Realising Weak Work Workflow with Declarative Flexible XML Routing in SOAP (DeFleX)

Adsett, C.; Bernardi, A; Liu, S; Spencer, Bruce

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

*Realising Weak Work Workflow with Declarative Flexible XML Routing in SOAP (DeFleX) **

Adsett, C., Bernardi, A, Liu, S, and Spencer, B.
May 2004

* published in the Proceedings of Business Agents and Semantic Web (BASeWEB'04); a workshop in conjunction with the Seventeenth Canadian Conference on Artificial Intelligence. London, Ontario, Canada. May 16, 2004. NRC 48062.

Canada

Realizing Weak Workflow with Declarative Flexible XML Routing in SOAP

Connie Adsett[2], Ansgar Bernardi[3], Sandy Liu[1,2], and Bruce Spencer[1,2]

[1] Institute for Information Technology – e-Business
National Research Council of Canada
46 Dineen Drive, Fredericton, New Brunswick, Canada E3B 9W4
http://iit-iti.nrc-cnrc.gc.ca/groups/il_e.trx
{*Sandy.Liu,Bruce.Spencer*}*@nrc.gc.ca*
[2] Faculty of Computer Science, University of New Brunswick
P.O Box 4400, Fredericton, New Brunswick, Canada E3B 5A6
http://www.cs.unb.ca/
Connie.Adsett@unb.ca
[3] The German Research Center for Artificial Intelligence GmbH
Erwin-Schrödinger-Straße D-67608 Kaiserslautern, Germany
http://www.dfki.de/
bernardi@dfki.uni-kl.de

Abstract. A weak workflow within an organization's information system allows processes to be defined as they are being performed. It requires general knowledge about the organization to be dynamically combined with specific information about a current workflow. This information, as well as the roles of agents involved, is declared in RuleML so that inferences can drive the workflow. We describe a use case of bug tracking system where agents need help to decide where to sent a document next. DeFleX is a prototype implementation of this architecture using standard Web Services technology and an open source inference engine, jDREW. DeFleX uses an often-ignored feature of SOAP, allowing intermediate locations to be dynamically determined, to realize weak workflows.

1 Introduction

Business process modeling and workflow systems are well-suited to handle processes which are repetitive by nature, where the work in question can be modeled *a priori*, and if information needs and support opportunities are determined once and for all. However, many interesting and valuable work activities do not fit into this static scheme. Static process models might even hinder the development of the intended innovation. Distributed cross-organizational workflows in dynamic and *ad hoc* cooperations are difficult to model *a priori*, and complex but unique problems result in complex but unique solutions, which do not justify the effort of complete *a priori* modeling. The concept of weak workflow is therefore introduced to handle incomplete process models and to intertwine modeling and enactment of workflows.

With the distributed and composable nature, the Web Services architecture is considered to be suitable for realizing weak workflow applications. SOAP, the core protocol that enables Web services, offers a lightweight approach for exchanging structured information in a distributed environment. In this paper, we utilize the extensibility of the SOAP header to assemble weak workflows, in which SOAP intermediaries are used to model agents or nodes in workflows and the path for message exchange signify the change of the responsibility among participating agents. To distinguish weak workflows from static ones, the complete message routing path are not specified until run-time.

We proposed a Web services based architecture enabling just-in-time service composition in accordance with the context and content. A deductive inference engine is employed to perform reasoning services for all the participating agents, or nodes. In this architecture, an organization can define a set of declarative processing rules as general policy and allow insertion of specific process rules and facts at runtime by each intermediary nodes. A prototype DeFleX(**De**clarative **Fle**xible **X**ML routing) is implemented to demonstrate how dynamic message passing is made possible in a weak workflow environment.

We describe a simplified use case based on a bug tracking scenario where the initial document is a bug report with incrementally appended information about the attempts to fix the bug, including any test activities that were subsequently performed, a log history of contributing personnel and affected modules, and *ad hoc* rules and facts inserted by agents to express routing information used for directing this document to appropriate agents, and anything else dynamically determined to be relevant.

Following some background on weak workflows, SOAP and WS-Routing, and reasoning with RuleML, the paper describes the DeFleX architecture, its application in the bug tracking use case, and the state of the current DeFleX prototype. After related proposals are discussed, the final section combines conclusions and future work.

2 Background

2.1 Weak Workflow

The modeling of business processes and their enactment in workflow systems is a well-established approach. Explicit process models facilitate the documentation of business work activities, represent crucial know-how, and are the basis for reflection and re-organization of work practices. Their enactment in workflow systems results in improved control and traceability of work, guaranteed observation of approved processes, and various possibilities for automatic and semi-automatic support like information routing, task and role assignment, load balancing, or logging and archiving services.

Furthermore, process models and workflow instances can be seen as an explicit representation of the encompassing application context of individual tasks in an enterprise. While the context is exploited to adequately describe, store,

and retrieve information, the current workflow may trigger automatic information delivery to the user or similar proactive services. The paradigm of Business Process-Oriented Knowledge Management relies on this basis to support acquisition, utilization, and distribution of knowledge in modern enterprises [8, 15, 10].

Nevertheless, many processes, especially knowledge-intensive work are typically difficult problem solving [11, 12]where the solution and the solution process are invented and evolve in parallel. Thus, task sequences are not known in advance, and details of the work are not repetitive by nature. To retain the advantages of process-oriented knowledge support and workflow approaches in face of these ill-structured but interesting work activities, the FRODO project [20, 9] developed the notion of weak workflows as an approach to handle incomplete process models and to intertwine modeling and enactment of workflows. This approach requires only a minimum of *a priori* modeling workload and imposes minimal restrictions on the knowledge worker.

The design of weak (or weakly-structured) workflows is characterized by

- Support for lazy and late modeling: Work may start with an abstract and incomplete process description which is completed and refined during the actual work when necessary.
- Interleaving of modeling and execution: In order to enable the dynamic refinement during work, the individual workflow instance (which reflects a process model) has to be accessible for modification at runtime. The traditional workflow's separation between modeling phase and execution phase is thus blurred.
- Hierarchical refinement of task descriptions: The most important way to realize the late/lazy modeling of the workflow is to allow the replacement of some abstract process step by a more detailed sub-process at runtime. This approach reflects the phenomenon that knowledge-intensive activities are often well-known on an abstract level but all details have to be worked out carefully in the individual case.
- Rich explicit process logic: The interdependence between different work steps in a knowledge-intensive work process might depend on conditions which are not completely known *a priori*. Thus the traditional modeling of task sequences may not be sufficient. Instead, constraint-like descriptions of pre- and post-conditions of individual tasks and appropriate reasoning mechanisms allow for the dynamic configuration of the work process at runtime.

The restriction to hierarchical refinement preserves the guarantees and assertions of the given abstract process model. Beyond that, the weak workflow allows for arbitrary modifications at runtime. This allows total modeling flexibility but reduces the guarantee which the process model can offer: In extreme cases, the pre-given models are reduced to references or examples which may be copied but which are not binding for any given instance. Given these characteristics, the concept of weak workflows balances the need for formal representation (which enables automatic support) against reduced costs for modeling and increased flexibility. Furthermore, the interleaving of modeling and execution

leads to dual results of a knowledge work process: Besides the intended solution, an individually-tailored work process is created and retained. Such individual process knowledge is a valuable basis for the build-up of solid organizational know-how.

2.2 SOAP and WS-Routing

The SOAP[2] protocol, recommended by the W3C as the means of communication between Web services[7], offers a lightweight approach for exchanging structured information in a distributed environment. Often SOAP is used merely to carry a payload from a message sender directly to the ultimate receiver. The specification, however, allows for the use of intermediaries along the message path between sender and ultimate receiver. In addition, the *actor* attribute can be used in a SOAP header to indicate which part of a message is intended for a given SOAP receiver. These intermediaries may be actual deployed Web Services themselves, or they may be another resource as long as they can be identified by a URI and are capable of receiving, processing, and sending SOAP messages. They must act on the SOAP message received and then pass the SOAP envelope on to the next node in the message path, which may be another intermediary or the ultimate receiver.

The ability to use intermediaries along the message path provides a flexible mechanism for service composition where one individual Web Service or SOAP node is often incapable of performing all tasks desired by the initial message sender. The potential set of distributed value-added services provided by an active SOAP intermediary could be many such as security services, annotation services, and content manipulation services[3]. This feature makes a SOAP node a sensible unit to compose a weak workflow.

Despite the implied SOAP message model, SOAP does not define any routing or forwarding semantics corresponding with a message path. For example, an initial sender A can indicate which part of the message is for node B, C, and D, but it cannot specify the message is intended to travel from A, via B, via C, then to D. The Web Services Routing protocol (WS-Routing) fills in this gap by defining a message path model for exchanging SOAP messages from an initial sender to an ultimate receiver, potentially via a set of intermediaries[16]. WS-Routing is also transport independent. The routing path can be clearly expressed despite the use of different transport protocols (such as HTTP, SMTP, etc.) which SOAP can travel over. In addition, it also provides an optional reverse message path that enables two-way messaging. This feature potentially facilitates roll back procedures to be defined in a weak workflow.

2.3 Rule-based Systems, Representation of Rules, and Reasoning with Rules

Each organization, large or small, has some general policies either for codifying or streamlining the business processes. While they may not be written down explicitly, a rule-based system allows these policies and tacit human knowledge

to be captured as a set of declarative rules. Rules can also be built incrementally and each rule can be altered independently when a certain policy has changed. A declarative approach to weak workflow allows them to be flexible, easily explained, and reusable.

While SOAP is an XML protocol, RuleML[5] is developed as the canonical Web language for rules using XML markup. RuleML covers the entire rule spectrum, from derivation rules to transformation rules to reaction rules. RuleML can thus specify queries and inferences in ontologies, mappings between ontologies, and dynamic behaviors of workflows, services, and agents. In this paper we use RuleML version 0.85 to express general company policies, case specific policies, and facts with regard to the context of a specific message.

To make sense of the rules an inference engine is required. The open source *j*DREW (***j***ava **D**eductive **R**easoning **E**ngine for the **W**eb)[1, 18] provides an application programming interface to both a bottom-up and a top-down reasoning engine. Our DeFleX prototype interfaces with the bottom-up reasoning module to derive appropriate information for handling a specific message.

3 Realizing Weak Workflow with DeFleX

The use of SOAP intermediaries along with WS-Routing specification enables workflows to be executed through SOAP messages. Currently, there exists the capability to make use of SOAP intermediaries along the message path, but there is little emphasis on developing a standard method for determining which intermediaries to send the SOAP message to if they are not known by the initial message sender. Being able to determine intermediaries at later points in the message path is crucial in situations where the message path corresponds to a weak workflow instead of a strong workflow. There needs to be a common method of determining where to direct the SOAP message to next at points along that path according to the message context. In this section, we propose a method and an implementation, namely DeFleX, to realize weak workflow. There are two key concepts in our approach; one is the usage of SOAP intermediaries to resemble operation nodes in a workflow, and the other is rule based routing for directing messages at each hop. At least one router that has such reasoning capability must be defined in each DeFleX system.

For our use case we consider a software development company that offers clients the capability to report bugs and have them dealt with through SOAP enabled weak workflow. These bug report messages are received by a Quality Controller(QC) who then determines the type of bug reported, sets the priority, and sends it to the appropriate developer. The developer then attempts to fix the bug which, in some cases, may involve sending the bug information to other developers known to them. The developer then sends the message to a tester who tests the fix developed. Based on the observed result, the tester will report back to the QC with an okay flag or with an indication that the bug is not yet fixed. The workflow will be terminated when the bug is fixed and testing is successful. This should include regression testing to detect whether any new bug

is introduced with the current fix. If testing does not succeed, the tester will pass the report back to the QC and who then triggers another workflow cycle. This model is an example of a weak workflow because the complete message path cannot be outlined by the client sending the SOAP message.

Since workflows are mostly defined within an organization, we assume that there exists a commonly agreed upon ontology that all the participating parties are aware of; thus the terminology used in encoding the processing rules and facts can be understood by all parties without further translation. If this is not the case then semantic integration of the ontologies may be needed, and deferred for future work. To take advantage of previous efforts in process modeling, a company may specify a set of generic processing rules that defines the general guideline for workflow management. For instance, in the bug report and debugging workflow a high level processing rule can be "All bug fixes must go through a tester", which in turn can be encoded[1] as:

```
fixedBug(X):- verified(tester,BugFix(X))
```

In order to reduce the size of each SOAP message, we use a URL that points to a file storing these global processing rules to be embedded in a SOAP message instead of the whole set of rules. As weak workflow should support dynamic refinement of the work at runtime, each SOAP intermediary should be able to add more specific processing rules and facts that are known only locally to the current node. Again these *ad hoc* rules can be stored in a file and be referred to as a file pointer within the SOAP message.

Figure 1 depicts an abstract message path. There is a critical inference service, the DeFleX router, introduced here. This inference service will perform the following tasks when a SOAP message is received:

1. Retrieve the generic policy file, if specified;
2. Retrieve the specific policy file(s), if specified;
3. Unpack the rules and facts embedded in the SOAP header;
4. Infer which agent or node the message should be routed to next;
5. Construct a new SOAP message indicating the next node in the message path according to the inferred results.
6. Route this newly constructed SOAP message back to the agent who sent the original message forward.

As Figure 1 shows, the DeFleX router provides inference services to process all the declarative information, thus freeing all the other intermediaries from knowing how to process rules.

It is also assumed that the initial sender either knows the next hop in the workflow or some rules and facts about the message to be sent to the inference service. These rules and facts are encapsulated within a SOAP header. By processing them with an inference engine, the next node in the message path can be determined. At each node more information about the message and its path is

[1] To save space, rules are represented in Prolog as oppose to RuleML in this paper, although we use RuleML in our prototype implementation.

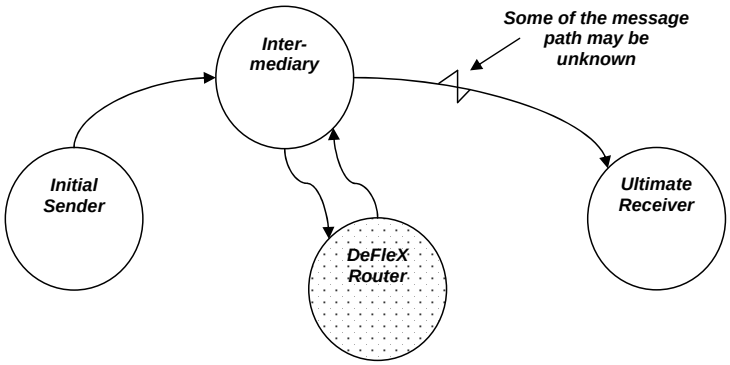

Fig. 1. An Abstract Message Path with SOAP intermediaries

revealed based on the context of the node itself and its ability to gather relevant information from the content of the SOAP message. The message path may then be created dynamically, as the SOAP envelope is passed along the path, thus facilitating a weak workflow.

Referring back to the use case, the client only need to be aware of the first step in the path, the QC or the DeFleX router. The software company can even mask this information from the end user by providing an client software, with which the client only needs to press a button to send the bug report to the destination that is pre-defined by the user's application.

The facts the client software would be able to include could be:

```
currentLocation(user).
messageID(message1).
about(message1,bug).
```

The first fact states that the current location of the message is with a specific `user`, the second fact gives the message an ID number as `message1`, and the last fact states that the `message1` is about a `bug`.

Assuming there exists a file that contains a set of general policy rules in terms of dealing with bug fix indicating:

```
goTo(Next) :- currentLocation(Current), nextNode(Current,Next).
nextNode(user, QC) :- messageID(X), about(X,bug).
```

The first rule says if the current location is `Current` and the next node of `Current` is `Next`, then go to `Next`. Please note that the terms starting with a capital letter such as `Current` and `Next` are variables. The second rule says the next node of the `user` is `QC` if a given message ID is about a `bug`.

The QC, with its capabilities and knowledge of some specific bug fix routine, can then add some rules to the set of processing rules stated by including:

```
nextNode(QC,developerComp) :- messageID(X), about(X,compilers).
nextNode(QC,developerUI) :- messageID(X), about(X,userInterface).
```

These rules specify how the bug report should be routed. If the bug is about `compilers`, then pass the message from `QC` to a specific compiler developer, `developerComp`; if the bug is about the user interface, then pass the message from `QC` to a specific GUI developer, `developerUI`.

The QC can also modify the facts to read:

```
currentLocation(QC).
messageID(message1).
about(message1,bug).
about(message1,compilers).
```

With this method the message path from the user to the tester can be dynamically created.

Because SOAP itself is an XML-based protocol, rules and facts can be naturally expressed in RuleML. In order to determine where to direct the SOAP message to next, the intermediary must be able to also process the RuleML contained in the header and use the inferred facts (e.g. the query result of goTo) to modify the message path. Although this may be feasible for some intermediaries, it is unlikely that all intermediaries are capable of doing this independently. For this purpose, the DeFleX (Declarative Flexible XML) router is designed to provide inference services. The following section outlines its implementation details.

3.1 Current Prototype

In the current prototype, the DeFleX router acts as a SOAP intermediary itself. The intermediary which is incapable of determining the next node forwards the SOAP message to the router after having added any known rules and facts to the RuleML header. The router then reads the RuleML header, processes the rules and facts, and obtains inferences through the reasoning engine jDREW. It does not modify the header in any way, but through the results from jDREW, determines the next node in the message path. We created a SOAP header following the WS-Routing specification to express the message path. The details of a *fwd* element in WS-Routing's path can be modified to allow the message path to be created dynamically by intermediaries.

Unfortunately, both WS-Routing and the SOAP intermediary have not been widely supported by many SOAP implementations albeit some commercial engines are available. We have chosen the highly configurable open source Axis as our SOAP engine, so that it could be customized to our needs.

The architecture of Axis is based on the concept of handlers and chains. A handler is described as "a reusable class which is responsible for processing a MessageContext in some custom way" and a chain is a group of related handlers which are invoked sequentially[6]. Axis has three levels of operation: transport, global, and service. Handlers, as well as chains, can be deployed at each level. The transport level takes care of all processes which relate to the transport the SOAP message was sent over. The global level handles all processes relating to general SOAP issues. The service level is responsible for processes relating to

the SOAP message for a specific Web Service. Intermediary handlers must be deployed at the global level because they are not attached to an individual Web Service, therefore this is the location that the DeFleX handlers are deployed. At the global level, they can access all SOAP messages received by the server and not just those addressed to a specific service.

The DeFleX router implementation is broken into two handlers; one to deal with the header containing the message path rules, and one to deal with the WS-Routing header. When SOAP messages are received by the server they are checked for the related headers and processed if they are present. The handler which operates on the rule and fact header interacts with jDREW in order to obtain the necessary inferences. From these inferences, the handler makes the necessary modifications to the WS-Routing header. The message is then sent back to the node which forwarded the message to the DeFleX router and from there it can be forwarded to the next applicable node, as specified in the WS-Routing header. If neither of these headers are present in a SOAP message received by the Axis SOAP server, the DeFleX functionality is not triggered and the message is handled as befits a SOAP message without path rules and facts.

In the case of the software development company described earlier, the DeFleX router is necessary to direct the SOAP message from the QC to the correct developer if the QC was not able to process RuleML itself. After processing the rules and facts with jDREW, the DeFleX router can determine that the next node in the message path is `developerComp` because the message is a `bug` about `compilers`. The QC agent can then modify the path header to state:

```
<path>
   <action>http://www.sftwrco.org/reportBug</action>
   <fwd>
      <via>soap://www.sftwrco.org/developerComp</via>
   </fwd>
</path>
```

The message would continue to be passed along the path (being sent to the DeFleX router for further steps as necessary) until the bug is fixed and fully tested.

4 Discussion and Related Work

While there is discussion about the need for the capability to express weak workflows (or dynamic message paths) using SOAP, what is often overlooked is where the information necessary to determine nodes in the workflow is located and how it is specified. It seems that most often the SOAP router is expected to have all required information within itself and not require input from external sources. Although this may be feasible in some instances, there are situations (such as the software development company example) where the information may need to be different for individual SOAP messages. For example, rules and facts about a message path may change for messages depending on the time

they were sent at or other characteristics of its context. In such a case, it is more logical to provide the information needed for routing within the SOAP message itself because it is unique to that message.

The WS-Referral Draft[13] in the Global XML Architecture Specifications (of which WS-Routing is also a part) offers a solution for containing information about the message path within referral statements. These can either be sent separately from the SOAP message or with the message as an additional header. However, the information about the path which can be contained in such a statement is limited. A paraphrased sample WS-Referral statement, provided in the Web Services Referral Protocol Draft, states "For any SOAP actor name matching the set of SOAP actors listed in the for element, if the set of conditions listed in the if element is met and hence the statement is satisfied, then go via one of the SOAP routers listed in the go element." [13] The conditions which may be listed are currently limited to *ttl*(time-to-live) and *invalidates*. Although further conditions may be specified, there is no provision for further rules and facts to be provided apart from those listed independently in the *if* element. The referral statement basically consists of one rule and the recipient must be able to provide the necessary facts about the message and use this combined with the given rule to determine what action to take. In short, WS-Referral provides a solution to dynamic routing, but it does not go quite far enough. Only being able to specify one rule at a time and not having the capability to provide any facts constricts the flexibility of the routing and limits its scope of use.

Other XML content-based routing solutions[14, 19, 17] are often based on XPath [21]. XPath provides an effective means for users to express the node(s) of interest within XML documents. Hence, XPath based XML routing solutions often require a user to specify explicit XPath expressions to be matched against received XML documents. In other words, users have to know the structure or schema of the documents to be received. This introduces a new problem when the users don't know what XML documents to expect or when the set of available documents becomes too large to manage by the user. Moreover, since XPath knows nothing about the message path, the destination the match should be sent to cannot be specified with XPath. Additional resource is needed in order to route the filtered or aggregated documents to the next node. DeFleX, on the other hand, supports message routing natively through SOAP intermediaries and WS-Routing. Each header block can be targeted to different *actors*(nodes), XPath expressions can be easily inserted in the header to indicate which block in the SOAP payroll should be processed.

Nonetheless there are still issues regarding the DeFleX router architecture, message path expression, common predicate names, etc. that we have not been fully dealt with. Although a DeFleX prototype has been constructed, we must consider alternative implementations before being sure of which architecture is the best suited to the task. Because the prototype treats the router as a SOAP handler, it is difficult to expose its functionality to the public as a Web Service. An alternative would be to expose the router as a web service, but this also has drawbacks. The node using the router would have to then extract information

from the SOAP message it received and construct a new SOAP message to send to the router. This requires more knowledge on the part of the individual nodes in the message path. This relates to the issue of simplicity. As much as possible, the DeFleX router must be straightforward and intuitive to become a practical solution to the industry. It should not be difficult for a given node to add contextual rules and facts about the message and its path and then send the message to the router to determine the next node in the path. There must also be a common manner of stating the message path which can be understood by the participating sender, intermediaries, Web Services, and router. Currently we assume such problem is being dealt with by a commonly agreed upon ontology.

The DeFleX prototype uses WS-Routing but this is not currently the common method of stating a message path. At present, SOAP is usually dependant on its transport protocol to handle message path data. In addition, neither WS-Routing nor RuleML has been ratified by standard bodies, although efforts for standardizing a rule language for the semantic web are well underway [4].

There are also issues surrounding how to handle the possibility of the wrong intermediaries modifying the SOAP message (and potentially the message path) need to be looked after. Fortunately, there are a set of security standards and proposals that are intended to provide better solutions for XML and Web Service security. In the DeFleX application, one basic step to improve the integrity of a message could be mandating each intermediary to sign the message.

5 Conclusion and Future Work

The need to have a dynamic message path from the sender to an ultimate web service via intermediaries arises in information systems that must adapt to changes in workflows, as workflows are invented by participants dealing with *ad hoc* cases. This need is only now being addressed by new flexible weak workflow systems. The DeFleX router realizes one solution based on standard web services and open source technology. Information, expressed in RuleML rules and facts, describes the contents of a SOAP message, the location of the current agent, other relevant context information, the path taken by the message to get to this point, general knowledge about the participating agents and their capabilities, the organization's general policy on workflow, and any specialization of it that might have been necessary to suit this specific message. This information is given to an inference service, the DeFleX router, to dynamically determine the next node in the path, and it is also stored with the message to create a history from which a trace analysis can be done, and *ad hoc* workflows can be extracted as candidates for general workflow policies. Yet without common naming conventions, inference cannot be drawn. While our use case assumes a common ontology, some semantic integration of terminology culminating in a common ontology is required. This is not a focus of this paper, but reserved for future work which may of interested to the Semantic Web community.

References

1. A Java Deductive Reasoning Engine for the Web. www.jdrew.org. Accessed 2004 Jan 12.
2. SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0 -20030624/. Accessed 2004 Jan 12.
3. SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/soap12-part1/#forwardinter. Accessed 2004 Feb 20.
4. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.daml.org/2003/11/swrl/. Accessed 2004 March 6.
5. The Rule Markup Initiative RuleML. www.ruleml.org. Accessed 2004 Jan 12.
6. Axis User's Guide. http://ws.apache.org/axis/java/user-guide.html, 2003. Accessed 2004 Jan 12.
7. Web Services Glossary, Accessed 2004 Jan 12.
8. Andreas Abecker, Ansgar Bernardi, Knut Hinkelmann, Otto Kühn, and Michael Sintek. Toward a Technology for Organizational Memories. *IEEE Intelligent Systems*, 13(3):40–48, June 1998.
9. Andreas Abecker, Ansgar Bernardi, and Ludger van Elst. Agent Technology for Distributed Organizational Memories - The FRODO project. In *5th International Conference on Enterprise Information Systems – ICEIS 03*, volume 2, pages 3–10, Angers, France, April 23-26 2003.
10. Andreas Abecker, Knut Hinkelmann, Heiko Maus, and Heinz-Jürgen Müller, editors. *Geschäftsprozessorientiertes Wissensmanagement*. xpert.press. Springer Verlag, June 2002.
11. S. Buckingham Shum. Negotiating the Construction and Reconstruction of Organisational Memories. *Journal of Universal Computer Science*, 3(8):899–928, 1997. Auch als Report KMI-TR-56, Knowledge Media Institute, The Open University, Milton Keynes, UK, URL: `http://kmi.open.ac.uk/publications/techreports.html`.
12. Thomas H. Davenport, Sirkka L. Javenpaa, and Michael C. Beers. Improving Knowledge Work Processes. *Sloan Management Review*, 37(4):53–65, Summer 1996.
13. S. Lucco H. F. Nielsen E. Christensen, D. Levin. Web Services Referral Protocol (WS-Referral). msdn.microsoft.com/library /en-us/dnglobspec/html/ws-referral.asp, 2001. Accessed 2004 Jan 12.
14. KnowNow Inc. KnowNow Solution Overview. www.knownow.com, 2002. Accessed 2004 Feb 18.
15. H. Maus. Workflow Context as a Means for Intelligent Information Support. In *Akman,V. and Bouquet, P. and Thomason, R. and Young, R.A. (Eds.): Modeling and Using Context. 3rd International and Interdisciplinary Conference, CONTEXT'01, Dundee, UK, Proceedings*, volume 2116 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.
16. H. F. Nielsen and S. Thatte. Web Services Routing Protocol (WS-Routing). msdn.microsoft.com/library/en-us/dnglobspec/html/ws-routing.asp, 2001. Accessed 2004 Jan 12.
17. PolarLake. PolarLake Technology Overview White Paper. http://www.polarlake.com/resources/whitepapers/technologyoverview/4.shtml, 2004. Accessed 2004 Feb 18.
18. Bruce Spencer. The Design of j-DREW: a Deductive Reasoning Engine for the Web. In Kung-Kiu Lau Manuel Carro, Claudio Vaucheret, editor, *First Colognet*

Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems, pages 155–166. Universidad Politécnica de Madrid, September 2002. CLIP4/02.0.
19. Todd Sundsted. Using the JMS API and XML in Content-based Routing. 2000. Accessed 2004 Feb 18.
20. Ludger van Elst, Felix-Robinson Aschoff, Ansgar Bernardi, H eiko Maus, and Sven Schwarz. Weakly-structured Workflows for Knowledge-intensive Tasks: An Experimental Evaluation. In *Proc. 12th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2003)*, pages 340–345. IEEE Press, 2003.
21. W3C. XML Path Language (XPath) Version 1.1. http://www.w3.org/TR/xpath, 1999. Accessed 2004 Jan 12.