The Knowledge Engineering Review, Vol. 17:4, 389–405. © 2002, Cambridge University Press
DOI: 10.1017/S0269888903000547 Printed in the United Kingdom

A survey of coordination middleware for XML-centric applications*

PAOLO CIANCARINI,[1] ROBERT TOLKSDORF[2] and FRANCO ZAMBONELLI[3]

[1] *Dipartimento di Scienze dell'Informazione, Università of Bologna, Mura Anteo Zamboni, 40126 Bologna, Italy; e-mail: cianca@cs.unibo.it*
[2] *Freie Universität Berlin Institut für Informatik, Takustrasse 9, D-14195 Berlin, Germany; e-mail: tolk@inf.fu-berlin.de*
[3] *Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Via Allegri 13 – 42100 Reggio Emilia, Italy; e-mail: franco.zambonelli@unimo.it*

Abstract

This paper focuses on coordination middleware for distributed applications based on active documents and XML technologies. First, the paper introduces the main concepts underlying active documents and XML, and identifies the strict relations between active documents and mobile agents ("document agents"). Then, the paper goes into details about the problem of defining a suitable middleware architecture to support coordination activities in applications including active documents and mobile agents, by specifically focusing on the role played by XML technologies in that context. A simple taxonomy is introduced to characterise coordination middleware architectures depending on the way they exploit XML documents in supporting coordination. The characteristics of several middleware infrastructures are then surveyed and evaluated, also with the help of a simple example scenario in the area of distributed workflow management. This analysis enables us to identify the advantages and the shortcomings of the different approaches, and the basic requirements of a middleware for XML-centric applications.

1 Introduction

The Internet and the Web are intrinsically document-centred. We use the Web mostly for accessing *contents*, that is, information contained in some sort of document and/or accessible via specific Web services. In addition, several Internet applications explicitly deal with document exchanges and manipulations. These include, for instance, digital libraries, systems for Computer-Supported Cooperative Work (CSCW) and electronic marketplaces.

As a consequence of the above scenario, scientists and engineers are very active in developing novel methods, tools and infrastructures for document management. More generally, we envision a shift from process- and service-centred computing models towards *document-centric computing models*, specifically centered around the concepts of *active and mobile documents* (Chang & Znati, 2001; Lange *et al.*, 1999; Satoh, 2000; Dourish *et al.*, 2000). On the one hand, documents will be no longer only be the passive part of a software system but, instead, will integrate active behaviour (e.g. internal code and threads) and will be able to handle themselves and to coordinate with other application components (Chang & Znati, 2001; Lange *et al.*, 1999). On the other hand, such behaviour will include the capability for documents to move themselves over a network (Chang & Znati, 2001; Satoh, 2000). The success of XML technologies (World Wide Web Consortium, 2001a; World Wide Web

* This paper has been partially supported by a grant of Microsoft Research Europe, by Italian MIUR project SALADIN, and by Italian MURST Project MUSIQUE.

Consortium, 2001b) concurs to accelerate this trend towards active and mobile documents by providing easy document processing and data portability (Ciancarini *et al.*, 1999b; DecisionSoft Limited, 2001). However, for such a shift to become viable, it is necessary to clarify the role and the characteristics of the middleware that should support active document applications.

In the presence of multi-component and multi-document applications (as, e.g., CSCW systems and digital libraries), suitable coordination middleware infrastructures (Ciancarini *et al.*, 1999a; Tolksdorf, 2000) are needed to orchestrate the activities of components and active documents (e.g. to maintain consistency across related documents and to synchronise accesses to documents). Interestingly, when focusing on XML-based active documents, it is possible not only to conceive a coordination middleware to support activities in XML-based multi-component applications (Bompani *et al.*, 1999), but also to exploit XML-based active documents as the core of a general-purpose coordination middleware (Cabri et al., 2001; Mascolo *et al.*, 2002; Tolksdorf, 2002), i.e. as the active artefacts in charge of mediating and ruling coordination in a generic multi-component application (Ricci *et al.*, 2002).

The contribution of this paper is to analyse the role that XML can play in modern coordination middleware for applications centered on active and mobile documents. A very simple taxonomy is introduced to identify and characterise document-centric middleware and the possible exploitations of XML in that context. Then, several XML-centric middleware infrastructures are surveyed and evaluated according to this taxonomy, also with the help of a simple example scenario. By this, we try to identify the advantages and the drawbacks of the different approaches and identify several questions and problems that remain as future research challenges.

The remainder of this paper is organised as follows. Section 2 introduces the basic concepts underlying active documents. Section 3, after having introduced the taxonomy and the example scenario, surveys and evaluates several examples of XML-centric coordination middleware, supporting and exploiting XML active documents at different levels. Section 4 discusses some open research issues. Section 5 draws our conclusions.

2 Documents as agents

What are active documents? From a software designer perspective, an electronic document is any kind of data structure that applications can exchange and process. By definition, a document has some kind of contents (e.g. data, text, images, music), representing the very information the document is intended to store. In addition, any document typically includes some sorts of structural information (e.g. index, tags, formatting commands) used by external, document-processing entities (e.g. humans, search engines, browsers, printers and so on) required to understand and elaborate the content. Thus, in general,

$$\textit{(passive) document} = \textit{content} + \textit{structure}.$$

A glossary in a book, or the header of a .bmp file are examples of meta-level structural information needed to understand the content of the document itself. Tags in HTML files and in TEX documents are examples of structural information needed to elaborate the document.

When dealing with structure representation in a document, it is quite important to distinguish the declarative power of structural information (e.g. a tag specifying that a paragraph is the title of one section) from the procedural interpretation that is necessary to render or generically process a document according to its structure (e.g. the fact that section titles must be rendered with a specific font style). In many approaches, the structural information is mixed up with the procedural interpretation in badly structured ways. For instance, HTML mixes the structural information with the procedural: an `h1` header in a file specifies both that the enclosed text is a section title and that it has to be properly emphasised when rendered in the browser. Still, HTML does not fully associate the procedural behaviour with the document, leaving different browsers free to render `h1` tags in different ways.

One of the key factors in the success of XML technologies is their capability to overcome the above problems by promoting a clear and well-structured approach to the representation of structural and procedural information. XML tags, while able to effectively represent structural information, are not associated with any predefined procedural behaviour (WorldWide Web Consortium, 2001a). Procedural information, for the rendering or manipulation of XML documents can be associated with the XML document using a companion XSL stylesheet (WorldWide Web Consortium, 2001b). The XSLT language component of XSL allows one to define rules manipulating a document (i.e. transforming it into a different XML tree), whereas the XSL-FO language component can be used to define the rendering behaviour. Moreover, it is also possible to use any programming language (usually a Turing-equivalent language) instead of XSLT and XSL-FO, to specify how a document should be manipulated and rendered. In this way, a document can be associated with any kind of Turing-expressable behaviour. This is the case, for instance, of an XML document associated with external JavaScript or ActiveX functions in charge of dynamically manipulating its content and its rendering.

The above characteristics make XML intrinsically prone to the definition of active documents, i.e. of documents including (or being associated with) computational behaviour, other than content and structure. In other words,

$$(active)\ document = content + structure + behaviour.$$

2.1 *Towards document agents*

When a document encapsulates document-specific behaviour, determining how the document itself has to be handled, it can no longer be considered simply a document. Instead, such an active document can be assimilated to a software component or – in some cases – even to a software agent (Jennings & Wooldridge, 1999). In particular, we can recognise two different classes of document that can be considered active:

- When the internal behaviour of a document is intended as a service, to be used by external applications or components to handle the document, the document can be assimilated to an object and, as such, its nature is simply *reactive*: the internal activity of the document is triggered by requests to access the document. A large number of examples of reactive documents can be found both in the literature (Gaines & Shaw, 1999; Dourish *et al.*, 2000) and in commerce (for instance, JavaScript-enriched documents can be assimilated to reactive documents).
- When the document integrates autonomous threads of control (e.g. a Java thread running within the document or,[1] in the simple case, a Java applet included in the document), the document can exhibit *proactive* behaviour and, as such, it can be somehow assimilated to a software agent (Jennings & Wooldridge, 1999). For this class of document, we use the term *document agents* to characterise their twofold nature as documents and as software agents.

Several research works have recently suggested interesting applications of document agents. For instance, a proactive agenda can be able to alert users and proactively reorganise the schedule of a meeting by interacting with the proactive agendas of the other users involved in the meeting (Satoh, 2000). A proactive Web-based document can look in the Web for further related documents of potential interest for the user (Ciancarini *et al.*, 1999a).

2.2 *Mobility and coordination*

If active documents can be assimilated to software components – whether they are objects or software agents – they can be used as building blocks for the development of complex distributed applications. This requires providing documents with two additional features: the capability of transferring

[1] Of course, a thread runs in the operating systems, not in the document. However, if the thread is intrinsically associated with the document and executes by accessing and manipulating document data, it can be perceived and abstracted as part of the document itself.

themselves over the nodes of a network and the capability of coordinating their actions with other active documents.

The first feature, mobility, is intrinsic to the very concept of information and, so, of document: a document is created to transfer and move information around. By adopting open data formats, like XML, mobility of passive documents is automatically achieved. When the document includes behaviour and threads of execution that enable it to move from one place to another, there is another requirement. Such a mobile agent (Cabri *et al.*, 2002) requires also code portability as well as the presence of a software infrastructure – i.e. of a middleware – enabling and supporting active document *mobility* (Satoh, 2000).

The second feature, coordination, is necessary for the building of complex multi-component (or, better, multi-document) applications. When only reactive documents are involved in an application, coordination between documents often assumes the form of simple client-server interactions. This is also what happens in object-oriented applications: as reactive documents, objects are entities that can only act upon requests for services, thus promoting a client-server approach to application development. However, as soon as the application is built by making use of document agents, interactions and coordination activities are needed to express more complex and dynamic patterns, as can be the case of an active agenda trying to reschedule a meeting. In fact, as it happens in multi-agent applications (Jennings & Wooldridge, 1999), proactivity of components enables services and data to be not only requested but negotiated. Of course, such complex coordination activities can take place only in the presence of a suitable infrastructure capable of supporting a variety of coordination patterns in a flexible way (i.e. a trivial Remote Procedure Call (RPC) infrastructure to support simple client-server forms of interaction is not enough).

3 Coordination middleware and XML

Middleware is conceived as a software layer to support the execution of distributed applications by abstracting from the heterogeneous characteristics of different architectures, operating systems, programming languages and networks. It integrates this heterogeneity into one uniform system, capable of providing functionalities and services according to a given common programming abstraction implemented on top of the named heterogeneous components.

Among the various services typically offered by middleware, we are most interested in facilities for the coordination of document-centric activities. Coordination is usually considered to be the management of dependencies amongst activities (Malone & Crowston, 1994). As such, coordination middleware is intended to integrate functionalities and services needed to enable and rule (Ossowski, 2001; Tolksdorf, 2000a) the coordination activities of heterogeneous software components.

Coordination middleware is difficult to design. The provided abstraction has to deal with the central issues of how data are communicated and how activities are started and synchronised.[2] The heterogeneity ranges from RPCs, object invocations and component usage to agent interaction with different characteristics such as one-to-one or one-to-many communication and synchronisation. In addition, modern middleware has to effectively support mobility of application components, users and devices.

3.1 Document-centric middleware

With the beginning of the 1990s, several companies tried to establish standards for middleware architectures supporting active documents and their coordination (Adler, 1995).

These middleware architectures – grounded in the works of distributed object applications and middleware, like CORBA – established notions of documents in which "components" or "objects"

[2] In its general terms, middleware is intended to support a larger set of services, including lookup services, directory services, load-balancing services, transactions and so on. However, when speaking of "coordination middleware" we explicitly restrict our focus to coordination services.

were included. The components contained data or software to manipulate the data in other components. They were displayed to the user and accepted input for direct manipulation. Some control infrastructure offered services to coordinate via client-server interactions the internetworking of different components. As the components could be of difference sources, different services serving different components were integrated into an application represented as a document.

The two major players in the middle of the 1990s were *OpenDoc* from Component Integration Laboratories, a consortium supported by Apple, IBM, Taligent, Novell and SunSoft, and *OLE2* from Microsoft. Both offered similar functionality with some differences in the object models.

In contrast to today's XML-oriented middleware, objects and data were represented in a binary format in both OpenDoc and OLE2, and both frameworks were rather heavy. While OpenDoc was not able to gain wider acceptance, OLE2 can be considered one of the grandfathers of Microsoft's COM and.NET frameworks. The latter, however, has definitely lost the document-orientation of its ancestors, being considered a pure object-oriented middleware. {CORBA} established a component- and object-standard at the same time that found great acceptance as a general-purpose middleware in the mid-1990s. However, CORBA did not (and does not currently) incorporate a strong document metaphor.

With XML, document-centric abstractions are revitalised, and several interesting middleware systems for coordination have been proposed since the late 1990s in which XML and document agents play a central role. We discuss in the following what role XML can play in middleware for modern document-centric applications, with a specific focus on coordination. The systems under review fall into three main categories (Figure 1):

- systems that can offer services not based on XML for the use of XML-based document agents (middleware for XML agents),
- at the other extreme, systems can offer coordination services based on XML technologies and XML active documents (XML middleware for agents), or
- systems can adopt a fully integrated approach for XML-based coordination services in a world of XML document agents (XML middleware for XML agents).

The analysis of the above classes of XML-centric systems will be the focus of the following section. To better proceed with it, however, we introduce an example scenario that will be used to evaluate the characteristics of the systems in these classes.

As a simple example for a conceptual comparative analysis of XML-centric middleware systems, we use a small scenario from financial services which is motivated by (Andrade & Fiadeiro, 1999). Assume that a person has two bank accounts, A and B. If he or she wants to withdraw an amount from account A which is larger than the current balance there, the banking system shall automatically try to transfer the missing sum from B and proceed with the transaction. Only if A and B together hold less money than requested does the transaction fail. Aside from those data and services needed to represent and handle accounts A and B, an additional coordinator service has to offer the functional interface to the user and, more relevant for our purposes, it has to be able to coordinate (or support) actions such as evaluating whether a transfer is necessary from different accounts and providing for these withdrawal operations. Central issues for the coordination middleware used here are (1) to provide its service in a rather transparent manner and (2) to integrate A and B even though they might be located at different banks possibly using different systems.

3.2 Middleware for XML document agents

The first category we look at concerns middleware that offers services for agents that are specified using XML and "run" in an XML environment. The world the agents live in is completely XML-oriented and the middleware under study offers services to make documents become active and to let them coordinate with the outside world, although the middleware itself is implemented outside the XML world, i.e. without exploiting XML technologies.

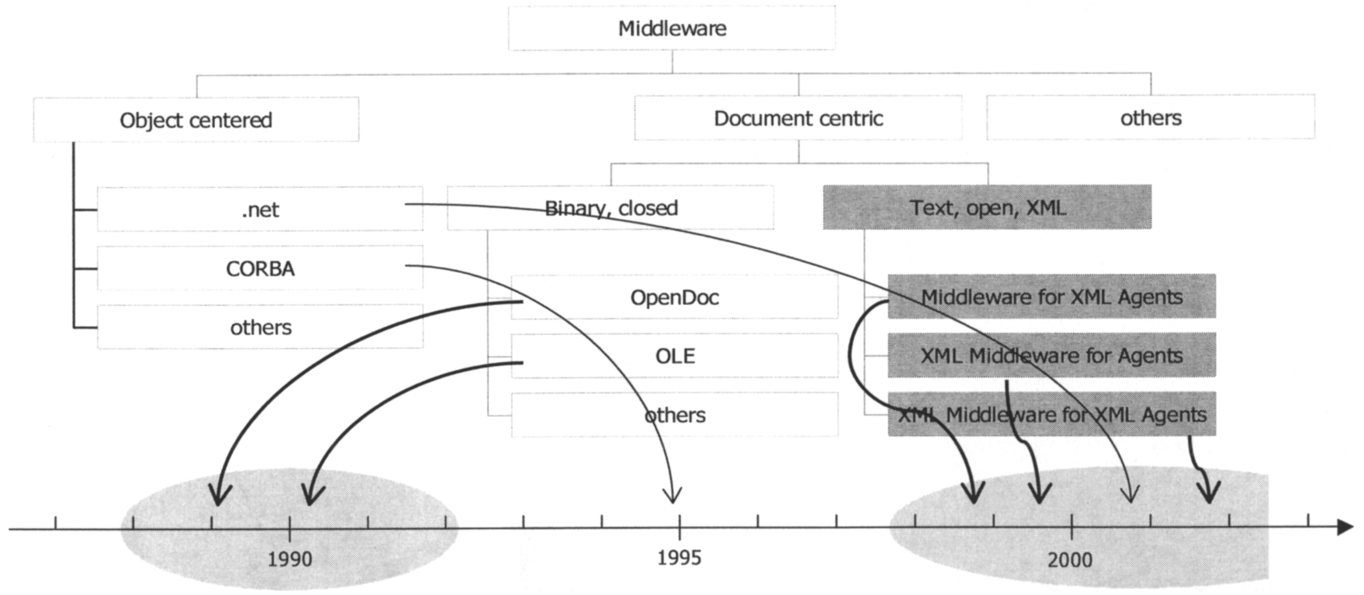

Figure 1 Document-centric middleware

3.2.1 Displets

The basic idea of the *Displets* approach (Ciancarini *et al*., 1999b) is to provide an active document environment, where XML documents can be enriched with application-specific behaviour in order to, say, let them be effectively rendered or transferred over a network. Specifically, displets are software modules that are attached to an XML document and activated when some pre-declared tags are parsed during the manipulation of the document. In short, a displet supports the specification of the treatment of either existing or new tags. A displet may print text strings, display images or make use of graphical primitives, and more generally do any needed action in the context of a multi-document application.

The first release of displets was proposed mainly for creating HTML extensions in a structured and general way. The idea was to be able to support new tags on a per-document basis, without any explicit support from commercial browsers, and to provide the document with the procedural rendering support needed to create in a document and visualise any kind of graphical object with styles, font, images and graphical primitives. With the advent of XML, the displets approach has been adopted as a tool for the rendering of XML documents. Now, displets are going to become a general-purpose environment for the definition and the execution of XML document agents.

The central idea of displets is to attach behaviour, in terms of Java classes, to XML documents. An XML transformation stylesheet (XSLT) can be defined to transform a “normal” XML document into an active one. The displets parser transforms the document into a DOM tree, after which the XML stylesheet can transform it into a different tree. The latter step is performed also by attaching to the generated tree the specification of Java classes devoted to associating specific behaviour with a specific portion of the tree. The new XML document obtained from this transformation can thus be an active document. There, Java classes determine the behaviour of the document when manipulated by external applications (e.g. browsers and printers), and runnable threads can determine the autonomous behaviour of the document when launched for execution.

A private internal behaviour can be associated with displets document agents, in order to define the behaviour of the document itself, as a stand-alone entity. However, it is also possible to think of attaching to a document a behaviour related to the interaction of a document with other documents, in the context of a multi-component application. Figure 2 illustrates the displets approach to coordination: in addition to the behaviour related to the internal handling of a document, a set of documents can share and being attached the behaviour devoted to implement and control the execution of coordination patterns among the set.

In the example scenario, a client document agent could be in charge of receiving inputs from the client, storing them internally in XML format, and rendering back to the client the XML data reporting the results of the account operations. All these operations are handled via some proper behaviour

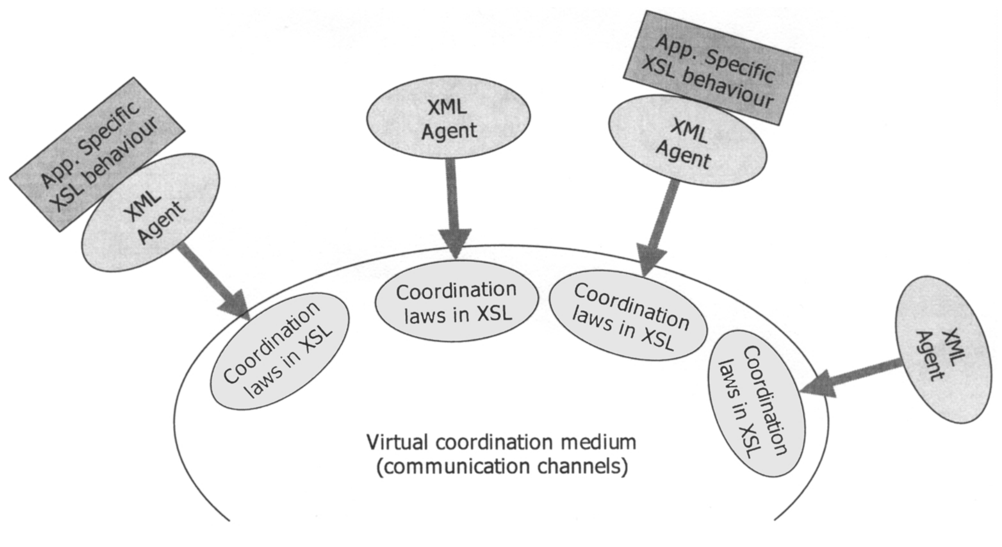

Figure 2 The displets approach

attached to the document agent. In addition, it is possible to attach to the client document agent the behaviour needed to coordinate – i.e. to negotiate withdrawal – with the document agents devoted to managing bank accounts. The document agents handling bank account, then, can integrate the coordination policies needed to handle the situation in which a client requests a sum which is not locally available, by making it start a negotiation with the document agents handling the other accounts of the user.

The main problem of the displets approach is that document behaviour, which includes the behaviour devoted to the implementation of coordination patterns, is hardwired into documents at compile time (i.e. during the transformation of the XML document into an active document). This can make it hard to exploit displets in open environments and in a mobile setting, where a document can move across different sites and needs to interact with different documents according to different coordination patterns. For the example scenario, changes to the policies adopted by the banks to handle accounts and withdrawals would require a change in the coordination behaviour attached to an applet, and would require rebuilding the document.

3.2.2 Other approaches

Other proposals exist that provide frameworks for making XML documents active by enriching them with some sort of internal behaviour. JXML (La Forge, 2001) is one such proposal. However, most of these frameworks are quite limited with regard to multi-document coordination. In most cases, coordination between documents simply amounts to enabling client-server object-oriented interactions, and there is no possibility of expressing more complex coordination patterns and coordination laws.

An interesting approach is adopted in the *adlets* system for information retrieval (Chang & Znati, 2001). There, the basic idea is to enrich Web-based documents (XML, but not necessarily) with a proactive declarative behaviour. The goal is to make a document able to look autonomously in the network for related documents. To this end, the adlets middleware enables a document to proactively move across the Internet (as if it were a mobile agent) and to coordinate with other documents to discover relations between documents and, possibly, to return to users clusters of related documents.

3.3 XML middleware for document agents

The coordination middleware described in this subsection exploits XML at the very core of the middleware. In particular, these systems assume that the coordination activities of application agents occur and are ruled via accesses to shared XML information spaces, in which the laws ruling coordination reside and are enacted. To some extent, these systems make information spaces in themselves become active document agents, active artefacts mediating coordination activities and specifying the laws according to which they can be accessed and modified by application agents.

3.3.1 XMLSpaces

In the Linda coordination language, coordination activities takes place via exchanges of structured tuples, i.e. ordered set of primitive (typed) fields. In particular, two active entities in an application can exchange data and/or synchronise with each other via a built-in mechanism of pattern-matching over a shared memory of tuples.

While the pattern-matching approach is indeed powerful and simple, Web-based and document-oriented systems may be in need of adapting it so as to support richer forms of data than ordered sets of typed fields. In particular, there is the need to capture any needed application-specific data structures easily, while encoding them in the form of primitive tuples would be quite complex. The format has to be open so that new types of data can be specified. Moreover, it has to be standardised in some way, so that data-items can be exchanged between entities that have different designs. XML fulfils all these criteria.

XMLSpaces (Tolksdorf & Glaubitz, 2001) is an extension to the Linda model which serves as middleware for XML. Here, XML documents can be placed into tuple fields and are considered for matching with templates.

Table 1 Matching relations in XMLSpaces

Relation	Meaning
Exact equality	Exact textual equality
Restricted equality	Textual equality ignoring comments, processing instructions etc.
DTD	Valid towards a DTD
DOCTYPE	Uses specific DOCTYPE name
XPath	Fulfils an XPath expression
XQL	Fulfils an XQL expression
AND	Fulfils two matching relations
NOT	Does not fulfil matching relation
OR	Fulfils one or two matching relations
XOR	Fulfils one and only one matching relation

A multitude of relations amongst XML documents can be used for making two documents match with each other, and consequently support a process of pattern-matching much more flexible than in Linda. While the basic pattern-matching relations shown in Table 1 are automatically supplied by XMLSpaces, the system is open for extension with further relations. XMLSpaces is distributed so that multiple dataspace servers at different locations form one logic dataspace. A clearly encapsulated distribution policy can easily be tailored to different network restrictions. Distributed events are supported so that clients can be notified when a tuple is added or removed somewhere in the dataspace.

For the example scenario, the state of the accounts would be represented in some XML format and stored in XMLSpaces. The documents could be protected from access by unauthorised parties by additional encryption following the XML security framework (World Wide Web Consortium, 2002; World Wide Web Consortium IETF, 2002). An appropriately extended matching mechanism would have to be supplied to the XMLSpaces. The coordinator service would have to be implemented in some language running on the Java Virtual Machine.

3.3.2 MARS-X

The MARS-X coordination architecture (Cabri *et al.*, 2001), implemented as an extension of the MARS architecture (Cabri *et al.*, 2000), defines a Linda-like middleware model to enable agents (specifically, mobile Java agents) to coordinate their activities via Linda-like access to shared spaces of XML documents.

Unlike XMLSpaces, which operates at the granularity of XML documents, MARS-X adopts a more fine-grained approach, and considers any XML document in terms of unstructured sets of tuples. For instance, the records of an XML document describing bank accounts with data values tagged as *name*, *number*, *amount*, could be interpreted as a bag of tuples in the form *account(name,number,amount)*. Accordingly to this perspective, a document and its data can be accessed and modified by exploiting the associative operation typical of the Linda model, and agents can coordinate with each other via exchange of document tuples, and via synchronisation over tuple occurrences. Specifically, MARS-X provides agents with a JavaSpace interface to access a set of XML documents in terms of Java object tuples. This choice forces agents to be Java agents.

To support wide-area computation, MARS-X promotes an architecture based on a multiplicity of independent XML dataspaces, each to be considered as a local resource of an Internet node or of a local domain of nodes (see Figure 3). By moving across the Internet, mobile agents can access different XML dataspaces: when an agent arrives at a node, it is automatically provided with the reference to a MARS-X tuple space interface associated with the XML dataspace.

A peculiar characteristic of MARS-X dataspaces is that their behaviour in response to agent accesses can be programmed to implement specific access methods and specific synchronisation and coordination patterns. Both administrators and mobile agents (the latter in a quite restricted way) can

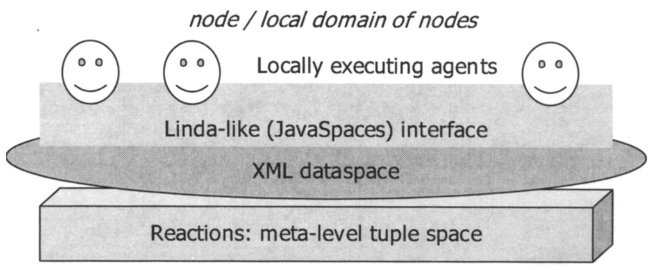
node / local domain of nodes
Locally executing agents
Linda-like (JavaSpaces) interface
XML dataspace
Reactions: meta-level tuple space

Figure 3 The MARS-X architecture

install in an XML dataspace reactions associated with specific access operations, performed by specific agents, with specific parameters. These reactions override the default behaviour of the performed operations and, for instance, can modify the result of the operations they are associated with, manipulate the content of the XML dataspace, and access whatever kind of external entity they need to access.

The programmability of MARS-X dataspaces makes the XML dataspace itself become an active document. In fact, although agents can access the dataspace always with the same limited set of operations, the dataspace itself can react to these accesses by behaving in different ways. The reaction in the dataspace can decide who and when can read and/or modify which XML documents. In addition, since coordination between agents occurs via data exchanged by means of the dataspace, the behaviour of the dataspace can be used globally to rule the activities of multi-agent applications.

Coming back to the example scenario, and by assuming the availability of the MARS-X middleware, one can think that each bank makes an XML dataspace with data account available to agents. When in need of withdrawal, the client can send his personal agent to account A first, to query the dataspace for his own data, to check the needed availability. On availability, the client agent can eventually withdraw the required amount by putting a specific tuple in a specific XML document. The insertion of that tuple can trigger the activity of the object devoted to managing account data, which will take care of actually performing the transaction and sendingthe result back to the client agent, again in terms of a tuple inserted in the dataspace.

The programmability of the tuple space can be effectively exploited in the example scenario to orchestrate, transparently to client agents, the cross-checking for availability in different accounts, and the possible need of withdrawing a portion of the total sum from different accounts. An example is the situation where the client agent requests an amount from account A which is more than the current balance of A.

The reactions in the dataspace can then cause another agent to become active. This agent is in charge of going to the account B dataspace to check if enough further money is available there. If so, it can let the account A dataspace reply to the client agent with the amount requested after withdrawing it in part from A and in part from B.

The possibility of controlling the execution of complex coordination patterns via specific behaviour of the XML dataspace and transparently to agents is, beyond the example scenario, a general advantage of the MARS-X approach.

A drawback of the MARS-X approach is that it introduces a big mismatch between the format of the data in the dataspace and the format of the data privately managed by the agent – the former being XML documents, the latter Java objects. This can make an application more complex. For instance, let us suppose that the client agent of the example scenario has to report back to the client its results via an XML page. In MARS-X, this activity report is fully in charge of the client agent, while there is no possibility of directly reporting in terms of XML documents the information that the agent has retrieved from the accessed dataspaces. This would require the client agent to directly manipulate and represent its world in XML terms or, in other words, would require agents to be themselves XML document agents, as in displets. A more detailed analysis of the ideal requirements for an XML middleware will be analysed later in this section.

3.3.3 XMIDDLE

The XMIDDLE middleware (Mascolo *et al.*, 2002) implements a coordination architecture somewhat similar to that of MARS-X: coordination between agents occurs via accesses to shared XML documents, and a limited form of programmability is made possible to rule these accesses. However, XMIDDLE implements specific architectural solutions to make it a suitable middleware for mobile computing and, specifically, for ad hoc networking.

The basic idea of XMIDDLE is to make coordination among agents (or, in general, among the processes of a distributed computation) occur by accessing a shared XML tree, via a specific language for querying and manipulating semi-structured data. However, in mobile settings, where processes or agents can disconnect and reconnect at any time, this introduces peculiar problems related to the accesses to the tree. In fact, in XMIDDLE an agent can access and modify the data on an XML tree, as well as its structure (see Figure 4). When that process disconnects from the network or becomes out of reach in the case of an ad hoc network, it is provided with a local replica of the tree (or of one of its sub-trees). When the agent reconnects, or is in reach again, the global tree has to be reconstructed, as it could have been possibly independently modified by different agents. To handle this situation, XMIDDLE enables the programmability, in the tree, of specific event handlers, in charge of implementing application-specific reconciliation policies, devoted to coherently reconstructing the structure of a tree.

In the example scenario, it is possible to conceive that a bank makes available one or more XML trees with the bank account data, to be accessed, as in MARS-X, by client agents. But unlike in MARS-X, these client agents could also be PDA and mobile devices, and XMIDDLE could automatically handle the problems related to mobility. In addition, since agents can directly manipulate the XML tree (while in MARS this manipulation occurred in the form of Java tuple objects), XMIDDLE can facilitate agents in directly reporting back XML data.

However, XMIDDLE has only a limited form of programmability of the XML tree, devoted to the handling of connection events. This makes it difficult to implement any complex coordination pattern in terms of transparent coordination policies, which include that required to withdraw partial amounts of money from different accounts. In XMIDDLE, this coordination pattern has to be directly implemented by the agent code.

3.3.4 Other approaches

There are some other approaches for XML middleware. Most prominent is the current XML Protocol activity of the World Wide Web Consortium (2001a). XML Protocol is an approach to follow up on SOAP and XML-RPC in order to have distributed peers communicate by using XML as the communication language. For communication amongst objects, for example, this boils down to representing a method invocation with name and parameters in a simple XML document.

The XML Protocol approach offers only a low-level abstraction for coordination and currently supports only client-server-style interactions.

3.4 Self-contained XML middleware

XML is a standard for representing data in networked documents. However, the specification of activities can also be expressed as an active document. Thus, if scripts and so on can be XML documents, a complete system can be based on XML representation and even activity and its coordination can be expressed within that framework. Thus the agents can be represented as some XML documents as well as the data they operate on and the laws ruling their coordination activities. The main effect of this self-containment is the uniformity of the language used. On the one hand, this makes programming easier since one does not have to switch to an external language like Java. On the other hand, and even more relevantly, the scripts themselves then become first-class data objects for the middleware. The same infrastructure used for applications can be used to manage the programs themselves.

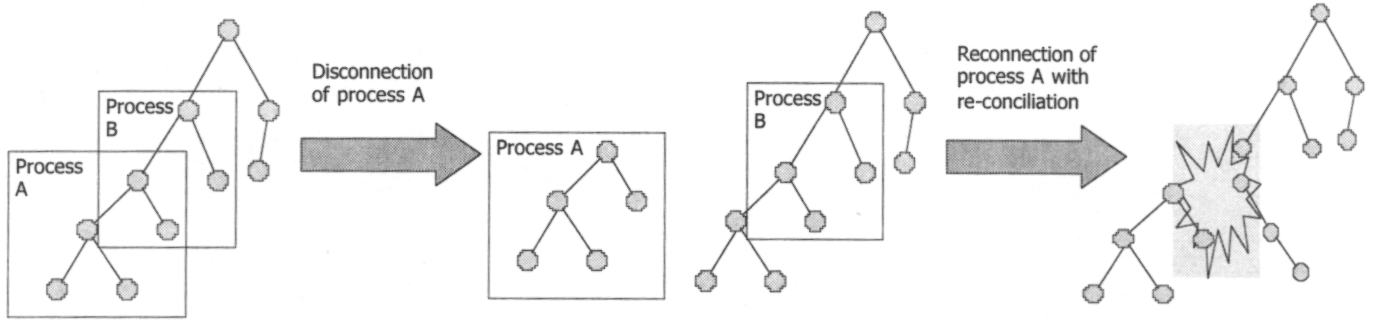
Process B
Process A
Disconnection of process A
Process A
Process B
Reconnection of process A with re-conciliation

Figure 4 Connections and disconnections on XMIDDLE trees

3.4.1 WorkSpaces

WorkSpaces (Tolksdorf, 2002b) combines workflow concepts with standard Internet technologies. The documents involved in the workflow are assumed to use application-specific markup languages expressed in XML. A workflow is composed of *steps* which are represented as XSL rules that are executed by an extended XSL processor, the WorkSpaces engine. It reads such a step and tries to retrieve the respective input document. It then applies transformations to the matching document found to generate the output document. The medium used to store all XML documents is XMLSpaces described in Section 3.3.1.

Figure 5 shows the flow of XML documents in the system. First a step description is retrieved with an in-operation that asks for a document that validates the step-DTD (interactions 1 and 2). Then the document to which the step should be applied is retrieved (interactions 3 and 4). This document is specific to a running instance of a workflow. This instance is represented by a unique identifier which is marked as an attribute of an XML element. This identifier is used during matching to find the specific document. After the step is performed, the resulting document is put into the repository by an out (interaction 5).

There are several classes of step. *Automatic steps* are pure document transformations and require only activity of some transformation component within the system. *External steps* involve applications that take a document as input, let the user perform some activity on it, and generate an output document. *User steps* are performed by a user without any support from the system. *Coordination steps* only coordinate the flow of work. Workflow procedures describe temporal and causal dependencies among activities represented as steps. The management of these dependencies is the central issue for any workflow system.

Steps are not specified individually. The whole workflow is represented as a graph of steps using the *WorkSpaces Coordination Language*, WSCL. WSCL is, again, an XML language and is based on the Workflow Process Description Language as defined by the WfMC in the Interface 1 of the Reference Model (Workflow Management Coalition, 1998).

In order to execute such a workflow, the WSCL description must be transformed into individual steps first. One can consider the workflow graph as the "program" written in a higher-level language and the individual steps as "instructions" as in microprocessors. The necessary "compilation" of the graph into steps is the transformation of one XML document into a set of XML documents. In WorkSpaces, it is called a *meta-step*. The compiler itself is another XSL script that is started by the workflow designer.

The unique distinctions of this approach from other workflow management systems with proprietary workflow engines are universal accessibility and ease of deployment due to Internet standards, and support for distribution and uncoupled operation due to coordination technology.

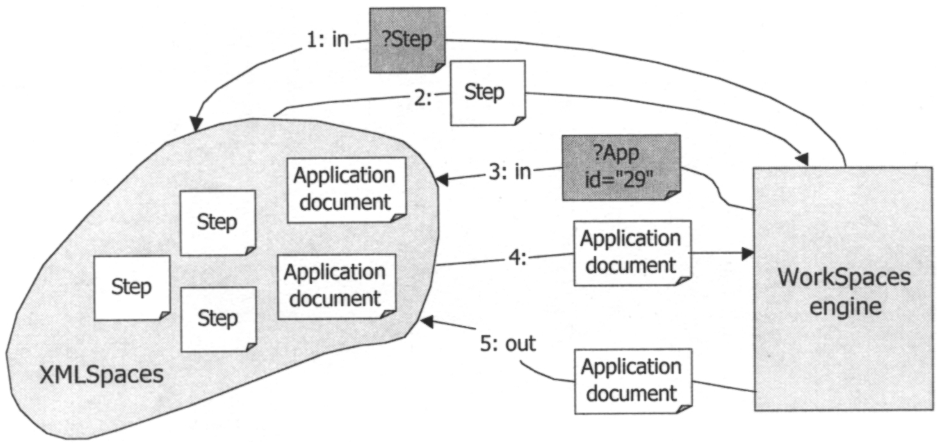

Figure 5 Access to documents in WorkSpaces

The example scenario above would be implemented in WorkSpaces as a workflow. The documents considered would represent the respective accounts in some XML-grammar, just as with the XMLSpaces example scenario. The coordinator component, however, would be "implemented" by a workflow of several steps that access the accounts by matching the account documents and by the selection of one of three branches in a so-called SPLIT-step (which is a coordination step) of the workflow depending on the current balances.

3.4.2 Other approaches

There is not much fully XML-integrated middleware such as WorkSpaces. With some limitations, one could consider XML-based scripting languages as middleware. Currently, two scripting languages with both the script and the data manipulated represented as XML are offered: XSL by the World Wide Web Consortium (2001b) and XML Script (DecisionSoft Ltd, 2001). While XSL is a transformation language for XML trees with strong declarative influences, XML Script is a rather traditional imperative scripting language. Both take an XML document as input and generate an output document as the result of the computation.

However, both offer no support for coordinating multiple activities. Thus their middleware service capabilities are very limited.

The Agent Definition Format (ADF) (Lange *et al.*, 1999) is slightly more powerful. It offers a way to specify agents in an XML representation. Agents have their own state and coordinate with others using HTTP communication. Calls to other agents' functionalities are encoded as parts of the URLs that reference agents. The underlying model of coordination is again client-server. Moreover, the coordination behaviour of document agents is totally mixed with their computational behaviour, thus providing no separation between computation and coordination.

3.5 Discussion

The above analysis has identified the main features – as well as the main limitations – of several middleware systems for XML-centric applications. The results of the analysis can be summarised as follows.

- Displets is the most suitable system for the definition and implementation of document agent applications, in that it enables one to embed behaviour in XML documents and enables this behaviour to directly manipulate the XML data they represent. Unfortunately, the displet approach is too static to meet the needs of open coordinated applications, in that it does not enable the dynamic definition of coordination patterns, which have to be statically hardwired into documents.
- MARS-X is particular suited for the definition of complex coordination patterns, even dynamically, in the access and manipulation of shared XML documents by mobile agents. However, it restricts the application to using Java agents, and therefore limits the possibility of defining coordinable document agents directly manipulating XML data.
- XMIDDLE coordinates activities over XML documents in the presence of mobility, but the possibility of defining suitable coordination laws is very limited.
- WorkSpaces provides more uniformity than the above systems, by exploiting XML both at the level of application agents and at the coordination level: XML document agents execute in the context of a common XMLSpaces, where also the definition of the coordination patterns (i.e. of the workflow rules) can be expressed in terms of XML documents. Still, WorkSpaces lacks an explicit support for mobility and – being mainly oriented to workflow applications – it is not clear whether it would be general-purpose enough to meet the need of any kind of multi-component application.

In an ideal scenario we envision that a suitable middleware is available that integrates the best features of all the systems analysed in this paper. These include the capabilities of:

- directly handling, at the application level, the activities of XML document agents, as in displets;
- making coordination activities occur in terms of manipulations of (portions of) shared XML documents, as in MARS-X, XMIDDLE and XMLSpaces;

- being flexible enough to support user-defined XML grammars and relations among them as in XMLSpaces;
- effectively handling mobility and associated issues, as in XMIDDLE;
- enabling the ruling of the coordination activities between application-level document agents in a dynamic way, as in MARS-X; and
- expressing not only document agents' behaviour but also the laws ruling their coordination activities in term of XML documents and XML rules, as in WorkSpaces.

In our opinion, the large amount of research efforts currently carried on in the area, including ours, will lead soon to the definition of further XML-centred coordination infrastructures, closely approaching our ideal requirements, and possibly defining solutions to a number of additional open research issues.

4 Open research directions

In addition to the need of defining a suitable coordination middleware, as from Subsection 3.5, there are several other issues that, in our opinion, need to find suitable solution for XML document agent applications to be effectively engineered and developed.

First of all, there may be the need of defining new "document-oriented" computational models, able to take into account and somehow formally analyse properties of coordinated applications based on XML document agents. Such models should take into account that the execution of a document-centric application can be better perceived in terms of manipulations on XML data rather than, as in more traditional computational models, in terms of state transitions in processes. A promising approach in that direction is represented by the work of Luca Cardelli (1999) on semi-structured computation. The basic intuition is that not only can manipulations of XML documents be represented in terms of a few basic tree transformations, but also the execution of a mobile computation can be modelled the same way, thus leading to a uniform model of XML document agents' computations in a mobile setting.

Strictly related to the above problem is the issue of handling with proper abstractions and middleware infrastructures the presence of mobility. In today's application scenarios, mobility comes in different forms, i.e. the logical mobility of software components migrating over different sites during their lives (as may be the case of an XML document agent being transferred from one site to another) and the physical mobility of devices such as laptops and PDAs (which necessarily implies mobility of the software components and of the documents installed on such devices) (Picco *et al.*, 2000). Thus a proper coordination middleware should be able to deal with both types of mobility in a uniform way, to diminish the overall complexity of application design. In addition, it is being recognised that handling mobility in an open and dynamic world requires application components with the capability of explicitly handling changes in the surrounding computational environment, i.e. in the "context". Accordingly, the concept of context and of context-aware computing must be properly supported by a coordination middleware (Cabri *et al.*, 2002).

A further promising research issue relates to the fact that, more and more, Web-based applications – and so document agent applications – tend to resemble, in their architecture, human and social organisations. This is mainly due to the fact that, first, applications are often intended to support the activities of some real-world organisations, and mimic them accordingly and, second, the autonomy of application components invites considering them in terms of individuals playing specific roles in an ensemble rather than in terms of components in charge of providing mechanical functionalities. Therefore, those software engineering approaches exploiting the research results of organisational management and social sciences may provide, in the near future, effective methodologies for the design and development of Web-based document agent applications and useful guidelines for the development of coordination middleware (Zambonelli *et al.*, 2000; Omicini, 2001).

As a final note, we think that the dramatic increase of embedded computer-based and software components, envisioning a future where uncountable multitudes of interconnected autonomous and mobile components will always execute and interact with each other, will challenge most of today's approaches to software development as well as today's model of coordination and associated middleware (Tennenhouse, 2000; Tolksdorf, 2002; Zambonelli & Parunak, 2002).

5 Conclusions

XML is emerging as a suitable technology for representing not only data but also computations, leading to the concept of XML document agents, as autonomous software components embedding both data and behaviour. However, for complex applications to be developed in terms of XML document agents, suitable middleware is needed to effectively enable and rule the coordination activities of application components.

In this paper, we have analysed several middleware systems proposed and implemented so far that, to different extents and with different architectural solutions, aim at providing a coordination framework for a world of XML document agents. The analysis, performed with the help of a simple example scenario, has outlined the main features and limitations of these systems, and has permitted us to sketch the requirements for an "ideal" coordination middleware for XML document agents.

Our current research focus deals with understanding how to make the identified ideal middleware an implemented system, although these may require facing further design and implementation issues not identified by this paper. In addition, we feel that it is important to investigate further general issues related to the engineering of complex distributed applications. These issues include the proper modelling and handling of mobility (Cardelli 1999; Picco *et al.*, 2000) and openness (Zambonelli *et al.*, 2000) in software systems, and the effective engineering very large-scale and embedded applications (Tennenhouse, 2000).

References

Andrade, LF and Fiadeiro, JL, 1999, "Interconnecting objects via contracts" *Proceedings of the 2nd International Conference on the Unified Modeling Language* (UML'99) 566–583.

Bompani, L, Ciancarini, P and Vitali, F, 1999, "Active documents in XML" *ACM SigWeb Newsletter* **8**(1) 27–32.

Cabri, G, Leonardi, L and Zambonelli, F, 2000a, "MARS: a programmable coordination architecture for mobile agents" *IEEE Internet Computing* **4**(4) 26–35.

Cabri, G, Leonardi, L and Zambonelli, F, 2000b, "Mobile-agent coordination models for Internet applications" *Computer* **33**(2) 82–89.

Cabri, G, Leonardi, L and Zambonelli, F, 2001, "XML dataspaces for mobile agent coordination" *Journal of Applied Artificial Intelligence* **15**(1) 35–58.

Cabri, G, Leonardi, L and Zambonelli, F, 2002, "Engineering mobile agent applications via context-dependent coordination" *IEEE Transactions on Software Engineering* **28**(11) 1039-1055.

Cardelli, L, 1999, "Semistructured computation" *Proceedings of DBLP 99* 1-16.

Cardelli, L and Gordon, AD, 2000, "Mobile ambients" *Theoretical Computer Science* **240**(1) 177–213.

Chang, S and Znati, T, 2001, "Adlet: an active document abstraction for multimedia information fusion" *IEEE Transactions on Knowledge and Data Engineering* **13**(1) 112-123.

Ciancarini, P, Omicini, A and Zambonelli, F, 1999, "Coordination technologies for Internet agents" *Nordic Journal of Computing* **6**(3) 215–240.

Ciancarini, P, Vitali, F and Mascolo, C, 1999, "Managing complex documents over the WWW: a case study for XML" *IEEE Transactions on Knowledge and Data Engineering,* **11**(4) 629–638.

DecisionSoft Limited, 2001, "XML Script" (http://www.xmlscript.org/) last checked 29 August 2001.

Dourish, P, Edwards, W, Howell, J, La Marca, A, Lamping, J, Petersen, K, Salisbury, M, Terry, D and Thornton, J, 2000, "A programming model for active documents" *Proceedings of the ACM Symposium on User Interface and Software Technology* 1-5.

Gaines, B and Shaw, M, 1999, "Embedding formal knowledge models in active documents" *Communications of the ACM* **42**(1) 57–74.

Jennings, N and Wooldridge, M, 1999, "Intelligents agents: theory and practice" *The Knowledge Engineering Review* **10**(2) 115-152,

La Forge, B, 2001, "The JXML home page" www.jxml.com.

Lange, D, Hill, T and Oshima, M, 1999a, "A new internet agent scripting language using XML" *Proceedings of the AAAI-99 Workshopon AI in Electronic Commerce* 115-128.

Malone, TW and Crowston, K, 1994, "The interdisciplinary study of coordination" *ACM Computing Surveys* **26**(1) 87–119.

Mascolo, C, Capra, L, Zachariadis, S and Emmerich, W, 2002, "XMIDDLE: a data-sharing middleware for mobile computing" *Personal and Wireless Communications* **21**(1).

Omicini, A, 2001, "SODA: societies and infrastructures in the analysis and design of agent-based systems" *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*.

Omicini, A, Zambonelli, F, Klusch, M and Tolksdorf, R (eds), 2001, *Coordination of Internet Agents: Models, Technologies, and Applications* Springer-Verlag.

Ossowski, S, 2001, "Constraint-based coordination of autonomous agents" *Electronic Notes in Theoretical Computer Science* **48**.

Picco, GP, Roman, GC and Murphy, A, 2000, "Software engineering and mobility: a roadmap" *Proceedings of the 22nd International Conference on Software Engineering* (ICSE 2000) 241-258.

Ricci, A, Omicini, A and Denti, E, 2002, "Activity Theory as a framework for MAS coordination" *3rd International Workshop "Engineering Societies in the Agents World"* (ESAW'02) 195–208.

Richard M Adler, 1995, "Emerging standards for component software" *IEEE Computer* **28**(3) 68–77.

Satoh, I, 2000, "MobiDoc: a framework for building mobile compound documents" *Proceedings of the 2nd International Symposium on Agent System, Applications, and Mobile Agents* (ASAMA 2000) 19-28.

Stauch, M and Tolksdorf, R, 2001, "Design and implementation of an XSL-T and XML-based workflow system" *Proceedings of XML Europe 2001* 229-241.

Tennenhouse, D, 2000, "Embedding the Internet: proactive computing" *Communications of the ACM* **43**(5) 43.

Tolksdorf, R, 2000a, "Coordination technology for workflows on the Web: Workspaces" *Proceedings of the Fourth International Conference on Coordination Models and Languages* (COORDINATION 2000) 36–50.

Tolksdorf, R, 2000b, "Coordinating work on the Web with Workspaces" *Proceedings of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises WET ICE 2000* 225-229.

Tolksdorf, R, 2000a, "Models of coordination" *Engineering Societies in the Agent World First International Workshop* (ESAW 2000) 78–92.

Tolksdorf, R, 2002b, "Workspaces: A Web-based workflow management system" *IEEE Internet Computing* **6**(5) 18–26,

Tolksdorf, R and Glaubitz, D, 2001, "Coordinating Web-based systems with documents in XMLSpaces" *Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems* (CoopIS 2001) 356–370.

Tolksdorf, R and Glaubitz, D, 2001, "XMLSpaces for coordination in Web-based systems" *Proceedings of the Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (WET ICE 2001) 171-180.

Tolksdorf, R and Stauch, M, 2001, "Using XSL to coordinate workflows" *Proceedings of Kommunikation in Verteilten Systemen* (KiVS) 127–138.

Workflow Management Coalition, 1998, "Interface 1: process definition interchange process model" http:/ /www.wfmc.org.

World Wide Web Consortium IETF, 2002, "XML signature" (http://www.w3.org/Signature/) last checked 25November 2002.

World Wide Web Consortium, 2001a, "XML protocol activity" (http://www.w3.org/2000/xp/) last checked 29 August 2001.

World Wide Web Consortium, 2001b, "XSL Transformations (XSLT) Version 1.0" (http://www.w3.org/TR/xslt) last checked 29 August 2001.

World Wide Web Consortium, 2002, "XML encryption" (http://www.w3.org/Encryption/2001/) last checked 25 November 2002.

Zambonelli, F and Van Dyke Parunak, H, 2002, "From design to intention: signs of a revolution" *Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent* Systems 445-446.

Zambonelli, F, Jennings, NR and Wooldridge, M, 2000, "Organizational abstraction for the analysis and design of multiagent systems" in P Ciancarini and M Wooldridge (eds) *Agent-Oriented Software Engineering* Springer-Verlag 235–251.

Zambonelli, F, Jennings, NR, Omicini, A and Wooldridge, M, 2000, "Agent-oriented software engineering for internet applications" in A Omicini, F Zambonelli, M Klusch and R Tolksdorf (eds) *Coordination of Internet Agents: Models, Technologies, and Applications* Springer-Verlag 320–341.