Novi Sad J. Math.
Vol. 36, No. 2, 2006, 65-79

LESSONS LEARNED FROM THE IMPLEMENTATION OF A WORKFLOW MANAGEMENT SYSTEM USING MOBILE AGENTS

Zoran Budimac[1], Dragoslav Pešović[1], Mirjana Ivanović[1], Nataša Ibrajter[1]

Abstract. In [1, 8] we proposed a complete design of a workflow management system using mobile agents. The proposed solution was highly distributed and almost without centralized control. Therefore, it had better characteristics than other corresponding designs, including ones that also use mobile agents as the infrastructure. In this paper we concentrate on the experience that we gained during the implementation of the system. The highlights are: the changes in the original design that are enforced by the choice of underlying mobile-agent system, distribution of responsibilities among entities of the system, and possibility to use stationary service agents. We also discuss one "worker" and its architecture.

AMS Mathematics Subject Classification (2000): 68U99

Key words and phrases: distributed programming, workflow management, mobile agents

1. Introduction

An infrastructure for implementation of workflow management system using mobile agents was proposed in [1, 8]. Since that time, the basic parts of the proposed infrastructure have been implemented. However, the proposal went through a series of modifications, regarding certain design features, as well as the choice of the mobile agent system used for implementation. The essence of the proposed infrastructure was three-part architecture consisting of the abstract class *Task*, as well as of the work-server and work-host classes.

Abstract class *Task* represented an abstract work in the proposed workflow system. This class is a descendant of a class that represents a mobile agent in a chosen mobile agent system. Objects of the class *Task* thus become mobile as well. The class contained attributes containing work identification, deadlines, owner of the work, etc. and methods for work externalization, internalization, and presentation of user-interface. The class also contains an itinerary. The itinerary is a list of triples of the following form: (node, condition, methods). It represents a flow of work-agent through a network. A node represents an address of a node where the work will transfer itself from the current node. Only methods enlisted in the list methods are active on the current node. The

[1]Institute of Mathematics and Informatics, Faculty of Science, University of Novi Sad, Trg D. Obradovića 4, 21000 Novi Sad, Serbia, e-mail: {zjb,dragoslav,mira,natasha}@im.ns.ac.yu

work-agent will transfer itself to the node when and if the condition (a logical function) is fulfilled.

Every node contains one work-host that can be implemented as a stationary agent. Its main role was to offer basic data about all work-agents that currently reside on the node. The host-agent enables the user to choose one of the work-agents. It establishes connection between the user and every residing work-agent using appropriate work-agent attributes and invoking its methods.

Work-server represented a program that executes all the time on every node in the workflow system. Its basic tasks were the following:

- "Listens the designated port, waits for incoming agents, internalizes them, puts them into a list of agents on that host, and activates them.
- Informs the user about an arrival of a new work.
- For every work-agent periodically calls the function condition for the current node.
- If the work-agent is too long on the same node, alerts the user directly or using some existing service (e-mail, for example).
- At the end (before switching off the computer), externalizes all agents that are currently under its supervision.
- At the beginning of its execution (after the computer is switched on), internalizes all saved agents." [1, 8]

In the new design, duties of a work-server have been distributed over other participants in the process.

The rest of the paper is organized as follows: second section gives a short overview of mobile agent systems. Third section focuses on most important changes in the original design. Fourth section describes one of stationary agents offering additional service, while fifth section introduces one characteristic worker. Sixth section concludes the paper.

2. The Choice of the Mobile Agent System

Many of the agent systems developed so far have been research prototypes, and only a few of these have been employed outside of their own university or research institute [3, 4, 13, 16, 21]. In the process of choosing the mobile agent system intended to be used for the implementation of the workflow system, the major research issues of the crucial significance for the development and application of mobile-agent programming systems have been distinguished. Moreover, several prominent mobile agent systems have been surveyed, indicating the variety of approaches that system designers have taken to address these issues. The choice of programming model varies from script-based agents, useful for quickly automating simple tasks, to object-oriented agents, which are better suited for more complex applications.

Of the mobile agent systems considered, Telescript is undoubtedly the best system for implementing mobile agents. It is one of the oldest systems developed, but certainly the most complete one. The Telescript system directly addresses each of the issues specified. It has a language which has been designed specifically for this purpose. The problem with Telescript is that it is proprietary software and a closed standard. Moreover, the fact that programmers have to learn a new language also influences the overall acceptance of the system.

The Java language is multi-purpose, but it has necessary capabilities for writing mobile agents. Java is inferior to Telescript in the areas of support for agent migration, communication between agents and interfacing access to host computer resources. In the other areas however Java at least equals Telescript. Java's advantage over Telescript is that it has an open specification. The breakdown of a technically impressive system like Telescript indicates that popular general-purpose languages like Java are more likely to succeed than special purpose ones like Telescript.

Java-based systems themselves (such as Mole, IBM Aglets, Odyssey, and Concordia) have different features developed depending on which aspects the designers have focused their research. Some of the systems provide good communication infrastructure or agent naming and finding services, another are focused on security problems, while the third are concerned with agent monitoring and control implementing an audit trail mechanism. It is therefore hard to say which of these systems is the most satisfying.

Agent Tcl is a high-level scripting language that has many of Telescript's capabilities regarding agent migration and communication. Agent Tcl and Java systems are not in direct competition, since they offer different capabilities.

The major difficulty preventing the widespread acceptance of the mobile agent paradigm is the security problems it raises. By our opinion, no current system solves these security problems satisfactorily, and thus mobile agent security remains an open research area.

So far, designers have paid little attention to the application level issues such as the ease of agent programming, control and management of agents, dynamic discovery of resources, etc. The literature on the use of basic templates is only just starting to appear. As larger and more complex systems of roving agents are deployed, programmers will need reliable control primitives for starting, stopping and issuing commands to agents. The agent system itself will have to incorporate robustness and fault tolerance mechanisms to allow such applications to operate over unreliable computer networks.

For the implementation of workflow management system, the mobile agent system Mole [7,13] has been chosen, instead of IBM Aglets [16]. The main reasons for choosing Mole are:

- Its rich features for agent cooperation (i.e., group communication),
- Its own well-developed agent-server that can take over some of the duties of our work-server.

3. Overview of the Current Architecture

Mole is based on concepts of agents and places. Every place consists of one engine and (possibly) several locations, thus allowing the existence of several locations at one physical network node.

3.1. Changes in the Original Architecture

The functionality of Mole place allowed us to delegate some duties of work-server to places. Duties of listening the designated port, waiting for incoming agents, their internalization, insertion into a list of agents on that host, and activation, as well as tasks of agent externalization (before switching off the computer) and internalization (after the computer is switched on), are delegated to Mole engine and locations.

The worker itself takes over the following duties that previously belonged to the work-server:

- Notifying the user about its arrival,
- Alerting the user if it is too long on the same node,
- Checking conditions for leaving the place.

These three new duties of the worker contribute to the higher degree of autonomous behavior of workers (high degree of worker autonomicity is intended to be one of the main strengths of our design [1].) On the other hand, methods for externalization and internalization are general (i.e., independent on the structure of the mobile worker) and therefore there is no need for them to be worker's methods. This functionality has been delegated to the work-server, i.e., to the Mole place, as explained above. This way a worker is also of smaller size than before.

Itinerary is represented by the more complex structure, to comply with more complex flow patterns that could be needed by workflow applications. Instead of the simple list of nodes that the worker should visit sequentially, it is now possible to define an itinerary, which includes AND and OR splits, used for enabling parallel works and alternative paths, as well as constructs that enable the creation of loops (block activities or reversible control connectors), etc. In the previous design proposal, complex workflows through the system were achieved by means of additional, specialized mobile agents. However, building complex paths into the itinerary itself, lessen the overall load of network with mobile agents and consequently improve the performance of the whole system.

To strengthen security, workers are not allowed to access the system resources including the screen. To present a user interface, worker should only create an instance of the window class. The window will be shown on the screen by the worker host, which interfaces between the user and the worker.

3.2. Current Architecture

The current architecture is essentially a two-part architecture consisting of work-agents (workers) and worker hosts. Duties of the work-server (present in previous architecture) are delegated to the agent-server of the underlying mobile agent system (in our case Mole), workers, and work-hosts.

Workers. As before, mobile agents are work-items that are passed to different users and autonomously take care of their current position and further itinerary. They are called workers and contain work identification, work deadlines, work owner, and possibly other important information.

Every worker must follow the itinerary, which is now *the directed graph* of triples of the following form: (node, condition, methods). A node represents the address of the current network location, where only methods enlisted in the list methods are available to the user on that node. Worker will transfer itself to the next node when and if the condition (a logical function) is met. It is the responsibility of the worker itself to check condition periodically or at any other appropriate moment.

Concrete workers are descendants of the abstract Worker and contain attributes that describe the work and methods that can be used to process the work (Fig. 1).

Workers behavior is almost entirely defined with its itinerary. Most of the abstract worker's methods are for setting and getting of worker's attributes. For illustration, we shortly introduce just some of other attributes.

- arrive (abstract) is called immediately after arrival at every node. Can be redefined in concrete workers.
- depart (abstract) is called immediately before departure from every node. Can be redefined in concrete workers.
- getHost returns the pointer to worker-host
- getHostInterface returns the pointer to worker-host's user interface.

Agent-server. Mobile agent system Mole, being the basis of our workflow system, takes care of all the agent-related activities of workers. For instance, Mole engines, residing on every node in the system, have the following tasks:

- Listen the designated port, waits for incoming agents, internalizes them, puts them into a list of agents on that host, and activates them.
- At the end (before switching off the computer), externalizes all agents that are currently under its supervision.
- At the beginning of its execution (after the computer is switched on), internalizes all saved agents.

Although these are characteristics of the particular mobile agent system, they are present in most of other mobile agent systems as well. If some of

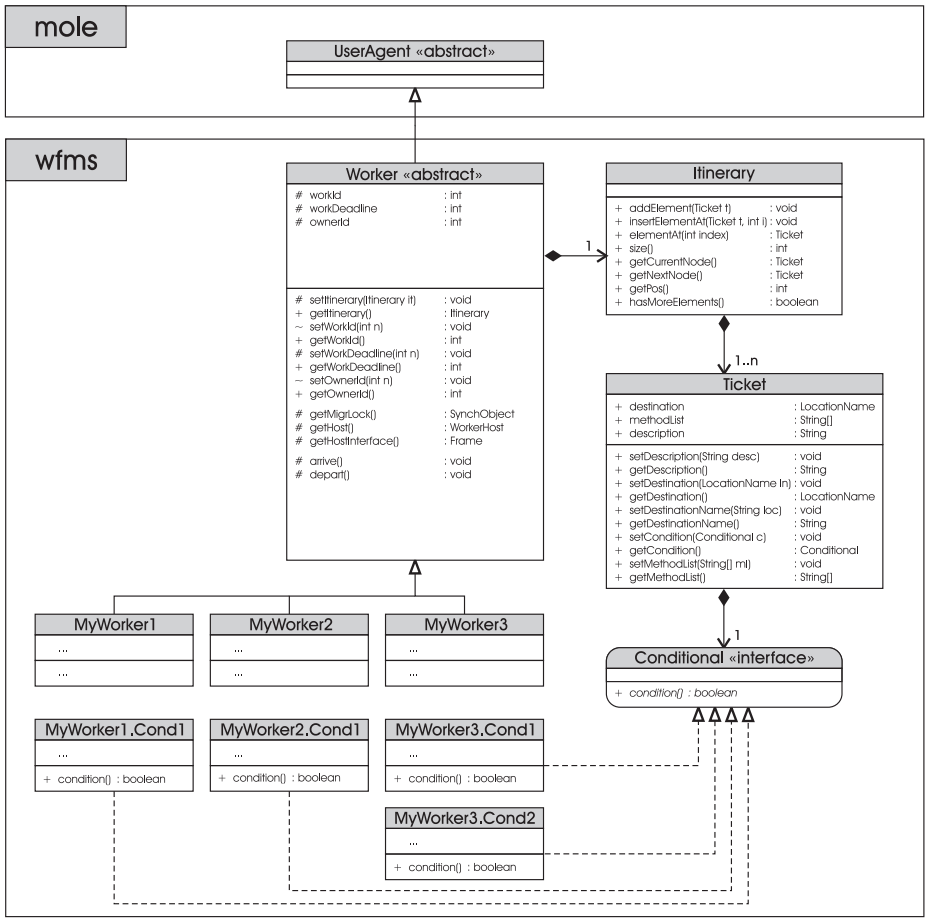

Figure 1: The class Worker

these characteristics are not present in some other mobile agents system used for workflow management system, then they should be implemented as separate stationary services.

Worker-host. Every node in the network contains a worker-host that is implemented as a stationary system agent, having special privileges for the access to host system resources. Worker host represents the interface component between the (operating) system, workers and users. Every worker host has a user interface, which enables the interaction with the user of the workflow system.

Every worker is automatically placed in the list of workers residing on the current node, and the corresponding icon appears on the user interface of the host. If the icon of the worker is selected, the list of all methods available at the current node is presented to the user, allowing him to invoke any of them.

During its visit on the node, worker itself will periodically check if the condition for the transition to the next node in its itinerary is fulfilled. When this condition is met, the worker reports its departure to the host, resulting in the corresponding icon being removed from the user interface. The worker is then transferred to the next node specified in its itinerary. When the itinerary is exhausted, worker's journey is complete, and the worker is removed from the system.

Worker-host has four methods that workers can invoke (Fig. 2). They are all used for bookkeeping of workers by worker-hosts:

- arrival will be called by workers when they arrive at the node. Host will first check it and then add the worker to its list of active workers.
- departure will be called by worker just before they leave the node.
- workCompletion is similar to the above, but called when the worker has finished its work (and it will be destroyed).
- migrationFailure will be called when worker cannot migrate to the next node in its itinerary.

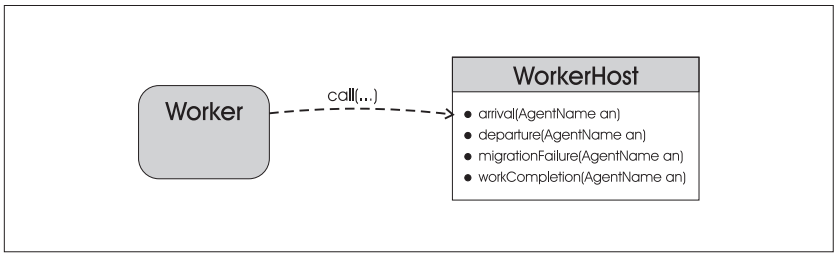

Figure 2: The worker invokes host methods

Workers and work-hosts are the only software components that are really necessary for the workflow system. However, if the system is to be user-friendlier and more flexible, additional tools and specialized agents need to be included.

4. An Example of a Stationary Service Agent

To increase efficiency of the whole system, workers should be of minimal possible size. Therefore, "standard" and additional features of the whole system must be implemented as separate stationary agents. These agents can lower the degree of worker autonomicity, because now workers partly depend on external services that may be not available all the time. To diminish the bad consequences of external functionalities, the additional services are widely spread over the whole workflow network. Of stationary services our system currently has services for e-mail, FTP, and database access. We shortly describe a service for electronic mail as an illustration of a stationary service.

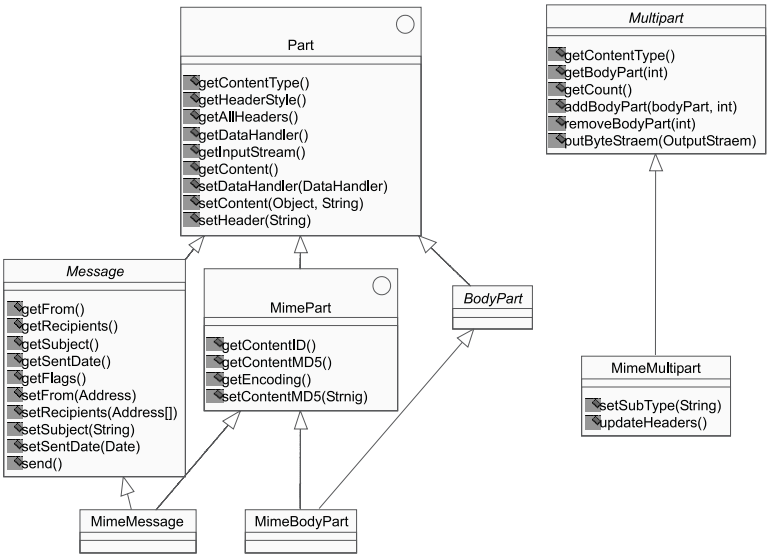

Figure 3: Major classes and interfaces of JavaMail API

The service is implemented as a combination of Mole system agent and JavaMail API [14, 15]. The latter one is pretty complicated (see Figure 3 for a reduced diagram of JavaMail API).

Message class is an abstract class that defines a set of attributes and content for a mail message. Attributes specify addressing information and define the structure of the content (DataHandler object), including the content type. The Part interface defines attributes that are required to define and format data content carried by the Message object. The Message class adds From, To, Subject, Reply-to and other attributes necessary for message routing. JavaMail API supports multipart Message objects, where each Bodypart defines its own set of attributes.

The stationary agent is implemented as the wrapper for JavaMail API. It inherits Mole system agent and implements only one method encapsulating functionality that is most needed by a worker. Nevertheless, our agent imports the

complete JavaMail API as its part. By inheriting Mole system agent, our service becomes able to easily communicate with workers (mobile agents).

In order to send mail, the worker only has to pass the following arguments to the agent method: type of message (text/plain, html, file), recipient address, subject, name of the file containing the message, and optionally name of the attachment file (Fig. 4). The method first sets the required system values, creates the session and then creates the message. The message is filled with content and sent. If something goes wrong, exception will be generated.

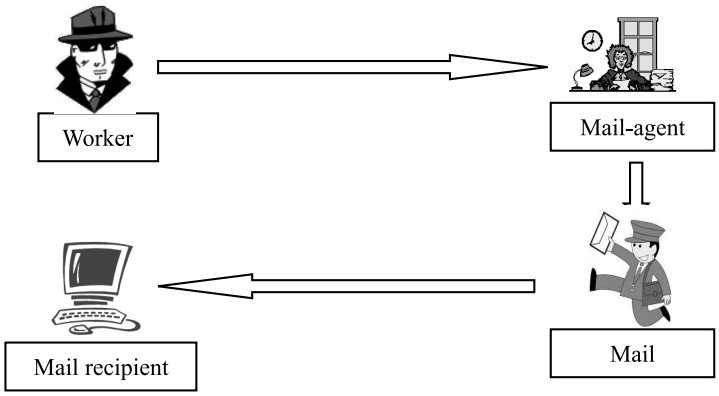

Figure 4: Worker sending a mail

4.1. Alternative solutions

Our solution has one obvious disadvantage – the network is populated with stationary agents offering services that might not be needed at all. Instead of making separate stationary agents for additional services, we could build these services into:

- Some mobile agents,
- Worker hosts,
- Java beans library that could reside on some nodes.

The first two alternatives are not appropriate because of the need to keep the sizes of workers and hosts minimal. The third alternative deserves additional analysis and investigation. However, until experimental results prove that this solution might be significantly more efficient, our solution is conceptually the best one – the system is uniform and consists solely of agents, without any additional facilities and tools.

5. Example Scenario

The application of the proposed system will be illustrated by the example of the worker used to assist in writing a joint paper. The first node in the worker's itinerary can be the node of the mentor, who is supposed to specify directions for paper writing and load the file containing the preliminary version of the paper. The worker can then move to the node of the other author presenting the directions and offering the method for paper extracting. When the other author have concluded the paper, he can analogously enter the reply message and load the new version of the paper. The worker can move back to the node of the mentor presenting the reply message and offering the possibility of paper extracting (Fig. 5).

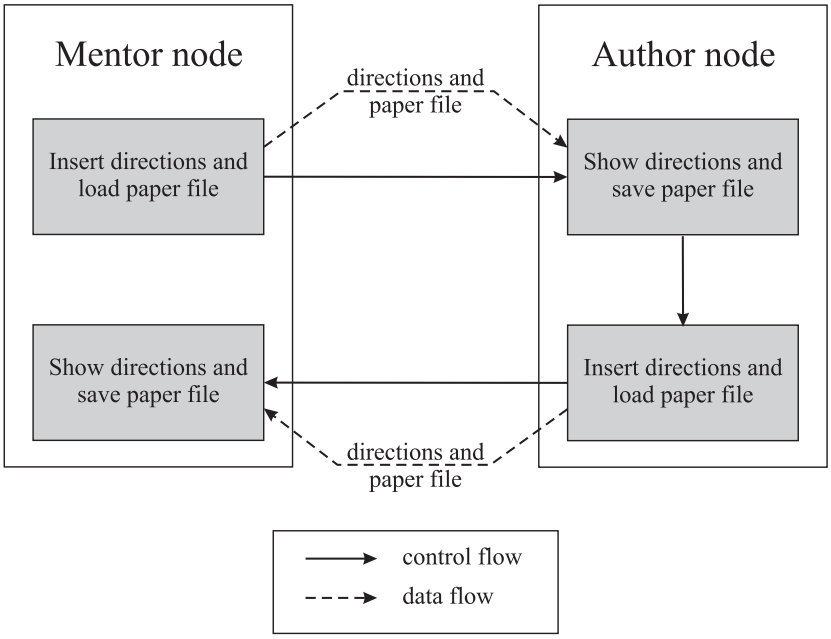

Figure 5: Current architecture of 'joint-paper' worker

After the worker is activated and arrives at the first ('mentor's) node, the user will be notified about the arrival of the new worker. He/she will call the available method for that node and will be presented by the following user interface (Fig. 6).

After the worker concludes that all conditions for that node are fulfilled (i.e., the message has been entered and the document has been attached), it will transfer itself to the next node in its itinerary. After the user on that node calls one of the two methods of the worker, he/she will be presented with the user-interface (Fig. 7).

When the user is ready to submit the second version of the paper, he/she

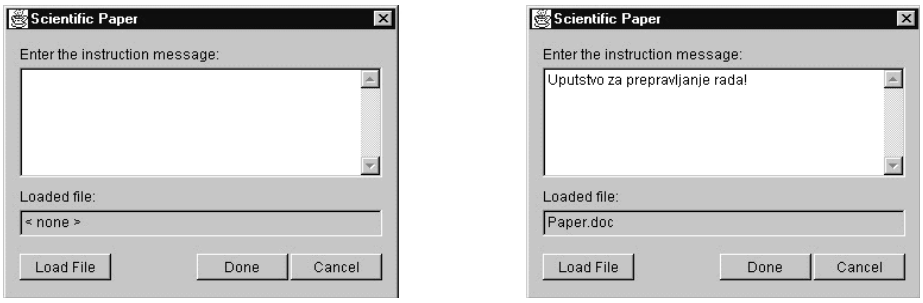

Figure 6: The window for entering writing directions and attaching the first paper version

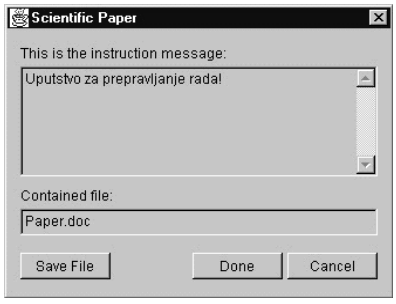

Figure 7: The window that shows the writing directions and offers the document for extraction

will call the second worker's method to do so. Worker will continue its travel according to its itinerary.

In the advanced version of this worker, the first node in the worker's itinerary can be the node of the mentor, which is supposed to specify directions for paper writing. The worker can then move to nodes of other authors presenting the directions and potentially offering a method for paper editing. The worker can move over the nodes of authors sequentially, or it can visit these nodes in parallel (Fig. 8 and 9). Moreover, this part of itinerary can also be set as iterative, i.e. the worker can repetitively circulate between mentor and other author nodes (carrying the current version of the paper) until the final version of the paper is created. Finally, when the paper is completed, the worker can send an e-mail to the conference organizer or reviewers of the paper using the service of the stationary mail agent.

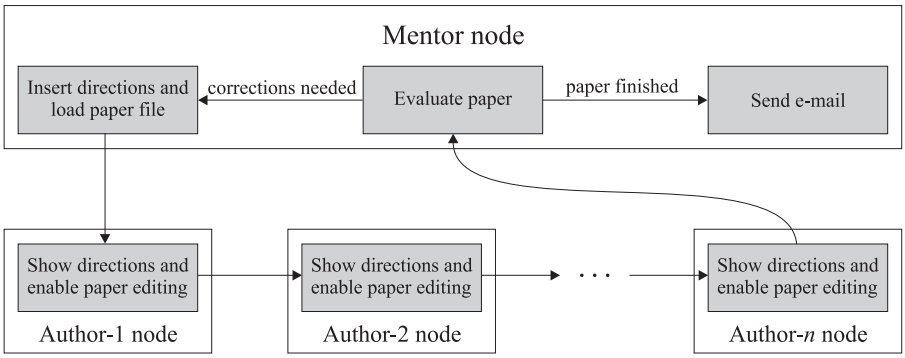

Figure 8: Sequential routing

The presented scenario has been implemented and tested within the department and it proved how naturally various workflow control patterns can be realized in our system by making use of social abilities of agents, which are the key feature used for execution of activities in parallel, as well as for their coordination and synchronization. In order to achieve the flow of work, agents split the work in logical parts, cooperate together and synchronize themselves.

6. Interoperability

To facilitate cooperation between different workflow management systems, the Workflow Management Coalition has defined several standards for workflow terminology, interoperability and connectivity between workflow products. The workflow reference model [20] describes a generic architecture for workflow management systems, distinguishing the functional components of a workflow system and recognizing the major interfaces between them. There are 5 interfaces identified (as shown on Fig. 10.), each of which is a potential point of integration between the workflow enactment service and other infrastructure or application components.

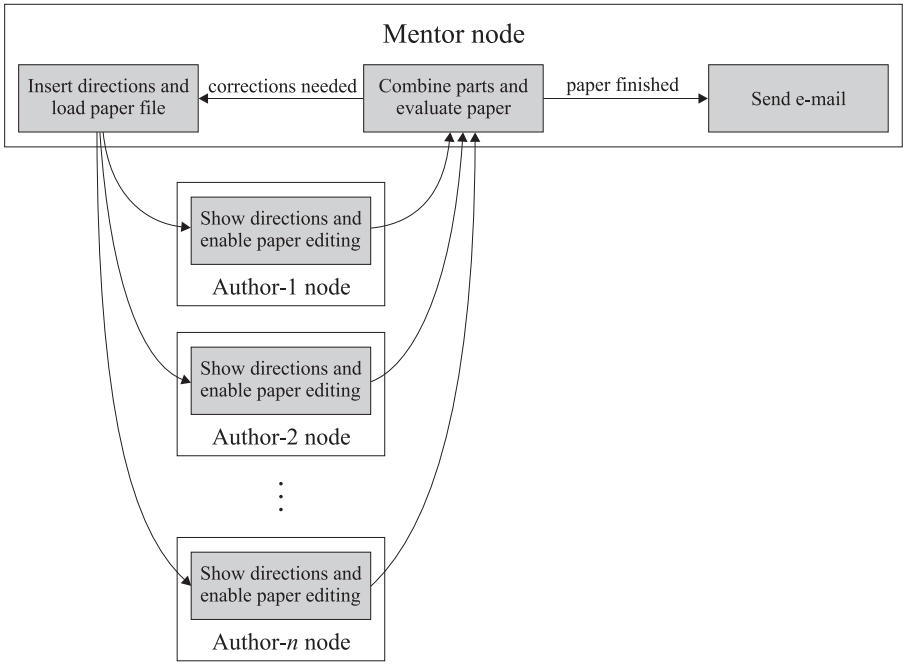

Figure 9: Parallel routing

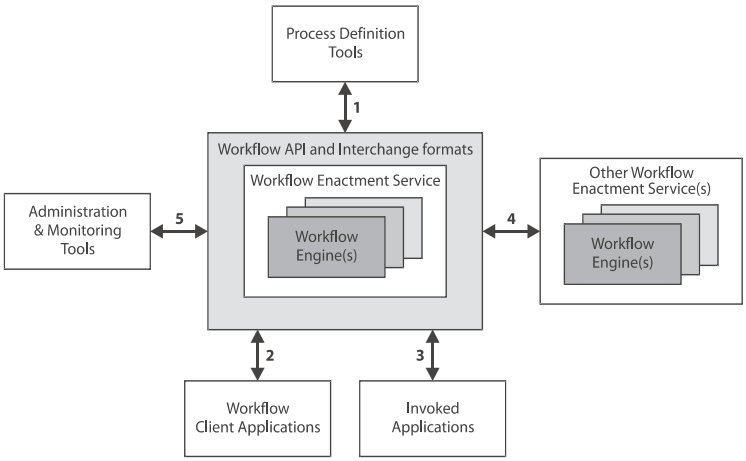

Figure 10: Workflow Reference Model (bidirectional arrows denote interfaces)

A mobile agent system is used for implementation of: workflow enactment services, workflow administration and monitoring tools, and workflow client applications. Moreover, agents are used as a uniform interface to invoked applications and other workflow enactment services.

The ongoing research is directed towards Interface 1. The specification of this interface includes the basic workflow meta-model, which describes the common set of entities contained within a process definition, their relationships and attributes. Furthermore, XML Process Definition Language is proposed allowing for the definition and exchange of process definitions using the entities and attributes defined within the meta-model.

The conformance to the proposed meta-model is achieved by adapting the structure of individual components of our system, namely workers and worker hosts, to comply with the meta-model specification. Moreover, an import/export layer is provided to support the mapping of XPDL definition to/from the internal worker representation.

7. Conclusion

During implementation of the original design, we adopted two-part architecture, instead of the original three-part architecture. What earlier were the duties of work-server, now are duties of work-host, worker itself and agent server of underlying mobile agent system. Agent-server has to exist at every node where the mobile agent will possibly reside and its functionality is directly used by the workflow management system. Other duties of work-server are delegated to workers, in order to increase their autonomous behavior. Apart from duties that are strictly related to concrete work, workers now also notify hosts and users about their arrival, departure, and about possible breaking of deadlines. To increase the safety of the whole system, workers are forbidden to directly access any system resources including the screen. Complex paths through the network are now built into the worker's itinerary.

Although workers are now almost fully autonomous they may need additional services to finish their work. Those services cannot be implemented directly into the workers, to keep them as small as possible. Therefore, those services are implemented separately, as stationary agents. By placing those services at a sufficient number of nodes in the network, the autonomy of workers is not seriously diminished.

The experience of using the system is positive. As planned, it is robust, highly distributed, and useful. The work on the system continues with implementation of more specialized agents and utilities for user-friendly administration of agents.

References

[1] Budimac, Z., Ivanović, M., Popović, A., Workflow Management System Using Mobile Agents. In: Proc. of ADBIS '99, Lecture Notes in Computer Science 1691, Springer Verlag, Berlin, (Maribor, Slovenia) (1999), 169 - 178.

[2] Debenham, J., Constructing an intelligent multi-agent workflow system. Proc. of AI'98, Brisbane, Australia (1998) 119-126,

[3] General Magic: Introduction to Odyssey API, Homepage of General Magic.

[4] Gray, R. S., Agent Tcl: A Flexible and Secure Mobile Agent System. Ph.D. thesis, Dartmouth College, Hanover, NH, USA, 1997.

[5] Ibrajter N., Agent Services in a Network Environment, Master Thesis, University of Novi Sad, Novi Sad, 2004, pp. 141.

[6] Meng, J., Helal, S., Su, S., An ad-hoc workflow system architecture based on mobile agents and rule-based reasoning. Proc. of Int. Conf. on parallel, and distributed computing techniques and applications, Las Vegas, 2000.

[7] Pešović, D., The Analisys and the Application of a Mobile Agent System. Diploma Thesis, Faculty of Science, University of Novi Sad, Novi Sad, 1999, pp. 74.

[8] Pešović D., The Implementation of a Workflow Management System Using Mobile Agents. Master Thesis, University of Novi Sad, Novi Sad, 2002, pp. 162.

[9] Pešović D., Joint-Paper Worker. Proceedings of XVI Conference on Applied Mathematics "Prim 2002", Zlatibor, Yugoslavia, May 29 – June 2, 2002., 143-152.

[10] Pešović, D., Budimac, Z., An advanced joint-paper worker, CCS journal, 4th issue (2003), 44-46.

[11] Pešović, D., Budimac, Z., A Comparative Analysis of Several Mobile Agent Systems. Novi Sad J. Math. Vol. 30 No. 2 (2000), 95-111.

[12] Stormer, H., A flexible agent-based workflow system. Workshop on ,,Agent-based approaches to B2B" 2001.

[13] Strasser, M., Baumann, J., Hohl, F., Mole – a Java Based Mobile Agent System. Home-page of Stuttgart University, Stuttgart, Germany, 1996.

[14] Sun Microsystems Inc.: JavaMail API Design Specification, http://java.sun.com/products/javamail, 1998.

[15] Sun Microsystems Inc.: JavaMail Guide for Service Providers, http://java.sun.com/beans/glagow/jaf.html, 1998.

[16] Venners, B., Under the Hood: The Architecture of Aglets. Java World, 1997., www.javaworld.com/javaworld/jw-04-1997/jw-04-**hood**.html.

[17] Versteeg, S., Languages for Mobile Agents, http://www.cs.mu.oz.au/∼scv/thesis.html, 1997

[18] Waardenburg M., van Emmerik M., Workflow Management in an Internet Environment, http://www.axxerion.com, 2005.

[19] Workflow Management Coalition, Workflow Process Definition Interface – XML Process Definition Language, Homepage of Workflow Management Coalition, 2002.

[20] Workflow Management Coalition, Workflow Reference Model, Homepage of Workflow Management Coalition, 2002.

[21] Wheeler, T., Voyager Architecture Best Practices, Homepage of Recursion Software, 2005.

Received by the editors June 6, 2006