

A Survey on Coordination Middleware for XML-centric Applications

Paolo Ciancarini[†] Robert Tolksdorf[‡] Franco Zambonelli^{*}

October 10, 2002

[†] Dipartimento di Scienze dell'Informazione,
Università of Bologna, Mura Anteo Zamboni, 40126 Bologna, Italy.
cianca@cs.unibo.it

[‡] Freie Universität Berlin,
Institut für Informatik,
Takustrasse 9, D-14195 Berlin, Germany.
tolk@inf.fu-berlin.de

^{*} Dipartimento di Scienze e Metodi dell'Ingegneria,
Università di Modena e Reggio Emilia,
Via Allegri 13 – 42100 Reggio Emilia, Italy.
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 characterize 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-centered. 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-support 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-centered computing models toward *document-centric computing models*, specifically centered around the concepts of *active and mobile documents* [8, 16, 24, 12]. On the one hand, documents will be no longer only the passive part of a software system but, instead, will integrate active behaviors (e.g., internal code and threads) and will be able to handle themselves and to coordinate with other application components [8, 16]. On the other hand, such behaviors will include the capability for documents of moving themselves over a network [8, 24]. The success of XML technologies [30, 31] concurs to accelerate this trend towards active and mobile documents, by providing easy document processing and data portability [10, 11]. 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 [9, 26] are needed to orchestrate the activities of components and active documents (e.g., to maintain consistency across related documents and to synchronize 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 [3], but also to exploit XML-based active documents as the core of a general-purpose coordina-

tion middleware [5, 18, 27], i.e., as the active artifacts in charge of mediating and ruling coordination in a generic multi-component application [23].

The contribution of this paper is to analyze 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 characterize 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 organized 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 XML-centric coordination middleware, supporting and exploiting XML active documents at different levels. Section 4 discusses some open research issues. Section 6 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 documents must have some kind of contents (e.g., data, text, images, music), representing the very information the document is intended to store. In addition, any document must also include some sorts of structural information (e.g., index, tags, formatting commands) used by external, document-processing entities (e.g., humans, search engines, browsers, printers, etc.) required to understand and elaborate the content. Thus, in general:

(passive) document = content + 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 one: e.g., a H1 header in a files specify both that the enclosed text is a section title and that it has be properly emphasized when rendered in the browser. Still, HTML does not fully associate the procedural behavior with the document, leaving different browsers free to render, e.g., H1 tags, in different ways.

One 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 to any predefined procedural behavior [30]. Procedural information, for the rendering/manipulation of XML documents can be associated to the XML document using a companion XSL stylesheet [31]. The XSLT language component of XSL allows 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 behaviors. Moreover, it is also possible to use any programming language, even a Turing-equivalent one, instead of XSLT and XSL-FO, to specify how a document can be manipulated and rendered. In this way, a document could be associated to any kind of Turing-expressible behavior. 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 behaviors, other than content and structure. In other words:

(active) document = content + structure + behaviors.

2.1 Towards Document Agents

When a document encapsulate document-specific behavior, determining how the document itself has to be handled, it can no longer be considered simply a document. Instead, such an active document can rather be assimilated to a software component or – in some cases – even to a software agent [15]. In particular, we can recognize two different classes of documents that can be considered active.

- When the internal behavior 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 that, its nature is simply *reactive*: the internal activity of the document is triggered by requests of accessing the document. A large number of examples of reactive documents can be found both in the literature [14, 12] 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 document¹ or, in the most simplistic case, a Java Applet included in the document), the document can exhibit *proactive* behavior and, as that, it can be somehow assimilated to a software agent [15]. For this class of documents, we use the term *document agents* to characterize their twofold nature of documents and of software agents.

Several research works have recently suggested interesting applications of document agents. For instance, a proactive agenda can be able of alerting users and proactively re-organizing the schedule of a meeting by interacting with the proactive agendas of the other users involved in the meeting [24]. A proactive Web-based document can look in the Web for further related documents of potential interest for the user [8].

2.2 Mobility and Coordination

If active documents can be assimilated to software components – whether objects or software agents – then they can be used as building blocks for the development of complex distributed applications. However, this requires providing documents with two additional features: the capability of transferring themselves over the nodes of a network and the capability of coordinating their actions with other active document.

The first feature, mobility, is intrinsic in the very concept of information and, so, of documents: a document is created to transfer and move around information. By adopting open data formats, like XML, mobility of passive documents is automatically achieved. However, when the document may include behaviors and threads of execution, enabling it to move from one place to another – as a mobile

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

agent [6] – requires also code portability as well as the presence of a software infrastructure – i.e., of a middleware – enabling and supporting active document mobility [24].

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 assume 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 can express more complex and dynamic patterns, as it can be the case of an active agenda trying to re-schedule a meeting. In fact, as it happens in multiagent applications [15], 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 RPC infrastructure to support simple client-server forms of interaction is not enough).

3 Coordination Middleware & 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 [17]. As such, coordination middleware is intended to integrate functionalities and services as needed to enable and rule [21, 26] 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 activi-

ties are started and synchronized². The heterogeneity found ranges from RPCs, object invocations, component usage to agent interaction with different characteristics such as one-to-one or one-to-many communication and synchronization. In addition, modern middleware has to effectively provide support for mobility of application components, users, and devices.

3.1 Document-Centric Middleware

With the beginning of the 90ies, several companies tried to establish standards for middleware architectures supporting active documents and their coordination [1].

These middleware architectures – grounded in the works of distributed object applications and middleware, like CORBA – established notions of documents into which “components” or “objects” 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 source, different services serving different components were integrated into an application represented as a document.

The two major players in the middle of the 90ies 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 used.

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 Microsofts COM and .NET frameworks. The latter, however, has definitely lost the document-orientation of its ancestors, being considered a pure object-oriented middleware. With *CORBA*, a component- and object-standard was established at the same time that found great acceptance as a general-purpose middleware in the mid 90ies. However, CORBA did not (and does not currently) incorporate a strong document metaphor.

With XML, document-centric abstractions are revitalized, and several interesting middleware systems for coordination have been proposed since the late

²In its general terms, middleware is intended to support a larger set of services, there included lookup services, directory services, load balancing services, transactions, etc. However, when speaking of “coordination middleware” we explicitly restrict our focus to coordination services.

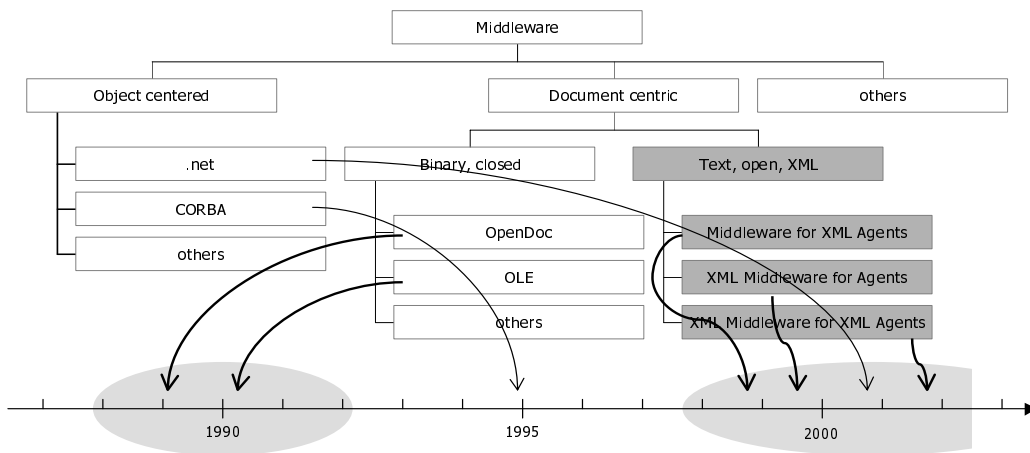


Figure 1: Document-centric Middleware

90ies 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):

- they can offer services not based on XML for the use of XML-based document agents;
- at the other extreme, they can offer coordination services based on XML technologies and XML active documents; or,
- they can adopt a fully integrated approach for XML-based coordination services in a world of XML document 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 [2]. 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 lesser money than requested, the transaction must 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 for the user and, more relevant to our purposes, it has to be able to coordinate (or support) actions such as evaluating whether a transfer is necessary from different account and providing for these withdrawal operations. Central issues for the coordination middleware used here is to provide its service in a rather transparent manner and to integrate A and B which 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.

3.2.1 Displets

The basic idea of the *Displets* approach [10] is to provide an active document environment, where XML documents can be enriched with application-specific behavior 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, or 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 visualize 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 behaviors, in terms of Java classes, to XML documents. An XML transformation stylesheet 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 associated specific behaviors to specific portion of the tree. The new XML document obtained from this transformation can thus have become an active document. There, Java classes determine the behavior of the document when manipulated by external applications (e.g., browsers and printers), and runnable threads can determine the autonomous behavior of the document when launched for execution.

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

In the example scenario, it is possible to think of having a client document agent in charge of receiving inputs from the client, storing it internally in XML format, and of rendering back to the client the XML data reporting the results of the account operations to it. All these operations are handled via proper behavior attached to the document agent. In addition, it is possible to attach to the client document agent the behavior needed to coordinate – i.e., to negotiate withdrawal – with the document agents devoted to manage 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 behaviors, which include the behaviors devoted to the implementation of coordination patterns, are hardwired into documents at compile time (i.e., during the transformation of the XML document into an active one). This can make it hard to exploit Displets in open environments and in mobile setting, where a document can move across different sites and needs to interact with different documents according to dif-

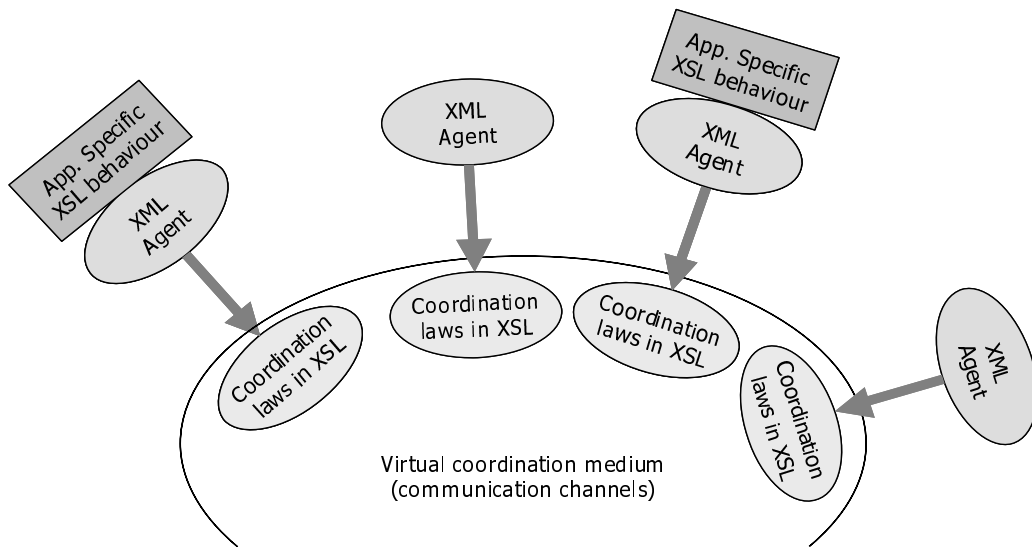


Figure 2: The Displets Approach

ferent coordination patterns. For the example scenario, changes to the policies adopted by the banks to handle accounts and withdrawal would require a change in the coordination behavior 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 behavior. For instance, JXML [13] is one of such proposals. However, most of these frameworks are quite limited with regard to multi-document coordination. In most of the cases, coordination between documents simply amounts at 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 [8]. There, the basic idea is to enrich Web-based documents (XML, but not necessarily) with a proactive declarative behavior. The goal is to make a document able to autonomously look in the network for related documents. To this end, the Adlets middleware enable a document to proactively move across the Internet (as if it were a mobile agent) and to coordinate itself with other documents to discover

relations between documents and, eventually, 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 artifacts 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 synchronize 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 set of typed fields. In particular, there is the need of capturing any needed application-specific data structure 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 standardized in some way, so that data-items can be exchanged between entities that have different design-origins. XML fulfills all those criteria.

XMLSpaces [28] is an extension to the Linda model which serves as middleware for XML. In XMLSpaces, XML documents are fields within the coordination space. Thus, ordinary tuples are supported in terms of simple documents, while complex XML documents can be simply perceived as simple tuples with a single field (i.e., the root of the document).

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 the one of Linda. While the basic pattern-matching relations shown in Table 1 are automatically supplied by 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	Fulfills an XPath expression
XQL	Fulfills an XQL expression
AND	Fulfills two matching relations
NOT	Does not fulfill matching relation
OR	Fulfills one or two matching relations
XOR	Fulfills one and only one matching relation

Table 1: Matching relations in 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 separated 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 unauthorized parties by additional encryption following the XML Signature framework. 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 [5], implemented as an extension of the MARS architecture [4], defines a Linda-like middleware model to enable agent (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 docu-

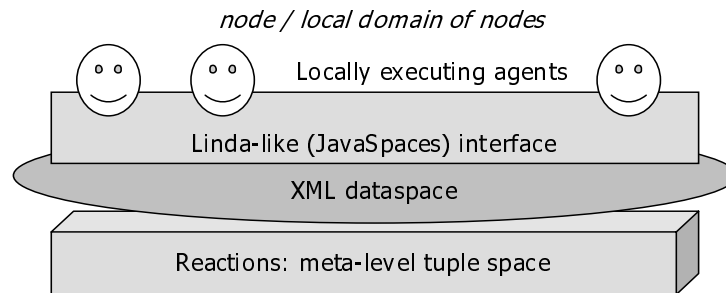


Figure 3: The MARS-X Architecture

ment describing bank accounts with data values tagged as *name*, *number*, *amount*, can 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 synchronization over tuple occurrences. Specifically, MARS-X provides agents a JavaSpace interface to access to 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 to different XML dataspaces: when an agent arrives in a node, it is automatically provided with the reference to a MARS-X tuple space interface associated to the XML dataspace.

A peculiar characteristic of MARS-X dataspaces is that their behavior in response to agent accesses can be programmed to implement specific access methods and specific synchronization and coordination patterns. Both administrators and mobile agents (the latter in a quite restricted way) can install in an XML dataspace reactions associated to specific access operations, performed by specific agents, with specific parameters. These reactions override the default behavior of the performed operations and, for instance, can modify the result of the operations they are associated with, can manipulate the content of the XML dataspace, and can 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 agent can access the dataspace always with the same limited set of operations, the dataspace itself can react to

this 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 occur via data exchanged by mean of the dataspace, the behavior of the dataspace can be used to globally rule the activities of multiagent applications.

Coming back to the example scenario, and by assuming the availability of the MARS-X middleware, one can think that each bank makes available to agents an XML dataspace with data account. 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 manage account data, that will take care of actually performing the transaction and sending back the result 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 for withdrawing portion of the total sum from different accounts. An example is the situation when the client agent requests an amount from account A which is more than the current balance of A.

The reactions in the dataspace then can cause another agent to become active. It 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 client agent accordingly 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 behavior of the XML dataspace and transparently to agent 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 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 a 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 dataspace. This would require the client agent to directly manipulate and represent its world in XML

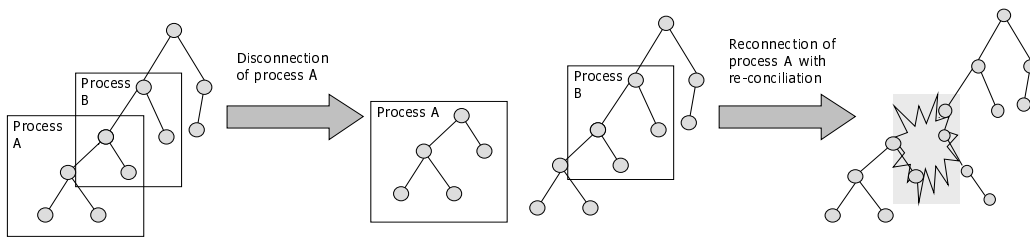


Figure 4: Connections and Disconnections on XMIDDLE Trees

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 analyzed later in this section.

3.3.3 XMIDDLE

The XMIDDLE middleware [19] implements a coordination architecture somewhat similar to the MARS-X one: 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 a specific architectural solutions to make it a suitable middleware for mobile computing and, specifically, for ad-hoc network.

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 setting, where processes/agent can disconnect and re-connect 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-tree). When the agent re-connects, or is in reach again, the global tree has to be re-constructed, 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 reconstruct 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 the one required to withdraw partial amounts of money from different account. 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 prominently, this is the current XML Protocol activity by the World Wide Web consortium [30]. 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 the communication amongst objects, for example, this boils down to represent 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. It is unclear whether this activity will aim at providing such a higher-level model, or puts the technical integration of several existing solutions into its center.

3.4 Self-contained XML Middleware

XML is a standard for representing data in networked documents. However, as seen in the previous subsection, the specification of activity can also be expressed as an active document. Thus, if scripts etc. 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 important, the scripts them-self then become first-class data objects

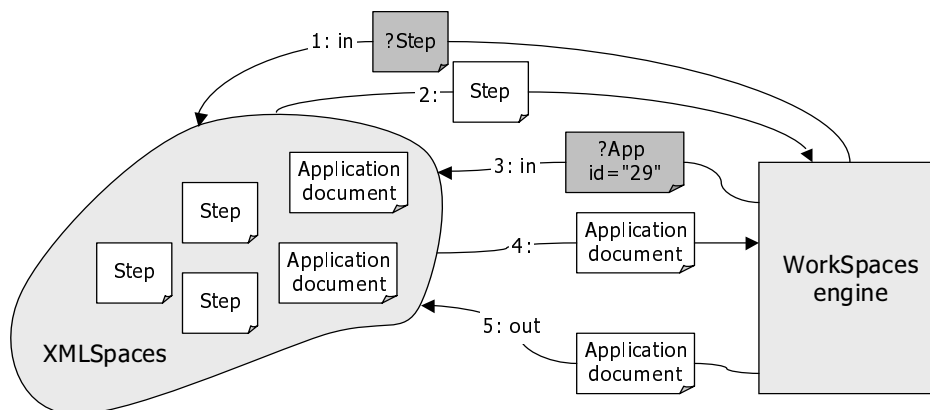


Figure 5: Access to documents in WorkSpaces

for the middleware. The same infrastructure used for applications can be used to manage the programs them-self.

3.4.1 WorkSpaces

WorkSpaces [27] combines workflow concepts with standard Internet technology. 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, tries retrieve the respective input document and to apply transformations on the match that 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 to the step-DTD (interactions 1 and 2). Then, the document to which the step should be applied to is retrieved (interactions 3 and 4). This document is specific to a running instance of a workflow. It is matched to some specific value for a unique identifier. After the step is performed, the resulting document is put into the repository with an out (interaction 5).

There are several classes of steps. *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 by a 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 [29].

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.

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 in a suited manner 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 are not many 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 [31] and XML Script [11]. While XSL is a transformation language for XML trees with strong declarative influences, XML Script is a rather traditional imper-

ative 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 [16] 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 in the URLs of references to agents. The underlying model of coordination is again client/server. Also, the coordination behavior of document agents is totally mixed with their computational behavior, 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 summarized as follows:

- Displets is the most suitable system for the definition and implementation of document agent applications, in that it enables to embed behavior in XML documents and enables this behaviors 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 dynamic definition of coordination patterns, which have to be statically hardwired into documents.
- MARS-X is very 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 use Java agents, and therefore limits the possibility of defining coordinable document agents directly manipulating XML data.
- XMIDDLE is coordinating activities over XML document 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 XML documents. Still, Workspace lacks an explicit support for mobility and – being mainly oriented to workflow applications – is not clear whether it would be general-purpose enough to meet the need of any kind of multi-component application.

The ideal scenario we envision is the one in which a suitable middleware is available integrating the best features of all the systems analyzed 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 to support user-defined XML grammars and relations amongst 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;
- expressing not only document agents behavior 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, there included ours, will lead soon to the definition of further XML-centered 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 analyze 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 on semi-structured computation [7]. The basic intuition is that not only manipulations of XML documents can be represented in terms of a few basic tree transformations, but also the execution of a mobile computation can be modeled as that, 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 into different forms, i.e., the logical mobility of software components migrating over different sites during their lives (as it may be the case of an XML document agent being transferred from a 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) [22]. 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 recognized that handling mobility in an open and dynamic world requires application components 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 [6].

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 organizations. This is mainly due to the fact that *(i)* often, applications are intended to support the activities of some real-world organizations, and mimic them accordingly; *(ii)* 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 organizational 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 [32, 20].

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 be always executing 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 [25, 26, 33].

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 behavior. 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 analyzed several middleware systems proposed and implemented so far that, to different extent 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 modeling and handling of mobility [7, 22] and openness [32] in software systems, and to the effective engineering very-large scale and embedded applications [25].

Acknowledgments

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

References

- [1] Richard M. Adler. Emerging standards for component software. *IEEE Computer*, 28(3):68–77, March 1995.
- [2] Luis Filipe Andrade and Jose Luis Fiadeiro. Interconnecting Objects via Contracts. In *Proceedings 2nd International Conference on the Unified Modeling Language (UML'99)*, volume 1723 of *LNCS*, pages 566–583. Springer, 1999.
- [3] L. Bompani, P. Ciancarini, and F. Vitali. Active Documents in XML. *ACM SigWeb Newsletter*, 8(1):27–32, 1999.
- [4] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A Programmable Coordination Architecture for Mobile Agents. *IEEE Internet Computing*, 4(4):26–35, July/August 2000.
- [5] G. Cabri, L. Leonardi, and F. Zambonelli. XML Dataspaces for Mobile Agent Coordination. *Journal of Applied Artificial Intelligence*, 15(1):35–58, January 2001.
- [6] G. Cabri, L. Leonardi, and F. Zambonelli. Engineering Mobile Agent Applications via Context-Dependent Coordination. *IEEE Transactions on Software Engineering*, 28(11), November 2002.
- [7] L. Cardelli. Semistructured Computation. In *Proceedings of DBLP 99*. 1999.
- [8] S. Chang and T. Znati. Adlet: an Active Document Abstraction for Multimedia Information Fusion. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 2001.
- [9] P. Ciancarini, A. Omicini, and F. Zambonelli. Coordination Technologies for Internet Agents. *Nordic Journal of Computing*, 6(3):215–240, 1999.
- [10] P. Ciancarini, F. Vitali, and C. Mascolo. Managing complex documents over the WWW: a case study for XML. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):629–638, July/August 1999.
- [11] DecisionSoft Limited. XML Script. <http://www.xmlscript.org/>. Last checked Aug. 29 2001.

- [12] P. Dourish et al. A Programming Model for Active Documents. In *Proceedings of the ACM Symposium on User Interface and Software Technology*, 2000.
- [13] B. La Forge. The jxml home page. *www.jxml.com*, 2001.
- [14] B. Gaines and M. Shaw. Embedding Formal Knowledge Models in Active Documents. *Communications of the ACM*, 42(1):57–74, 1999.
- [15] N. Jennings and M. Wooldridge. Intelligents Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2), 1999.
- [16] D. Lange, T. Hill, and M. Oshima. A New Internet Agent Scripting Language Using XML. In *Proc of AAAI-99 Workshop on AI in Electronic Commerce*, 1999.
- [17] T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.
- [18] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: a Data Sharing Middleware for Mobile Computing. *Personal and Wireless Communications Journal*, 21(1), 2002.
- [19] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis, and Wolfgang Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications*, To appear.
- [20] A. Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, volume 1957 of LNCS. Springer Verlag, 2001.
- [21] S. Ossowski. Constraint-based Coordination of Autonomous Agents. *Electronic Notes in Theoretical Computer Science*, 48, 2001.
- [22] G. P. Picco, G. C. Roman, and A. Murphy. Software Engineering and Mobility: A Roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, 2000.
- [23] Alessandro Ricci, Andrea Omicini, and Enrico Denti. Activity Theory as a framework for MAS coordination. In Paolo Petta, Robert Tolksdorf, Franco

- Zambonelli, and Sascha Ossowski, editors, *3rd International Workshop "Engineering Societies in the Agents World" (ESAW'02)*, pages 195–208, Madrid, Spain, 16–17 September 2002. Universidad Rey Carlos, Madrid.
- [24] I. Satoh. MobiDoc: A Framework for Building Mobile Compound Documents. In *Proceedings of the 2nd International Symposium on Agent System, Applications, and Mobile Agents (ASAMA 2000)*. 2000.
- [25] David Tennenhouse. Embedding the Internet: proactive computing. *Communications of the ACM*, 43(5):43, May 2000.
- [26] Robert Tolksdorf. Models of coordination. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agent World First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000*, number LNAI 1972, pages 78–92. Springer Verlag, 2000.
- [27] Robert Tolksdorf. Workspaces: A Web-Based Workflow Management System. *IEEE Internet Computing*, 6(5):18–26, 2002.
- [28] Robert Tolksdorf and Dirk Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFICIS International Conference on Cooperative Information Systems (CoopIS 2001)*, number LNCS 2172, pages 356–370. Springer Verlag, 2001.
- [29] Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model, 1998. <http://www.wfmc.org>.
- [30] World Wide Web Consortium. XML Protocol Activity. <http://www.w3.org/2000/xp/>. Last checked Aug. 29 2001.
- [31] World Wide Web Consortium. XSL Transformations (XSLT) Version 1.0. <http://www.w3.org/TR/xslt>. Last checked Aug. 29 2001.
- [32] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organizational Abstraction for the Analysis and Design of Multiagent Systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*. Springer-Verlag: Heidelberg, Germany, 2000.
- [33] F. Zambonelli and H. Van Dyke Parunak. From design to intention: Signs of a revolution. In *Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems*. Bologna, Italy, July 2002.