Electronic Communications of the EASST
Volume 3 (2006)

Proceedings of the Third Workshop on Software Evolution through Transformations: Embracing the Change (SeTra 2006)

Towards Distributed BPEL Orchestrations

Luciano Baresi , Andrea Maurino and Stefano Modafferi

14 pages

Guest Editors: Jean-Marie Favre, Reiko Heckel, Tom Mens
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Distributed BPEL Orchestrations

Luciano Baresi [1], **Andrea Maurino** [2] **and Stefano Modafferi** [1]

[1] Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32, I-20133 Milano, Italy

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano - Bicocca
Via Bicocca degli Arcimboldi 8, I-20126 - Milano, Italy

Abstract: Web services are imposing as the technology to integrate highly heterogeneous systems. BPEL, the standard technology to compose services, assumes a single ”orchestrator” that controls the execution flow and coordinates the interactions with selected services. This centralized approach simplifies the coordination among components, but it is also a too heavy constraint. To this end, the paper introduces the idea of *distributed orchestrations* and presents a proposal to couple BPEL and distributed execution in mobile settings. The approach —exemplified on a simple case study— transforms a centralized BPEL process into a set of coordinated processes. An explicit meta-model and graph transformation supply the formal grounding to obtain a set of related processes, and to add the communication infrastructure among the newly created processes. The paper also presents a communication infrastructure based on tuple spaces to make the different orchestrators interact in mobile contexts.

Keywords: WS-BPEL, Graph transformation, Distributed system

1 Introduction

Web services are a well-known technology to implement and deploy *service-oriented* systems. Their flexibility is playing a key role to integrate highly heterogeneous systems. Specific technologies are hidden behind simple XML interfaces written in WSDL (Web Services Description Language [CCMW01]) and services interact by means of SOAP (Simple Object Access Protocol) messages —another XML-based technology. The same ease and freedom do not apply to the paradigms with which services can be composed to support business processes. WS-BPEL (Web Services Business Process Execution Language [BEA03]), the most prominent standard technology to compose Web services, only allows for a ”strict” centralized approach to composition, where a single *orchestrator* controls the execution flow and coordinates the interactions with selected services.

This centralized approach to execution helps simplify the coordination among components, but it is also a too heavy constraint for those systems that have a distributed nature and imply the cooperation of decentralized (maybe nomadic) actors. The Web services community is already pushing more decentralized approaches to composition. Choreographies (WS-CDL, Web Ser-

vices Choreography Description Language [Nic04]) —instead of orchestrations— support peer-to-peer interactions among services, and event-based architectures, with WS-Notification [Aka04] as one of the emerging standards in this direction, support even looser cooperations among Web services, where the interaction is limited to reacting to events generated within the composition. Unfortunately, available technology is still limited and it is not as widely accepted as WS-BPEL.

Even if we stick to a workflow-like view of the Web services composition, in many cases, the centralized process needs to be deployed in a distributed setting. The ”conceptual” monolithic process must be partitioned into a well-organized set of sub-processes, and the implied coordination and synchronization actions among the parts must be suitably designed and implemented. This is the case, for example, of cross-departments business processes, where each division is in charge of a particular fragment of the process, and they all concur to the success of the global process. Nowadays, we can also envisage another, completely different, scenario with the centralized model split because it has to be executed by a set of federated mobile devices. No single device has the capability of executing the whole process, but they all can contribute with the execution of a dedicated portion. The scenario might become even bolder if we thought of moving the partitioning process at runtime to dynamically select and allocate the pieces of a process to the devices that are available, and have enough resources, at a given moment. Nomadic users are supposed to move with their devices, and thus the set of cooperating devices can change. This scenario could be useful for disaster recovery situations, and more in general in all those cases where a stable and reliable network is not available, but the process must rely on a mobile ad-hoc network.

Given the wide diffusion of WS-BPEL, and the good set of supporting tools, the paper tackles the problem of decentralizing the execution of WS-BPEL processes by fostering the idea of *decentralized orchestration*. It proposes an automatic and formal approach to partition WS-BPEL processes, and to add the necessary synchronization and coordination stuff. The approach transforms a centralized WS-BPEL process into a set of coordinated processes. An explicit and precise meta-model [Obj02] and graph transformation systems [BH02] supply the formal grounding to slice a single WS-BPEL process, obtain a set of related processes, and add the communication infrastructure among the newly created processes to both exchange data and propagate the execution flow. The paper demonstrates the approach on a simple case study.

The approach is supported by two different families of tools, which are currently under development. We are working on a CASE tool, based on AGG (Attributed Graph Grammar [Bey92]), that implements *partitioning rules* and automatize the whole ”slicing” process. We are also working on the infrastructure to execute such federated WS-BPEL processes.

The results presented in this paper are an evolution of those presented in [BMM04]. More specifically, the novel contributions of this paper are: (1) an explicit meta-model for WS-BPEL specifications, (2) partitioning rules that address the information exchange among cooperating processes, (3) partitioning rules for fault, compensation, and event handlers, and (4) a more complete example application.

The rest of the paper is organized as follows. Section 2 introduces the formal ingredients, that is, the meta-model and partitioning rules. Section 3 discusses the approach on an example application. Section 4 surveys related approaches, and Section 5 concludes the paper.

2 Partitioning rules

This section introduces the approach to transform a WS-BPEL model into a set of federated models. The first part of the section introduces the meta-model, specified to render the key elements of WS-BPEL, and graph transformation systems. The second part presents *partitioning rules*. It describes one rule thoroughly, and concentrates on the key aspects for the others. Lack of space does not allow us to present all rules in detail, but interested readers can refer to [BMM05] for the whole graph transformation system.

2.1 Meta-model

Even if WS-BPEL is a complex workflow language for describing Web services compositions, Figure 1 only comprises those elements that are used by our approach, that is, that are handled within partitioning rules. The meta-model presented here borrows some concepts from the work presented in [Ga03], but we explicitly decided to get rid of all those elements that, even if are part of the WS-BPEL specification, and thus should be part of a complete meta-model, are not considered by the partitioning process.

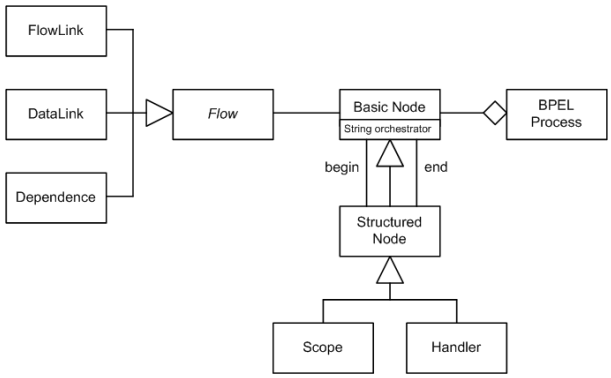

Figure 1: Dedicated WS-BPEL metamodel

Straightforwardly, a `BPEL Process` comprises several `Basic Nodes`. This class is the key component of the whole meta-model. `Basic Nodes` correspond to the basic activities that can be performed by a WS-BPEL process, like `invoke` and `receive`. `Structured Nodes`, which are a specialization of the first ones, correspond to the typical constructs of workflow languages supposed by WS-BPEL, like `switch`, `flow`, `sequence`, and `pick`. We treat each `Structured Node` by means of two `Basic Nodes` to identify the first and the last element of the composed statement. `Scopes` and `Handlers` are seen as special-purpose `Structured Nodes` since they both embed a set of `Basic Nodes`. We do not use containment relations to render the embedding of some nodes into others, but we exploit associations `begin` and `end` to identify the scope of each structured construct.

Nodes are characterized by the `orchestrator` that is in charge of their execution and are connected through `Flows`. These, in turn, can be `FlowLinks`, to render order of execution

among nodes, `DataLinks`, to define data dependencies among nodes, and `Dependences`, to relate the pairs of `invoke/receive Basic Nodes` added to synchronize components that have to be executed by two different executors (orchestrators).

This is not a "complete" and constrained meta-model. This means that we cannot be used to probe the consistency of WS-BPEL specifications. The meta-model is simple and "unprecise" since we assume that initial WS-BPEL models are correct and that the partitioning rules, that is, the graph transformation rules, only produce correct models[1].

2.2 Partitioning rules

Partitioning rules must be precise enough to allow for correct and automatic transformations and are supposed to work on the graph structure (i.e., an instance of the meta-model) that is behind any WS-BPEL process. These two requirements led us to consider graph transformation systems [BH02], along with the tool support they offer, as the right means to specify them.

Partitioning rules are introduced here incrementally. We start introducing the rules to control the execution flow, already presented in [BMM04], then we move to those that oversee data exchange and manage handlers. Since all rules are "similar", here we only describe one rule in detail and we invite readers to refer to [BMM05] for the complete set. All the other rules are presented by concentrating on the problems behind them, instead of showing how the different instances of meta-model elements are created, deleted, and modified.

In this paper, we introduce *partitioning rules* for controlling the execution flow by means of the example of Figure 2, which shows the meta-model of a simple WS-BPEL `Switch` statement with a `Basic Node` in each branch. The figure also shows that different nodes are associated with different orchestrators: this means that the designer has already decided how to split the process, that is, how to allocate the different sub-processes on available executors.

To partition the simple example process of Figure 2, we start by setting links to state the dependence, in terms of execution flow, between two WS-BPEL activities that are controlled by different orchestrators. Figure 3 shows the rule. The left-hand side describes when the rule can be applied, while the right-hand side shows the final result. The rule starts with two nodes that are associated with different orchestrators and introduces two new `Basic Nodes` to synchronize the execution of the first part with the execution of the second part. These two nodes are a pair of `invoke/receive` activities to forbid the second activity to start before the completion of the first one.

The partitioning process continues by identifying all the other pairs of nodes that must be synchronized and by adding special-purpose `Basic Nodes` to notify the branch followed by the `Switch` to all involved orchestrators.

The partitioning process also takes into account data distribution. According to the WS-BPEL specification, the scope of variables can be global (all activities can use them) or local (e.g., constrained in a given handler or scope). After partitioning, the result is a set of variables (both local and global) that are accessible by their local orchestrators, and a way to propagate their values to the other orchestrators. To cope with this, we added data-dependence graphs to our approach. As discussed in the literature for liveness analysis, in some configurations it is not

[1] This is not explicitly demonstrated in this paper, but readers can refer to [BMM04] for a first informal demonstration limited to the rules that deal with the execution flow.

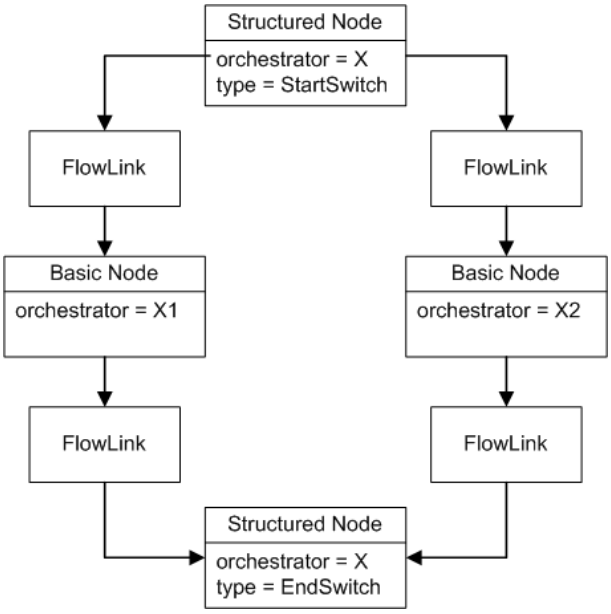

Figure 2: Example meta-model of a simple `Switch` construct

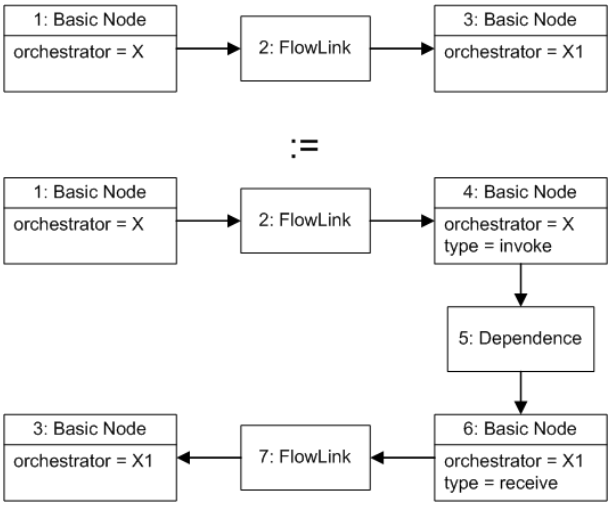

Figure 3: Rule AddDependence

possible to detect, at design time, suitable data dependencies between producers and consumers[2]. In such situations, the original WS-BPEL description, with that specific assignment of activities to orchestrators, cannot be partitioned. Thus, we also added an analysis phase, before starting the partitioning process, to understand if the approach is applicable.

Since execution and data flows are similar, it is possible to use the same rules defined for partitioning functional dependencies also for building a distributed data dependence graph. Nevertheless, in a data dependence graph, a node can be reached by several nodes if producers belong to structured activities.

For example, if we considered a `Flow` statement and we suppose that two nodes A and B are executed in parallel and produce the same variable v for node C, in a centralized environment the last activity executed would set the value for v. To preserve such a behavior, we need to add before the consumer node a `Pick Structured Node` with two branches: the first to receive the datum from B and the second from A. The first branch is followed after receiving a message *receiveA*, while the second is triggered by a message *receiveB*. This means that the branch followed is driven by the first node that produces data, but the value is received from the second node.

If we considered `While` statements, producers and consumers can be organized in different ways:

- The producer node is executed before the loop and the consumer is executed within the loop. In this case, we add a `Switch Structured Node` before the consumer. If the iteration is the first, the corresponding switch branch has a node to receive data from the producer; otherwise no operations are performed.

- Producer and consumer are in the loop. The producer is executed before the consumer. This simple case can be directly solved by using the rules already presented in [BMM04].

- The producer is within the loop and the consumer is executed after the end of the loop. In this case, we add a `Basic Node` after the end of the loop to send produced data.

- A producer is before the loop, but another producer is within the loop and after the consumer. To cope with this situation, we add a `Switch Structured Node` before the consumer to distinguish between the first iteration, when the data come from the external producer, and the other iterations, when data come from the internal producer.

Even if our meta-model considers exception handlers as structured activities, they need some specific considerations. First of all, we use the general term exception handler to identify fault, event, and compensation handlers. They comprise two parts: a part that model the raising of the exception, and a part that models the behavior of the handler by using "normal" WS-BPEL activities. Thinking of distributed executions, the second part can be managed with the rules already used for the "normal" behavior, while we need to address more thoroughly the raising of the exception and its propagation to involved orchestrators.

[2] Two WS-BPEL activities are interpreted as producer and consumer if the former sets the value of a given variable that is then used by the latter, without any other activity that changes the value between the two activities.

Each exception is associated with a scope and the relevant orchestrators are the same as those involved in the corresponding scope. For this reason, we add a synchronization mechanism at the end of each scope to ensure that if an exception is raised in a scope all orchestrators are properly synchronized. The other operation we need to insert is the propagation of the fault to all involved orchestrators. To cope with this, we propose a solution that is very similar to that used for propagating the value of a variable in `Switch` statements (an orchestrator evaluates it and then propagates the result to the others).

Figure 4 shows how to propagate the fault to all the orchestrators involved in a scope. The `Switch` is necessary to distinguish if the controller is also the catcher of the exception (e.g., a controlled activity failed) or if it is only involved in the scope.

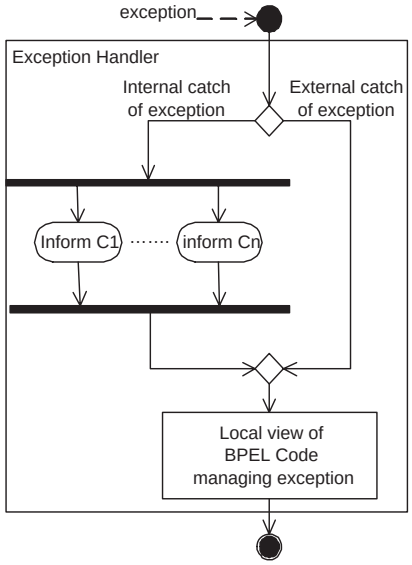

Figure 4: Distributed exception handling

This propagation mechanism, along with the synchronization required before the end of the scope, ensures that each exception is properly forwarded to all interested orchestrators.

To create the local processes, we have a dedicated set of rules. Given the orchestrator for which we want to produce the view, the rules are applied as follows. They: (1) remove all the `Basic Nodes` whose execution is not controlled by the selected orchestrator; (2) translate all the `Structured Nodes`, with the exception of `Sequences`, that do not include "local" activities into `Sequences` with no nodes. The same process, applied iteratively, produces the partial WS-BPEL models for the orchestrators that are part of the system.

3 Example

To explain the approach, we render as workflow a standard procedure defined and realized for carrying out maintenance by the Italian railways. Specifically, we consider the situation in which maintenance and testing of electrical lines and signals have to be performed. This process is par-

tially automatic and involves a central management unit (CMU) and several teams that perform required actions on site. CMU manages the activation/deactivation of railroad traffic on a given track. There are also two teams, one for putting and checking signals and the other one for managing electrical lines. The latter is further divided in the Electric Maintenance team (EMT) and the Electric Test Team (ETT). There is also a control post (CP) that controls the power supply on tracks. In this example, CP only exposes a Web service that allows to inhibit or reactivate electricity on tracks.

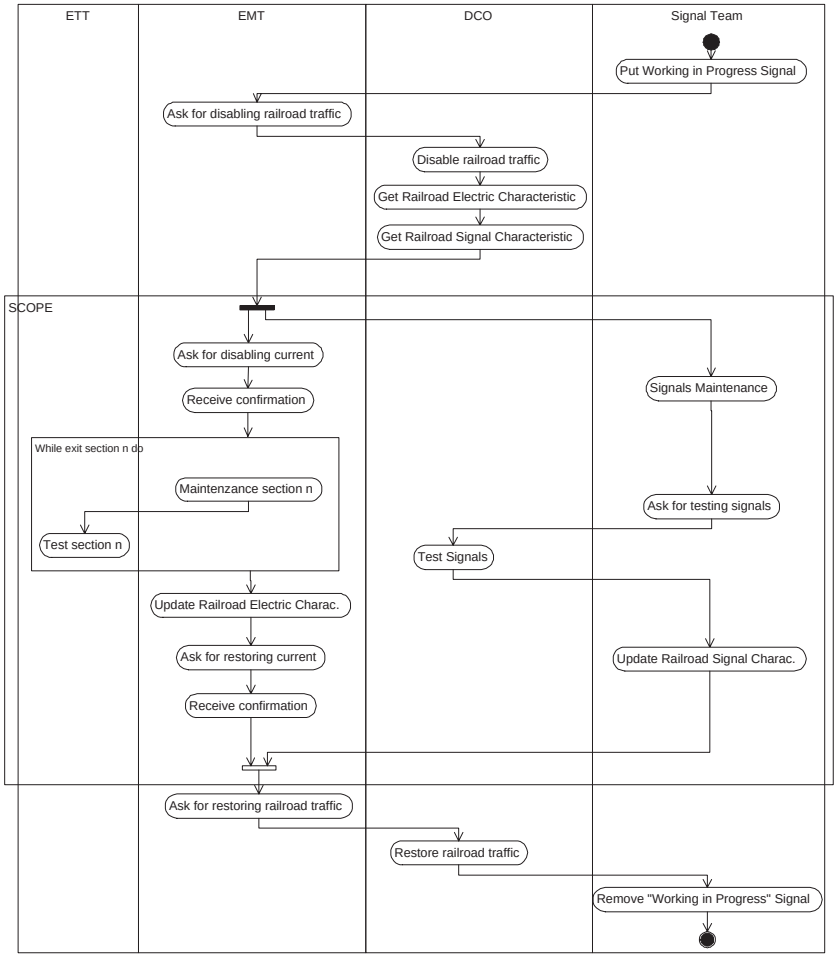

Figure 5: Example workflow

Figure 5 shows the example WS-BPEL process rendered as activity diagram. Each swimlane corresponds to a different controller (orchestrator). For our purposes, we consider each operation as a single task that can be modelled as a WS-BPEL basic activity. A fault handler is associated with the process. It models the possibility of stopping the maintenance activity, and thus resume the availability of the track for the normal train traffic. Notice that the operative teams could be off-line and perform part of their tasks while disconnected.

The workflow starts with the Signal Team that raises a working in progress signal on a given

track. Then, the EMT asks the CMU for disabling the railroad traffic. CMU disables the traffic and gets from the database the information about the electric characteristics and signals of the considered track. After this, the Electric Team and the Signal Team perform their activities in parallel. Electric maintenance is carried out by asking for the inhibition of the track, by dividing it in sections, and by starting maintenance and testing activities on each section. At the end, the power supply is restored. The maintenance of signals is performed by the Signal Team and then signals are tested by the CMU. At the end of their work, both EMT and Signal Team update the corresponding parts of the database. All operations are enclosed in a scope with an associated fault handler that allows the process to be interrupted to restore the normal traffic on the track. After completing maintenance, the EMT asks for restoring the normal traffic and after the CMU does this, the Signal Team removes the working in progress signal.

After explaining the problem, we are ready to apply our partitioning approach. Rules are applied recursively to insert specific coordination activities in the original workflow.

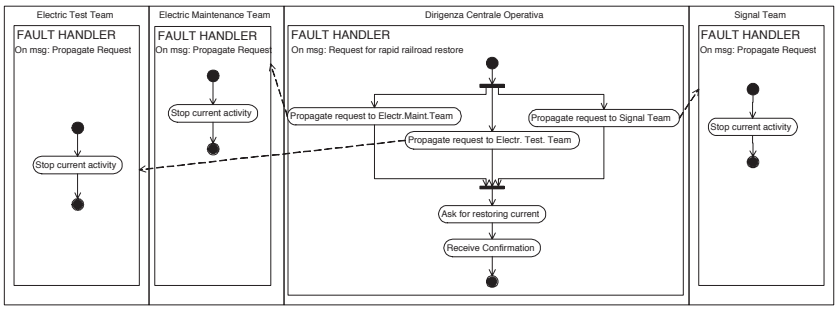

Figure 6: Decentralized fault handler

In our example, we have one fault and it can only be caught by the CMU. With this assumption, Figure 6 shows the fault handlers of the different controllers. After the CMU captures the request for restoring the normal situation, it propagates a message to the other orchestrators to raise an exception in each workflow. After this propagation, it asks the CP for restoring electricity. For security reasons, the procedure would require double confirmation to stop the activities of the operative teams (both CMU and CP would be involved), but this is not important for this example since it would only require that the fault handler be more complex.

The local view of each orchestrator, that is, its sub-process, can then be extracted from the modified workflow: Figure 7 shows the dedicated sub-process for the CMU orchestrator limited to the execution flow.

As far as data are concerned, the example shows that variables railroad electric characteristic and railroad signal characteristic are used by different roles (orchestrators). They store information that will be updated during the process. In Figure 5, the activities that use these variables are marked with x and +, respectively. In the centralized model, they are global variables, but in the decentralized scenario they have to be propagated to the correct orchestrators. It is possible to determine how these variables are used by building the data flow graph. Then, Figure 8 shows the activities that must be added to the original workflow to manage their propagation.

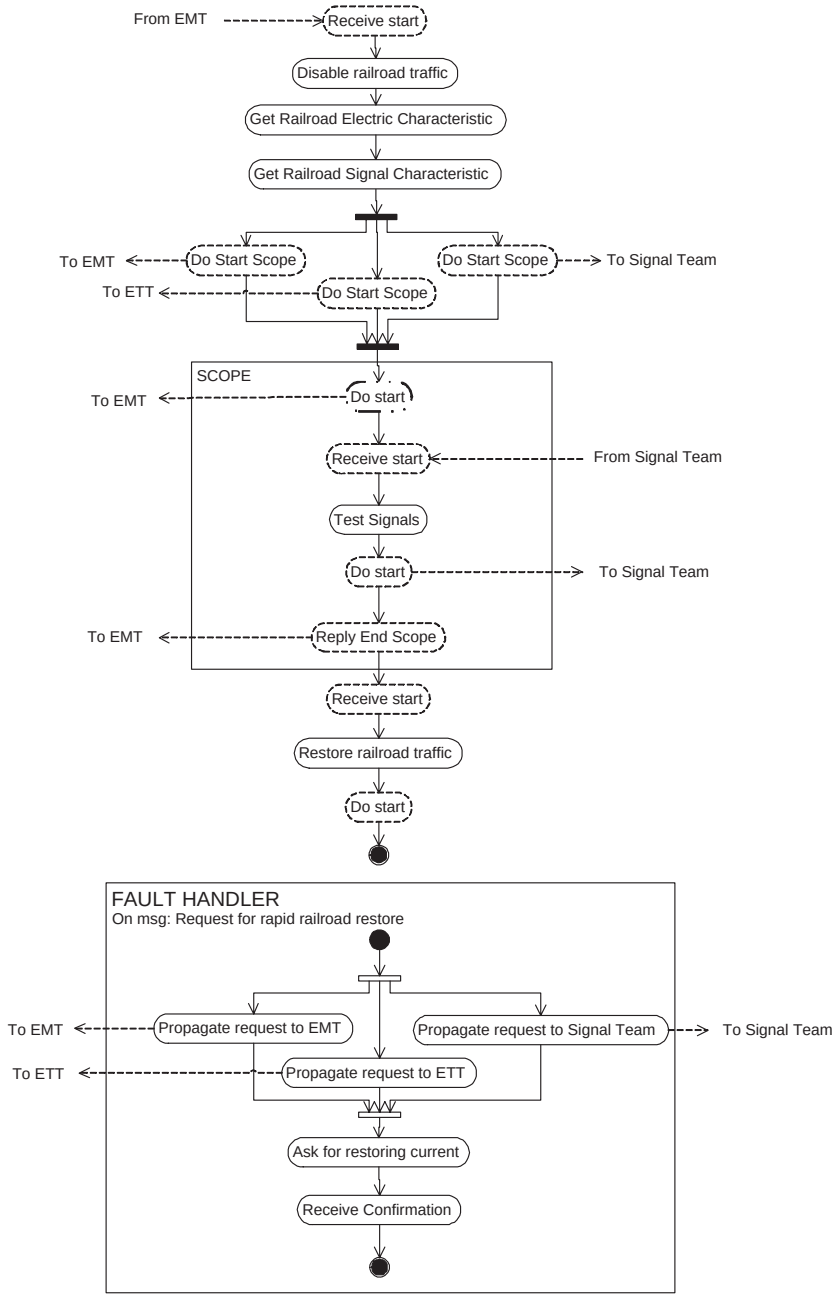

Figure 7: Local view for the CMU

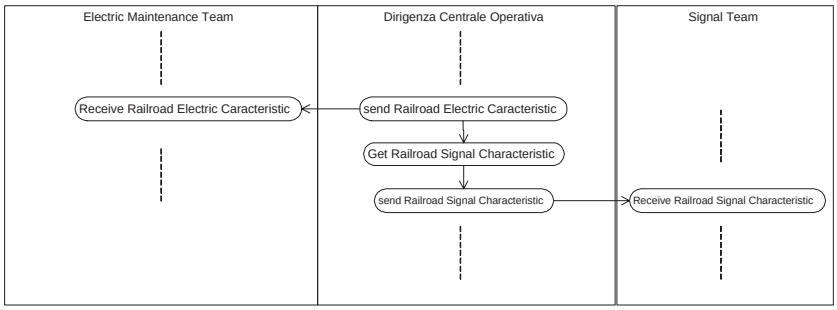

Figure 8: Activities inserted for propagating variables

4 Related work

The problem of workflow partitioning has been well-studied in the field of business process design for some ten years. The issue of workflow distribution still offers interesting aspects to study because of mobile information systems and Web services with the problems that come with them. In [JSH+01], the authors present a comparison among the different approaches to distribution.

Cross-Flow [GAHL00] aims at providing high-level support to workflows in dynamically-created virtual organizations. High-level support is obtained by abstracting services and offering advanced cooperation support. Virtual organizations are created dynamically by contract-based match-making between service providers and consumers. In Agent Enhanced Workflow [JOSC98], the agent oriented solution presents the interesting aspect of building execution plans using a goal approach. Event-based Workflow Process Management [EP99] includes an event-based workflow infrastructure and model constructs for addressing time aspects of process management. The main feature of ADEPT [RD98] is the possibility of modifying workflow instances at run-time. MENTOR [MWW+98] 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) [ASC+03] 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 applications which involve legacy information systems and that have geographically distributed and heterogeneous hardware and software environments, spanning multiple organizations. It also provides support for dynamic workflow processes, error and exception handling, recovery, and QoS management. Exotica [MAGK95] is characterized by the possibility of disconnected operations. It does not permit complete decentralization because it maintains a central unit and all operations obey a client/server paradigm. WISE [AFH+99] exploits the Web for its engine and offers an embedded fault handler. WAWM [Rie98] focuses on the problems related to the workflow management in wide area networks. Mobile [JB96] is developed to support inter-organizational workflows and is strongly based on modularity. This characteristic alleviates change management and also allows users to customize 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.

Described systems 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 runtime like ADEPT. Moreover, we argue that the mobile environment needs a language strongly oriented to the automatic execution like WS-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 demands them to the specific business process and application domain.

Many of these cited approaches do not consider Web services as available instruments for decentralizing business processes. A different comment is for [CCMN04], 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 minimize 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 WS-BPEL constructs and activities.

5 Conclusions and future work

The paper has presented our approach towards *distributed orchestrations*. It is the continuation of the results already presented in [BMM04], and extends them with the partitioning rules for handling data exchange among orchestrators and for dealing with handlers. Again, the first results are encouraging and are motivating us to continue working on these ideas.

Besides refining the rules, and applying them on further and more complex examples, the key element of our future work is the definition of the infrastructure to accommodate the execution of our federated WS-BPEL processes. We are working on different solutions that allow us to solve different problems. The first option is the use of standard WS-BPEL engines. This is the simplest solution, but it requires that all synchronization and communication problems among orchestrators be faced and solved in the process specifications. We are thinking of adopting an infrastructure based on a tuplespace middleware (like JavaSpace [Sun04]) to increase the

reliability of the supporting infrastructure, cope with the problems that come from producer-consumer analysis, and allow nomadic users to store and retrieve their data in/from dedicated tuples. Finally, we are investigating how to move the synchronization among orchestrators from the WS-BPEL models to the supporting infrastructure by adopting event-based architectures (e.g., WS-Notification) as the means to make the orchestrators communicate.

References

[AFH+99] 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.

[Aka04] Akamai Technologies, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM, SAP AG, Sonic Software, TIBCO Software. Web Services Notification. 2004.

[ASC+03] 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.

[BEA03] BEA and IBM and Microsoft and SAP and Siebel. Business Process Execution Language for Web Services Version 1.1. 2003.

[Bey92] M. Beyer. *AGG1.0 - Tutorial*. Technical University of Berlin, Department of Computer Science, 1992.

[BH02] L. Baresi and R. Heckel. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In *Proceedings of the First International Conference on Graph Transformation (ICGT 2002)*, volume 2505 of *Lecture Notes in Computer Science*, pages 402–429. Springer-Verlag, 2002.

[BMM04] 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.

[BMM05] Luciano Baresi, Andrea Maurino, and Stefano Modafferi. Partitioning rules for ws-bpel processes. Technical report, Politecnico di Milano, 2005.

[CCMN04] 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.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C, March 2001.

[EP99] 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.

[Ga03] T. Gardner and al. Draft uml 1.4 profile for automated business processes with a mapping to the bpel 1.0. IBM alphaWorks, 2003.

[GAHL00] 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.

[JB96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson, 1996.

[JOSC98] D. Judge, B. Odgers, J. Shepherdson, and Z. Cui. Agent enhanced workflow. *BT Technical Journal*, (16), 1998.

[JSH+01] 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.

[MAGK95] 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.

[MWW+98] 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.

[Nic04] Nickolaos Kavantzas and David Burdett and Greg Ritzinger. Web Services Choreography Description Language Version 1.0. 2004.

[Obj02] Object Management Group. Meta Object Facility (MOF) Specification - v.1.4. Technical report, OMG, March 2002.

[RD98] M. Reichert and P. Dadam. Adeptflex – supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[Rie98] G. Riempp. *Wide Area Workflow Management*. Springer, London, UK, 1998.

[Sun04] Sun Microsystems. Jini network technology, 2004. wwws.sun.com/software/jini/.