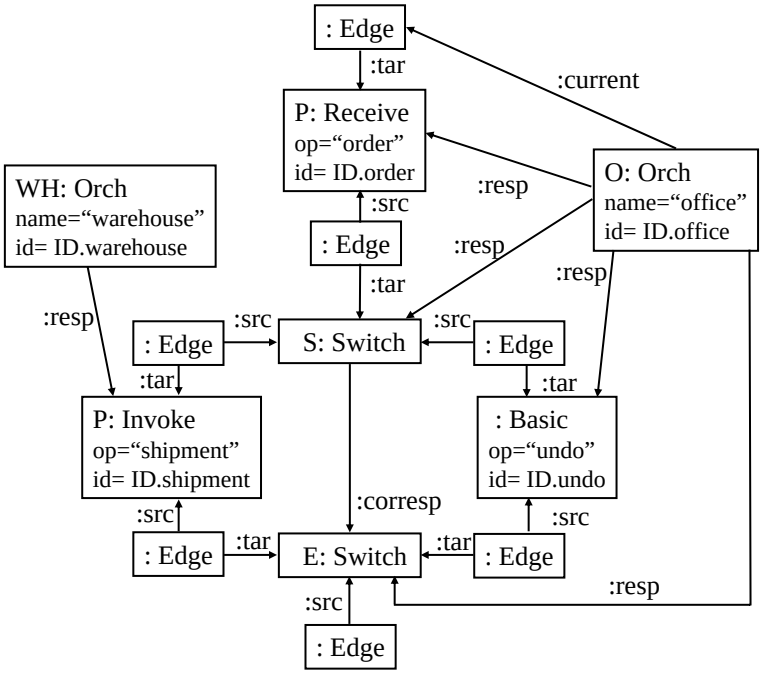

Fig. 3. The example of Fig. 1 as an instance of the metamodel of Fig. 2

The meta-model of Fig. 2 is incomplete, missing both a number of constraints and further types, but sufficient to represent the example process in Fig. 1. The corresponding metamodel instance is shown in Fig. 3.

3.2 The partitioning rule

The partitioning identifies parts of the original process which can be externalised and redirects the execution flow accordingly. Consider the rule in Fig. 4 delegating the execution of a block of activities that are in the domain of a different orchestrator. The part to be delegated is situated between `1:Node` and `2:Node` in the left-hand side of the rule. The `corresp` edge, removed in the transformation, represents the derived information that these two nodes are indeed connected by a workflow. It has to be set accordingly before executing the rule.

In our example model in Fig. 3 the part to be delegated consists of a single activity `P:invoke` only. In this case both `1:Node` and `2:Node` are mapped to `P:invoke`.

The delegation subprocess is started by the first orchestrator `8:Orch` via `:Invoke`. The second orchestrator `3:Orch` executes the process between the pair of `:Receive` and `:Reply` nodes, activated by the new `:current` edge. The negative application condition `NAC` ensures that the subprocess to be delegated is not already under execution by `3:Orch`. Since it is invoked synchronously, the invoking process has to wait for a reply of the executed subprocess.

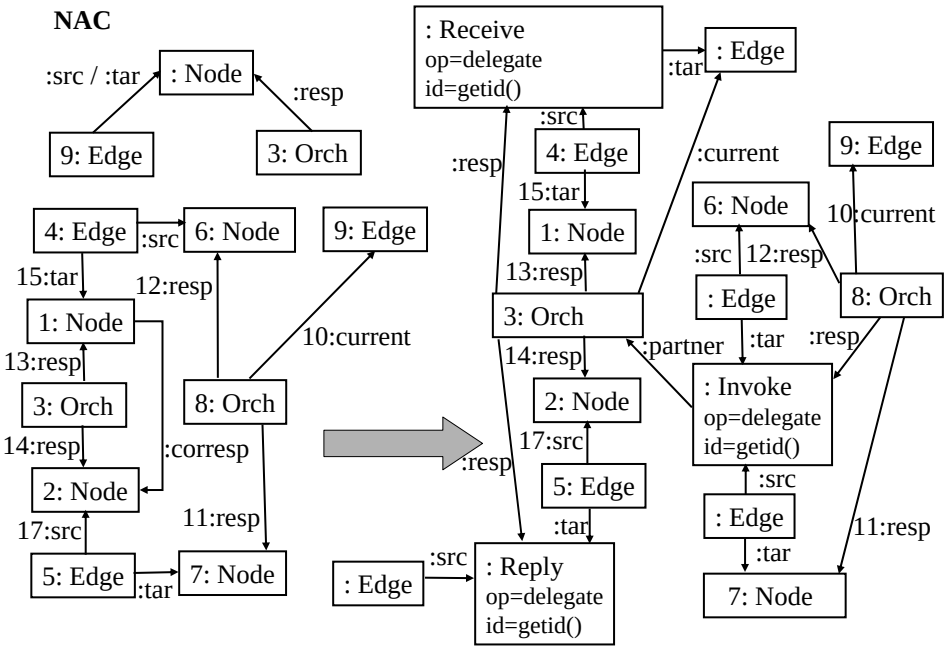

Fig. 4. Transformation rule `delegate`

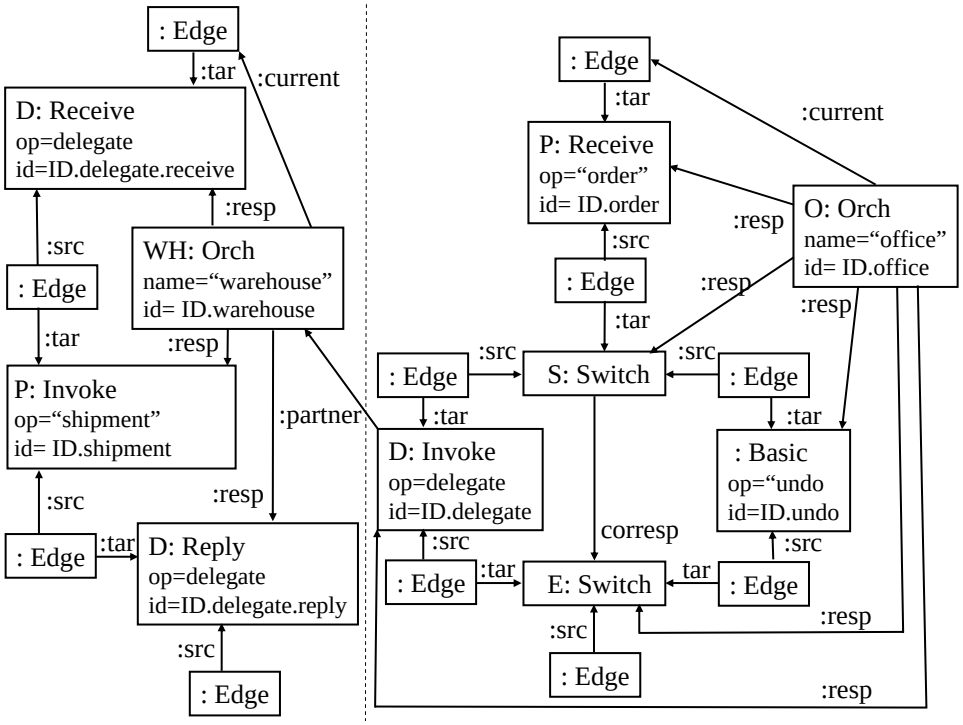

Fig. 5. Resulting subprocesses

4 Operational Semantics

Graphical operational semantics has been introduced as an extension of meta modelling (the specification of abstract syntax and/or static semantic by means of class diagrams) to deal with the dynamic aspect of modelling languages [10]. In this section we are using the approach to model the operational semantics of executable business processes, based on their graphical representation from their previous section. We focus on the rules needed for explaining the behaviour of the example (represented by the meta model instance in Fig. 1), briefly sketching the remaining rules. Then we define observations and derive a labelled transition system.

4.1 Operational Rules

Fig. 6 shows the operational rule for executing the *1:Invoke* action, sending message `:Msg`. The operation mentioned by the message is the same of the action node i, as described by the condition `op = i.op` in the `:Msg` node. A new unique identifier is supplied by `getid()`. As specified be the `partner` edge from the invoke node, the message is created by the orchestrator `3:Orch` for the orchestrator `4:Orch`.

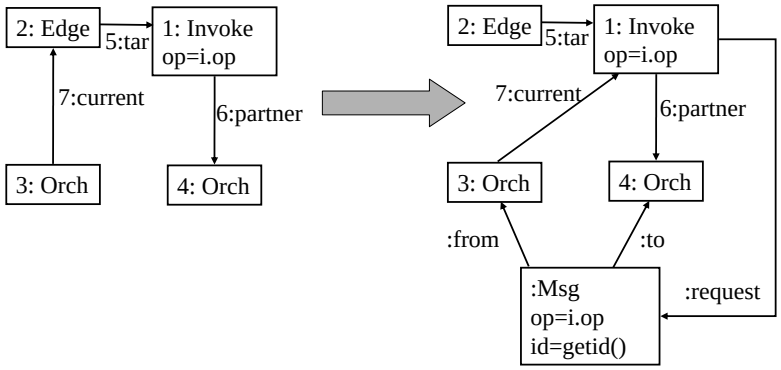

Fig. 6. Operational rule `invoke`

Fig. 7 shows the rule by which orchestrator `3:Orch` executes the *1:Receive* action, accepting a message `4:Msg` with the operation name `r.op` of `1:Receive`. The `current` edge, previously pointing to the Edge before the next action Node is advanced from `2:Edge` to `5:Edge`.

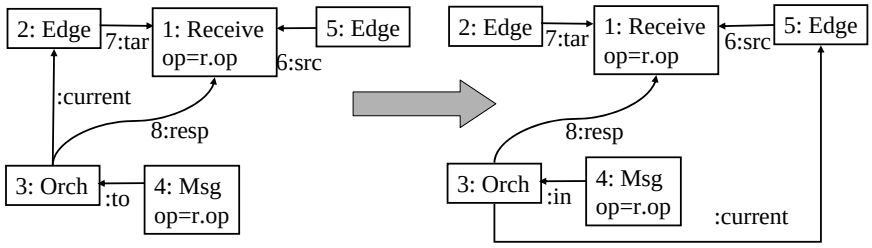

Fig. 7. Operational rule `receive`

Fig. 8 shows the rule by which orchestrator `3:Orch` is replying to message `4:Msg` with the new message `:Msg` which is sent to the invoking orchestrator `10:Orch`.

Fig. 9 shows the rule by which orchestrator `5:Orch` is handling the response of orchestrator `6:Orch` by deleting the request and response messages and advancing the `:current` edge.

The rule `switch` in Fig. 10 represent an example of how the semantics of control structures are described. The rule implements both the split operation, moving the *:current* edge to one of several branches outgoing from the *1:Switch* node, and the joining of several alternative branches. Notice that split is non-deterministic because, due to the lack of data types in our model, we do not specify any guards. In practice we can expect that switches are deterministic.

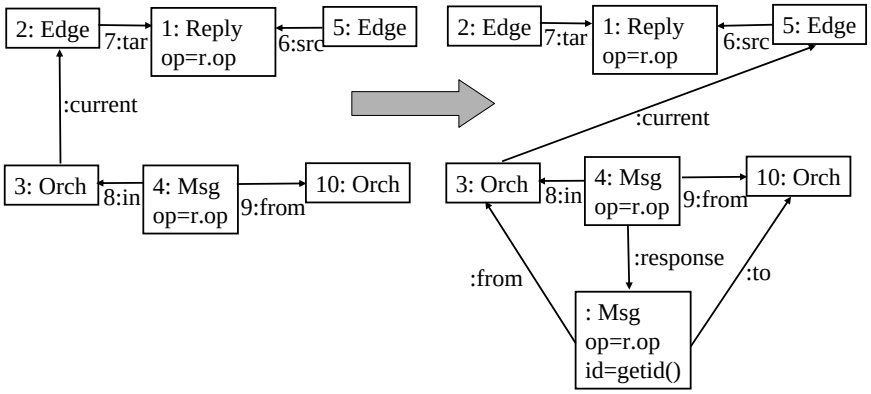

Fig. 8. Operational rule `reply`

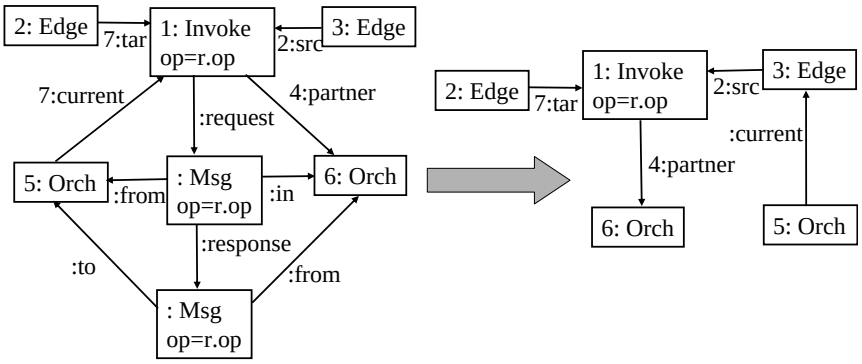

Fig. 9. Operational rule `response`

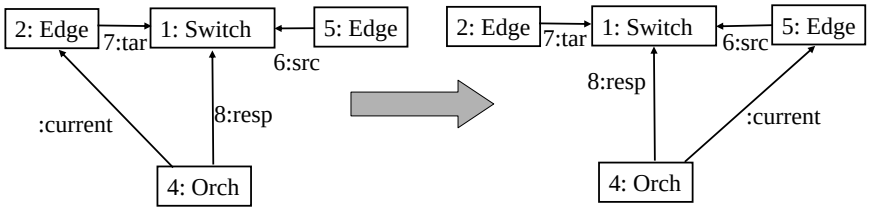

Fig. 10. Operational rule `switch`

The rule `partner` in Fig. 11 selects a partner orchestrator in a non-deterministic way. This is an under-specification of a potentially complex protocol for selecting services. Like the non-deterministic switch rule this will lead to extra transitions, which do not conflict with our aim of demonstrating the preservation of the operational semantics.

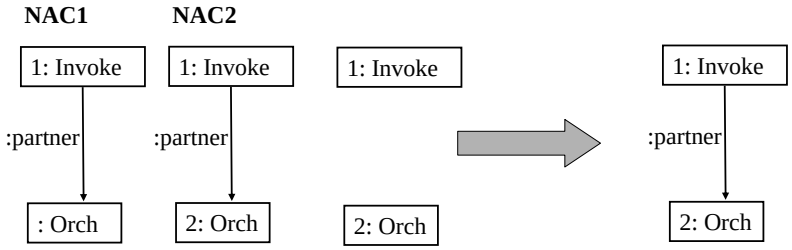

Fig. 11. Operational rule `partner`

Finally the rule `reinit` in Fig. 12 sets the `:current` edge from the end to the beginning after a subprocess has completed its execution to allow a new execution in another context.

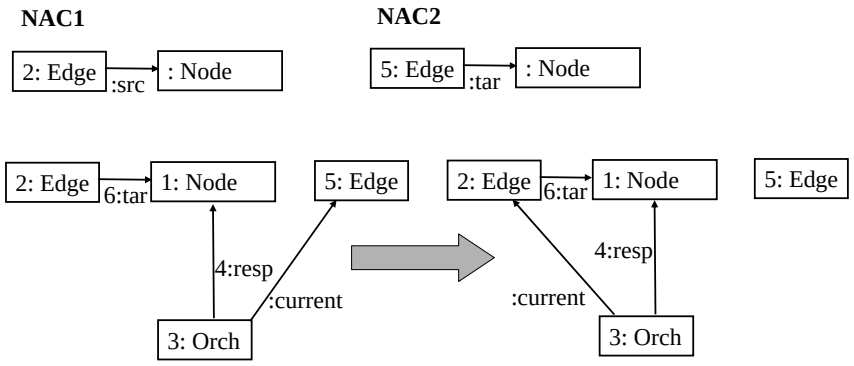

Fig. 12. Operational rule `reinit`

Additional operational rules considered in the complete version of the model include

- flow (fork and sync): the creating and synchronisation of concurrent flows of control;
- pick (split and join): like guarded/external choice in process calculi, where the incoming message determines which of a number of alternative paths is chosen;
- init and final: dealing with the start and termination of processes;

4.2 Labels and Transition System

Observations on rules define the labels of the transition system representing the operational semantics of processes. They contain the name of the rule and list

the `id` attributes of some key elements. For example, `inv(i.id, m.id)` refers to the application of the rule `invoke` and observes the identity of the `invoke` action i executed and the message m created. Similarly, the remaining labels include

- `rec(r.id, m.id)` performing receive action r on message m;
- `reply(r.id, m1.id, m2.id)` replying to $m1$ with $m2$;
- `resp(i.id, m2.id)` receiving response message $m2$ for invocation i;

Control flow rules like `switch` represent internal steps and are uniformly labelled with the silent action τ.

In transformations, formal parameters from the rules are replaced by identities of the actual nodes in the graphs representing system states. Given a transformation $G \stackrel{p(o)}{\Longrightarrow} H$ with, for example rule $p =$ `invoke` (see Fig. 6) and match o, the label `inv(i.id, m.id)` produces the observation `inv(`o`(i).id,` o`(m).id)` using the values of the `id` attributes for the images of nodes `i` and `m` under occurrence o.

Given a graph transformation system $\mathcal{G} = (TG, P)$ with start graph G_0, we derive a labelled transition system $LTS(\mathcal{G}, G_0) = (S, L, \rightarrow)$ with all graphs reachable from G_0 by applications of rules as states S, observations on rules as labels L, and transformations as transitions.

The set O of observations excluding τ is potentially infinite. Those that can actually be produced by a transformation system $\mathcal{G}$ from a given graph G are limited to such expressions `rule(params)` where `params` is a list of `id` attribute values that already occur in G. We call this the *alphabet* $\mathrm{alph}(G)$ of graph G.

5 Verification

Based on the definitions in the previous section we are now able to define a relation of semantic compatibility between processes. The idea is to require weak bisimilarity after hiding all labels that are not in the intersection of the alphabets of the two processes. On a labelled transition system, the operation of hiding replaces all occurrences of a given label by a silent action τ.

Definition 1 (semantic compatibility). *Given a graph transformation system $\mathcal{G}_{OP} = \langle TG, OP \rangle$ (specifying operational semantics), two graphs G_1 and G_2 (representing processes) are* semantically compatible *if they are weakly bisimilar after hiding all labels not in $alph(G_1) \cap alph(G_2)$.*

In our case O_1, the set of observations produced by a centralised process, will be a subset of O_2, the observations of the distributed process. This is because of the additional communication actions required to coordinate the different local processes. The following theorem establishes a condition for semantic compatibility. For a regular expression r, by $s \xrightarrow{r} t$ we denote a sequence of transitions $s \xrightarrow{l_1} \cdots \xrightarrow{l_n} t$ such that $l_1 \ldots l_n$ is in the language described by r.

Theorem 1 (semantic compatibility of transformations). *Assume graph transformation systems $\mathcal{G}_{OP} = \langle TG, OP \rangle$ (operational semantics) and $\mathcal{G}_T = \langle TG, T \rangle$ (model transformations) such that for all operational steps $P_1 \xrightarrow{l} Q_1$ and transformation steps $P_1 \Longrightarrow P_2$*

1. *$l \in alph(P_2)$ implies that there exist $P_2 \xrightarrow{\tau^* l \tau^*} Q_2$ and $Q_1 \Longrightarrow Q_2$;*
2. *$l \notin alph(P_2)$ implies that there exist $P_2 \xrightarrow{\tau^*} Q_2$ and $Q_1 \Longrightarrow Q_2$;*

and the same is true for the inverse of $\mathcal{G}_T$, obtained by reversing all productions.

$$\begin{array}{ccc} P_1 & \longrightarrow & Q_1 \\ \Downarrow & & \Downarrow \\ P_2 & \longrightarrow & Q_2 \end{array}$$

Then, whenever there exists a transformation $G_1 \Longrightarrow G_2$ in $\mathcal{G}_T$, typed graphs G_1 and G_2 are semantically compatible.

Proof. Recall that a weak bisimulation is a relation R on states such that P_1 R P_2 implies

1. $P_1 \xrightarrow{l} Q_1$ implies $P_2 \xrightarrow{\tau^* l \tau^*} Q_2$ and Q_1 R Q_2
2. $P_1 \xrightarrow{\tau} Q_1$ implies $P_2 \xrightarrow{\tau^*} Q_2$ and Q_1 R Q_2

while the same is true for the inverse R^{-1}. The relation R on TG-typed graphs is defined by G_1 R G_2 iff $G_1 \Longrightarrow G_2$. It is easy to see that this satisfies the properties 1 and 2 above.

The inverse transformation system produces the inverse of the transformation relation R, thus making the above true for the symmetric closure of the relation.

Notice that Theorem 1 is based on a notion of local confluence, restricted by assumptions on the labels of derived transitions. Such a property, called *mixed confluence* in [8], can be verified statically by critical pair analysis and search: First we check if all pairs consisting of a model transformation rule and a semantic rule are parallel independent, i.e. there are no critical pairs between them. If this fails, we have to demonstrate confluence for all critical pairs, searching for compatible transformation sequences with the right labels that lead to a common successor state.

For our case study, this means that we have to analyse the critical pairs between `delegate` and the operational semantic rules: Critical pair analysis is supported by the attributed graph grammar tool environment AGG [22, 23]. Critical pairs formalise the idea of a minimal example of a conflicting situation. From the set of all overlapping graphs the objects and links are extracted which cause conflicts or dependencies.

Fig. 13 shows the critical pair analysis in AGG where the model transformation rule `delegate` is indeed independent of all operational semantics rules (last row). The opposite is not true since we have critical pairs in the last column

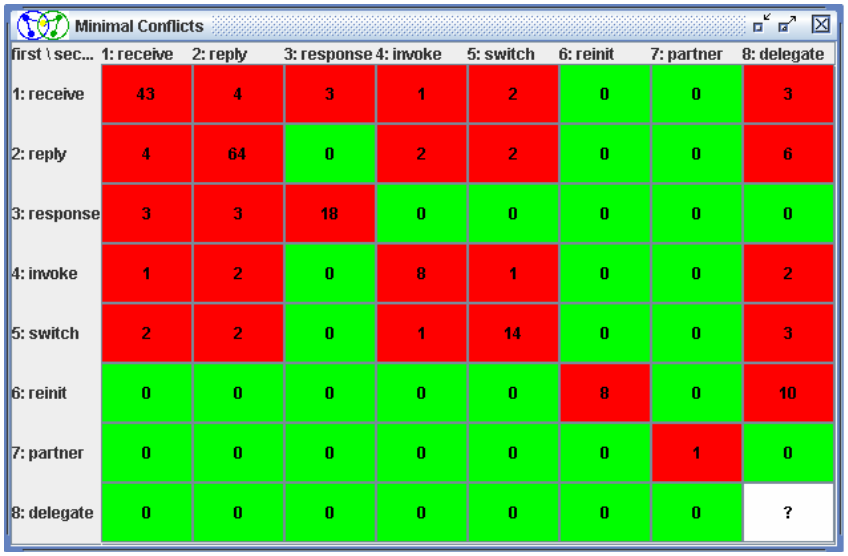

Fig. 13. Critical Pair Analysis in AGG

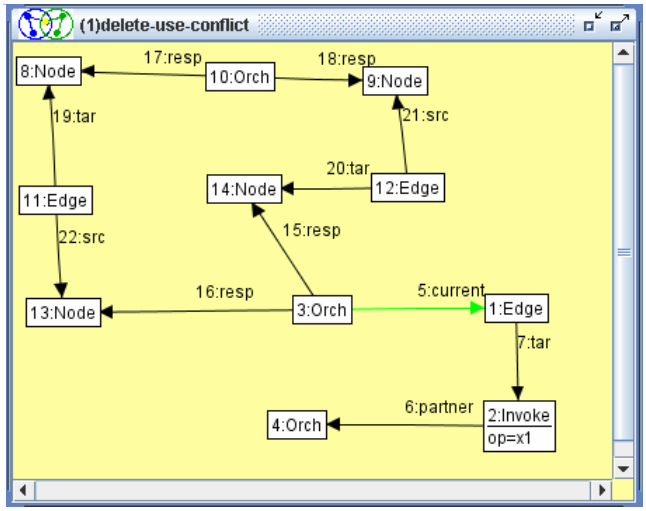

Fig. 14. Critical Pair between `invoke` and `delegate`

which are all caused by a delete-use-conflict between `10:current` in `delegate` and the semantic rules.

According to [17] two direct graph transformations $G \overset{(p_1,m_1)}{\Longrightarrow} H_1$ and $G \overset{(p_2,m_2)}{\Longrightarrow} H_2$ are in delete-use-conflict (resp. use-delete-conflict) if rule p_1 (resp. p_2) deletes part of the graph G, which is used by rule p_2 (resp. p_2) in the second (resp. first) direct transformation. Fig. 14 shows one of these critical pairs between `invoke`

and `delegate`. (Note that the critical pairs between operational semantics rules, or of rule delegate with itself, are not relevant for mixed confluence.)

Intuitively, the problem is that rule `delegate` carries an application condition to check that the subprocess to be separated from the main one is not currently active. Therefore it is obvious that this should be in conflict with semantic rules advancing the control flow of the process, thus potentially entering the subprocess.

The solution to this problem consists in providing additional transformation rules to deal with the delegation in the case that the subprocess is indeed active. But for its application condition, this rule coincides with `delegate`, with the additional effect that a request message would be created to represent the fact that the (then delegated) subprocess has been invoked from the main one by a message, rather than just by advancing the control flow.

Such an extension of the transformation system to deal with non-confluent cases can be seen as a variant of Knuth-Bendix completion for non-confluent critical pairs in term rewriting. With this extension, the resulting system enjoys the mixed-confluence property.

As a result we obtain mixed confluence, consequently, the compatibility of distributed process with the centralised ones they have been obtained from using rule `delegate`.

6 Related work

The problem of workflow partitioning has been studied in the field of business process design for some ten years. It still offers interesting issues to study because of mobile information systems and web services, and the novel problems that come with them. In [15], the authors present a comparison among the different approaches to workflow distribution.

Cross-Flow [12] aims at providing high-level support to workflows in dynamically-created virtual organisations. High-level support is obtained by abstracting services and offering advanced cooperation support. Virtual organisations are created dynamically by contract-based match-making between service providers and consumers. Agent Enhanced Workflows [16] adopt the interesting approach, inspired by agent-oriented systems, of building execution plans from predefined goals. Event-based Workflow Process Management [7] use an event-based infrastructure and support modelling constructs for addressing the timing issues of process management. The main feature of ADEPT [20] is the possibility of modifying workflow instances at run-time. MENTOR [19] provides an autonomous workflow engine. In this approach the workflow management system is based on a client-server architecture. The workflow itself is orchestrated by appropriately configured servers, while the applications that invoke workflow activities are executed on the client sites. The METEOR (Managing End to End OpeRations) [2] system leverages Java, CORBA, and Web technologies to provide support for the development of enterprise applications that require workflow management and application integration. It enables the development of complex workflow appli-

cations which involve legacy information systems and that have geographically distributed and heterogeneous hardware and software environments, spanning multiple organisations. It also provides support for dynamic workflow processes, error and exception handling, recovery, and QoS management. Exotica [18] is characterised by the possibility of disconnected operations. It does not permit complete decentralisation because it maintains a central unit and all operations obey a client/server paradigm. WISE [1] exploits the Web for its engine and offers an embedded fault handler. WAWM [21] focuses on the problems related to the workflow management in wide area networks. Mobile [14] is developed to support inter-organisational workflows and is strongly based on modularity. This characteristic alleviates change management and also allows users to customise and extend aspects individually.

The analysis of presented models suggests two different and dual approaches to the problem of workflow coordination. The first approach supports the integration of autonomous and preexisting workflows and it aims mainly at the coordination of different and independent actors. The second approach supports the decomposition of single workflows to support their autonomous execution by means of different engines. Cross-Flow, Agent Enhanced Workflow, Event-based Workflow process Management, Adept, WISE and WAWM belong to the first approach; Mentor, Exotica and Mobile belong to the second one.

The systems described offer three different solutions for the definition of partitioning and allocation rules. The first solution proposes specific definition languages (Cross-Flow, Agent enhanced workflow, Mentor, Exotica). The second approach proposes the extension of workflow languages with distribution rules (Cross-Flow, ADEPT, WISE, WAWM, Mobile). The third approach does not consider the language for distribution rules (Event-based, Workflow Process Management). Cross-Flow belongs to more than one class because the distribution rules are split into several definition parts.

Our delegation model supports disconnected components like Exotica, the independence of workflow engines like MENTOR, and the possibility of modifying the workflow instance at run-time like ADEPT. Moreover, we argue that the mobile environment needs a language strongly oriented to the automatic execution like BPEL, but we do not forget the need for lightness that is a mandatory feature if the system runs on portable devices in ad-hoc networks. As far as the definition of rules is concerned, our approach defines partitioning rules, but does not define allocation rules. It defers them to the specific business process and application domain.

Many of the cited approaches do not consider web services as available instruments for decentralising business processes. An exception is made by [5], which presents an approach very similar to ours. The authors use BPEL as workflow model and use the term *Composite Web Service* to refer to a standard workflow. However, they focus on the problem of assigning workflow portions to specific orchestrators to minimise the traffic among nodes, but they do not provide any specific information about the partitioning rules and/or any proof of their validity. They introduce the concepts of *Control Flow Graph* and *Program*

Dependence Graph without providing how they refer to BPEL constructs and activities.

7 Conclusion

Reporting on an application of the *mixed confluence* method to show semantic correctness of a transformation from centralised to distributed BPEL processes, the motivation of this paper was two-fold. First, the correctness of the transformation is a practical problem which arose independently and whose solution is potentially relevant to the acceptance of the idea of distributed processes for web services. Second, the problem represents an interesting case for the method of mixed confluence, whose feasibility was validated in the process.

It turns out that, specifying the operational semantics required a non-trivial refinement (and we like to believe improvement) of the meta model over the originally proposed one, which was only used to describe the transformation. Also, the importance of efficient tool support became evident, in particular with respect to the scalability to non-trivial examples.

As future work we intend to consider more transformations consisting of several steps as part of a transaction, which will allow us to consider more complex transformation scenarios.

References

1. G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to business e-commerce. In *RIDE*, pages 132–139, 1999.
2. K. Anyanwu, A. Sheth, J. Cardoso, J. Miller, and K. Kochut. Healthcare enterprise process development and integration. *Journal of Research and Practice in Information Technology*, 35(2), 2003.
3. L. Baresi, A. Maurino, and S. Modafferi. Workflow partitioning in mobile information systems. In Kluwer, editor, *In Proc. of IFIP TC8 Working Conference on Mobile Information Systems*, volume 158 of *IFIP International Federation for Information Processing*, 2004.
4. Luciano Baresi, Andrea Maurino, and Stefano Modafferi. Partitioning rules for bpel processes. Technical report, Politecnico di Milano, 2006. In preparation.
5. G.B. Chafle, S. Chandra, V. Mann, and M.G. Nanda. Decentralized orchestration of composite web services. In *In Proc. of the Int. World Wide Web conference on Alternate track papers & posters*, pages 134–143, New York, NY, USA, 2004. ACM Press.
6. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) version 1.1.* W3C, March 2001. http://www.w3.org/TR/wsdl.
7. J. Eder and E. Panagos. Towards distributed workflow process management. In *In proc. of Workshop on cross-Organizational Workflow Management and Coordination*, San Francisco, USA, 1999.
8. H. Ehrig and K. Ehrig. Overview of Formal Concepts for Model Transformations based on Typed Attributed Graph Transformation. In *Proc. International Workshop on Graph and Model Transformation (GraMoT'05)*, Electronic Notes

in Theoretical Computer Science volume 152, Tallinn, Estonia, September 2005. Elsevier Science.
9. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.
10. G. Engels, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to operational semantics. In *Proc. OOPSLA'99 Workshop on Rigorous Modeling and Analysis with the UML: Challenges and Limitations, Denver, CO, USA*, November 2 1999.
11. T. Gardner and al. Draft UML 1.4 profile for automated business processes with a mapping to the BPEL 1.0. IBM alphaWorks, 2003.
12. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering*, 15(5):277–290, 2000.
13. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. *Business Process Execution Language for Web Services version 1.1*, May 2003. http://www.ibm.com/developerworks/library/ws-bpel/.
14. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson, 1996.
15. S. Jablonski, R. Schamburger, C. Hahn, S. Horn, R. Lay, J. Neeb, and M. Schlundt. A comprehensive investigation of distribution in the context of workflow management. In *In proc. of International Conference on Parallel and Distributed Systems ICPADS*, Kyongju City, Korea, 2001.
16. D. Judge, B. Odgers, J. Shepherdson, and Z. Cui. Agent enhanced workflow. *BT Technical Journal*, (16), 1998.
17. Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Conflict Detection for Graph Transformation with Negative Application Conditions. In *A. Corradini et al. (Eds.): Proceedings of the Third International Conference on Graph Transformation (ICGT 2006)*, volume 4178 of *Lecture Notes in Computer Science*, pages 61–76. Springer-Verlag, 2006.
18. C. Mohan, G. Alonso, R. Gunthor, and M. Kamath. Exotica: A research perspective of workflow management systems. *Data Engineering Bulletin*, 18(1):19–26, 1995.
19. P. Muth, D. Wodtke, J. Weisenfels, A. Kotz Dittrich, and G. Weikum. From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, 10(2):159–184, 1998.
20. M. Reichert and P. Dadam. Adeptflex − supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
21. G. Riempp. *Wide Area Workflow Management*. Springer, London, UK, 1998.
22. G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of So ftware. In J. Pfaltz, M. Nagl, and B. Boehlen, editors, *Application of Graph Transformations with Industrial Relevance (AGTIVE'03)*, volume 3062 of *Lecture Notes in Computer Science*, pages 446 – 456. Springer, 2004.
23. Technical University of Berlin, Department of Computer Science. *AGG Version 1.4.1 - http://tfs.cs.tu-berlin.de/agg*, 2006.