XRL/Flower: Supporting Inter-organizational Workflows Using XML/Petri-Net Technology

H.M.W. Verbeek, A. Hirnschall, W.M.P. van der Aalst

Faculty of Technology Management
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
h.m.w.verbeek@tm.tue.nl

Abstract. In this paper, we present the architecture of XRL/Flower. XRL/Flower is a software tool, which benefits from the fact that it is based on both XML and Petri nets. Standard XML tools can be deployed to parse, check, and handle XRL documents. The Petri-net representation allows for a straightforward and succinct implementation of the workflow engine. XRL constructs are automatically translated into Petri-net constructs. On the one hand, this allows for an efficient implementation. On the other hand, the system is easy to extend: For supporting a new routing primitive, only the translation to the Petri-net engine needs to be added and the engine itself does not need to change. Last, but not least, the Petri net representation can be analyzed using state-of-the-art analysis techniques and tools.

Keywords. Workflow, Inter-organizational, enactment, XML, WF-net, XRL, PNML

1 Introduction

Today's corporations often must operate across organizational boundaries. Phenomena such as E-commerce, extended enterprises, and the Internet stimulate cooperation between organizations. Therefore, the importance of workflows distributed over a number of organizations is increasing [2, 3, 16, 24]. Inter-organizational workflow offers companies the opportunity to re-shape business processes beyond the boundaries of their own organizations. However, inter-organizational workflows are typically subject to conflicting constraints. On the one hand, there is a strong need for coordination to optimize the flow of work in and between the different organizations. On the other hand, the organizations involved are essentially autonomous and have the freedom to create or modify workflows at any point in time. These conflicting constraints complicate the development of languages and tools for cross-organizational workflow support.

Looking at existing initiatives, it can be noted that (until recently):

- process support for cross-organizational workflow has been neglected since lion's share of attention has gone to data and
- mainly pre-specified standardized processes have been considered (such as, market places, procurement, and so on).

Based on these observations, we developed the *eXchangeable Routing Language* (XRL). The idea to develop a language like XRL was raised in [22] and the definition

Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 93–108, 2002.

of the language was given in [7]. XRL uses the syntax of XML, but contains constructs that embed the semantics of control flow. Moreover, XRL supports highly dynamic one-of-a-kind workflow processes. For example, we consider the "first trade problem," that is, the situation where parties have no prior trading relationship [25]. Clearly, the "first-trade problem" is the extreme case of highly dynamic one-of-a-kind workflow processes and therefore also the most difficult. To support highly dynamic one-of-a-kind workflow processes, XRL describes processes at the instance level. Traditional workflow modeling languages describe processes at the class or type level [17, 23]. An XRL routing schema describes the partial ordering of tasks for one specific instance. The advantages of doing so are that:

- the workflow schema can be exchanged more easily,
- the schema can be changed without causing any problems for other instances, and
- the expressive power is increased.

Note that workflow-modeling languages typically have problems handling a variable number of parallel or alternative branches [5]. In our research on workflow patterns [5], we compared the expressive power of many contemporary workflow management systems including COSA, HP Changengine, Eastman, Flower, Domino Workflow, Forté Conductor, I-Flow, InConcert, MQ Series Workflow, R/3 Workflow, Staffware, Verve, and Visual WorkFlo using a set of workflow patterns (See *http://www.tm.tue.nl/it/research/patterns/*). Based on the workflow patterns supported by these systems, and their relative use in practice, we carefully selected the most relevant constructs for XRL. Note that the expressive power of XRL far exceeds that of each of the workflow management systems mentioned above.

As was shown in [7], the semantics of XRL can be expressed in terms of Petri nets [28, 30]. Based on these semantics, we developed a workflow management system, named *XRL/Flower*, to support XRL. XRL/Flower benefits from the fact that it is based on both XML and Petri nets. Standard XML tools can be deployed to parse, check, and handle XRL documents. The Petri-net representation allows for a straightforward and succinct implementation of the workflow engine. XRL constructs are automatically transformed into Petri-net constructs. On the one hand, this allows for an efficient implementation. On the other hand, the system is easy to extend:

> *For supporting a new routing primitive, only the transformation to the Petri net format needs to be added and the engine itself does not need to change.*

There is a direct transformation from XRL to WF-nets, that is, the semantics of XRL is given in terms of WF-nets [33]. WF-nets are a special subclass of Petri nets which possess an appealing correctness notion (the soundness property [1]), are based on strong theoretical results (such as, the link between soundness, liveness, and boundedness [1]), and are supported by powerful software (such as, the tool Woflan [32]). The transformation has been implemented in XSLT (eXtensible Stylesheet Language Transformations) and resulted in the tool XRL/Woflan [33].

The language XRL and the mapping of XRL onto WF-nets has been presented in earlier papers [7, 8, 33]. This paper presents the architecture of XRL/Flower and describes parts of its implementation.

The remainder of this paper is organized as follows. Section 2 introduces XRL. Section 3 introduces WF-nets. Section 4 describes the formal semantics of XRL in terms of WF-nets. In Section 5 we present the architecture. Section 6 presents parts of the implementation of the tool XRL/Woflan. Section 7 relates this paper to known research. Section 8 concludes the paper. The Appendix shows the DTD of XRL.

2 XRL: An XML Based Routing Language

The syntax of XRL is completely specified by the DTD [11] shown in the Appendix. An XRL route is a consistent XML document, that is, a well-formed and valid XML file with top element route (see the Appendix).

The structure of any XML document forms a tree. In case of XRL, the root element of that tree is the route. This route contains exactly one so-called *routing element*. A routing element (RE) is an important building block of XRL. It can either be simple (no child routing elements) or complex (one or more child routing elements). A complex routing element specifies *whether*, *when* and in *which order* the child routing elements are done.

XRL provides the following routing elements:

- *Task*. Offer the given step to some resource, wait until the step has been performed, and afterwards set all events for which a child event element exists.
- *Sequence*. Start the child routing elements in the given order and wait until all have been performed.
- *Any_sequence*. Start the child routing elements in any order and wait until all have been performed.
- *Choice*. Start one of the child routing elements and wait until it has been performed.
- *Condition*. If the given condition holds, start the child routing elements of all true child elements in parallel and wait until all have been performed. Otherwise, start the child routing elements of all false child elements in parallel and wait until all have been performed. A condition may have any number (even none) of true and false child elements.
- *Parallel_sync*. Start the child routing elements in parallel and wait until all have been performed.
- *Parallel_no_sync*. Start the child routing elements in parallel but do not wait for any of them.
- *Parallel_part_sync*. Start the child routing elements in parallel and wait until the given number of child routing elements has been performed.
- *Parallel_part_sync_cancel*. Start the child routing elements in parallel, wait until the given number of child routing elements has been performed and cancel the remaining child routing elements if possible.

- *Wait_all*. Wait until either all events for which an eventref child exists are set, or wait until the given deadline of some child timeout element has expired. If this timeout element has a child routing element, start it and wait until is has been performed.
- *Wait_any*. Wait until either at least one of the events for which an eventref child exists is set, or wait until the given deadline of some child timeout element has expired. If this timeout element has a child routing element, start it and wait until is has been performed.
- *While_do*. As long as the given condition holds, start the child routing element and wait until it has been performed.
- *Terminate*. End this workflow instance.

As mentioned before, the routing elements of XRL are based on a thorough analysis of the workflow patterns supported by leading workflow management systems [5]. This, in addition to the fact that XRL is instance based, extensible, and grounded in Petri-net theory, makes XRL an interesting proposition.

3 Workflow Nets

As is shown in [33], the semantics of XRL can easily be expressed in terms of *Work-Flow nets* (WF-nets). Although the semantic transformation onto WF-nets is complex, it allows us to use standard tools and theoretical results [1, 32]. Before we sketch this new transformation, we briefly introduce some of the concepts related to WF-nets. We assume some basic knowledge of Petri nets [28, 30].

A Petri net that models the control-flow dimension of a workflow, is called a *WF-net* [1]. A WF-net is a Petri net with one unique input place and one unique output place. The procedure represented by the WF-net is created (with only the input place marked) when it enters the workflow management system and is deleted (when only the output place is marked) once it is completely handled by the workflow management system, that is, the WF-net specifies the life-cycle of a case. Moreover, in a WF-net there are no 'dangling' tasks and other unconnected parts, that is, every node of the WF-net should be on some path from its input place to its output place. Recall that a WF-net specifies the dynamic behavior of a single case in isolation.

For sake of completeness, we mention that the original definition of WF-nets did not include arc weights (sometimes also called multiple arcs). However, as mentioned in [32], it is straightforward to extend WF-nets by allowing arc weights. For the remainder of this paper, we assume that arc weights are allowed in WF-nets.

The WF-net requirements can be verified statically, that is, they only relate to the structure of the Petri net. However, for proper workflow behavior, there is another requirement that should be satisfied:

> *For any case, the procedure will terminate eventually and upon termination there is a token in the output place and all the other places are empty.*

Moreover, there should be no dead tasks, that is, it should be possible to execute an arbitrary task by following the appropriate route through the WF-net. These two additional requirements correspond to the so-called *soundness property.*

Note that the soundness property relates to the dynamics of a WF-net. In [1], it is shown that there is an interesting relation between soundness and well-known Petri-net properties such as liveness and boundedness. A WF-net is sound if and only if the short-circuited net (that is, the net obtained after adding a transition with the output place of the WF-net as its only input and the input place of the WF-net as its only output) is live and bounded. This result illustrates that standard Petri-net-based analysis techniques can be used to verify soundness.

4 Semantics of XRL in Terms of WF-Nets

The DTD shown in the Appendix only describes the syntax of XRL and does not specify the semantics. To provide operational semantics of the routing elements we transform each routing element mentioned in the DTD into a Petri net. Such transformations were given in [7, 33]. In this paper we assume the transformation as given in [33].

In Section 2 we already observed that the structure of an XRL document forms a tree, with the route element as root. Many routing elements interface only with their parent element and their child elements. For this reason, we propose to 'copy' this tree structure to the resulting WF-net: Every routing element is replaced by some Petri-net fragment that interfaces with the Petri nets associated with its parent element net and child elements nets. The exceptions to this rule are the terminate routing elements and the task, wait_all and wait_any routing elements (when events are involved). As an

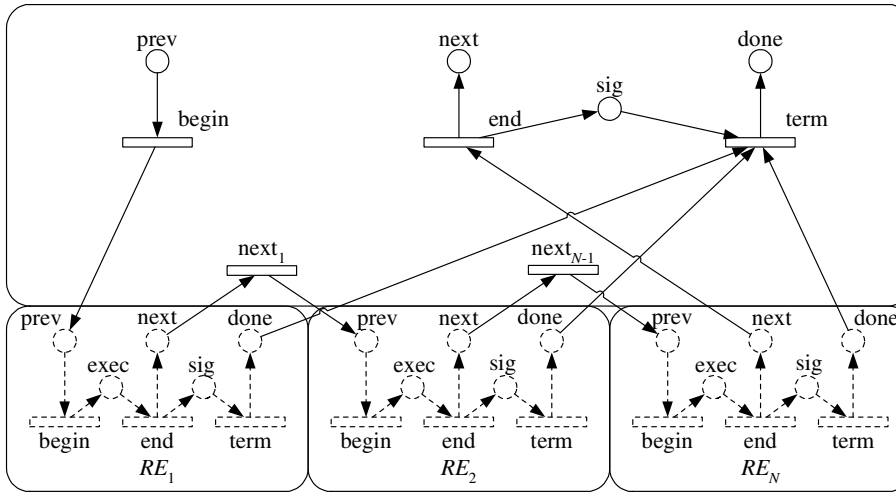

Fig. 1. The Petri-net fragment of the sequence routing element.

example Fig. 1 shows the Petri-net fragment of the sequence routing element. A detailed discussion of the XRL semantics in terms of WF-nets can be found in [33].

5 Architecture

Fig. 2 shows the architecture of the toolset involving XRL/Flower and XRL/Woflan. Using both the control flow data for the workflow case and the case specific data, the Petri-net engine computes the set of enabled tasks, that is, the set of *work items* that are *ready*. The engine sends this set to the work distribution module. Based on information on the organizational roles and users, the work distribution module sends e-mails offering the work item to certain users who are qualified to work on it. A user would receive an e-mail notification with a URL pointing to new work item(s) waiting for

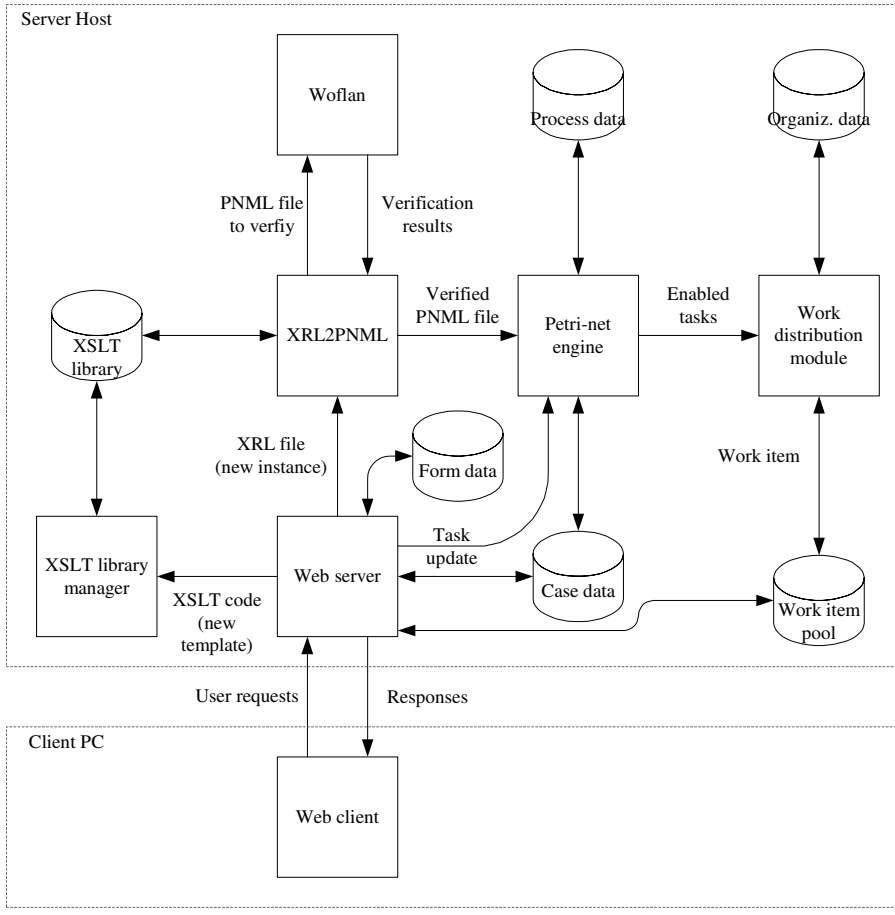

Fig. 2. A detailed architecture for implementing inter-organizational workflows using XRL/Woflan.

him. By clicking on the URL, the user accepts the work item; thus, the *work item* becomes an *activity* for a specific user, and other users to whom the work item was also offered are notified that it has already been accepted and is no longer available to them. A user who has accepted an activity may perform work on it either at acceptance time or later. In order to enable a user to perform an activity, the web server fills the appropriate form template with the case specific data for the activity. The user indicates completion of an activity by, say, pressing a submit button. The web server stores the updated case data and signals the Petri-net engine that the activity has been completed. The Petri-net engine then recomputes a new set of work items that are ready. The user can also start an XRL instance by sending the corresponding XRL file to the web server. The Web server forwards the XRL file to the XRL2PNML module that transforms XRL to PNML (Petri-Net Markup Language), which is a standard representation language for a Petri net in XML format [21].

6 Implementation

Fig. 3 shows the relevant part of the overall architecture taken out of Fig. 2 that is partly implemented: the Petri-net engine, the Web server, and XRL2PNML.

6.1 XRL2PNML

The XRL2PNML component first transforms an XRL instance into a WF-net using the semantics as described in Section 4. Second, it verifies whether or not the WF-net is sound. If sound, it delivers the WF-net using the PNML format to the Petri-net engine. Otherwise, it returns the WF-net together with diagnostic information on its unsoundness to the client. Because the transformation takes XRL as input and generates PNML

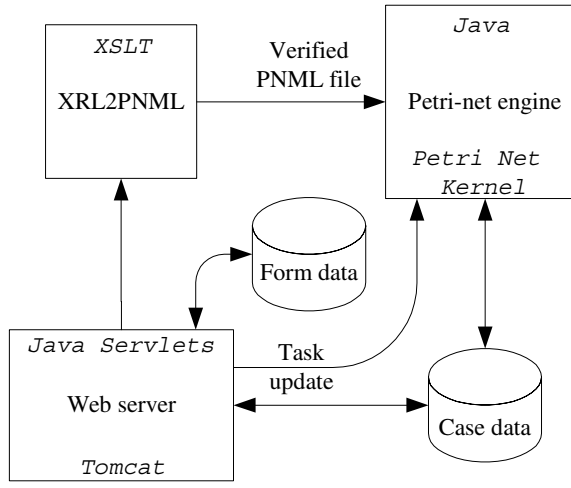

Fig. 3. Choice of Technology for Component Implementation.

as output, which are both XML-based formats, XSLT is used for the transformation step [33]. For verifying the resulting WF-net, the existing Woflan tool is used [31].

6.2 Petri-Net Engine

In an academic setting the implementation of algorithms for the analysis, and simulation of Petri nets often fails because the development effort can not be justified. Too much implementation time is spent on programming details. The main goal of the Petri Net Kernel (PNK) [15] is the reduction of programming time, which allows a better focus on algorithmic ideas. Thus, the PNK provides an infrastructure offering methods for the administration and modification of Petri nets and supports a quick, modular, object oriented implementation and integration of Petri-net algorithms and applications. The PNK also supports the PNML format, that is, the format delivered by the XRL2PNML component. A simple interface allows access to basic net information.

Fig. 4 shows a conceptual model of the Petri Net Kernel, which consists of layers. The PNK is responsible for the administration of graphs and nets. The main classes of the PNK one may take into consideration for application development are Net, Place, Transition, and Arc. The Editor visualizes Petri nets and allows their manipulation. Applications interface with the PNK to which they can easily be docked on and are independent of particular editors and their specific functionalities. The net type interface describes how an application programmer may define his own Petri-net type with specific extensions. Main classes of the net-type interface are Marking, Mode, Inscription, and Extension, which are represented as attributes in the class Specification. This specification of the net type is passed on to the PNK as a parameter.

With the help of the PNK it is relatively straightforward to build a Petri-net engine as described in Fig. 3 for the enactment of a WF-net. The code of Fig. 5 does not only sketch the essential lines of an enactment application but also shows how easily and

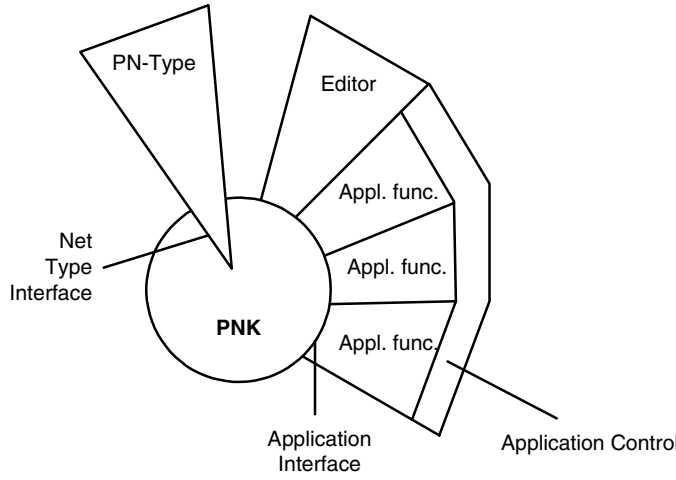

Fig. 4. Conceptual Model of the Petri Net Kernel.

straightforward applications can be integrated with the PNK in general. The code of Fig. 5 is described in further detail according to [29].

The `Enactment` class extends the `MetaApplication` class (line 6). Every application has to extend this class to inherit important base functionality. The class constructor has to take an `ApplicationControl` object as parameter and propagate it with the `super()` call (line 13) to the `MetaApplication`'s class constructor. The application may define its name in a static class field `staticAppName` so the `ApplicationControl` can display the application's name in its menubar (line 9). The application then implements a `run()` method (lines 16–20). This method is called by the `ApplicationControl` to start the application. When run, the `Enactment` requests a net's extension using the net field inherited from the `MetaApplication` class (lines 17–18) and starts simulating the net (line 19).

The `SimpleRule` class of the Petri Net Kernel implements the interface responsible for firing rules: `de.huberlin.informatik.pnk.netElementExtensions.base.FiringRule`. This interface defines methods for the firing of transitions and the inquiring of concessioned, that is, enabled, transitions [19]. The simple *firing rule* works as follows:

- A transition is said to be *enabled* iff each of its input places contains at least one token.
- An enabled transition may *fire*. If transition fires, it consumes one token from each of its input places and produces one token for each of its output places.

The `SimpleRule` object is net type specific and has to be implemented with the net type. Furthermore the method `simulateWithUserInteraction(MetaApplication app)` of the sample code in Fig. 5 is declared in the SimpleRule class. The implementation of this `simulateWithUserInteraction()` method in Fig. 6 (lines 3–24) is interesting in many aspects. Especially one sees the use of the Firing-

```
1  package de.huberlin.informatik.pnk.app;
2  import de.huberlin.informatik.pnk.app.base.MetaApplication;
3  import de.huberlin.informatik.pnk.appControl.ApplicationControl;
4  import de.huberlin.informatik.pnk.netElementExtensions.llNet.SimpleRule;
5
6  public class Enactment extends MetaApplication {
7
8      // this application's name
9      public static String staticAppName = "Enactment";
10
11     // class constructor
12     public Enactment(ApplicationControl ac) {
13         super(ac);
14     }
15
16     public void run() {
17         SimpleRule rule =
18             (SimpleRule) net.getExtension("firingRule");
19         rule.simulateWithUserInteraction(this);
20     }
21 } // Enactment
```

Fig. 5. Sample Code for Petri net Enactment.

Rule interface. Further, the code shows the simple algorithm that first detects all concessioned transitions of a Petri net (line 9). In a loop (lines 16-23) the firing of concessioned transitions takes place, which leads to the marking of their output places.

6.3 Web Server

Choosing the PNK and hence Java as technology of implementation is additionally supported by the implementation of the Web server. The Web server handles requests of Web clients and responds to them. This indicates that Java, and in particular Servlet technology in combination with a Tomcat server may be applied. In [18] Tomcat is the servlet container that is used in the official Reference Implementation for Java Servlet and Java Server Pages technologies. In [20] Java Servlets technology extends and enhances web servers. Servlets provide a component-based, platform-independent method for building web-based applications with good performance and allow access to the entire family of Java APIs.

7 Related Work

7.1 Petri Nets and Workflow

The semantics of XRL is expressed in terms of Petri nets. Petri nets have been proposed for modeling workflow process definitions long before the term "workflow management" was coined and workflow management systems became readily available. Consider for example the work on Information Control Nets, a variant of the classical Petri nets, in the late seventies [12, 13]. The readers interested in the application

```
1  public class SimpleRule extends FiringRule {
2      ...
3      public void simulateWithUserInteraction(MetaApplication app) {
4
5          checkContextAndParseExtensions();
6          ApplicationControl ac = app.getApplicationControl();
7          Net net = (Net) getGraph();
8
9          Vector concessioned = getAllConcessioned( ); //all concessioned transiti
10
11         if(concessioned == null || concessioned.isEmpty()) return;
12
13         Transition transition = (Transition)
14             (new SelectObjectAction(ac, net, app, concessioned)).invokeAction();
15
16         while(transition != null) {
17             fire(transition);
18             concessioned = getAllConcessioned();
19             if(concessioned == null || concessioned.isEmpty()) return;
20             transition = (Transition)
21                 (new SelectObjectAction(ac, net, app,
22                                         concessioned)).invokeAction();
23         }
24     }
25     ...
26 } // class SimpleRule
```

Fig. 6. Sample Code for Firing According to the Simple Rule.

of Petri nets to workflow management may refer to the two recent workshops on workflow management held in conjunction with the annual International Conference on Application and Theory of Petri Nets [6, 27] and an elaborate paper on workflow modeling using Petri nets [1].

We also would like to refer to two existing approaches toward inter-organizational workflows based on Petri-nets. The first approach uses Documentary Petri Nets (DPN's), that is, a variant of high-level Petri nets with designated places for message exchange, to model and enact trade procedures [10, 25, 26]. The Interprocs system is based on these nets. The main difference between the Interprocs language and XRL is that XRL is instance based and supports less structured and more dynamic processes. Another approach combining Petri nets and inter-organizational workflows is the P2P approach described in [9]. This approach uses inheritance to align local workflows. In [9] this approach is used to design an inter-organizational workflow for a fictitious electronic bookstore similar to amazon.com or bn.com. A predecessor of the P2P approach has also been applied to an inter-organizational workflow in the Telecom industry [4]. An interesting topic for future research is to see how the inheritance concepts used in [9, 4] translate to XRL.

7.2 Workflow Standards

Clearly, the work presented in this paper is related to the standards developed by the Workflow Management Coalition (WfMC, [23]). XPDL (XML Process Definition Language [34], is the XML version of WfMC's language to exchange workflow process definitions (cf. Interface 1 of the reference architecture). Wf-XML [35] is an XML-based language to support interoperability between multiple enactment services (cf. Interface 4 of the reference architecture). The scope of XRL can be compared to the combination of XPDL and Wf-XML. However, there are some striking differences:

- XRL supports an abundance of routing constructs while XPDL supports only the very basic ones (AND/XOR-split/join and loops),
- XRL is extensible with new routing primitives while XPDL only allows for additional attributes,
- XRL is instance based, and
- XRL has formal semantics.

7.3 Electronic Commerce

Recent development in Internet technology, and the emergence of the "electronic market makers", such as ChemConnect, Ariba, CommerceOne, Clarus, staples.com, Granger.com, VerticalNet, and mySAP.com have resulted in many XML-based standards for electronic commerce. The XML Common Business Library (xCBL) by CommerceOne, the Partner Interface Process (PIP) blueprints by RosettaNet, the Universal Description, Discovery and Integration (UDDI), the Electronic Business XML (ebXML) initiative by UN/CEFACT and OASIS, the Open Buying on the Internet (OBI) specification, the Open Application Group Integration Specification (OAGIS), and the BizTalk Framework are just some examples of the emerging standards based

on XML. *These standards primarily focus on the exchange of data and not on the control flow among organizations.* Most of the standards provide standard Document Type Definitions (DTDs) or XML schemas for specific application domains (such as procurement). Initiatives that also address the control flow are RosettaNet and ebXML.

RosettaNet. The Partner Interface Process (PIP) blueprints by RosettaNet do specify interactions using UML activity diagrams for the Business Operational View (BOV) and UML sequence diagrams for the Functional Service View (FSV) in addition to DTDs for data exchange. However, the PIP blueprints are not executable and need to be predefined. Moreover, like most of the standards, RosettaNet is primarily focusing on electronic markets with long-lasting pre-specified relationships between parties with one party (such as the market maker) imposing rigid business rules.

ebXML. Electronic Business XML (ebXML) is an interesting new framework for the conduct of business between different enterprises through the exchange of XML based documents. It consists of a set of specifications that together enable a modular, yet complete electronic business framework. Among other things, the ebXML architecture provides a way to define business processes and their associated messages and content [14].

The ebXML initiative is designed for electronic interoperability, allowing businesses to find each other, agree to become trading partners and conduct business. It is a joint initiative of the United Nations (UN/CEFACT) and OASIS, developed with global participation for global usage. Another important feature of ebXML is the systematic representation of company capabilities to conduct e-business in the form of a Collaboration Protocol Profile (CPP). CPPs give companies a common XML format to describe the industries, business processes, messages, and data-exchange technologies that they support in a structured way. With CPPs companies can agree on the business processes, messages and technologies used to exchange business messages for a specific trading need. These are expressed in a Collaborative Protocol Agreement (CPA), which is itself an ebXML document. Thus, the CPA provides the technical features of the agreement in automated form. The ebXML messages use the SOAP (Simple Object Access Protocol) specification. SOAP is an XML application that defines a message format with headers to indicate sender, receiver, routing and security details.

The ebXML proposal looks promising; however, it lacks many of the richer routing constructs that are present in XRL. Moreover, routing is modeled somewhat indirectly by means of a Document envelope sent by one role and received by another. Nevertheless, it appears that ebXML can solve the first-trade problem mentioned in the introduction.

8 Conclusion

In this paper, we presented the architecture of XRL/Flower. XRL is an XML-based language for describing WF-nets. One component of XRL/Flower converts XRL code into PNML so that a Petri net engine is capable of its enactment. The already existent Petri Net Kernel can be used and easily extended with an application for PNML enact-

ment. Since the PNK is implemented in Java, this can be considered a determining factor for the choice of technology. Subsequently Java Servlets and Tomcat are suggested for supporting the Web server of XRL/Flower.

References

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[2] W.M.P. van der Aalst. Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. Information and Management, 37(2):67–75, March 2000.

[3] W.M.P. van der Aalst. Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow. Information Systems, 24(8):639–671, 2000.

[4] W.M.P. van der Aalst and K. Anyanwu. Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. In A. Dognac, E. van Heck, T. Saarinnen, and et. al., editors, Proceedings of the Second International Conference on Telecommunications and Electronic Commerce (ICTEC'99), pages 141–157, Nashville, Tennessee, October 1999.

[5] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, pages 18–29. Springer-Verlag, Berlin, 2000.

[6] W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors. Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98), Lisbon, Portugal, June 1998. UNINOVA, Lisbon.

[7] W.M.P. van der Aalst and A. Kumar. XML Based Schema Definition for Support of Inter-organizational Workflow. Information Systems Research (forthcoming).

[8] W.M.P. van der Aalst, H.M.W. Verbeek, and A. Kumar. Verification of XRL: An XML-based Workflow Language. In W. Shen, Z. Lin, J.-P. Barthès, and M. Kamel, editors, Proceedings of the Sixth International Conference on CSCW in Design (CSCWD 2001), pages 427–432, London, Ontario, Canada, July 2001.

[9] W.M.P. van der Aalst and M. Weske. The P2P approach to Interorganizational Workflows. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), volume 2068 of Lecture Notes in Computer Science, pages 140–156. Springer-Verlag, Berlin, 2001.

[10] R.W.H. Bons, R.M. Lee, and R.W. Wagenaar. Designing trustworthy interorganizational trade procedures for open electronic commerce. International Journal of Electronic Commerce, 2(3):61–83, 1998.

[11] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. eXtensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, 2000.

[12] C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, pages 225–240, Boulder, Colorado, 1979. ACM Press.

[13] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, Application and Theory of Petri Nets 1993, volume 691 of Lecture Notes in Computer Science, pages 1–16. Springer-Verlag, Berlin, 1993.

[14] Enabling Electronic Business with ebXML, White Paper, http://www.ebxml.org/white_papers/whitepaper.htm.

[15] C. Ermel, M. Weber. Implementation of Parameterized Net Classes with the Petri Net Kernel of the "Petri Net Baukasten".

[16] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational Workflow Management in Dynamic Virtual Enterprises. International Journal of Computer Systems, Science, and Engineering, 15(5):277–290, 2001.

[17] S. Jablonski and C. Bussler. Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London, UK, 1996.

[18] The Jakarta Project, Available on the Web at http://jakarta.apache.org/site/binindex.html, 27.03.2002.

[19] Java Generated Documentation of the Petri Net Kernel, Available on the Web at http://www.informatik.hu-berlin.de/top/pnk/doku/pnk21-doc/index.html, 27.03.2002.

[20] Java Servlet technology, Available on the Web at http://java.sun.com/products/servlet/index.html, 27.03.2002.

[21] M. Jungel, E. Kindler, and M. Weber. The Petri Net Markup Language. In S. Philippi, editor, Proceedings of AWPN 2000 - 7thWorkshop Algorithmen und Werkzeuge für Petrinetze, pages 47–52. Research Report 7/2000, Institute for Computer Science, University of Koblenz, Germany, 2000.

[22] A. Kumar and J.L. Zhao. Workflow Support for Electronic Commerce Applications. Decision Support Systems (forthcoming).

[23] P. Lawrence, editor. Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons, New York, 1997.

[24] A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The WISE Approach to Electronic Commerce. International Journal of Computer Systems, Science, and Engineering, 15(5):345–357, 2001.

[25] R.M. Lee. Distributed Electronic Trade Scenarios: Representation, Design, Prototyping. International Journal of Electronic Commerce, 3(2):105–120, 1999.

[26] R.M. Lee and R.W.H. Bons. Soft-Coded Trade Procedures for Open-edi. International Journal of Electronic Commerce, 1(1):27–49, 1996.

[27] G. De Michelis, C. Ellis, and G. Memmi, editors. Proceedings of the Second Workshop on Computer-Supported Cooperative Work, Petri nets and Related Formalisms, Zaragoza, Spain, June 1994.

[28] T. Murata. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 77(4):541–580, April 1989.

[29] PNK User's Manual, Available on the Web at http://www.informatik.hu-berlin.de/top/pnk/doku/pnk-guide/quickStart.html, Last updated: 7.8.2001.

[30] W. Reisig and G. Rozenberg, editors. Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1998.

[31] H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, Application and Theory of Petri Nets 2000, volume 1825 of Lecture Notes in Computer Science, pages 475–484. Springer-Verlag, Berlin, 2000.

[32] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes Using Woflan. The Computer Journal, 44(4):246–279. British Computer Society, 2001.

[33] H.M.W. Verbeek, W.M.P. van der Aalst, and A. Kumar. XRL/Woflan: verification of an XML/Petri-net-based language for inter-organizational workflows. BETA working paper 65, Eindhoven University of Technology, Eindhoven, The Netherlands, 2001.

[34] Workflow Management Coalition Workflow Standard - Workflow Process Definition Interface - XML Process Definition Langauge. WFMC-TC-1025, Draft 0.03a, May 22 2001.

[35] Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding. WFMC-TC-1023, Version 1.1, November 14 2001.

Appendix: The DTD of XRL

```
<!ENTITY % routing_element
"task|sequence|any_sequence|choice|condition|parallel_sync|parallel_no_sync|parall
el_part_sync|parallel_part_sync_cancel|wait_all|wait_any|while_do|terminate">
<!ELEMENT route ((%routing_element;), event*)>
<!ATTLIST route
   name ID #REQUIRED
   created_by CDATA #IMPLIED
   date CDATA #IMPLIED>
<!ELEMENT task (event*)>
<!ATTLIST task
   name ID #REQUIRED
   address CDATA #REQUIRED
   role CDATA #IMPLIED
   doc_read NMTOKENS #IMPLIED
   doc_update NMTOKENS #IMPLIED
   doc_create NMTOKENS #IMPLIED
   result CDATA #IMPLIED
   status (ready|running|enabled|disabled|aborted|null) #IMPLIED
   start_time NMTOKENS #IMPLIED
   end_time NMTOKENS #IMPLIED
   notify CDATA #IMPLIED>
<!ELEMENT event EMPTY>
<!ATTLIST event
   name ID #REQUIRED>
<!ELEMENT sequence ((%routing_element;|state)+)>
<!ELEMENT any_sequence ((%routing_element;)+)>
<!ELEMENT choice ((%routing_element;)+)>
<!ELEMENT condition ((true|false)*)>
<!ATTLIST condition
   condition CDATA #REQUIRED>
<!ELEMENT true (%routing_element;)>
<!ELEMENT false (%routing_element;)>
<!ELEMENT parallel_sync ((%routing_element;)+)>
<!ELEMENT parallel_no_sync ((%routing_element;)+)>
<!ELEMENT parallel_part_sync ((%routing_element;)+)>
<!ATTLIST parallel_part_sync
   number NMTOKEN #REQUIRED>
<!ELEMENT parallel_part_sync_cancel ((%routing_element;)+)>
<!ATTLIST parallel_part_sync_cancel
   number NMTOKEN #REQUIRED>
<!ELEMENT wait_all ((event_ref|timeout)+)>
<!ELEMENT wait_any ((event_ref|timeout)+)>
<!ELEMENT event_ref EMPTY>
<!ATTLIST event_ref
   name IDREF #REQUIRED>
<!ELEMENT timeout ((%routing_element;)?)>
<!ATTLIST timeout
   time CDATA #REQUIRED
   type (relative|s_relative|absolute) "absolute">
<!ELEMENT while_do (%routing_element;)>
<!ATTLIST while_do
   condition CDATA #REQUIRED>
<!ELEMENT terminate EMPTY>
<!ELEMENT state EMPTY>
```