Published Version: Godlewska M., Smart Document-Centric Processing of Human Oriented Information Flows, COMPUTING AND INFORMATICS, Vol. 37, iss. 3 (2018), pp. 673-692, doi: 10.4149/cai 2018 3 673

SMART DOCUMENT-CENTRIC PROCESSING OF HUMAN ORIENTED INFORMATION FLOWS

Magdalena Godlewska

Institute of Informatics
Faculty of Mathematics, Physics and Informatics
University of Gdansk
Wita Stwosza 57, 80-308 Gdansk, Poland
e-mail: maggod@inf.ug.edu.pl

Abstract. Usually people prefer to focus on creative rather than repetitive and schematic work patterns. Still, they must spend a lot of time complying with the procedures, selecting the information they receive and repeatedly restoring the previous state of work. This paper proposes the Mobile INteractive Document architecture (MIND) – a document-centric uniform interface to provide both effective communication of content and coordination of activities performed on documents. MIND documents are proactive, capable of initiating process activities, interacting with individuals on their personal devices and migrating on their own between collaborators. Each MIND document is a mobile agent that has built-in migration policy to control its own workflow and services enabling proper processing of contained information. The architecture supports users in the implementation of procedures, and selection of services needed to work on the document. A Personal Document-Agent (PDA) is a further development of MIND aimed at preserving continuity of state of individuals' work to support their creativity and comfort of their daily work.

Keywords: Human-computer interaction, electronic documents, multi-agent systems, collaborative work, workflow management, task coordination, knowledge-based organization

Mathematics Subject Classification 2010: 68-M10, 68-M11, 68-M12, 68-M14, 68-U35

1 INTRODUCTION

Despite of the intensive development of artificial intelligence and machine learning, people are still the key intellectual resource in almost all areas of life. Yet, supporting people's interaction with various knowledge resources would contribute to their productivity and well-being, and help them to focus their activities on creative work, and to put less effort to perform simple, structured tasks.

Working in a group, as well as individually, humans perform certain processes. In an organization of many people, in particular in a *knowledge-based organization* [1], a collaboration process is often implementation of the established procedure. Moreover, collaboration with other workers, in accordance with the organizational procedures, enables converting knowledge of individuals to a collective organization knowledge. The purpose of the knowledge-based organization is to implement the *knowledge process*, in which the human mind is an important element.

There are several problems that hinder effective collaboration, in particular:

1. The worker needs to know the organizational procedures, which are often very confusing. Especially in procedures that are rarely performed, it could take a long time before the worker finds out to whom a particular document should be sent.
2. Collaborators perform procedures manually, and can make mistakes such as sending (receiving) some information several times to (from) a wrong person. This leads to *information overload* phenomenon, that often leaves the worker confused and unable to make a decision [2].
3. Workflow process automation often requires large amounts of work to define the entire process before performing. Moreover, this is often infeasible as the process flow often depends on individual decisions made during its performance.
4. Documents usually play a passive role in the process, which means that they are opened, filled, sent, etc. They generally do not give any support to users, or do that only in a very limited form.

Also in individual work, there are some problems affecting the efficiency of the performed tasks:

5. A user is the only one person who knows his devices and applications installed on them. Nowadays, this is a problem because users use multiple devices to continually perform the same task at work and home including PCs, tablets and smartphones which must be synchronized. A question arises how to synchronize entire systems, not just separate files.
6. A user needs to install the same or similar applications and configure peripherals for each device or OS.
7. Even if a person uses cloud services, like as Google Drive or One Drive, he has to find the right documents and recreate the state of his recent work after changing a device or rebooting its operating system.

8. A user enacts his own accustomed processes while working or resting, which are not supported in any way.

This paper presents the Mobile INteractive Document architecture (MIND) [3] and a special workflow enactment application, a Local Workflow Engine (LWE) [4], enabling a loosely-coupled agent system, capable of coping with the above points.

The MIND architecture is a model of a document-centric uniform interface to provide both effective communication of content and coordination of activities performed on documents. MIND documents are *proactive*, i.e. they are capable of initiating process activities, interacting with individual workers on their personal devices and migrating on their own between collaborators. Thus, each MIND document is a mobile agent, called a *document-agent*. Document-agents have built-in migration policy to control their own workflow and services to properly process contained information. Section 3 provides a more detailed overview of the MIND architecture.

The migration path of a document-agent contains all information and status of the workflow process to perform it locally on users' devices. A document is transferred between users in a serialized form via an available protocol – the MIND architecture does not impose any specific implementation of that. The choice of a concrete protocol depends on the requirements of the organization, in particular on security levels, a number of employees, a comfort of use, etc. In virtual knowledge-based organizations, email can be used as basic medium for exchanging digital documents of any kind. Implementation of the document transfer protocol is discussed in Section 7.

The LWE application mention before is installed on each worker's device participating in the process. It has a workflow enactment capability, i.e. a functionality to activate document-agents and switch documents between the activity and transition phases of the workflow. All LWEs participating in the process, and performing independently, form together both a technologically independent loosely-coupled agent system and a distributed workflow enactment service. Section 4 outlines generic functionality of LWE and the idea of a distributed workflow enactment service.

In the LWE-based MIND system, workers perform activities on documents independently, using their personal devices, and yet collaborate on achieving a common goal. It is possible owing to a migration policy embedded in each document. This policy defines for each document a workflow process composed of specific document-flow patterns [5] that provide process-wide coordination. The document-flow patterns are a result of analysis of the coordination patterns proposed by van der Aaalst [6]. This shows that a relatively small and well defined set of collaboration patterns contains building blocks of arbitrary complex workflow processes in real organizations. Thus, document-flow patterns proposed in this paper, which are based on these collaborations patterns enable modelling and coordination of any workflow process with MIND documents. Moreover, the proposed distributed workflow enactment service allows defining dynamically the workflow process during its actual

execution. Section 5 discusses briefly the document-flow patterns and shows how to modify a workflow dynamically.

The proposed solution allows for significant reduction of the problems with group and individual work mentioned above. Points 1–4 present the problems for which the MIND architecture was developed. The system enabling group work through performing of knowledge processes has been implemented and validated as a part of the MENAID (Methods and Tools for Next Generation Document Engineering) project [7] – Section 7 presents the results of this work. An attempt to solve the problems introduced in points 5–8 is based on a concept to apply the model of the MIND architecture to improve individual work on different devices, with a Personal Document-Agent (PDA) outlined in Section 6.

Section 2 opens this paper by reviewing work related to the presented research, while Section 8 concludes the paper and introduces the possibilities of further development of the presented solutions.

2 RELATED WORK

The idea of an active document is not new. Already in 1996, the Multivalent Document architecture MVD [8] was presented and it was the first significant step in the document-based processing. MVD allowed for treating the document as an object the content of which can be manipulated dynamically. It introduced active functionality with dynamically loaded objects called *behaviors*. The MIND embedded services are similar to the concept of behaviors, however MIND introduces also local and external services, to manipulate a document content, but are not components of documents. This gives documents more flexibility, adjusting them to exploit local resources of visiting devices and to easily add a new functionality.

The Placeless Documents [9] extend document functionality with active properties that can not only allow to manipulate a document content, but also can manage of a document structure and its workflow. These are also the main features of the MIND architecture. However, the Placeless Documents are reactive, i.e., they respond to external events, while MIND documents are proactive – they initiate their own behavior as they have their own embedded functionality (services).

It is worth mentioning the document-agent MobiDoc platform [10] as the concept of a proactive document, capable of travelling between computers under its own control. This platform was closely related to the particular technology, thus lacked forward compatibility, and, consequently was difficult to implement in a large-scale. The idea of a document-agent is very interesting, as openness and technological independence are very important features of modern systems. In the LWE-based MIND system, a technological independence is one of its priorities. In a special case, when each user has a different operating system, each LWE may be implemented in a different technology. The document transfer protocol can be also adapted to the requirements of implementation. It gives the opportunity to create an agent platform with all benefits of multi-agent systems, but without a need to implement

a full-size agent platform that depends on the chosen technology, has to be updated regularly and requires additional skills from administrators.

Among more recent solutions that allow agents to perform a workflow is WADE (Workflow and Agents Development Environment) [11] agent platform based on JADE (Java Agent DEvelopment framework) [12]. WADE agents embed a micro-workflow engine, capable of executing workflows compiled before launching a workflow. Performing activities may be delegated by one agent to another and in principle it is not related to agent mobility. This solution follows a classic central workflow enactment philosophy, and differs from it only in decentralization of a global process state into a process states controlled by micro-workflow engines running inside agents. This solution makes the WADE platform different from the MIND architecture. In MIND, a workflow object specified formally with XPDL (XML Process Definition Language) [13] is bundled with a document and contains its internal state of which it is of full control. Then, a workflow is enacted outside of agents by local workflow engines (LWEs). More details on the concept of distributed workflow enactment can be found in Section 4. The advantage of such a solution is that a respective XPDL file may be modified during process execution. Moreover, MIND document-agent is the only communication interface, which makes it technologically independent in a loosely coupled and heterogeneous distributed system.

There are some interesting solutions, such as AMODIT [14], that use elements of artificial intelligence to improve workflow processes. The main concept is to analyse the content of the documents and previous decisions, in order to suggest the next steps of the workflow. The mentioned system is a commercial product based on a client-server model, which does not exhibit openness, technological independence and multi-agent approach, as the MIND architecture does. The idea of adopting a Multi-Agent System and machine learning to support cooperation based on documents was introduced in [15]. The presented idea was based on the analysis of the documents and users' behaviors, omitting the problem of process definition. The MIND architecture, allowing for the dynamic modification of the process, opens the possibility of learning the flow of the processes and behavior of users.

One of the problems in individual work, mentioned in point 7, is saving a state of a recent work. This problem occurs, for example, when the system needs to be rebooted. The user often loses the information about all documents or applications opened before. Sometimes, it can be really frustrating. Users of the Mac OS receive the greatest support on this issue. They can simply decide if they want to re-open applications in the same state they left them before logging out [16]. It is also possible, although not so easy, in Linux systems (e.g. Ubuntu). There is the `dconf-editor`, where the `auto-save-session` check box may be selected. The `dconf-editor` is not preinstalled by default, so even in experienced users may have a problem using it. Finally, in the Windows systems it is not possible to re-open active programs on reboot without installing an external application. There is only the possibility to automatically re-open any Explorer windows that were opened before rebooting. There is a list of external applications that enable to restore programs or folders after system reboot, e.g. Cache My Work [17] or SmartClose [18].

Restoring applications after a system reboots solves the problem on just one operating system. Currently, users would like to work on many devices, e.g. to start work in the office, continue it on the way home, and end at home. On each device, users would have to recreate a state of their current work. Google Drive [19] or One Drive [20] cloud based solutions could be very useful for sharing files through different devices, but they cannot ensure continuing work at the point where it previously has been interrupted.

The MIND architecture allows for implementation of a Personal Document-Agent (PDA), which can store a global state of the work, interact with a user, operating systems and services. This solution differs from the idea of Virtual Personal Assistant (VPA), like Apple Siri [21], Google Assistant [22] or Amazon Alexa [23], which are based on interaction with a user. The latter act as local applications or web services and they support a user in everyday duties, e.g. by turning on an alarm, checking the weather, reminding a meeting or making purchases. PDA proposed in this paper is a smart middleware between operating systems and applications rather than a yet another application like VPA (it can, however, use VPA as a local or external service).

3 THE MIND ARCHITECTURE

Traditionally, electronic documents have been static objects downloaded from a server or sent by an email. MIND allows static documents to be converted into a set of dynamic components that can migrate between collaborative workers according to their migration policy.

The concept of the MIND document lifecycle is illustrated in Figure 1.

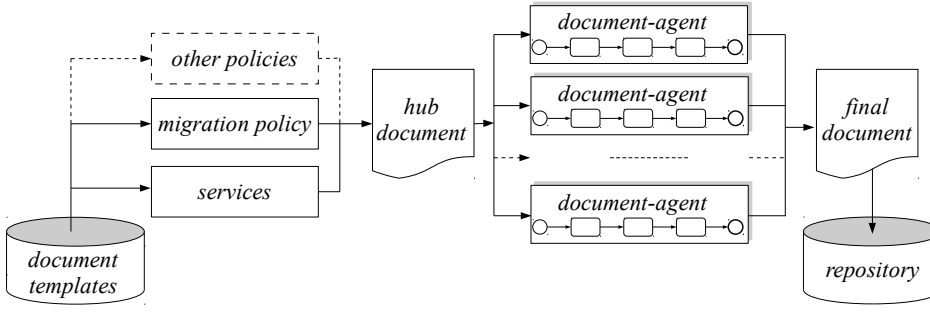

Figure 1. The MIND document lifecycle [4]

A *hub document* is formed on the basis of document templates that includes *migration policy*, which specifies steps of the knowledge process and services that will be performed on different parts of a document during the process. The hub document that contains static elements (e.g. XML files), is *unmarshalled* to mobile objects called *document-agents*, which perform their mission in a distributed agent

system. Each document-agent migrates across an organization and interacts with its workers.

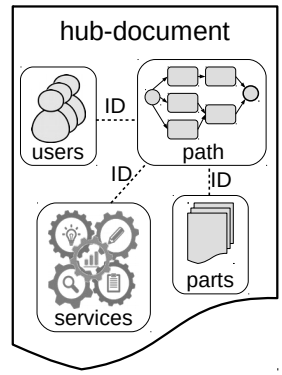

Figure 2. Components of a MIND document-agent

The *hub document* contains five components, outlined in Figure 2:

hub-document, a headline of the MIND document. It contains its ID and other related information necessary to identify a document-agent with a given process.

users, containing data about users who participate in a process.

parts, containing information about constituent documents, which are actual documents on which work will be performed, e.g. PDF files, MSOffice documents, HTML forms, etc. Such constituent documents are parts of the MIND document, which does not mean that they always migrate with document-agents. Sometimes, a part can describe an external location of a constituent document.

services, containing information on a document functionality available during a process. Three types of services are possible: *embedded*, transferred together with a document-agent, *local* which may be acquired by a document-agent from a local user's device, and *external*, to be called on remote hosts by a user's system at the request of an arriving document-agent. Services provide document-agent functionality and make it proactive as it was mentioned in Section 2.

path, defining migration policy (workflow) of each part of a document. It specifies steps of a process and activities that should be performed at each step of a process. The other components refer to the path and are distributed according to the path during a process.

The presented document-agent model complies with the Belief-Desire-Intention (BDI) definition of an agent [24]. BDI is an agent architecture that reflects model of human practical reasoning, developed by Bratman [25]. The main goal of the MIND architecture is to support users in making reasonable decisions, so BDI model fits

well to document-agents, because it represents the natural knowledge processes of a human mind.

A model considers *beliefs* as a knowledge about the world, *desires* as goals of an agent and future-directed *intentions*, which are composed of plans, as an important and irreducible concept. Particularly, for the MIND document-agents:

- the *world* is a knowledge-based organization, in which users have devices and applications that can be an environment for agents and these devices are from time to time connected to a network enabling migration or communication of agents.
- *beliefs* are components of document-agents: hub-document, users, parts and services. An agent does not need to know all users or services of an organization. It is enough for the agent to know a subset of the world that is needed to perform a designated process. Especially, during a process, a document-agent can "discover" the world, i.e., add users, parts or services.
- *desires* are represented by the path component that reflects steps of a knowledge process striving to achieve an organization's goal.
- *intentions* are related to the agent's autonomy. In the Bratman's model, plans are initially only partially conceived, with details being filled in as they progress. The MIND document-agent autonomously follows its path, which can be modified dynamically during the process. It is also able to designate the services needed to perform the appropriate action on a constituent document.

Although MIND meets requirements of the BDI model, it is not its direct implementation, as in the latter, agents do not have any specific mechanisms to learn from past behavior and adapt to new situations. The MIND document collects all information about its performance, which can be an appropriate training set to teach agents to perform a process better in future, for example negotiation [26].

4 DISTRIBUTED WORKFLOW ENACTMENT

A key feature of the MIND architecture is physical distribution of business process activities, performed dynamically on a system of independent personal devices. The MIND documents have built-in process definition and functionality (the respective path and embedding service components mentioned in the previous section). This makes them document-agents, which are autonomous and mobile. Especially, they are independent of any particular platform supporting workflow enactment and they are capable of launching individual activities onto various users' devices, which maintain process coordination across an organization.

A standard WfMC *Workflow enactment service* [27] interprets the process description and controls sequencing of activities through one or more cooperating workflow engines. Even if workflow engines are distributed, workflow enactment is centralized in most of the implementations, because control data must be available to all engines. Contrary to that, in the MIND architecture, all data needed

for workflow enactment are embedded in documents, and allow for implementation of a really distributed workflow enactment service, consisting of Local Workflow Engines (LWEs).

The idea of the distributed workflow enactment service built on top of LWEs is illustrated in Figure 3.

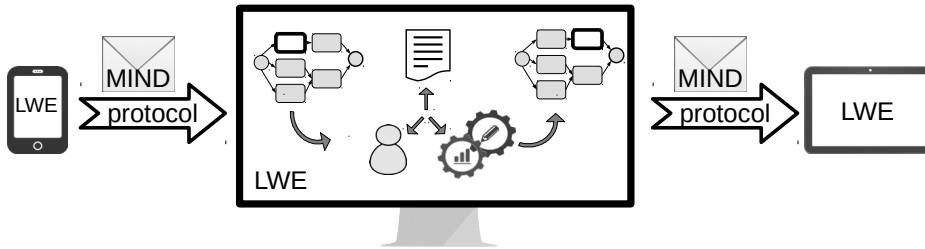

Figure 3. Distributed workflow enactment service based on LWEs

Each LWE is independent of other LWEs, so it can be implemented in any technology and adapted to requirements of particular devices, especially mobile devices such as tablets and smartphones.

A MIND document-agent is sent in a static, serialized form via any network protocol available, enabling transmitting a document-size data. For example, it can be a FIPA MTP standard protocol [28], ordinary HTTP protocol, or just email protocols such as SMTP or IMAP. Section 7 outlines several possible implementations of transport layer for the MIND documents.

The initial state of workflow enactment is when LWE downloads a serialized document on a local device and activates it, which means unmarshalling and launching its embedded functionality. The activated document-agent begins its mission by obtaining the path component and determining the current activity that should be performed in this particular step of the process. The document-agent contains all data needed to determine the state of the workflow process locally. Thereafter, the document-agent may interact with users, their local systems and some external services. If the next activity is intended for another user, the document is serialized again, packed and sent to the next user via a network protocol.

All workflow process data are brought to LWE by a document. So document-agents are the only means of communication between LWEs in a distributed system. Since LWEs can be implemented in different technologies, they can be adapted to various hardware.

5 DOCUMENT-FLOW PATTERNS

To run the appropriate activity in a workflow process locally, a MIND document-agent must contain not only a process definition, but also its current execution state. This state contains an ID of a *current activity* assigned to a given user,

stored in an external XPDL attribute. But it is not enough in many situations in a process. In order to define all attributes that form a process state, a set of *document-flow patterns* has been defined. These attributes, namely *current activity*, *counter*, *sentinel*, *finish* and *semaphore* are implemented as internal variables of the MIND document and operated by a handling LWE as described below.

Based on the work of van der Aaalst [6] and characteristics of a distributed system three categories of the document-flow patters have been identified: distributed state patterns, coupled state patterns, and embedded state patterns [4].

5.1 Distributed State Patterns

These patterns describe situations in which a next activity or activities can be determined solely on the state of the *current activity*. Four patterns of this type have been distinguished:

Document sequencer involves a user transferring a document to another user. The document may be transferred in its entirety in one package or if it is too large for a transport layer protocols, it may be partitioned into several packages. If the document is partitioned into smaller pieces, the numbering of packages is entered and the last package is marked with the *sentinel* attribute. Thanks to that, a recipient knows if all packages of the document have been already received, even if they come in a different order.

Document splitter creates identical copies of a document or partitions it into separate fragments. Whether the document would be copied or decomposed depends on the functionality of the document provided by services. The resulting documents get new document IDs and they are transferred to respective users specified in a migration policy. They are modified parallelly while executing different activities. For each document, ID of an activity that would be executed on it in the next step is set as a *current activity* attribute value.

Depending on conditions defined in the workflow process, the splitter produces a different number of copied/fragmented documents. This number is stored in the *counter* attribute. If the splitter has n outgoing branches, the *counter* assumes a value from 1 to n: 1, if only one branch was chosen and n, if all branches were chosen.

Document merger complements the document splitter pattern and merges all received documents in one. Of course, it may involve various document functionality, depending on whether the preceding splitter has been cloning or decomposing. But before merging, all expected documents must be delivered. The LWE client on the basis of path component of the first received document (the value of the *counter* attribute) determines a number of expected documents that have to be merged.

Document iterator enables repeated execution of some sequence of activities controlled by a condition specified in a respective document migration policy. Many

workflow languages allow for creating unstructured loop with more than one entry or exit point, that do not need any specific looping operators [6]. In this case, the value of the one boolean *finish* attribute brought by document to logic gateway can decide, whether a workflow should continue a loop or exit from it.

5.2 Coupled State Patterns

Sometimes completion of an activity performed by one user may require a notification on a state of some activity performed by another user somewhere in an organization. That involves a notion of asynchronous signals, sent between different parts of a workflow process. Three document-flow patterns of this kind have been distinguished: deferred choice, milestone and cancel activity.

Milestone and deferred choice are used to deal with situations when the current activity of one user has to be blocked until a signal notifying on some external event has been received from another worker. Both patterns require the *semaphore* attribute and embedded functionality to handle it. Initial value of the *semaphore* is closed, so if a signal from another worker has not been received, the current activity is blocked. Upon receiving a signal, a waiting activity is resumed. Milestone just blocks some activity of one user by another. Thus, a signal does not have to have any specific value. Deferred choice is used when sending a given document has to be postponed until the user gets information to whom it should be sent. Thus, a signal should have a value identifying a next user, e.g. the user's ID.

Cancelling pattern. Implementation of this pattern depends on which process activities should be cancelled. If a particular activity should be cancelled, a cancellation signal is sent only to LWE responsible for its execution. The decision on cancelling the activity is immediate for a receiving device or does not make sense any more if a document has been sent to another user. More problematic situation is to cancel a document (one part of the MIND document), because it requires a designation of its location. It is possible to search for a document in all places indicated by the workflow, but this solution is expensive and can be unreliable. Another solution is to chase a document that can leave a trail in each visited LWE. It is worth mentioning that a document flow takes hours, even days, rather than seconds, so it would be a reasonable solution. For example, the Intel's Email Service Level Agreement defines the acceptable time frame for replying to emails in 24 hours [29]. Using an external "ground control" service [5], which introduces the ability to track document-agents globally, allows for simplification of a document cancellation.

The cancelling pattern does not need any additional attributes to enable cancellation. In principle, every activity or every document can be cancelled. The *cancel* attribute is added after the cancellation to indicate this fact, which may be needed for further analysis of a process.

5.3 Embedded State Patterns

Performing an activity by some user may require a subprocess delegated to someone else with activities not specified originally in a migration policy of an arriving document. States of such a subprocess are embedded in a state of a current activity referring to that subprocess. Two types of subprocess can be distinguished: internal and external.

Internal subprocess. If a current user is authorized to extend an original migration policy of a document with new activities, they constitute an internal subprocess. Neither a structure of the internal subflow nor identity of added users have to be known earlier to a workflow originator (designer).

This is a key pattern in the MIND architecture. Internal subprocesses can be added during the workflow execution. It allows for defining a relatively small initial workflow and its dynamic expansion during execution. The activity assigned to the user is converted into a subprocess activity. The main advantage of editing a process during its execution is that it can be built ad hoc of small pieces. Each worker can define a fragment (subprocess) of a workflow, i.e. each user can be a document originator.

External subprocess. A performed activity may call some external subprocess whose structure is unknown for both, a workflow designer and a performer of a current activity. The external subprocess is often performed outside of an organization.

6 A PERSONAL DOCUMENT-AGENT

A special case of an iterator is recursion, which means the ability of some activity to invoke itself during its execution [6]. For documents, it is a situation in which a user performs the same activity several times on different devices. From the perspective of an entire process, this is still one activity, but from the perspective of a document-agent, executing conditions on different devices can vary significantly. This situation requires creating a subprocess with one input and one output, in which a document will be transferred between user's devices and edited until the end of work.

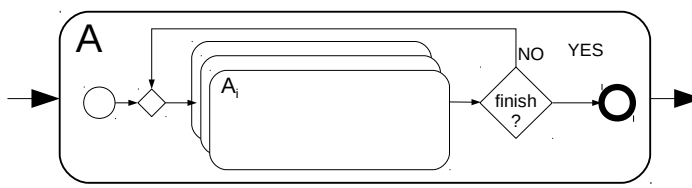

Figure 4. A recursive document-flow pattern

Figure 4 presents a recursive pattern as a subprocess added to the activity **A**, being a certain task that a user has to perform on a document part. In many cases, a user does not perform this task at once. Thus, the document has some intermediary states between receiving and sending. It can be opened, edited, saved and closed many times during one activity. LWE can put to sleep and next wake up a document-agent as many times as needed, including the restoration of necessary services. This is a typical behavior of the MIND document and does not require any subprocess. A subprocess would be needed if one activity would have to be performed on different devices of the same user. It is a common situation, as many people have several devices at their disposal. Each activity $\mathbf{A}_i$ is an i^{th} copy of the activity **A** started before adding the subprocess and each one expresses the same task **A**. After performing a certain stage of the task represented by $\mathbf{A}_i$, the user decides, if a work is completed. If not, it may happen that it would be continued on another device as $\mathbf{A}_{i+1}$.

A document-agent, when interacting with LWE, can recognize on which device it currently resides. Thus, it can dynamically adjust to various execution contexts provided by devices. It may have a certain performance strategy for a given device, which it can consult with users by negotiating with their device [30].

The recursive pattern proposed in Figure 4 can been exploited by the concept of a Personal Document-Agent presented in Figure 5.

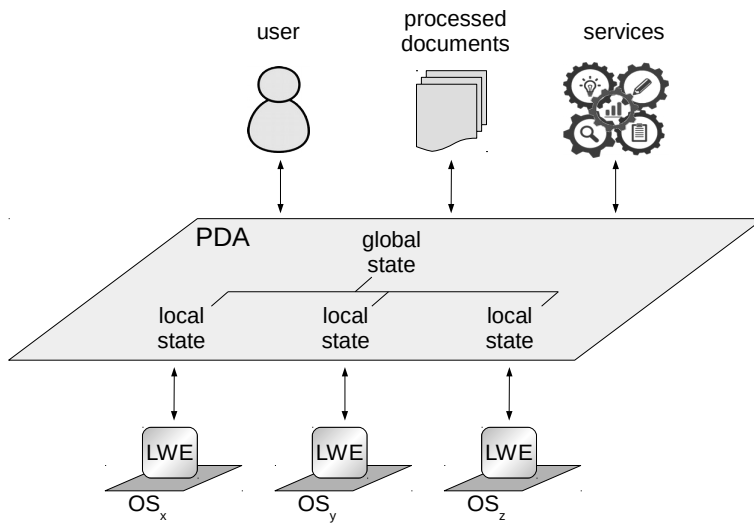

Figure 5. PDA as a middleware between user and his OSs

Assume that everything users do on their devices is the implementation of their *personal process*, and LWE has been installed on each of them. The special MIND document does not implement a path assigned in advance, but follows users to

support their work. This is a process that is being built ad hoc. In a personal process, a state of work is often more important than the sequence of activities performed. A user interrupts it at a certain point and after some time wants to resume it in the same point where it was interrupted. This is not a problem of just one device. Most operating systems allow for putting them into a sleeping or hibernating mode. Incidentally, the system is not able to wake up and then the state of work would be lost. It would be much more useful to provide a prospect for continuing work from the sleeping/hibernating point, but on another device.

PDA collects the data about tasks performed by the user, as snapshots of a state of work. Next, PDA distributes the obtained data into two sets: data important for the work on local operating system (a local state) and data that express the state of performing a certain task (a global state). This distribution is supported by the MIND service component (see Section 3). The local state contains information about local services that users used in their latest work, while the global state registers the use of external and embedded services. But not only, the global state also includes mappings between corresponding applications on different devices, for example, Adobe Acrobat Reader on PC with the Windows OS would correspond to Google PDF Viewer on a tablet with the Android OS. It also can indicate discrepancies, for example an AutoCAD application would be only available on a computer at work.

PDA also uses the part component to enable users interaction with the documents they have recently worked on. It keeps information not only about open documents, but also about specific places in these documents, if it is possible for a given format. Thanks to this, users changing the device, are redirected to the exact place in document.

Interaction between PDA and a user could evolve in time. PDA can use many services to cooperate with a user, including voice communication. However, turning on too many services at the beginning, would overload the system and a user might not be able to use so many of them. Instead, PDA should be kept as small as possible and should be able to collect data about the user's habits to better support the latter. For this purpose, various solutions are currently being adopted by the Author, such as machine learning to automatically built document migration, strategies or emotion recognition of its local user [31]. For example, let imagine the situation, that PDA "knows" a user named Bob. PDA knows, that Bob never works on Friday evenings. If he turns on the computer, he usually watches movies or plays video games. He gets angry when anything reminds him of work. Therefore, PDA would ask Bob what to turn on for him: his recently acquired video game or a Netflix service, and only one exception would be allowed – when the tight deadline to complete task is about to expire. Then Bob would accept the suggestion to continue his work instead. In such a case, PDA would readily recreate the state of his recent work.

PDA can collect some other data, not only a status of work. It is practical to collect data about peripheral devices, so as not to have to configure them for each device separately and to communicate with them faster and easier.

7 CASE STUDY AND VALIDATION

The MIND architecture was created to facilitate knowledge management in complex knowledge processes, in which a flow of electronic documents and extracting knowledge from them are crucial. Coordination of document workflows may be often enforced by law, especially when related procedures are implemented manually by workers – as it takes place in court trials, crash investigations or complex medical cases.

The first case study was a judicial proceeding system. A real judicial case in the form of complete files could reach an enormous size. The purpose of this exercise was to consult with court workers (judges, attorneys, counsellors, judicial officers, etc.) to verify the usefulness of the proposed document-flow patterns and their required functionality. In court trials, there are many constituent documents that have specified structure and workflows are precisely defined by legal procedures. So, using MIND-like documents would be essential to redirect attention of all stakeholders involved to the content of a court trial, rather then concentrating on complying with the legal rules governing it. A feasibility study of the MIND architecture was carried out in cooperation with lawyers and a company providing software for courts. This allowed for developing the MIND document model and defining the document-flow patterns.

The second case study involved the issue of evaluation of students in a typical university grading process. It allowed to test the validation of implementing the MIND architecture in a real environment, using the document-patterns.

A worker of Registrar's Office forms a grade roster hub document, and transfers it to a Course Leader. A Course Leader runs his own subprocess of collecting credits from instructors during the entire semester; structure and implementation of that subprocess is irrelevant to the Registrar's Office. While the Registrar's Office may use an online grade system for one-time roster submission and approval, a Course Leader is responsible for all subprocesses of collecting credits and has modification, control and cancellation permissions. Instructors receive only a class roster of their student's groups, which can be filled out at any time, before a specified deadline.

Several prototype applications were implemented to validate the MIND architecture and demonstrate its implementability in the context of the above mentioned case study. The main task of the implementation was to create an environment for document-agents: for their actions on users' devices and for transferring between devices. It was common for all prototypes to use XML [32] for MIND document implementation. The path component that describes the document workflow has been specified in XPDL. XML files can be easily transformed to other formats, tailored to the specific technology.

The first implemented prototype [3] used the JADE. Document-agents extended the JADE MobileAgent class and LWE was a component of a JADE container. The transport layer was built on top of the JADE IMTP protocol. However, users were hesitant to use this prototype as difficult to configure and maintain, subscribing too

much to the specific technology, and requiring troublesome inclusion of additional ports, often blocked by intermediary firewalls.

Further prototypes used email, and standard email's protocols, as a transport layer, with LWE implemented simply as a lightweight email client. Email is the most popular computer mediated communication in the workplace, as a simple textual form combined with a possibility to disseminate attachments in any format. This solution did not require additional skills from users and could support asynchronous work. LWE was implemented in several leading technologies: as a Java desktop application for PCs and laptops, and for mobile platforms: iOS, Windows Phone and Android [33]. These implementations used, however, different available libraries for email messaging, serialization and compression of documents, and XML data binding. LWE prototypes were tested simultaneously, while performing one process, so they formed a really heterogeneous system.

Installation of LWE on each device is recommended to take full advantage of a distributed system. The implementability of LWE was proven by prototypes implemented for various platforms. They were lightweight standalone applications and there was no need to configure any servers or databases. With email protocols used as the underlying transport layer, the configuration proceeded in the same way as the configuration of a typical email client. Alternatively, LWE could be provided as a Web service – especially in cases, when a user has a device for which LWE has not been implemented yet. Also, a user may play a marginal role in the process or just want to refuse installation of LWE.

This system based on email worked satisfactorily and has been considered by users as friendly. The Course Leader was free to implement his evaluation process in any way and course instructors could perform their activities using their personal mobile devices in any time, even if they were out of their campus network. The grading process involved both scheduled and unpredictable events, such as project assessment or homework collection for the former, and grade correction or disciplinary actions in a case of academic misconduct. These events were effectively handled with document-flow patters outlined in Section 5. The users' satisfaction was also influenced by: a simplicity of use, configuration, and easiness of the LWE application, as well as the ability to work without a permanent internet connection.

8 CONCLUSIONS AND FUTURE WORK

One of the main objectives for the presented MIND architecture has been its openness to new policies, services and diverse applications. Some of them have already been implemented, while others are still in the development phase.

Executability and *mobility* constitute the enabling services for MIND document-agents (see Section 7). The former involves unpacking, assembling and activating arriving document components to enable execution of the current activity, and after that packing them back before their departure, while the latter involves transporting them between personal devices of users to proceed with subsequent activities.

Next, *reliability* of the MIND agent system has been provided by a "ground control" external service [5] to make the distributed workflow enactment system more useful and trustworthy. It allowed for estimation of the global state of a distributed loosely coupled system, taking into account transport layer errors, unforeseen actions of users and process modifications. The "ground control" service together with LWE enabled communication between a persons responsible for executing a process and performers of a related activity. Additional permissions and rules allowed to determine which participant could control the process execution and make decisions in unforeseen or conflict situations. That also allowed for introducing the *choreography policy* [5] into the process enactment.

There is also a *security* issue, which answers the question: what to do if a document gets to an unauthorized person? LWE may require authentication of a user before unpacking and activating document components – to verify if the performer assigned to the current activity is the same person as the recipient of the document. An interesting solution for that has been proposed in [34]; it introduced a biometric face recognition mechanism built in MIND document-agents.

Finally, a *negotiation* capability was added to MIND documents to resolve possible conflicts between document-agents and user's devices they visit to execute a particular activity at their workflow [30].

Personal Document-Agent (PDA) presented in Section 6 is the next concept building on the MIND architecture. It explores executability and mobility of the MIND document-agents to improve the work of a specific user with his devices and peripherals. Thanks to this, a user may have an impression of continuing work from the point where it was interrupted, despite of changing the device and its location.

The MIND architecture enables dynamic modification of the workflow process. After the workflow process is completed, the document is archived (see Figure 1), and the data collected during it are a great base for analysis. Retracing already completed processes allows for their optimization in accordance with the real behavior of users. Machine learning approaches can be used to choose the best path or services in the process. During the process, users can assess the accuracy of the activity, that they performed. For example, if a document got to someone's device unnecessarily, it could learn not to follow such a path in the future.

Acknowledgment

This work was supported in part by the National Science Centre, Poland, under Grant No. DEC1-2011/01/B/ST6/06500.

REFERENCES

[1] Bhatt, G. D.: Organizing Knowledge in the Knowledge Development Cycle. Journal of Knowledge Management, Vol. 4, 2000, No. 1, pp. 15–26, doi: 10.1108/13673270010315371.

[2] Spira, J. B.: Overload!: How Too Much Information Is Hazardous to Your Organization. John Wiley and Sons, 2011.

[3] Godlewska, M.: Agent System for Managing Distributed Mobile Interactive Documents in Knowledge-Based Organizations. In: Nguyen, N. T. (Ed.): Transactions on Computational Collective Intelligence VI. Springer-Verlag, Berlin, Lecture Notes in Computer Science, Vol. 7190, 2012, pp. 121–145.

[4] Godlewska, M.—Wiszniewski, B.: Smart Email: Almost an Agent Platform. In: Sobh, T., Elleithy, K. (Eds.): Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. Springer-Verlag, Berlin, Lecture Notes in Electrical Engineering, Vol. 313, 2015, pp. 581–589.

[5] Godlewska, M.: Reliable Document-Centric Processing and Choreography Policy in a Loosely Coupled Email-Based System. International Journal on Advances in Intelligent Systems, Vol. 9, 2016, No. 1-2, pp. 1–13.

[6] Russell, N.—ter Hofstede, A. H. M.—van der Aalst, W. M. P.—Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, 2006.

[7] MeNaID. National Science Center, Poland, Grant DEC1-2011/01/B/ST6/06500, 2012-2014. Available on: http://menaid.org.pl, 2017.

[8] Phelps, T. A.—Wilensky, R.: Toward Active, Extensible, Networked Documents: Multivalent Architecture and Applications. Digital Libraries (DL '96), 1996, pp. 100–108, doi: 10.1145/226931.226951.

[9] Dourish, P.—Edwards, W. K.—LaMarca, A.—Lamping, J.—Petersen, K.—Salisbury, M.—Terry, D. B.—Thornton, J.: Extending Document Management Systems with User-Specific Active Properties. ACM Transactions on Information Systems (TOIS), Vol. 18, 2000, No. 2, pp. 140–170, doi: 10.1145/348751.348758.

[10] Satoh, I.: Mobile Agent-Based Compound Documents. Proceedings of the 2001 ACM Symposium on Document Engineering (DocEng '01), ACM, 2001, pp. 76–84.

[11] Telecom Italia. Workflows and Agents Development Environment. Available on: http://jade.tilab.com/wade, 2017.

[12] Telecom Italia. Java Agent Development Framework. Available on: http://jade.tilab.com, 2017.

[13] WfMC. Workflow Management Coalition: Process Definition Interface – XML Process Definition Language (Version 2.2). Technical Report WFMC-TC-1025, 2012.

[14] AMODIT Web Site. Available on: http://amodit.com/, 2017.

[15] Enembreck, F.—Barthès, J. P.: Agents for Collaborative Filtering. Cooperative Information Agents VII, 7th International Workshop Proceedings (CIA 2003), Helsinki, Finland, August 2003, pp. 184–191, doi: 10.1007/978-3-540-45217-1_14.

[16] Apple Web Site. Automatically Re-Open Windows, Apps, and Documents on Your Mac. Available on: https://support.apple.com/en-us/HT204005, 2018.

[17] Cache My Work Web Site. Available on: http://cachemywork.codeplex.com/, 2017.

[18] SmartClose Web Site. Available on: http://bmproductions.fixnum.org/smartclose/index.htm, 2017.

[19] Google Drive Web Site. Available on: `https://www.google.com/drive/`, 2017.

[20] One Drive Web Site. Available on: `https://onedrive.live.com/about/pl-pl/`, 2017.

[21] Apple Siri Web Site. Available on: `https://www.apple.com/ios/siri/`, 2017.

[22] Google Assistant Web Site. Available on: `https://assistant.google.com/`, 2017.

[23] Amazon Alexa Web Site. Available on: `https://developer.amazon.com/alexa/`, 2017.

[24] Rao, A. S.—Georgeff, M. P.: BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multiagent Systems (ICMAS '95), 1995, pp. 312–319.

[25] Bratman, M. E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA, 1987.

[26] Kaczorek, J.—Wiszniewski, B.: Document Agents with the Intelligent Negotiation Capability. Knowledge and Cognitive Science and Technologies (KCST 2015), Proceedings of the 19th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI 2015), Orlando, FL, USA, July 12–15, 2015, pp. 353–358.

[27] WfMC Workflow Management Coalition: Terminology and Glossary, WfMC, Winchester, UK, Technical Report WFMC-TC-1011, Issue 3.0, 1999.

[28] Foundation for Intelligent Physical Agents: FIPA Agent Message Transport Service Specification, Geneva, Switzerland, 2000.

[29] Spira, J. B.—Burke, C.: Intel's War on Information Overload: A Case Study. Basex, Inc., 2009.

[30] Kaczorek, J.—Wiszniewski, B.: Augmenting Digital Documents with Negotiation Capability. 13th ACM Symposium on Document Engineering (DocEng 2013), Florence, Italy, 2013, pp. 95–98, doi: 10.1145/2494266.2494305.

[31] Landowska, A.—Szwoch, M.—Szwoch, W.: Methodology of Affective Intervention Design for Intelligent Systems. Interacting with Computers, Vol. 28, 2016, No. 6, pp. 737–759, doi: 10.1093/iwc/iwv047.

[32] Bray, T.—Paoli, J.—Sperberg-McQueen, C. M.—Maler, E.—Yergeau, F.: Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium, Recommendation REC-Xml-20081126, 2008.

[33] Wiszniewski B.: Interactive Documents for Network Organisations. Adjacent Digital Politics, Ltd., 2013.

[34] Siciarek, J.—Smiatacz, M.—Wiszniewski, B.: For Your Eyes Only – Biometric Protection of PDF Documents. 2013 International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE '13), Las Vegas, USA, 2013, pp. 212–217.

Magdalena Godlewska received her M.Sc. in computer science from University of Gdansk. She received her Ph.D. degree from Gdansk University of Technology also in computer science. Area of her interest, in general, is document engineering. She got two scholarships for Ph.D. students: the first one awarded by the Poland Pomeranian Special Economic Zone and the second one co-financed by the European Union. She participated in Grant "Methods and Tools of Future Document Engineering – MENAID" – financed by the Poland National Science Centre. She is currently working at the Institute of Informatics at the University of Gdansk.