Postprint of: Godlewska M., Wiszniewski B. (2010), Distributed MIND – A New Processing Model Based on Mobile Interactive Documents. In: Wyrzykowski R., Dongarra J., Karczewski K., Wasniewski J. (eds) Parallel Processing and Applied Mathematics. PPAM 2009. Lecture Notes in Computer Science, vol 6068. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-14403-5_26

Distributed MIND - a new processing model based on mobile interactive documents

Magdalena Godlewska and Bogdan Wiszniewski

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
ul. Narutowicza 11/12, 80-952 Gdańsk
{magdal,bowisz}@eti.pg.gda.pl

Abstract. Collaborative computing involves human actors and artificial agents interacting in a distributed system to resolve a global problem, often formed dynamically during the computation process. Owing to the open nature of the system and non-cooperative settings, its computations are in general non-algorithmic, i.e. their outcome cannot be calculated in advance by any closed distributed system. Authors advocate for a new processing model, based on exchange of documents implemented as autonomous and mobile agents, providing adaptive and self-aware content as interface units.
Keywords: collaborative computing, intelligent agents, policy-driven management

1 Introduction

Traditionally documents are passive objects, being sent over to their responsible authors, or downloaded from servers by interested users for further processing. MIND assumes them to be active objects, which are able to migrate from an originating server to continue work at remote computers by interacting with local users and services, who can view, edit and expand arriving documents using their private, locally available resources. Upon completing their mission and return to their originating server, document components are integrated, serialized (marshalled) and filed in a repository. Novelty of the MIND concept comes from the fact that document components are implemented as autonomous agents. Owing to that a new distributed processing paradigm can be introduced, making possible a radical shift from *data-centric* distributed systems, with hard-coded functionality, to flexible *document-centric* ones, where only generic services are provided by local environments, while specialized functionality is embedded in migrating document components. Moreover, that functionality may be dynamically expanded by incorporating the notion of plug-ins, available to the migrating components from their originating servers.

2 Document-centric distributed processing

Collaborative computing processes are often based on knowledge that shall be created and acquired dynamically. It involves human actors who cooperate

within a certain organizational structure – mostly by creating and exchanging documents. Their cooperation is not algorithmic, as it relies on dynamic interaction in a distributed setting [14]. Consequently, process outcome cannot be calculated in advance by a computer system. Moreover, documents being exchanged are the only visible interfaces to the respective processes, and constitute independent units of information necessary to elaborate a final result upon which all related documents have to be archived in a form enabling their future reuse.

A key feature of this scheme is an exchange of information objects, which are created dynamically at any time and in any place in the system. Collaborative computing systems of today are typically based on a pool of servers dealing with these objects as static and passive information units being served in a closed *distributed processing system (DSP)*, rather then interactive and active objects migrating in an open *multiagent system (MAS)*. Making electronic documents functional (interactive) and readable simultaneously to human users and computers, paves the way for direct incorporation of human intelligence into MAS computations. Putting the concept of distributed mobile interactive documents (MIND) to work requires, however, solving the three problems:

1. Modeling a document architecture as a system of distributed component objects, with embedded functionality for interaction and migration, to make them capable of implementing arbitrary complex behaviors.
2. Introducing security mechanisms to enable access management to document components, encryption of their content, digital signatures and preservation of integrity.
3. Resolving reliability issues, such as global state monitoring, state recovery, global object naming and automatic adaptation of migrating objects to local contexts.

A conceptual view of MIND architecture is shown schematically in Figure 1a.

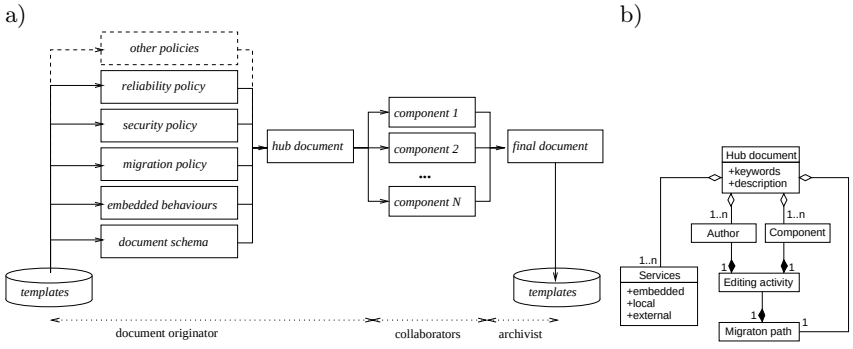

Fig. 1. Architecture of MIND: general view (a), hub document class diagram (b)

MIND builds upon two concepts: an automatic XML data binding, which allows for representing the information in an XML document as a set of objects

in computer memory, and mobile agents augmenting these objects to make them capable of migrating from one computer to another to continue execution. Owing to that, a static document template represented by a schema can be converted to component objects, which as agents can next implement their individual actions at designated destinations. This process is initiated by a *document originator*, who designs a *hub document* based on a schema defining a logical structure and functionality of a document template of interest. An UML class diagram representation of the logical structure of the hub document is shown in Figure 1b (its equivalent XML Schema representation has been omitted for brevity).

Key parts of the hub document are: one or more *components* with information (data) content, which upon unmarshalling become mobile objects, document *migration path* (specified in a form of a document workflow) and specification of one or more embedded, local and external services. *Embedded services* are implemented with plugins, expanding dynamically document functionality, *local services* are those that may be acquired by document components from target hosts upon arrival, while *external services* are provided by the originating server, in a fashion of a "ground control" facility. Users accessing component incoming locally act as *knowledge workers*, by interactively contributing to a component content, by making decisions based on the content of other components they have access to, and if necessary by creating new components in the form of dynamically added annotations, linked to various components of interest. Interaction of knowledge workers with components (agents) may be *individual* – when a document component has only one user, and *collective* – when a group of users has to share it and cooperate. In particular, the latter will take place when a set of dynamically created annotations will have to be integrated to form one coherent component content. Individual and collaborative editing will use locally available services to provide knowledge workers with a comfort of using their private tools, like editors or personal communicators. As mentioned before, document components migrate as agents across the Web according to their migration paths until they reach their individual goals. Upon reaching its goal, an object returns to its document originator, where it is marshalled with other objects back to an XML document. Files with XML documents are archived by an archivist on a server for reuse by other applications, especially for extracting accumulated knowledge.

3 Open document architecture

Adoption of agents enables documents to exhibit adaptive behaviors and facilitates their content to meet dynamically changing environmental requirements, such as in ad-hoc military networks [5]. The key concept is to provide for policy driven management of agents, and the architecture outlined schematically in Figure 2.

A policy, represented as a quadruple $< event, condition, action, scope >$ can be regarded as a goal that is to be observed by the entity to which the policy is applied. An agent whose behavior is to be controlled by applying a policy nor-

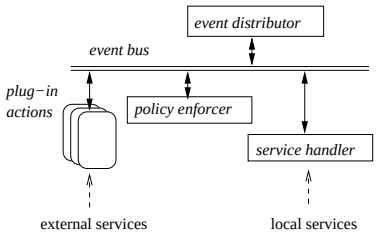

Fig. 2. Component MIND object architecture

mally consists of *plug-in actions*, which are individual software components that constitute augmented agent functionality, a local *service handler*, capable of detecting the current local context of an agent and determining its current role, a *policy enforcer* component monitoring events that could trigger the enforcement of relevant policies, and an *event distributor* component notifying all parties that registered their interest in particular events when they occur. Events published by the service handler and plug-in actions are passed by the event distributor to the policy enforcer. The latter evaluates the condition of each available policy and loads a plug-in action of the policy for which condition evaluates to true and scope includes the role determined by its service handler component. Owing to the flexibility of plug-in actions combined with the policy control, agents can select any specified objectives in a dynamically configurable and distributed fashion, and autonomously adapt their functionality to a wide range of environmental condition changes. MIND exploits this mechanism to make all document components adaptable to the locally available tools. This provides knowledge workers with their customary interaction and editing tools in a form of pluggable actions that could be performed from inside any popular Web browser.

4 Core technologies of MIND

As indicated before, implementation of the MIND concept involves specific combination of just a few basic Web technologies and no dramatic breakthrough in their development would be required. However, their survavibility as standards is important, to consider the MIND architecture as a reasonable alternative to data-centric computation models.

Below a quick review of the MIND core technologies and their state-of-the-art has been given to assess prospects of a new distributed processing model advocated in the paper.

4.1 Augmentation of document components

With the advent of XML, a uniform mark-up notation for describing logical structure of data has been established as a global standard. Representing the logical structure of documents and their components is important for MIND,

since according to the concept illustrated in Figure 1a documents will be converted to interactive components, enabling knowledge workers to manipulate their content. Several languages have been proposed to model a schema, to which a structure of a particular XML document should conform in a checking process called validation. Any schema language is a certain compromise between its power of expression, simplicity, and features, being more or less suitable for particular types of applications. The problem is to choose a schema language best fitted for the class of documents supporting collaborative computing processes today and in the future. Equally important is the support by organizations and vendors providing standards and tools to promise survival of the chosen language and guarantee forward compatibility of archived documents. For example, Document Type Definition (DTD), historically rooted in the SGML mark-up language, provides a very simple grammar mechanism and allows for structure rules to be parameterized and customized. It has also an excellent vendor support and prospects for survival as a default schema language. Unfortunately, its expressiveness is low, as DTD has been shown to be in the most restrictive subclass of regular tree grammar (RTD) languages, known as local tree grammar (LTG) languages. Formally, LTG is RTG without competing non-terminals, i.e. such that any two production rules do not share the same terminal symbol in their right-hand sides. Competition of non-terminals is important and makes document validation difficult, since more than one interpretation of a document may exist. The intent of XML Schema language [12] is to reconstruct the facilities provided by DTDs parameter entities and marked sections into a full type-lattice system with type inheritance, type extensions and type restriction. Although relatively more complex to use than DTD, XML Schema language has been shown to be less restrictive than DTD, by belonging to the subclass of single-type tree grammar (STG) languages. The class of STGs properly contains LTGs, as it does not allow competing non-terminals within a single content model, i.e., in the right-hand side of some production rule. Algorithms for constructing a unique interpretation of a document when validating it against a schema written in any STG language have been proposed. Finally, RELAX NG [3], has been so far the least restrictive schema language, as it belongs to the class of RTG languages; formally, RTG allows the right-hand side of a production rule to have just a regular expression over non-terminals. This freedom, however, implies difficulties in checking for ambiguity in matching regular expression patterns. Attempts to extend validation algorithms for LTG and STG classes of schema languages to the broader RTG class yielded non-deterministic algorithms that were not able to provide a unique interpretation of the document [7]. Later on, linear-time matching algorithms have been proposed based on the adoption of greedy regular expression matching [1]. This speeded up development of tools, increased interest in RELAX NG and improved its chances for survival.

Exploiting expressive power and ease of use of schema languages is just one aspect of the challenge. Another one is practical assessment and validation of the existing tools supporting the selected schema language. From this point of view XML Schema is best suited for MIND. An interesting alternative is the

newest technique of Java-XML data binding based on VTD-XML and XPath, which does not mandate schemas [4]. In order to convert an XML document into a set of objects enabling manipulation of data in that document one needs a tool for XML data binding to these objects. Existing tools for that have, however, a number of limitations, most of which are not serious in practice. They may be classified as round-tripping limitations and feature limitations. Round tripping of an XML document occurs each time it is unmarshalled and marshalled again. While all XML data binding tools can round-trip elements, attributes, text, and the hierarchical relationships among them, most of them cannot preserve comments and processing instructions, physical constructs such as entity references, and the order in which sibling elements and text occur. Feature limitations are caused by the fact that most XML data binding tools do not support XML Schema with regard to mixed content, wild cards, substitution groups, key/keyref and complex type restrictions. A more serious challenge is that XML data binding tool cannot operate on document fragments, i.e., cannot extract data from one or more fragments of an XML document, expose that data using schema-specific objects, and re-write those fragments to the document, leaving the rest of the document unchanged. This is a real problem in workflow scenarios, where many applications work on one document, each modifying and passing part of it to the next application. Architecture of MIND outlined in Figure 1a can cope with that challenge, since document fragments extracted once from the global hub document are bound with the application run by a document originator to objects with embedded functionality driven by externally defined policies and making these objects active "applications", capable of operating on their related document fragments. List of XML data binding tools includes products of both kinds: commercial, such as XML Schema Definition Tool in .NET Framework (Microsoft), XML Spy (Altova) or WebSphere Studio Application Developer (IBM), and open source, such as JAXB (GlassFish community project) or XMLBeans (Apache XML project). Experiments performed by authors with the prototype implementation of the MIND architecture have indicated that XML Beans performs very well in regard to the above.

4.2 Document component mobility

In order to enable migration of document components to remote sites for processing, they have to be converted to mobile objects (agents). Since their invention over a decade ago and dozens of different platforms developed, agents have been claimed to become a breakthrough in distributed computing with such obvious benefits as loose coupling, adaptability and support of heterogeneous systems. Surprisingly, it is relatively difficult to point to successful large-scale implementations of agent systems. Probably the reason is that the basic paradigm of agent technology involves functional specialization and autonomous interaction with the local environment, narrowing the class of realistic applications to just a few, such as resource management and maintenance of complex systems, or delivery of personalized content and e-commerce, for each of which a volume of agents should rather not be excessive. MIND concept of representing documents as

functionally autonomous and mobile objects requires, however, taking a closer look at scalability of the most popular agent programming platforms in terms of excessive message sending and agent migration actions, since the volume of component objects might be sometimes as large as few thousands for particularly complex collaborative computing processes (eg. court trials). A recent survey of a variety of agent platforms representing the current state in the field [10] including Aglets, Voyager, Grasshopper, Tryllian, JADE, Tracy, and SPRINGS indicates that for agent applications of a moderate size (up to 100 agents, each one performing few hundred movements and communications) the differences in reliability and performance are nor significant, and only some platforms have problems with reliable execution when agents move and communicate; agents that only move usually have less problems. Another experiment [6] aimed at flooding of a single node with incoming agents indicates that a JADE based application could reliably perform with up to 800-1000 incoming agents. These results are promising for MIND applications. Another issue is stability of the selected platform to guarantee survivability of MIND technology. An important point of reference here are the abstract platform and agent management specifications provided by the FIPA; so far JADE, offers a platform with the strongest resemblance to this specification. Experimental version of MIND based on JADE, implemented by authors is promising in that regard. Any MIND architecture implementation will have to build on top of FIPA to enable forward compatibility with agent platforms that will be developed in the future. Defining migration policies of MIND objects with standard mobile agent design patterns will further ensure this, as their implementability with JADE has been already proved [16].

4.3 Actionable meaning of document content

Knowledge workers collaborate in a virtual space by interacting with document components and to some extent by co-editing their content. Inspiration for that form of cooperation between users has been development of open source software by large groups of programmers. A key feature of that is a global scope of work, hard to achieve with paper-based documents. Each co-worker (co-author, co-editor, co-programmer) processes the assigned piece of content, by executing elementary operations like insert, cut and replace, while a system supporting the society of workers has to assure consistency and completeness of all so produced components. There are three possible approaches for such a collaboration:

1. *Pessimistic*, where a unique copy of the edited document is a shared at a central server. By using the mechanism of shared locks only one co-editor at a time can modify a document content, while other co-editors can only read it [11]. The main disadvantages are the requirement to stay on-line by all interested contributors, no provision for concurrency, and the overhead imposed by the management of locks with the increasing number of co-editors.
2. *Optimistic*, where before edition, a document is replicated and copies are sent over a network to each respective co-worker, who edits locally the received

copy. Messages on changes are broadcasted over the network to all interested co-editors. Editing operations performed individually by each author on his/her local document copy are rectified with messages received from remote authors doing their modifications in parallel [8]. One disadvantage of this model is a need to implement a complex mechanism for transforming editing actions performed locally with messages received from remote sites. Unfortunately, many algorithms already published in the literature have been found flawed and do not guarantee consistency of document copies edited in parallel [2].

3. *Realistic*, where a document is transformed into a set of objects forming a certain hierarchy implied by its logical structure, and having their own local memory and methods. An advantage is the potential of using well-defined object-oriented mechanisms for document processing, in particular open MAS, allowing for dynamic expansion and migration of document components – the key concept of MIND.

Implementation of interactive electronic documents as a single shared object is not an effective solution when the number of contributors is high, however has advantages when considering protection of a document content. On the other hand, the model based on replication allows for parallel actions of co-authors, but when the document volume is high (for particularly complex collaboration such as in court trials) would be impractical. Moreover, document replication precludes expansion of a document with new components, since adding a component would require its instant replication and redistribution over the network to maintain architectural consistency. Contrary to that, realistic document distribution appears to be the most efficient one of the three scenarios considered above. The content of distributed document components, augmented with mobility, enables information flows that can be organized into work processes involving intellectual resources of knowledge workers. These processes can be formally described with a *workflow definition language* as patterns of activities, which together with embedded behaviors of document components, implemented with plug-in actions, and predefined migration policies, implicitly related to the document component type and semantics, will constitute the actionable meaning of a document content. This meaning combines activities constituting the work process, partners (agents and users) involved in the process, messages exchanged between them, and exceptions raised in cases of errors.

The list of workflow definition languages and workflow management systems that have been developed so far is quite long, but any one of them may be criticized for being unsuitable to implement this or that particular pattern. A canonical set of patterns, independent of any particular implementation technology and application area, has been defined as a point of reference to evaluate any language already defined, or yet to be defined [15]. The set of interest considered by the authors included: XML Process Definition Language XPDL (Workflow Managament Coalition WfMC), Business Process Execution Language for Web Services BPEL (IBM and Microsoft), and Business Process Modeling Language BPML (Sun Microsystems). Prototype implementation of MIND used finally

XPDL, which proved to be sufficient to perform any realistic workflow. To increase prospects for forward compatibility of MIND, any workflow definition language based on XML should be allowed, what can be realistically provided owing to XSL transformations.

During processing along their respective workflows, components of a distributed document may need to refer to one another in the form of links, citations or bookmarks. The common addressing structure for the Web, based on the URLs, combining logical names with physical IP addresses to identify and locate digital objects, fails whenever the resources are moved between locations – is the case of mobile MIND objects. Moreover, some of these mobile objects may be dynamic in the sense that they are created after originating the workflows of document components defined by a schema of the document template. One example are dynamic annotations that may be created and linked by knowledge workers to other components at any time during the document lifecycle (see Figure 1a). The notion of a persistent identifier is needed, i.e. the one that tracks specific objects it refers to regardless of their physical location. Two primary persistent identifier applications have emerged and are strongly supported: the Persistent URL (PURL) [9], and the Handle System [13]. Both approaches provide registration and resolution services to map the persistent identifier to the current physical location of the digital object of interest and implement a sort of a redirection mechanism. PURL assigned to a mobile document points to the special resolver record in a resolver table maintained by a dedicated PURL server; the resolver record contains information to redirect the PURL to the current URL of the object, while the resolver table is updated each time any actual URL stored in it changes. Owing to this, the referred objects PURL does not change when the object migrates. The Handle System is an interoperable network of distributed resolver servers, linked through a Global Handle Server; a local Handle Server can resolve any handle through the global server to the current URL of the migrating object. Besides that the resolution of one handle to multiple objects, to other handles or even email accounts is possible. Persistent identification of MIND objects can adopt any of these mechanisms for dynamic annotations management. Note that objects that may be annotated will have to exhibit dual behaviors: one with regard to the user editing it – when the document component behaves as an interactive object exercising its embedded (plugged) functionality, and another with regard to the reader annotating it – when it is perceived as just a static content with resources to be indicated with specific XPointer and XLink expressions.

5 Conclusions

The proposed distributed MIND concept has a potential to provide a breakthrough in resolving non-algorithmic problems, to which collaborative computation paradigm seems to have no alternative. This is because document-centric processing model provides a natural mechanism for incorporating intelligence of human actors in computations of an open MAS. In particular, intelligent,

autonomous and adaptive document content advocated by MIND should significantly speed up and ease complex interaction and leverage knowledge bound organizations with virtual collaboration to enable resolving non-algorithmic decision problems, such as court trials, integrative bargaining, medical consultations, or crash investigations.

References

1. Frisch A. and Cardelli L. Greedy regular expression matching. In *Proc. ICALP'04*, 2004.
2. Oster G., Molli P., Urso P., and Imine A. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *Proc. Collaborative Computing: Networking, Applications and Worksharing*, 2006.
3. Clark J. and Murata M. RELAX NG specification. OASIS TC Specification 3 December 2001, www.oasis-open.org/committees/relax-ng.
4. Zhand J. Schemaless Java-XML data binding with VTD-XML. O'Reilly ONJava.com, 2007.
5. Cho-Yu J.Ch., Ritu Ch., Yuu-Heng Ch., Levi G., Shihwei L., and Poylisher A. A novel software agent framework with embedded policy control. In *Proc. MILCOM 2005*, 2005.
6. Chmiel K., Gawinecki M., Kaczmarek P., Szymczak M., and Paprzycki M. Efficiency of (JADE) agent platform. *Scientific Programming*, 13, 2005.
7. Murata M., Lee D., and Mani M. Taxonomy of XML schema languages using formal language theory. In *Proc. Extreme Markup Languages*, 2000.
8. Ressel M., Nitsche-Ruhland D., and Gunzenhuser R. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proc. Computer Supported Cooperative Work, CSCW'96*, November 1996.
9. PURLS. A project of OCLC research. OCLC Online Computer Library Center, www.purl.org.
10. Trillo R., Ilarri S., and Mena E. Comparison and performance evaluation of mobile agent platforms. In *Proc. 3rd Int. Conf. on Autonomic and Autonomous Systems, ICAS 2007*, 2007.
11. Greenberg S. Personalizable groupware: accommodating individual roles and group differences. In *Proc. Computer Supported Cooperative Work CSCW'91*, Amsterdam, 1991.
12. Gao Sh., Sperberg-McQueen C.M., and Thompson H.S. (eds). W3CXML Schema Definition Language (XSD) 1.1 Part 1: Structures, Part 2: Datatypes. W3C Working Draft 30 January 2009, www.w3.org/TR/xmlschema11-1/.
13. Handle System. Unique persistent identifiers for internet resources. www.handle.net.
14. P. Wegner. Why interaction is more powerful than algorithms. *Comm. ACM*, 40(5), May 1997.
15. Aalst W.M.P. and Hofstede A.H.M. YAWL: Yet Another Workflow Language. Technical Report QUT Tech. Rep., FIT-TR-2002-06, Queensland Univ. of Technology, , 2002, Brisbane, 2002.
16. Tahara Y., Toshiba N., Ohsuga A., and Honiden S. Secure and efficient mobile agent application reuse using patterns. *SIGSOFT Software Engineering Notes*, 6(3), May 2001.