# Coordination Middleware for XML-centric Applications

Paolo Ciancarini
Dipartimento di Scienze dell'Informazione
University of Bologna
Mura Anteo Zamboni – 40126 Bologna – Italy
cianca@cs.unibo.it

Robert Tolksdorf
Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
FR 6–10 Franklinstr. 28/29
D-10587 Berlin, Germany
tolk@cs.tu-berlin.de

Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria
Univ. Modena & 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. It introduces the main concepts underlying active documents and XML Then, the paper goes into details about the problem of defining a suitable middleware architecture to effectively support coordination activities in applications including active documents and mobile agents, by specifically focusing on the role played by XML technologies in that context. According to a simple taxonomy, the characteristics of several middleware systems are analyzed and evaluated. This analysis enables us to identify the advantages and the shortcoming of the different approaches, and to identify the basic requirements of a middleware for XML-centric applications.

## 1. INTRODUCTION

The convergence of Information and Communication Technologies offers new opportunities for industry, research, and teaching, and it is pushing the development of novel appliances, applications, and services. People who are using these technologies are mostly interested in communicating or accessing *contents*, that is information transmitted and stored in form of electronic documents. There is a wide, ever-increasing range of Internet-based applications and services that are document-centric, meaning that they are made of components which agree on some document ontology to exchange structured data in form of documents complying with such an ontology. Several Internet applications deal with document exchanges: for instance, CSCW systems typically deal with accesses to shared workplaces or document spaces.

From a software design viewpoint, people are actively developing novel methods, tools and infrastructures for document-centric applications, because it is still unclear how they should be designed at a world-wide scale. Document-centric computing models are needed in order to study, compare and design these applications. In this context, we envision a trend toward computing models centered around the concept of active and mobile documents. On the one hand, documents may be not only the passive part of a software system but, instead, can integrate active behaviors and can be able to handle themselves and to coordinate with other application components. On the other hand, such behaviors can include the capability of moving themselves over a network. The success of XML technologies concur to accelerate this trend by providing easy document processing and data portability, that is, by facilitating the shift towards active document. However, for such a shift to become viable it necessary to clarify the role and the characteristics of the middleware that should support active document applications.

Since several applications are multi-components and multi-documents, there is the need of suitable middleware to support the related coordination activities. Interestingly, the definition of a coordination middleware offers the possibility for the exploitation of XML-based active documents in the framework at different levels. While the role of XML for defining documents can be either purely passive, namely structuring data, or behavioral, namely defining its rendering, it is also possible to exploit XML in middleware as an integral part of the underlying coordination framework and, say, use it to define the coordination space as well as the coordination laws in terms of active XML documents.

Our goal in this paper is analyzing the role that XML can play in modern coordination middleware for document agent applications. A very simple taxonomy is introduced to identify the possible exploitations of XML in that context. Several systems are analyzed and evaluated according to this taxonomy. By this, we identify the advantages and the drawback of the different approaches and identify several questions and problems that are currently unanswered by these systems and remain as future research challenges.

The remainder of this paper is organized as follows. Section 2 introduces active documents as document-agents. Section 3 describes some XML-based coordination middleware supporting the ontologies of document-agents. 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 (e-document for short) is a kind of data structure that applications can exchange and process. By definition a documents must have some kind of contents: Data, text, images, music, money, etc. In addition, any document has a representation and a structure. These are defined by codes like ASCII, tags, formatting commands, etc. Thus, we can say that a *(passive) document = content + representation.*



SAC 2002, Madrid, Spain

E-documents can contain meta-level or structural information used by external, document-processing entities (e.g., humans, search engines, or printers) to index or even "understand" its contents. A glossary in a book, or a a header of a BMP file are examples of meta-level information. Instead, some tags in HTML files or in Tex documents are used to define structural information. It is quite important to distinguish the declarative power of structural information from the procedural interpretation that is necessary to render or generically process a document according to its structure. For instance, the rendering rules of HTML documents are built in the browser: there is no specific behavior associated to a document, thus different browsers can handle differently different tags. In contrast, XML tags are not associated to any predefined behavior of external applications, thus they are purely declarative [24].

The above characteristic – together with its intrinsic portability – is one of the main driving forces in the increasing success of XML, intrinsicaly promoting a shift from passive to active document. In fact, the computational code associated to the rendering/manipulation of XML documents (or to XML document type) can be associated to the XML document using a companion XSL stylesheet [25]. The XSL-T language component of XSL allows to define by rules the tree-based manipulation of a document, whereas using the XSL-FO language components we can define the rendering behaviors. It is also possible to use a fully fledged programming language instead of XSL-FO: in this way a document can be associated to any kind of behavior expressable using a Turing-equivalent language. For instance, a document representing a program could be associated to some way of animating its own symbolic execution. We define such e-documents carrying their own behaviors as "active", in contrast with passive documents which rely upon applications to be manipulated. More precisely, an active document is defined as *(active) document = content + structure + behaviors.*

## 2.1 Towards Document Agents

When a document encaspulate document-specific behavior, determining how the document itself has to be handled (for instance by specifying document-specific behaviors related to rendering or to its structural manipulation), it cannot be longer considerd 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 [13]. There are 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, is nature is simply *reactive*. For this class of active documents, of which a large number of examples can be found both in the literature [12, 10] and in commerce, the internal activity of the document is triggered by requests of accessing the document;

When, instead, the document integrate autonomous threads of control, the document can exhibit *proactive* behavior and, as that, it can be somehow assimilated to a software agent. For this class of documents, we use the term "document agents" to characterize their twofolds nature of documents and of software agents.

Several research works recently suggest interesting applications of document agents. For instance, a proactive agenda can be able of alerting users and re-organizing the schedule of a meeting by interacting with the proactive agendas of the other users involved in the meeting [18]. A proactive Web-based document can look in the Web for further related documents of potential interest to a user [7].

## 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 a building blocks for the development of complex distributed applications. However, this may require 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/move information. By adopting open data formats, like XML, mobility of passive documents is automatically achieved. However, when the document, other than data, may include behaviors and threads of execution, to enable it to move from one place to another – as a mobile agent [4] – requires also code portability as well as the presence of a software infrastructure – i.e., of a middleware – enabling and supporting active document mobility [18].

The second feature, coordination, is necessary for the bulding 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. However, as soon as the application is built by making use of document agents, interactions and coordination activities are likely to express more complex and dynamic patterns, as it can be the case of an active agenda trying to re-schedule a meeting. Again, a suitable infrastructure is necessary to support coordination activities of document agents.

# 3. COORDINATION MIDDLEWARE & XML

*Middleware* is conceived as a software layer that abstracts from the heterogenous characteristics of different architectures, operating systems, programming languages and networks in distributed systems. It integrates these into one system by providing services that provide functionality based on the given common abstraction and that are 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 [15]. As such, coordination middleware is intended to integrate functionalities to enable and rule the coordination activities of heterogeneous

Coordination middleware is difficult to design. The provided abstraction has to deal with the central issues of how data is communicated, how activities are started and synchronized. The heterogeneity found is very broad, ranging 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 support 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 – estab-

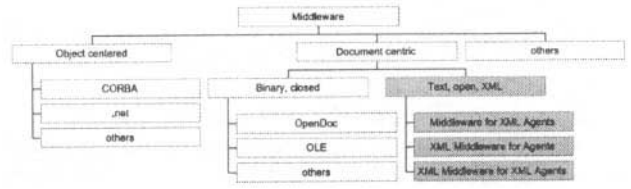


**Figure 1: Document-centric Middleware**

lished a notion of documents into which "components" or "objects" were included. The components contained data or software to manipulate that data. They were displayed to the user and accepted input for direct manipulation. Some control infrastructure offered services to coordinate via client-server interactions the interworking of different components. As the components could be of different source, these component software integrated different services into one 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 todays XML oriented middleware, objects and data were represented in a binary format in both and the frameworks were rather heavy. While OpenDoc was not able to gain wider acceptance, OLE2 is a grandfather of Microsofts COM and .NET frameworks. With *CORBA*, a component- and object-standard was established at the same time that found great acceptance which however did not incorporate a strong document metaphor.

With XML, document-centric abstractions are revitalized, and several interesting middleware systems for coordination have been recently proposed 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 document; or they can adopt a fully intgrated approach for XML-based coordination services in a world of XML document agents.

As a case study for the comparative analysis of these 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. If A and B together hold lesser money than requested, the transaction fails. 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 in itself is implemented outside the XML world, i.e., without exploiting XML technologies.

### *3.2.1 Displets*

The basic idea of the *Displet* approach ([8]) 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 principled, 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 XML, the displet 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, that the XML stylesheet can transform into a different tree, also by attaching to the tree 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 executed.

Displets document agents can have associated a private internal behavior, 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 document can share and have attached the behavior devoted to implement and control the execution of coordination patterns among the set.

In the case study, it is possible to think at having a client document agents, 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 being handled via proper behavior attached to the document agent. In addition, it is possible to attach to the client

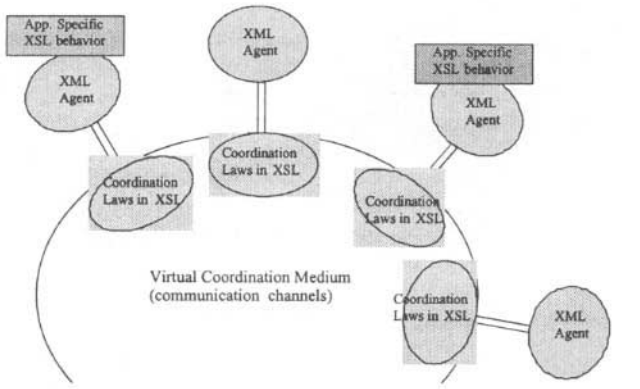


**Figure 2: The Displets Approach**

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 a user.

The main problem of the Displet approach is that document behaviors, which include the behaviors devoted to the implementation of coordination patterns, are hardwired into documents at compile time. 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 different coordination patterns. For the case study, 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 aim at providing frameworks for making XML documents active by enriching it with behavior, e.g., JXML [11]. However, most of this frameworks are quite limited with regard to multi-document coordination. In most of the cases, coordination between documents simply amount 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 [7]. 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 exploit XML at the middleware level in itself. In particular, they 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 the information space in itself become an active document agent, which is able to determine the laws according to which its data can be accessed and modified by application agents.

### *3.3.1 XMLSpaces*

In the coordination language Linda, tuples are primitive data without higher order values such as nested tuples, or mechanisms to express the intention of typing fields such as names etc. For Web-based systems, a richer form of data is needed. It has to be able to capture application specific higher data-structures easily without the need to encode them into primitive fields. The format has to be open so that new types of data can be specified. And 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 ([22]) 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, while XML documents can be represented as one-fielded tuples.

| 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 matching relation |

**Table 1: Matching relations in XMLSpaces**

A multitude of relations amongst XML documents can be used for matching. While the ones show in table 1 are supplied, 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.

The case study above would facilitate XMLSpaces to represent the state of the accounts in some XML representation. It would be very likely that some secure representation mechanism, ie. XML Signature, would be used and that a specific additional matching mechanism would ensure that account information is protected. The actual coordinator component would be implemented in some language running on the Java Virtual Machine. It would explicitly encode the rules for transferring money between the accounts using the respective mechanisms of the chosen programming language.

### *3.3.2 MARS-X*

The MARS-X coordination architecture ([5]), implemented as an extension of the MARS architecture ([3]), 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 document 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 it 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 in 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 modifies 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 case study with 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 case study 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. For instance, when the client agent request a total amount to account A, and that amount is not locally available, the reactions in the dataspace can trigger the activities of another agent, which

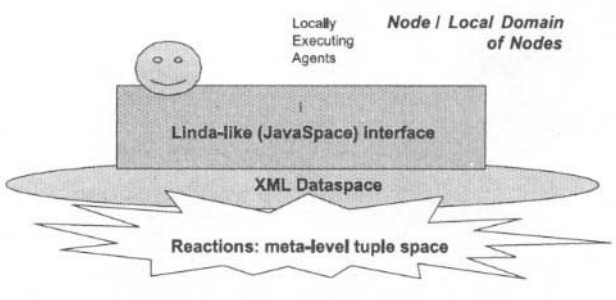


**Figure 3: The MARS-X Architecture**

is in charge of going to account B dataspace to check if enough further money is available there, and let account A dataspace reply to client agent accordingly to the total distributed amount that can be withdrawn. In a similar way, when the client eventually decides to withdraw, the XML dataspaces can coordinate the activities of the agents that will actually perform the partial withdraws from the different account. The possibility of controlling the execution of complex coordination patterns via specific behavior of the XML dataspace and transparently to agent is, beyond the case study, 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. Let us suppose that the client agent of the case study 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 dataspaces. This would require the client agent to directly manipulate and represent its world in XML terms. This would require agent to be not Java agents but, instead, XML document agents, e.g., Displets.

### *3.3.3 XMIDDLE*

The XMIDDLE middleware ([16]) 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 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 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

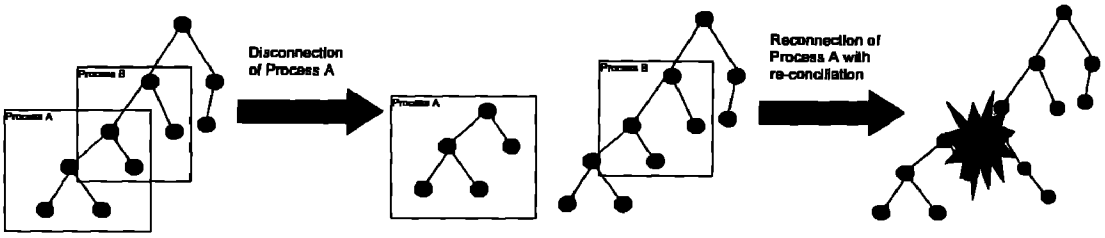


**Figure 4: Connections and Disconnections on XMIDDLE Trees**

implementing application-specific reconciliation policies, devoted to coherently reconstruct the structure of a tree.

In the case study, 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. In addition, 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 in terms of transparent coordination policies any complex coordination pattern, 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 ([24]). 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 a 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 are 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 – for programming one does not have to switch to an external language like Java.

### *3.4.1 WorkSpaces*

WorkSpaces ([20]) 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

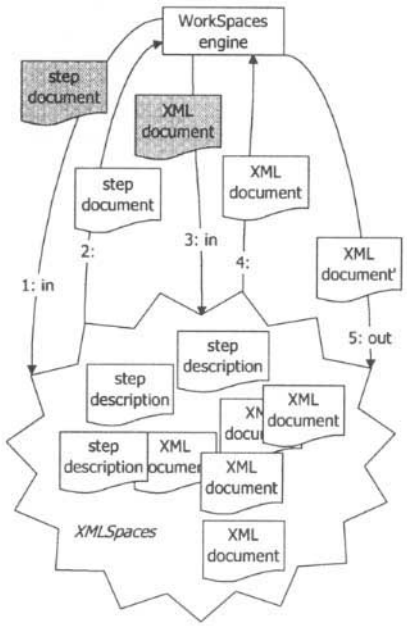


**Figure 5: Access to documents in WorkSpaces**

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.

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 an is based on the Workflow Process Description Language as defined by the WfMC in the Interface 1 of the Reference Model ([23]).

In a *meta step*, a set of individual steps is generated from this process description. While the workflow graph can be considered the "program" written in a higher level language, the execution of a workflow is the execution of individual steps, which resemble "instructions" in microprocessors. The "compilation" is performed by *meta steps* in WorkSpaces. XSL rule sets are by definition represented as XML documents following a syntax defined in the XSL standard. Thus, the compilation of the graph into steps is the transformation of one XML document into a set of XML documents, each containing an XSL rule for one step.

The unique distinction of this approach from other workflow management systems with proprietary workflow engines is universal accessibility and ease of deployment due to Internet standards, and support for distribution and uncoupled operation due to coordination technology. It also shows the power of XML and XSL as a fundament for a complex application, and enjoys characteristics such as universal access and distributed execution, thus being much more advanced than todays server-centric Web-services.

The case study 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 case study. The coordinator component, however, would be "implemented" by a series of 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 middlewares 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 ([25]) and XML Script ([9]). 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 most limited.

The Agent Definition Format ADF ([14]) is slightly more powerful. It offers a way to specify agents in a XML representation. Agents have their own state and coordinate with others using call encoded into agent references in URLs. The underlying model of coordination is again client/server. Also, the coordination behavior of document agents is mixed with their computational behavior, thus providing no separation between computation and coordination.

## 3.5 Discussion

The above analysis has identified the main features and 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 enable to embed behavior in XML documents and enable 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 complex coordination patterns to be defined, 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 more suitable for document agents, and its architecture seems very suitable for mobility, but the possibility of defining suitable coordination laws is very limited.

- WorkSpaces provides more uniformity, 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 explicit support for mobility and – being mainly oriented to workflow applications – may not be general-purpose for any kind of 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 XML document agents, as in Displets; making coordination activities occur in terms of manipulation 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 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.

# 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 application to be effectively engineered and developed.

First of all, there is the need to define new computational models, able to take into account and somehow formally analyse properties of coordinated applications based on XML document agents. A promising approach in that direction is represented by the work of Luca Cardelli on semi-structured computation ([6]). The basic intuition is that not only manipulations of XML documents can be represented in terms of a few basic tree transformation, 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.

The presence of mobility, in general, requires facing other important issues to enable and engineered approach to application design and development. One the one hand, there is need of clarifying the differences and the similarities between logical mobility of software components and physical mobility of devices ([17]). On the other hand, the concept of "context", intrinsic in mobility, must

be properly explored and its impact in modeling coordination activities must be clarified, and possibly taken into account in the definition of a suitable middleware ([4]).

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 support the activities of some real-world organizations, and mimic them accordingly; *(ii)* autonomy of application components invites considering them in terms of individuals playing specific roles in an ensemble rather them in terms of components providing functionalities. Therefore, those software engineering approaches exploiting the research results of organizational management may provide, in the near future, effective methodologies for the design and development of Web-based document agent applications and of coordination middleware ([26]).

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 ([19, 21]).

# 5. CONCLUSIONS

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

This paper has analyzed several middleware systems 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 case study, 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 such as the one related to the proper modeling and handling of mobility ([6, 17]) and openness ([26]) and to the effective engineering very-large scale and embedded applications ([19]).

**Acknowledgements**

Work partially supported by a grant of Microsoft Research Europe, by Italian MIUR project SALADIN, by Italian MURST Project MUSIQUE, and by a grant from Nokia Research Center.

# 6. 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] 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.

[4] G. Cabri, L. Leonardi, and F. Zambonelli. Engineering Mobile Agent Applications via Context-Dependent Coordination. In *Proceedings of the 23nd International Conference on Software Engineering (ICSE 2001)*, 2001.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. XML Dataspaces for Mobile Agent Coordination. *Journal of Applied Artificial Intelligence*, January 2001.

[6] L. Cardelli. Semistructured Computation. In *Proceedings of DBLP 99*. 1999.

[7] S. Chang and T. Znati. Adlet: an Active Document Abstraction for Multimedia Information Fusion. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 2001.

[8] 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.

[9] DecisionSoft Limited. XML Script. http://www.xmlscript.org/. Last checked Aug. 29 2001.

[10] P. Dourish et al. A Programming Model for Active Documents. In *Proceedings of the ACM Symposium on User Interface and Software Technology*, 2000.

[11] B. La Forge. The jxml home page. *www.jxml.com*, 2001.

[12] B. Gaines and M. Shaw. Embedding Formal Knowledge Models in Active Documents. *Communications of the ACM*, 42(1):57–74, 1999.

[13] N. Jennings and M. Wooldridge. Intelligents Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2), 1999.

[14] 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.

[15] T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[16] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis, and Wolfgang Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications*, To appear.

[17] 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.

[18] 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.

[19] David Tennenhouse. Embedding the Internet: proactive computing. *Communications of the ACM*, 43(5):43, May 2000.

[20] Robert Tolksdorf. Coordination Technology for Workflows on the Web: Workspaces. In *Proceedings of the Fourth International Conference on Coordination Models and Languages COORDINATION 2000*, LNCS, pages 36–50. Springer-Verlag, 2000.

[21] 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.

[22] Robert Tolksdorf and Dirk Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems (CoopIS 2001)*, 2001.

[23] Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model, 1998. http://www.wfmc.org.

[24] World Wide Web Consortium. XML Protocol Activity. http://www.w3.org/2000/xp/. Last checked Aug. 29 2001.

[25] World Wide Web Consortium. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt. Last checked Aug. 29 2001.

[26] 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.