Postprint of: Godlewska M. (2012), Agent System for Managing Distributed Mobile Interactive Documents in Knowledge-Based Organizations. In: Nguyen, N.T. (eds) Transactions on Computational Collective Intelligence VI. Lecture Notes in Computer Science, vol 7190. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-29356-6_6

Agent System for Managing Distributed Mobile Interactive Documents in Knowledge-Based Organizations

Magdalena Godlewska

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
ul. Narutowicza 11/12, 80-233 Gdańsk
magdal@eti.pg.gda.pl

Abstract. The MIND architecture of distributed mobile interactive document is a new processing model defined to facilitate obtaining a proper solution in knowledge processes carried out by knowledge-based organizations. Such organizations have an established structure that defines document templates and knowledge process. The aim of the MIND architecture is to change the static document to mobile agents, which are designed to implement the structure of the organization through autonomous migration between knowledge workers in accordance with the built-in policy. An extensible functionality of the agents also extends the static document to interface unit. The prototype of the agent system, based on the MIND architecture, shows the possibility of its implementation using available technology. The case studies provide the possibility of using architecture to solve specific decision problems with external knowledge of organization's workers.

Key words: Agent system, collaborative computing, knowledge-based organizations, policy-driven management, interactive documents, workflow patterns

1 Introduction

MIND architecture of distributed **M**obile **IN**teractive **D**ocument is a new processing model proposed by the author. Traditionally electronic documents have been treated as static objects downloaded from a server or sent by an e-mail. The MIND provides for exchange of static documents into a set of mobile agents, that can migrate between authors according to a fixed migration policy.

The MIND architecture makes 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.

In order to verify the assumptions of the MIND architecture and its possible implementation with the use of available technology, a set of prototype tools was implemented. This set of tools can be defined as an environment for designing

and managing document migration. Among these tools is a document editor that supports the design of documents in accordance with the assumptions of the MIND architecture and agent system that allows management components of documents.

The aim of research on this architecture is to develop a new model of object processing based on a system consisting of migrant components. Each document component is intelligent because of built-in functionality and mobile because of capabilities of migration in the system according to the inscribed policy. Workflow description language supported by the agent system will perform the workflow control-flow patterns, which are used in decision-making processes involving documents. Intelligent and mobile components form the MIND architecture that supports obtaining a proper solution by participants in collaborative computing of knowledge-based organizations. The main goal of the knowledge-based organizations is knowledge management and generation of new knowledge by knowledge workers through the documents. The documents are transferred between workers in accordance with the policy of migration.

In this paper, the author presents the model of a knowledge-based organization and general assumptions of the MIND architecture, presents also selected workflow patterns applicable to knowledge-based organizations, describes the elements of the agent system for managing distributed mobile documents and shows case studies that use the MIND architecture.

2 Related Work

The presented proposal combines existing technologies and new idea to extract some new functionality in the topics of distributed electronic document and collaborative environments. Unmarshalling and marshalling [1] allow XML documents [2] to be switched from static to dynamic form. Full distribution guarantees convenience of working in group and minimizes needs to connect with central server. Proposed solution differs from collaborative editing problem (algorithms solving that problem are introduced e.g. in [3–5]). Collaborative editing is the situation when users edit together the same part of document. This paper introduces problem when users work on different parts of one document, called constituent documents.

It is worth mentioning that the proposal is something different than shared workspace (e.g. BSCW - Basic Support for Cooperative Work [6]) or repository (e.g. CVS - Concurrent Versions System [7]). Those concepts are based on server. Users are able to upload and download some objects (documents, sources etc.), to communicate with other users, to be informed about some restrictions like deadlines. But most operations take place on the server. In the proposed concept, agent-objects have build-in functionality that has effect of shifting some services from server location to user location. That solution limits client-server communication. That feature distinguishes the proposal from client-server based applications like Collaboratus [8].

Full distribution and minimizing client-server communication are needed in workflow. Components of document do not have common location. They are able to migrate between users according to some policy. When there are many users in collaborative group (e.g. in court trials), the problem of too many client-server communicates appears. More independence form server allows also for implementing assumption that system managing documents should be as transparent to user as it is possible.

Workflow, based on the agent platform, is also implemented by WADE - Workflow and Agents Development Environment [9] software platform. The proposed concept differs from that implemented in WADE, because WADE is a data-centric approach and MIND is a document-centric one. In the MIND architecture, workflow (as a XPDL [10] file) is a part of the whole document. Workflow in the form of a document may be modified during the process execution. This allows the implementation of complex workflow patterns, i.e. Multiple Instances with a Priori Run-Time Knowledge or Multiple Instances without a Priori Run-time Knowledge presented in [11–13].

The MIND architecture proposed by the author is a completely different concept than the MIND architecture for heterogeneous multimedia federated digital libraries presented in [14–16]. The aim of the second architecture is integration of heterogeneous, multimedia non-co-operating digital libraries. This architecture consists of a single mediator and one proxy (composed of several proxy components) for every connected library and gives the user the impression of a single coherent system. The MIND architecture described in this paper is an abbreviation of **M**obile **IN**teractive **D**ocument. The aim of this MIND architecture is to change the static document to mobile agents, which are designed to implement the structure of the organization through autonomous migration between knowledge workers in accordance with the built-in policy. The similarity of the architecture names is coincidental.

3 Knowledge-Based Organization

Information can be defined as ordering and interpretation of data based on some patterns. The set of patterns in a certain context creates *knowledge*. New knowledge is built through the creation of new patterns. Knowledge-based organizations focus on processes based on collection, transfer and use of knowledge. Such processes are called *knowledge processes*.

Knowledge workers play an important role in the knowledge process. They generate new knowledge based on current information, their own knowledge and knowledge transfered by other workers. The purpose of the knowledge-based organization is to implement the knowledge process, in which the human mind is an important element. Knowledge is usually transferred between knowledge workers through the documents.

Tree resources of knowledge are used in the knowledge process: codified knowledge, established knowledge and personalized knowledge [17] (cf. Fig.1):

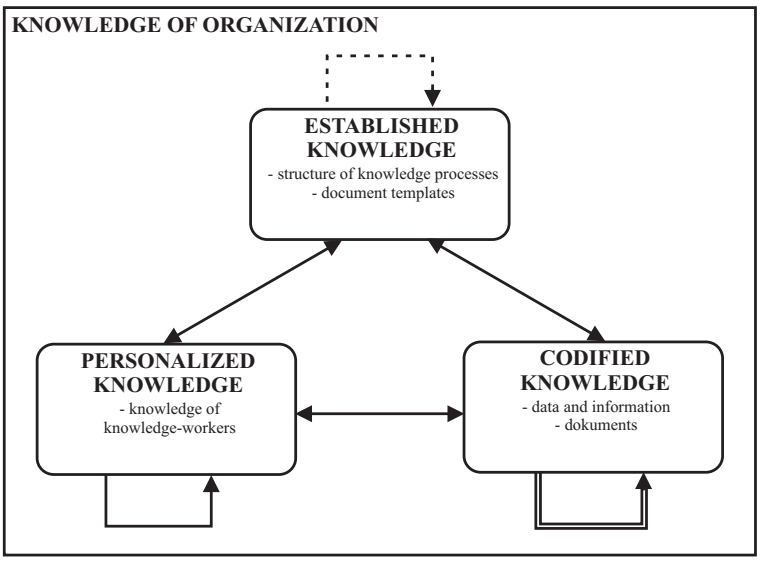


Fig. 1. Resources of knowledge in knowledge-based organizations

- *Established knowledge* includes the structure of knowledge processes. Processes are defined based on years of experience. Their aim is to ensure the execution of all necessary and sufficient activities to obtain the proper solution. The structure of the processes change rather slowly (dashed line in Fig.1) and is often conditioned by the law. Established knowledge includes also precisely defined document templates that can be used during the knowledge process.
- *Codified knowledge* includes a documentation of the process, ie information and knowledge transferred through the documents. That resource is dynamically changed during the process (double line in Fig.1).Codified knowledge is generated during the process by knowledge workers, who create new knowledge based on the data, information and knowledge contained in available documents. New knowledge is transferred to the next workers through the documents.
- *Personalized knowledge* is a knowledge of organization workers. The role of workers is creating new knowledge according to the rules of the process and based on codified knowledge. Workers collectively contribute to the proper solution of the process. Participation in the process also affects the development of knowledge workers (single line in Fig.1).

In the knowledge-based organizations, knowledge workers fill in relevant documents and transfer them to the next workers in accordance with a migration policy. In the process, the document is generally a static object, which means that: it is filled, it is sent, it is viewed, etc. In this paper the author presents a proposal to apply a distributed document, that consists of dynamic and intelligent

constituent documents. The constituent documents are knowledge storages and also interface units, that facilitate the extraction of knowledge and implement the migration policy defined in the knowledge process.

Distributed document - D - is created on the basis of a template S. The template $S = \{s_1, s_2, ..., s_n\}$ is a set of templates of constituent documents that may arise during the knowledge process. The templates stem from established knowledge of organization. Document D can be composed of multiple instances of some templates s_i.

The process is defined as a set of places P and a set of transitions T. The process also stems from established knowledge of organization. This definition is based on the model of Petri nets [18, 19]. Operations on documents are performed by knowledge-workers in places. Place is marked, when at least one constituent document is in this place. The constituent documents migrate between places in accordance with conditions defined in transition nodes. The process is graphically presented as a directed bigraph. The place nodes are separated by transition nodes in such a way that neither of the two places and neither of the two transitions are directly connected with each other. The transition nodes define the directed arcs between places. For example, a transition t_1 defines an arc "'form p_1 to p_2"'. It means, that there are two pairs: (p_1, t_1), (t_1, p_2) in a graph structure. The arcs in the graph form a set $A \subseteq P \times T \cup T \times P$. The conditions for transitions between places are also defined in the transition nodes.

The model of knowledge-based organization contains the sets of places and transitions that allow for defining the process and the set of constituent document templates that allows for the construction of the document D. The definition below summarizes the definition of the knowledge-based organization.

Definition 1. Knowledge-Based Organization *is a three-tuple* $KBO = (P, T, S)$, *where:*

1. P is a set of **places**.
2. T is a set of **transitions**, such that $P \cap T = \emptyset$.
3. S is a set of **constituent document templates**.

Information about the process (P, T) and the document template (S) allow for the construction of the proper document D, which task is the realization of the knowledge process. The definitions of sets and functions presented below show how to construct the document D.

Let $E = \{e_1, e_2, ..., e_n\}$ be a family of set of instances of all constituent document templates. $e_i = \{e_i^1, e_i^2, ..., e_i^k\}$ is set of instances of one constituent document template s_i. e_i^j is a constituent document, ie. specific instance of one constituent document template s_i. The constituent document is distinguished from others by the unique identifier. Let $f : S \to E$ be a function that assigns set of constituent documents to their template. Document D is a set consisting of the constituent documents, eg. $D = \{e_2^4, e_1^6, e_3^1, ..., e_3^2\}$. Functions: $pred : D \to D \cup \{nil\}$ and $succ : D \to D \cup \{nil\}$ define the order of the elements in D.

The set D is modified during the process, because the constituent documents can be deleted, copied, merged or created. The state of the document SD can

be defined as a set D in a specific point in the process. The document D is distributed and mobile thus it is necessary to define the function $g : e_i \rightarrow P$ that assigns the current place to the constituent document.

4 MIND Architecture

The main goal of the MIND architecture is to implement the model of knowledge-based organization by changing the static document to mobile components, that meet their mission in the distributed agent system. After the mission, components are merged into a static final document. The concept of document life cycle is illustrated in Fig. 2.

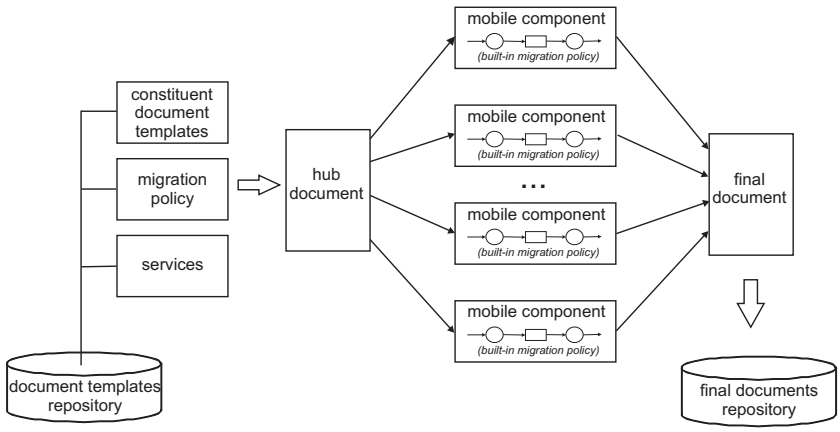


Fig. 2. The concept of MIND document life cycle

The MIND architecture has been proposed with a view of facilitating the collaboration of knowledge workers in knowledge-based organizations. In such organizations, a set of particular knowledge contributes to the final proper solution. There are some important elements in knowledge process: the structure of organization and knowledge of workers passed through the documents. In such a process the external data sources (knowledge of workers) appear. The examples of such processes are court trials, integrative bargaining, medical consultations, or crash investigations.

The MIND architecture involves the creation of a document D that contains structure of the process (sets P and T) and templates of constituent documents (set S). The constituent documents follow a defined process (migration policy) to move between their authors in the form of mobile agents and then, as the interface unit, assist in extracting knowledge from authors.

In the architecture some additional services can be defined. The purpose of these services is adaptation of the architecture to the needs of a particular

organization. From the user's perspective, these services can be `local` (installed on local computer), `external` (accessible from the outside) and `embedded` (build-in in agent).

The main components of the architecture are: *HDDL* (Hub Document Description Language) [20] document template, that is defined in *XML Schema* [2] and the agent system for document managing. The document template contains the following components:

- `document.xml` - basic information about the document D in the form of a header and a specification of services.
- `authors.xml` - data about the authors (knowledge workers).
- `templates.xml` - a set S of constituent document templates.
- `parts.xml` - a definition of the document parts. Each part contains a constituent document that is an instance of one template defined in the component `templates.xml`. This component defines the structure of vector D.
- `path.xml` - a definition of the process graph. This component connects the authors with the document parts and defines workflow of constituent documents.

Main component (`document.xml`)
Elements and attributes of the main component are presented on Fig. 3. Attributes describe identity and the basic properties of each MIND document:

- the unique identifier (`ID`) and the `title` – distinguish a specific template in the document templates repository;
- the `security` level and the ability to add new dynamic objects (components) in the document life cycle – determine the behavior of constituent documents in the form of mobile objects during interaction with knowledge workers.

The <`head`> element specifies metadata of document that is useful for searching the repositories that store documents in the static form. The following information may be contained in the <`meta`> collection: information about the tool by which the document was prepared (`name="Generator"`), basic information about the document like `version`, `keywords`, `author` and `description`, and other information defined by the user using their own names.

The <`services`> element specifies services of MIND document, that are available for each dynamic object created as a result of its conversion from a static to an object form. The services can be implemented as:

- <`embedded_service`> - services implemented with MIND application, expanding dynamically document functionality
- <`local_service`> - services acquired by document components from target hosts upon arrival
- <`external_service`> - services triggered by a document components on remote hosts

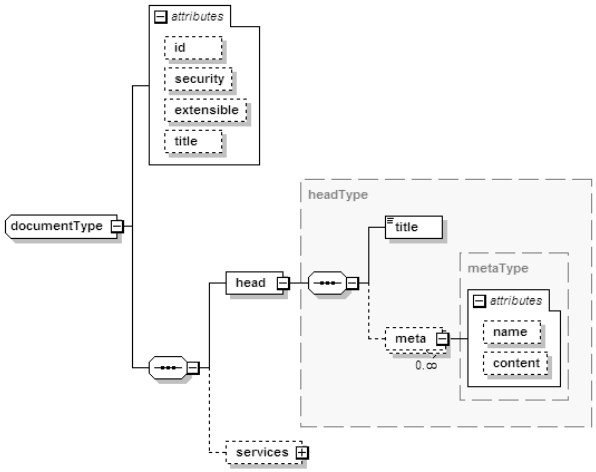


Fig. 3. Logical structure of main component

Authors component (`authors.xml`)

Each MIND document has at least one author, who is a document originator. The document originator creates a hub document at the beginning of the life cycle (cf. Fig. 2). Constituent documents which migrate as mobile components can be assigned to other authors. This assignment is defined in the built-in migration policy of the component. During component migration, also new authors can be added to the policy path.

Elements and attributes of the <`author`> element are presented on Fig. 4. Each author is assigned a unique identifier (`ID`). The `originator` attribute pointed the document originator and the `active` attribute specifies whether the author has the right to change the constituent document.

The <`name`> element contains the first and the last name of author or the name of company, the <`position`> element specifies the author's position in the organization. The <`description`> is some additional information about the author, for example, role in the document creating. The <`contact`> element contains the contact data of the author. IP-address is especially important information for the mobile component. It specifies the host on which the constituent document should be delivered. IP address can be defined in indirect way as a persistent number that distinguishes the author. Then the appropriate service assigns the current IP address of the author to the persistent address. This solution enables work on multiple computers or on mobile devices. The <`communication`> element specifies other information necessary for communication with the author.

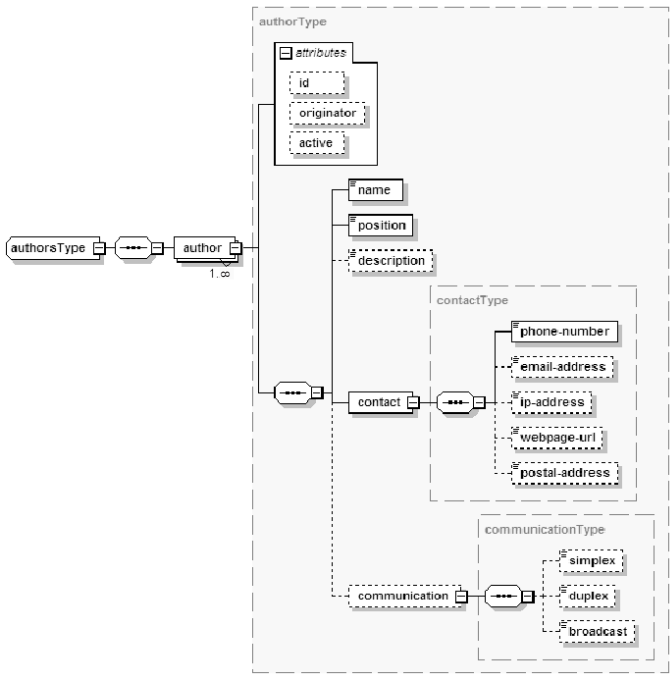


Fig. 4. Logical structure of authors component

Templates component (`templates.xml`)

The document D consists of any number of constituent documents which are documents of a certain type. The document types are defined by a set of templates ($S = \{s_1, s_2, ..., s_n\}$). Fig. 5 shows the specification of the constituent document templates.

Each template has its own unique identifier (`ID`) and `title`. The <`mime-type`> element contains the name of a template type compatible with the standard *Multipurpose Internet Mail Extensions* (MIME) [21]. The <`content`> element represents the content of the template encoded in a format *base64* [22].

Parts component (`parts.xml`)

The MIND document consists of constituent documents, which are converted into mobile objects. The constituent documents meet their mission of the transfer and collection of knowledge in the knowledge-based organizations. The constituent document is defined as a part in parts component. Its structure and format are defined by an appropriate template that is stored in the templates component (described earlier).

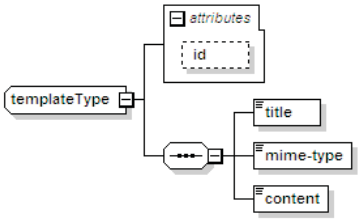


Fig. 5. Logical structure of templates component

With reference to the model of knowledge-based organization, the parts component defines a vector D. The constituent documents are placed in the specific order and distinguished from others by the unique identifier (attribute `ID`). Each constituent document is a document of a certain type and this type is specified by element <`template`> which contains the ID of the document template defined in the template component.

Fig. 6 shows the logical structure of document's part. Attributes specify a unique identifier (`ID`) and a `state` of part. The `state` attribute defines the state of part in the life cycle of document and can take values: `new`, `modified`, `verified-positive`, `verified-negative` and `done`.

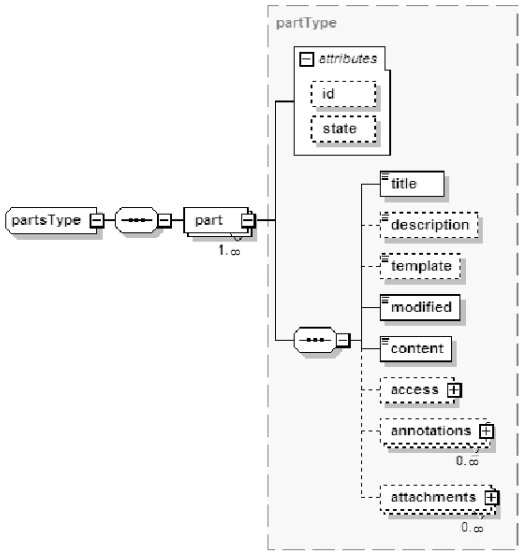


Fig. 6. Logical structure of parts component

The <title> element contains a title of part, the <modified> element contains date of last modification and the <content> element represents the actual content of part encoded in a format *base64*. The <description> element provides a full text description of the constituent document and <template> specifies the ID of document template.

The <access> element has child elements: none, toc, read, write and print and they specify access right of authors to the document. The access rights are: none - no right of access, toc - access to table of content, read - right to read the document, write - access to write and print - rigth to print. The child elements allow to assign the value of the author's ID attribute. All value of each element allows all the authors for the specified operation.

Each constituent document can have the <annotations> mapped dynamically at any time of the document life cycle. The <reference> child element of <annotations> allows also for putting new annotations to existing annotations. It is also possible to link some other document to MIND constituent document as an attachment.

Path component (path.xml)

The migration policy is defined also as a component of the document. The path.xml component is a *XPDL* [10] file. XPDL (XML Process Definition Language) is a dialect of XML (Extensible Markup Language). It is a language for workflow process design.

The main element of XPDL - <WorkflowProcesses> - may contain any number of <WorkflowProcess>. The logical structure of <WorkflowProcess> is shown in Fig.7. Each element <WorkflowProcess> represents a process consisting of <Activities> and <Transitions> between them.

<Activity> - element of the workflow process - combines the author with the part of document that is assigned to this author. In the workflow definition, places P in a process are called activities [23]. <Transition> element has attributes From and To that define directed paths between activities. <Transition> element may also specify the conditions for transitions between activities. Construction of dynamic objects representing the components of the document is presented in Fig. 8.

5 Workflow Patterns Applicable to Knowledge-Based Organizations

In knowledge-based organizations, the process is constructed from elementary operations. These operation are called workflow patterns. This section presents sample workflow patterns that are applicable to knowledge-based organizations. These patterns are selected from the complete list (more than 40) of workflow patterns presented in [11–13]. Some of these are supported directly by XPDL and others may be implemented by agent systems. These workflow patterns

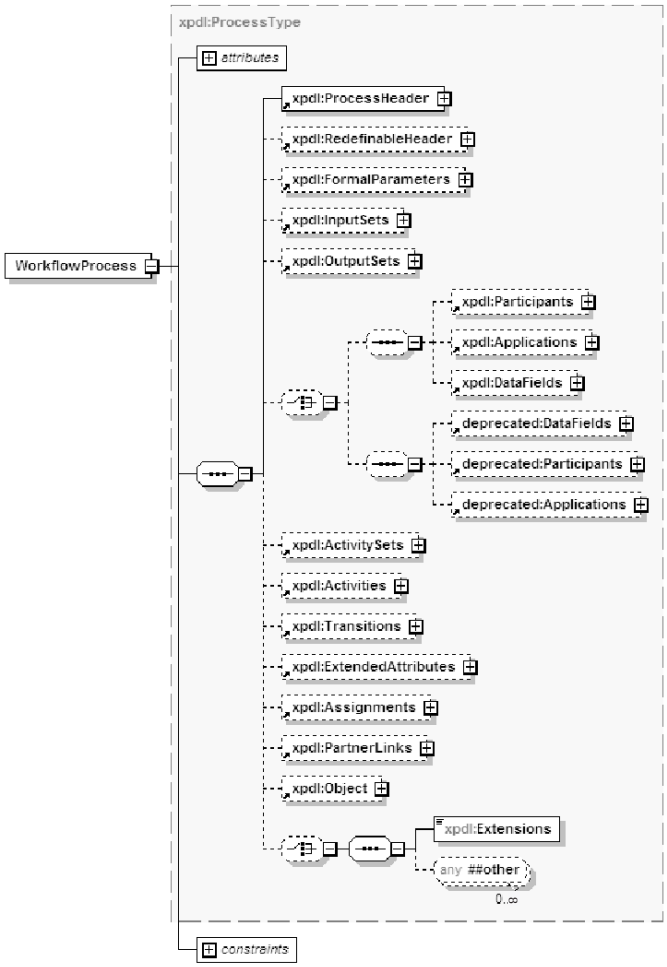


Fig. 7. Logical structure of XPDL workflow process

are modules of the path component. The complete list of modules allows for construction of any process of constituent document migration.

5.1 Basic control-flow patterns

To the basic control-flow patterns include: Sequence, Parallel Split, Synchronization, Exclusive Choice and Simple Merge. All these patterns are directly supported by XPDL.

Sequence is when an activity in a workflow process is enabled after the completion of another activity in the same process. This is the basic form of transferring constituent documents in the knowledge-based organizations. When one of

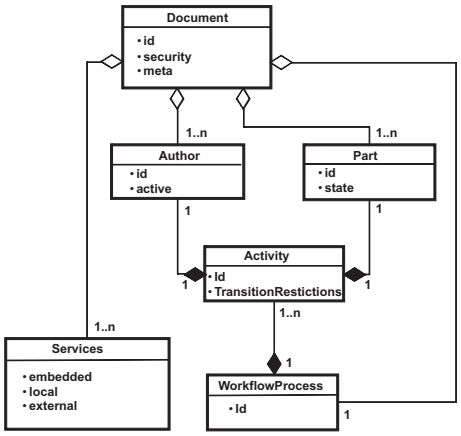


Fig. 8. MIND dynamic object

knowledge workers completes work on the document, this document goes to the next author, in accordance with the migration path. In MIND, agent reads, directly from the XPDL policy file, a target activity (designated by the transition) to which it has to move. Sequence is the simplest but also the most frequent form of the flow of constituent documents.

Parallel Split is a point in the process where a single thread of control splits into multiple threads of control that can be executed in parallel, thus allowing activities to be executed simultaneously or in any order. In the knowledge-based organizations, this is a case where one constituent document is copied and the copies are moved to the various knowledge workers. This places the new constituent documents in the vector D. For example, a few experts have to make independent decisions on the basis of the same research results. It is important in the parallel split, that the whole constituent document is duplicated and each of the knowledge workers get exactly the same copy.

After the Parallel Split, usually the *Synchronization* occurs in the process. This is a point in the process where multiple parallel branches converge into one single thread of control, thus synchronizing multiple threads. The merger activity is waiting for all branches to be completed. In the knowledge-based organizations, this is a case where all copies of one constituent document, that was edited by various knowledge workers are collected. A set of documents that differ from each other and containing different knowledge is appeared. The synchronization requires to wait for all copies of the document. Then the copies have to be integrated into one new constituent document. It also changes the dimension of the vector D. In the case of documents, integration does not always mean the same operation. It could be:

– *integrating into a single file*: this integration is basically gathering all copies in one document without interfering with its contents. Such situation can occur when the copy must remain unchanged. For example, in a court trial,

each expert gives an independent opinion on certain evidence. These expert reports are collected and presented in unchanged form during the trial. The particular opinion is essential.
- *integrating the document by the knowledge worker*: this is integrating the content of documents into one output document. The aggregate knowledge of one worker on the basis of expert reports from other workers is important. For instance, a doctor diagnoses a patient. He carries out a series of tests and sends the results to several specialists for consultation. After receiving all the opinions, the doctor integrates them and generates a new document containing a decision: diagnosis, referral for next tests or next consultations.
- *integrating the document based on the authors' agreement*: this is integrating the contents of the document on the basis of communication between the authors. For example, several authors writing a paper. First, everyone writes a part of the document and next, the authors integrate the whole paper together: they write an introduction and a conclusion, accept some consistent grammatical forms. In example, this is situation if few authors write one paper. First, each of them writes some section and then they establish together a common whole.
- *automatic integration*: content of the documents is automatically integrated this solution works only for simple documents with precisely specify structure, such as questionnaires.

In MIND, the merging point is defined in the policy file. This is the point where agents converge with the copies of document inside. The agents save the copies of document locally, then the copies are integrated into a single document without interfering with its contents or by the knowledge worker. Integrating the document based on the authors' agreement is more complicated. MIND architecture enables to define new applications (e.g. in services form), that facilitate and streamline the process of extracting information. In order to integrate the document, based on the authors' agreement, an application like communicator and/or an application, that allows for sharing the view and editing the document by many authors, could be used. The current version of the MIND architecture does not support directly the concept of integrating the document based on the authors' agreement. The automatic integration may also be performed by dedicated services for reading and integrating specific documents, i.e. questionnaires or tests. The aim of the MIND architecture is the support of the knowledge worker's interaction with the system, hence there is no direct support for automatic integration of documents. Regardless of the synchronization type, only one agent remains on the merging activity. This agent will continue its mission with a newly created document inside.

Exclusive Choice is a point in the process where, based on a decision or workflow control data, one of several branches is chosen. The choice is defined as a condition of the transition node. The corollary of Exclusive Choice is often a *Simple Merge* pattern. This is a point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel

with another one. In these cases, the problem boils down to interpretation of the earlier conditions that selecting one of the options of flow. After the correct interpretation of the conditions, the flow of the agent does not differ from the normal Sequence.

5.2 Advanced branching and synchronization patterns

To this group of patterns include: Multi-Choice, Synchronizing Merge, Multi-Merge and Discriminator.

Multi-Choice is a pattern similar to the Parallel Split, with the difference that there is selected a subset of threads to be executed in parallel. The corollary of the Multi-Choice is a *Synchronizing Megre* pattern, that merge the active threads. These patterns are directly supported by XPDL and in the MIND, they are connected with problems of synchronization and condition interpretation.

Multi-Merge is point in a process where two or more branches converge without synchronization, i.e. each incoming thread requires the same reaction in the activity. For example, after expert reports are received to the court, the feedback is sent. There is some workaround for this problem: the activity may be duplicated to each branch. In Together Workflow Editor (TWE) [9], there is a `WorkflowPatterns` directory, where the examples of patterns implementation can be found. There is some other solution for the Multi-Merge pattern with the use of subprocess executed asynchronously. In MIND, every agent has a built-in policy file, hence duplicating the activity is more obvious solution to this problem.

Discriminator is a point in the workflow process that waits for one (the few) of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and ignores them. For instance, the recruitment committee is waiting for receipt of applications; when the first 20 applications arrive, the recruitment process moves to the next stage, and the rest applications are ignored. The support of this pattern in XPDL is unclear. There is some workaround solution presented in TWE. In MIND, all agents have built-in policy, so method, that counts instances of documents, is shifted from the agent to the container object, where the agents arrive with documents. So, discriminator will be implemented in the application rather than in XPDL.

5.3 Multiple instance patterns

This group of patterns include: Multiple Instances without Synchronization, Multiple Instances with a Priori Design-Time Knowledge, Multiple Instances with a Priori Run-Time Knowledge and Multiple Instances without a Priori Run-Time Knowledge.

Within a one process, multiple instances of an activity can be created. These instances are independent of each other and run concurrently. In the first pattern, there is no requirement synchronization upon completion and in other pattern

the synchronization is required. For documents, the synchronization involves the integration of documents as in the case of Synchronization pattern.

The first three patterns can be implemented in XPDL by the LoopMultiInstance element but Multiple Instances without a Priori Run-Time Knowledge pattern can not be implemented in XPDL. In Multiple Instances without a Priori Run-Time Knowledge pattern, the number of instances to be created is not known in advance: new instances are created on demand, until no more instances are required. In the knowledge-based organizations, this is a case, when testimony of witnesses in a court trial are being collected. It is not known how many witnesses will testify and when they decide to testify. MIND allows for editing the policy file during a run time of the process. The policy file may be edited many time at each point in the process.

5.4 State-based patterns

This group of patterns include: Deferred Choice, Interleaved Parallel Routing and Milestone. *Deferred Choice* is a point in a process where one among several alternative branches is chosen based on information which is not necessarily available when this point is reached. The choice is not made immediately when the point is reached, but instead several alternatives are offered, and the choice between them is delayed until the occurrence of some event. *Interleaved Parallel Routing* is a pattern, when a set of activities is executed in an arbitrary order. Each activity in the set is executed exactly once. In any case, no two activities in the set can be active at the same time. *Milestone* is defined as a point in the process where a given activity has finished and another activity following it has not yet started.

These patterns use a different states of the process points. XPDL does not allow for the definition of states. Deferred Choice may fulfilled. The other patterns can be implemented by agent application. In the agent platform (e.g. JADE), agents can communicate with each other through the implemented communication protocols. They can also change some data locally and remotely. One agent ,who is in active state, may change state of other agents to inactive. So, XPDL can be extended on states by agent system.

6 Prototype System for Managing Distributed Documents

The static MIND document is designed in such way that it can be transformed into the object form, and then into a set of mobile agents. The mobile agents migrate between user's local workstations in accordance with a defined migration path. Based on the MIND architecture the prototype system for remote management of distributed components of the document has been implemented.

In the prototype system for managing distributed documents the following technologies are used. Static documents are implemented in *XML* language in accordance with the HDDL format designed for the MIND architecture. HDDL

format has been specified in *XML Schema*[2]. To design workflow process *XPDL* format [10] is used. Server application [24] has been written in *Java* and *JADE* platform [25] is used for the creation, migration and tracking of agents. *XML-Beans* [1] is a technology used for accessing XML by binding it to Java types, *Java Mail* platform [26] is used for transferring agents via e-mail and *Log4j* [27] tool is used for creation of event logs.

The MIND architecture is not dependent on the chosen technologies and for its implementation different ones may be chosen. To describe the structure of documents and workflow XML language was selected. The possibility of its transformation and processing with programming languages allows to use its also in the future. The advantage of easy archiving of XML documents is particularly important in the documents, which will remain valid for many years, such as e.g. property rights.

In order to facilitate the preparation of the documents in HDDL format the following tools have been implemented: *HDDLEdit* [20] for creating and editing patterns of document and *STMPEdit* [20] for creating and editing of constituent document templates.

6.1 The Life Cycle of a Distributed Document in the Agent System

The system for managing components of the document is designed to implement the concept of converting a static form of the document to the autonomous agents that will meet their mission in the system, and then create a final document (cf. Fig.9.).

Agents have built-in migration functionality defined in the MIND policy file. According to the next stage of the document's life cycle, the agents migrate between the authors computers with a constituent document. The constituent document's content is processed by the respective authors. Organization of the agents population is involved in the agent system. The components of the system are run on the users computers. They create a platform that enables the functioning of the agents on each workstation, provides access to memory and enables the use of communication protocols. These protocols allow agents to exchange messages, and migrate independently on the network. If the author's computer is offline in terms of the agent platform, the system sends the agent by e-mail (cf. Fig. 10).

During the completion of the life cycle of the document, the agents reach different states, which, in the form of a diagram, is shown in Fig. 11.

An *active* state is a state when the agent is run on the agent platform. The agent is in a *hibernate* state in a situation when the user of the current agent's container chooses to stop work and turn off the application. The agent in this state is saved as a file on user's local computer – this operation called serialization. When the system is restarted, the agent can be rerun (transition from *hibernate* to *active* state). When the next agent's author is offline in the agent platform, the agent is sent to him by e-mail. This additional functionality moves the agent to a *sent* state. Agent is in this state, as long as the agent is stored on the mail server. The agent is received from the e-mail server and

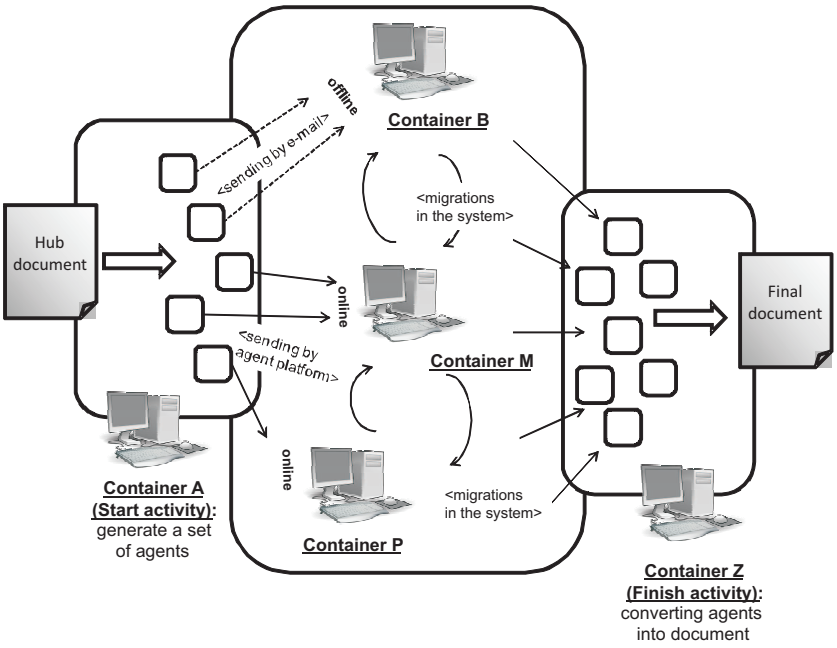


Fig. 9. The life cycle of MIND document in the implemented system

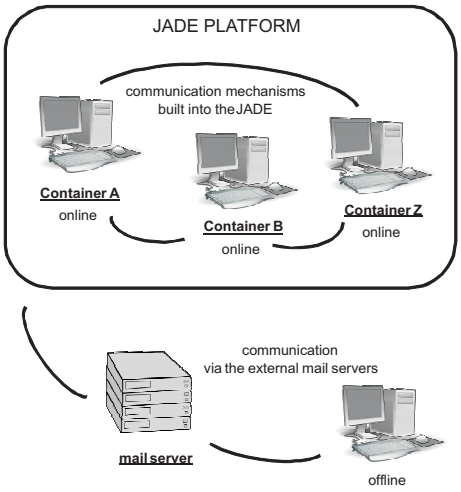


Fig. 10. The life cycle of MIND document in the implemented system

brought back to the *active* state after restarting the application. When the agent arrives to final activity in the process, it moves to the *final* state. The agent in the final state may be *deleted* or *archived* (saved to a file).

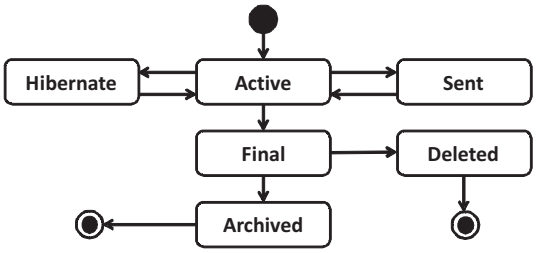


Fig. 11. Agent's states diagram

6.2 The Main Modules of the System

The application server for remote management of distributed document components consists of the following modules: agent server module, e-mail module, policy module, document module, mediator module and initial module.

Agent Server Module The basic element in the agent server module is a *JADEServer* class. The object of this class starts JADE platform. JADE platform provides a set of containers. One of them is called a main container. The container is JADE object that allows to run the agents. In the main container, three functional agents (AMS, DF, RMA) are run. They control the JADE platform and move the agents. Any other containers must liaise with the main container in order to join the JADE platform.

The cooperation of the various computers in one agent system is possible when all containers are connected to the main container. Users of containers must enter IP address of main container to connect to the platform. A successful connection to the main container determines that the local system will work online. If connection to the main container is not possible, the user must work offline. When the JADE platform is started, the agent server module is responsible for managing agents, i.e. for creating, awakening and deleting agents.

The agent is built on the basis of the XML document conforming to the MIND architecture. The agent gets as input arguments the following information: a unique name, policy file and constituent document ID. Awakening of the agent means running it from a file and changing it state from hibernate to active or from sent to active (cf. Fig. 11).

Functionality of migration enables the agent to transfer in the JADE system. For this purpose, some parameters are determined: the name of the agent to be sent, the place of its destination after a jump and the constituent document to be forwarded.

E-mail Module E-mails are not saved directly in a recipient's computer, but are stored on mail servers. Implementation of the e-mail module allows to transfer the serialized agent by e-mail when sending it to the JADE server is not possible. This allows the agent to reach to assigned recipient even if at the time of dispatch

his container is not registered in the system (i.e. from the viewpoint of the system the recipient is offline).

The e-mail module uses the Java Mail platform and it is equipped with a set of classes responsible for handling e-mail servers, sending and receiving e-mails with a specific name. This name allow to distinguish messages with agents from others. A main class of module (*MailServer* class) implements the operations of sending and receiving the agents. Before sending, the agent is serialized to a file. To take advantage of this mechanism, the authors specified on the migration path must have e-mail addresses.

Policy Module and Document Module The policy module is separated from the module that manages the other parts of the document, because of the very important role in the agents migration. The XPDL file defines the process of moving the agents in the system. The policy module provides a binding XPDL file to Java objects. For this purpose, XML Beans tool was used to generate *xpdl.jar* file based on XML Schema template for XPDL language. The *xpdl.jar* library allows for creation of the object representation of XPDL files within the system.

The task of the policy module is mapping a static representation of the policy file (path.xml) to the Java reference mechanism. The module opens the policy file (or files), interprets its structure and then creates Java objects.

Completion of work on a constituent document in current node triggers the movement of this document to another node. In this case, the policy server searches possible transitions. The module creates a collection of locations, which the agent must visit in the next step. In this case, the location means a container in the JADE system or e-mail address of the next user of the system.

The document module allows to load and manage other components of the MIND document. In particular, its task is to choose the appropriate constituent document in accordance with an activity element from the policy file.

This module also maps the static XML document to Java objects. XML Beans tool was used to generate *XMLHubDoc.jar* library based on XML Schema template defined for the MIND architecture.

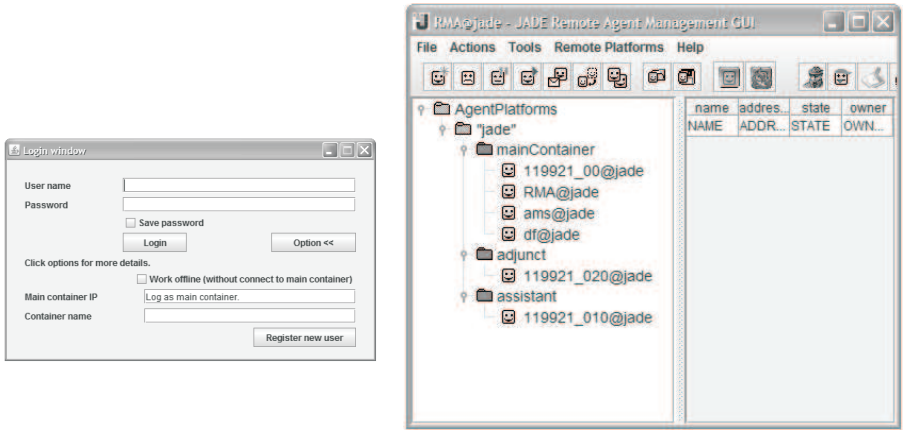
The document module is responsible for extracting the constituent document from the agent and saving it on the local computer of the author assigned to the activity. Then the author can perform on that document some actions. After that, the document's current version is re-loaded by the agent and moved to another location. When the agent reaches the final location, the constituent documents are integrated into a final document.

Mediator Module and Initial Module Unlike other modules, the mediator does not provide any operation. This service is implemented only to provide the communication between other modules and mediate in the running of methods. It also allows for easy implementation of subsequent modules in the evolution of the system.

The initial module runs all subsystems (modules). The task of the module is to take data from the user during the login to the system and booting the system.

6.3 Prototype Application's Performance

The system for managing of components is used to implement the life cycle of the MIND documents. The authors defined in the document should be the holders of the application, that creates the containers that forming a part of the JADE platform. After running the application, dialog box for authors authorization appears. During the login operation, the author gives the IP of the main container (unless the main container is created) and the name of the local container. The author may choose to work offline, i.e. a work without connection to the main container (cf. Fig. 12(a)).



(a) Login window (b) JADE window - RMA agent

Fig. 12. Dialog box for authors authorization (a) JADE mechanism for monitoring the agents and containers (b)

After the successful authentication, the particular modules are run. That leads to join the container to the JADE platform and runs the possibility of receiving, status changing and sending agents. Launching an application as a main container causes running an additional mechanism to monitor the agents and containers on the JADE platform. For this purpose, JADE provides a tool RMA (Remote Agent Management) in the form of a dialog box (cf. Fig. 12(b)). This tool allows for monitoring the global state of the agent system.

In the next step after running, the application converts the MIND document to objects (this operation is called *unmarshalling*). The policy and document modules are responsible for that. A set of agents is created on the basis of start

activities. A unique number (consistent with the activity ID) and ID number of the constituent document are assigned to each agent.

An agent interface is a simple dialog box with two possible actions: *Done* and *Exit*. The agent's GUI may be expanded depending on the system use cases. When the author finishes work on a constituent document, selects the *Done* action. The agent then reads from the policy file its next activities and moves to new locations. If the container of the target activity is present on the platform, the agent is sent by the JADE , otherwise the agent is sent by e-mail. In this way, the agent transfers the constituent document according to the defined workflow. Once the agent reaches a final location, *Done* action deletes the agent or transfers it to the archiving location. In both cases, the constituent document is saved as a file on the local computer.

If the author chooses the *Exit* action, it means that he has not finished to work on the constituent document yet. Then the agent is saved on his local computer as a file and the constituent document is also saved as a static XML document. When the system is rerun, the agent is awaken and the constituent document is upgraded in accordance with the new version from the local computer.

7 Case Studies of the MIND Architecture

The MIND architecture was created for facilitate obtaining a proper solution in complex knowledge processes, in which the electronic circulation of documents and extracting knowledge from them is crucial. In order to demonstrate this possibility of the MIND architecture, two case studies have been implemented. The first of these concerns the large-scale problem of judicial proceedings. Document in the form of complete files can reach an enormous size. The prototype system is not yet ready to carry out experiments in this target environment. The experiment took place to verify the adequacy of the used notation and the scope of the required functionality.

In a court trial, there are many documents that are often small and have specified structure. The workflow of the files is defined by organizational structure of the courts. In this case, there is a possibility to use the agent system for managing the electronic files.

The second case study involved the issue of evaluation of students in a typical university environment. This problem allows to test the mechanisms of the MIND architecture. In a teaching management system, a person, who receives a document template, is a student, and an evaluation card of one subject is a single MIND document. In the authors file, there are teachers that leads one subject for a group of students. The constituent document is transmitted from one teacher to another, according to the policy file. The teacher is responsible for the subject and the student receive the final version of the evaluation card. In the first case the agent is archived, and in the second, the agent is removed.

Fig. 13 shows the graph of one migration path of the constituent document in the agent system. AUT000 is the student's ID. The student is the author placed in the start activity (A119921_00 is a start activity ID). The constituent

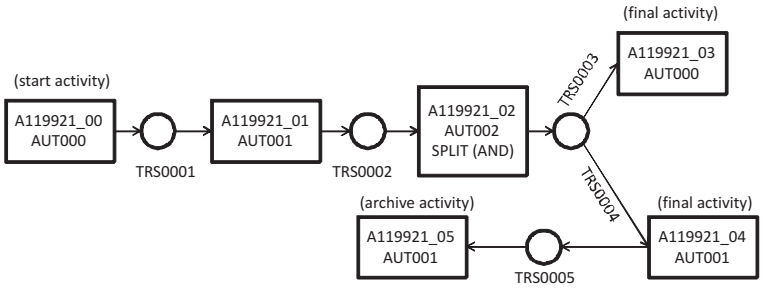


Fig. 13. Graphical representation of the policy file for one student and for one subject (constituent document ID: PAR0001)

document is copied in the activity A119921_02 and the copies migrate to authors: AUT000 and AUT001. There are two final activities. The student is the author of final activity no A119921_03. When the agent reaches this activity, the copy of constituent document is saved on the student's computer. Then the agent is deleted from the computer . The person responsible for the part of subject is the author of the final activity no A119921_04. Both the document and the agent are archived on his computer. This allows to trace the migration path of the document by the responsible author.

Teaching management system has been tested on four containers (the containers have been assigned to the authors: AUT000, AUT001, AUT002 and AUT003). The containers have been run on different computers on the same local network. Three constituent documents (PAR00001, PAR00002 and PAR00003) have migrated between the containers. The start activities have been: 119921_00, 119921_01 and 119921_02.

After the transition from the start activities to the next activities, the agent *Exit* action has been performed. Consequently, two files have been saved on the local computer: the serialized agent file and the XML constituent document file. After editing the document and running the container, the agent has brought back to the active state with modified document inside. If the author has been off-line, the agent with the document has been sent by mail. Sending agent on the mail server has proceeded quickly and without any problem. Receiving the agent from the e-mail server has proceeded successfully, but has taken a time period of few seconds. The whole process, the following the migration path and transferring the constituent documents have proceeded successfully.

The teaching management system in its current form is meeting its objectives, but it is not completely user-friendly. To edit the constituent documents on the system nodes the STMPEdit editor may be used as a local service. However, it is rather cumbersome solution. Hence, there are plans to implement tools that make the system more user-friendly. There are possibilities to implement new functionalities in the MIND architecture through a mechanism of embedded, local and remote services. Adjusting the system for this use case allows for the

execution of larger scale tests. This is the nearest plan for developing the agent system.

8 Conclusions

The prototype server for managing components in the current stage does not implement all the principles of the MIND architecture. In the next version of the prototype new functionality will be added, such as: adding new components, granting of various rights of access and views of the constituent documents, adding new authors and modifying the policy files. A persistent addressing will be implemented to identify author's localization by name instead of computer's IP. This functionality allows the author to work on different computers. A control of the agent migration and edition of the document will be extended by adding conditions of transitions and checking the time of execution of the various tasks. Workflow patterns applicable to knowledge-based organizations will be implemented in order to facilitate the construction of knowledge processes.

The next stage of development of the MIND architecture is to ensure the reliability of processing and security of document. Reliability of the process refers to the possibility of self-diagnosis of distributed document in a situation where some of its component are lost or where some components are changed by unauthorized persons. Security of document concerns encryption of its the content and digital signatures.

Acknowledgments. This research work was supported by the system project Innodoktorant Scholarships for PhD students, III edition. Project is co-financed by the European Union in the frame of the European Social Fund.

References

1. The Apache XML Project. http://xmlbeans.apache.org/sourceAndBinaries/.
2. W3C Recommendation. http://www.w3.org/XML/Schema.
3. Mella G., Ferrari E., Bertino E., and Koglin Y. Controlled and cooperative updates of xml documents in byzantine and failure-prone distributed systems. In *ACM Transactions on Information and System Security*, volume 9, pages 421–460, November 2006.
4. Oster G., Urso P., Molli P., Molli H., and Imine A. Optimistic replication for massive collaborative editing. Technical Report 5719, October 2005.

5. Oster G., Urso P., Molli P., Molli H., and Imine A. Proving correctness of transformation functions incollaborative editing systems. Technical Report 5795, December 2005.
6. OrbiTeam Software GmbH & Co. KG, Fraunhofer FIT. Bscw -basic support for cooperative work, version 4.4, October.
7. Price D.R. Cvs - concurrent versions system. http://cvs.nongnu.org/, December 2006.
8. Lowry P.B., Albrecht C.C., Lee J.D., and Nunamaker J.F. Users experiences in collaborative writing using collaboratus, an internetbased collaborative work. In *35th Hawaii International Conference on System Sciences*, 2002.
9. Workflows and Agents Development Environment. http://jade.tilab.com/wade/index.html.
10. WfMC. Workflow Management Coalition Workflow Standard: . Process definition interface - xml process definition language. Technical Report WFMC-TC-1025, Workflow Management Coalition, Padziernik 2008.
11. Russell N., ter Hofstede A.H.M., van der Aalst W.M.P., and Mulyar N. *Workflow Control-Flow Patterns: A Revised View*. BPM Center Report BPM-06-22, 2006.
12. van der Aalst W.M.P. Patterns and xpdl: A critical evaluation of the xml process definition language. Technical Report FIT-TR-2003-06, Queensland University of Technology, Brisbane, 2003.
13. Workflow Patterns Home Page. http://www.workflowpatterns.com/ patterns/.
14. Nottelmann H. and Fuhr N. MIND: an architecture for multimedia information retrieval in federated digital libraries, 2001.
15. Nottelmann H. and Fuhr N. The MIND architecture for heterogeneous multimedia federated digital libraries. In *Distributed Multimedia Information Retrieval'03*, pages 112–125, 2003.
16. Berretti S., James P. Callan J.P., Nottelmann H., Xiao Mang Shou, and Shengli Wu. MIND: resource selection and data fusion in multimedia distributed digital libraries. In *SIGIR'03*, page 1, 2003.
17. Mikuła B. *Organizacje Oparte na Wiedzy*. WAEK, Kraków, 2006.
18. Starke P.H. *Sieci Petri - podstawy, zastosowania, teoria*. PWN, Warszawa, 1987.
19. Jensen K. and Kristensen L.M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer-Verlag, Berlin, 2009.
20. Siciarek J. Środowisko narzedziowe do wytwarzania inteligentnych dokumentów elektronicznych. Master's thesis, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 2008.
21. Internet Assigned Numbers Authority (IANA): MIME Media Types. http://www.iana.org/assignments/media-types/.
22. S. Josefsson. Base-N Encodings, RFC 4648, Network Working Group. http://tools.ietf.org/html/rfc4648.
23. WfMC. Workflow Management Coalition. . Terminology and glossary. Technical Report WFMC-TC-1011, Issue 3.0, Workflow Management Coalition, Winchester, United Kingdom, Luty 1999.
24. Szczepański J. Serwer usług bazowych do zarzdzania konfiguracj inteligentnego dokumentu elektronicznego. Master's thesis, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 2008.
25. Java Agent Development Framework. http://jade.tilab.com/.
26. Java Mail: Sun Developer Network. http://java.sun.com/products/javamail/downloads.
27. Apache Log4j: Logging Services. http://logging.apache.org/log4j.