A PROTOCOL TO DEVELOP AGENT-BASED FORM FLOW SYSTEMS

PAUL JUELL AND
MD. AHSAN HABIB

COMPUTER SCIENCE DEPARTMENT

NORTH DAKOTA STATE UNIVERSITY, FARGO, ND 58102

PAUL.JUELL@NDSU.NODAK.EDU
MD.HABIB@NDSU.NODAK.EDU

ABSTRACT

We present a protocol to develop a computerized form flow system from an existing paper form flow system. Form flow systems involve the flow of information and control in a process. Our protocol maps the processing into an agent-based system. The agent-based methodology presents a rich model for form flow systems. The design methodology allows the designer to easily transform the requirements of a form flow system into an agent-based automated from flow system. The protocol is simple, flexible, and generic in nature to adapt to different kinds of form flow systems. We show the steps of the protocol for an example of a course drop/add form. The prototype we built for the example is a multi-agent system designed to run on multiple processors.

INTRODUCTION

This paper presents a protocol to develop a form flow or work flow system using agent based technology. The protocol converts the requirements of a document flow into an agent-based document flow system. While agent applications are becoming increasingly popular, there have not been many proposals for agent-oriented methodologies for analysis, design, and software development. We will follow object-oriented analysis and design methodology to reach our goal. Our approach is to identify the agents' role so that they can be arranged in a class hierarchy. Responsibilities are then assigned to each role, along with the services required to meet those responsibilities.

We have the following steps in the proposed design methodology for an agent-based document flow system:

1. Identify tasks or processes of the document flow system.
2. Identify entities in the document flow system.
3. Identify agents along with their tasks for the system.
4. Identify interaction among agents.

5. Build agents and implement the system.

Step one is basically the requirement specification of the document flow. We will identify the tasks involved in a business system in this step. In step two, entities will be identified. The tasks identified in step one are performed by these entities.

Once entities are identified, we will go to step three where we will recognize the agents required in the agent-based document flow system. Interaction among agents is handled via some messages. In step four, we will define the message format which will be handled by the agents in the system.

Finally, it is time to build the agents. To build agents, we need an agent platform. In our proposed system, we will not build our own agent platform. We will use an existing one to build the agents. Once agents are built, we will implement the whole system.

Identify tasks or processes of the document flow

In this step, we will basically define the business logic of the document flow system. The function of business process modeling is to render a business process from the real world into a formal definition which can easily be computer interpretable. The procedure to model the business process is a separate methodology itself. The modeling can be done by the use of analysis, modeling, and system definition techniques. The resulting definition is called a process model or process definition. This type analysis was proposed in the seventies [6] and has been refined since then [1] [2] [8].

In this study, we will not focus on a specific process model to define the business component of a document flow system. It is rather very open to the reader, but in our case, we will use the object-oriented analysis to recognize a set of tasks and business rules of a document flow system. While there are a range of tools to build document flow systems [1] [2] [14] [15] they typically do not come with a design protocol or deign techniques to develop the systems. We have found some work on developing these systems [7] [8] [9], and our work will follow the general patterns they propose. We set out a simple mapping protocol. We will continue to the next step in the methodology from there.

This step is very tied with a specific document flow system. We can recognize the processes and/or rules in the business system by first looking at the requirements of the document flow system to be automated. Answers to the following questions would give a good understanding of the process rules:

1. What is the content of the document being automated?
2. What part of the form will be processed at each level?
3. What are the rules associated with each processing?
4. What are the rules to flow the document to the next level?
5. How will the changes to the document be updated in the storage?

Along with the answers to these questions, we can look at the use case diagram of the requirements of the specific document flow system. This will help us to recognize tasks associated with each processor in the system.

The characteristics of the document flow system are a set of features that can be expected of document flow systems. It is very important to resolve issues of the state of the document. Each state of the document is stored in the database using a unique id (token). The token is passed around with the messages to the agents. The combination of the location of the token and the database entry give the state of the document. We will discuss more about the unique id later in the step four.

Identify entities in the document flow system

Once the tasks are defined, identifying entities is simple. The entity in the system is the user or the device that performs some tasks. Each entity plays a different role in the system. We can make out the entities by asking questions with "who." For example, "who does initiate the document?" would give us the answer "initiator." This initiator is one entity or actor of the system. Actor and entity are synonymously used throughout this paper.

If we look at how the system will be used, we can also find the actors in the system. Use cases are utilized to provide a description of how the system will be used. The use case diagram will be utilized to represent use cases.

From the characteristics the initiator, intermediate processors, and curator can be identified as the actors of the system. Database can also be treated as an actor of the system. The behavior of these actors is presented using a use case diagram. The generic use case diagram presents some high-level requirements of a document flow system. We obtained these requirements from the analysis of the two experimental document flow systems.

Irrespective of the kind of document flow system, the actors and their behavior presented in the generic use case diagram will be similar. For example, the initiator will always start up the flow of the document which will be processed by the intermediate processors. Similarly, all of the changes to the document will be updated into the database.

The designer can use this generic use case diagram and map the actors in the system that will be automated. In the specific system, use cases can be added or removed as needed. The one we showed here is a very simple diagram of a generic system. It can easily be altered according to the requirements of the specific system being automated.

Identify agents along with their tasks in the system

So far, we have talked about the document flow system itself. We have established the tasks or processes involved in the business level of the document flow system. We have also found the actors or entities along with their tasks in the document flow system in the previous steps.

Now we are at the execution phase of the methodology. Once we have the actors in the document flow system, we can map these actors in the system architecture of the agent-based document flow system. Each actor will eventually be an agent, and each agent will perform some specific tasks coupled with it. Tasks found in the first step will be performed by these agents, respectively.

Table 1.1 Generic Agents of a Document Flow System

Actors	*Agents*
Initiator	Initiator Agent
Processors	Processors Agent
Curator	Curator Agent
Database	Database Agent

Identify interaction among agents

Our agents need to interact with each other. They can do this in a variety of ways. They can talk to each other directly, provided they speak the same language, or they can talk through an interpreter or facilitator, provided they know how to talk to the interpreter and the interpreter can talk to the other agent.

There are two levels of language: basic and deeper level. While in the basic level, the syntax and the messages are involved; in the deeper level, the meaning or semantics are involved. When agents talk to each other, they need to understand the message format as well as the meaning of the message. Therefore, they need to have shared vocabulary of words and their meanings. This shared vocabulary is called ontology.

Our goal in this step is not to develop an Agent Communication Language (ACL); rather, we will explore an existing one which we will use in our system. And then we will discuss how the messages can be formatted to interact with other agents.

We will use the most widely used Agent Communication Language, KQML (Knowledge Query and Manipulation Language). KQML [17] provides a framework for programs and agents to exchange information and knowledge. It focuses mainly on the message formats

and the message-handling protocols between agents. It defines the operations that agents may attempt on each others' knowledge base.

KQML messages are called performatives. There are a large number of performatives defined in KQML specifications. Most agent-based systems support only a small subset of it. Using performatives, agents can ask other agents for information, tell other agents facts, subscribe to the services of agents, and offer their own services.

KQML messages encode information in three different architectural levels: content, message, and communication. An example of a KQML message from agent "john" telling "hello" to agent "jason" might be encoded as follows:

```
(tell
        : sender john
        : receiver jason
        : content (hello)
        : language English
        : ontology Standard
    )
```

The KQML performative is tell. The :content parameter completely defines the content level. The :sender and :receiver parameters specify information at the communication level. The performative name, the :language specification, and the :ontology name are part of the message level.

There are many implementations of the KQML framework. A Java implementation of a subset of the specification was developed at Stanford University [16]. The implementation is one of the layers in JATLite (Java Agent Template Lite).

In our proposed agent-based document flow system, the document data are stored in the database. At each level of the flow, the document is updated by the agent at that level. The changes in the document are updated in the database as well. The document is stored with a unique identifier. The agent sends a message with this identifier to the next level agent and the database agent. The receiver agent gets the document from the database using the unique id sent with the message.

Choosing a unique identifier is a tricky part of the system. It gets more difficult for a complicated system. For a simple document flow system, the initiator agent id would be sufficient, but it is up to the designer what the document identifier will be. In our case, we make a combined unique identifier using the initiator agent id plus the time stamp.

Build agents and implement the system

So far, we have analyzed the system and gone through the steps to identify artifacts in the document flow system. In step one, we identified tasks that will be performed by the agents. In step two, we recognized the entities in the system. Entities found in step two

become agents that we mapped in step three. We identified the interaction between agents and the message formats to communicate among them in step four.

Before going to build the agents, we will look at the database we need. We will have a central database to store the document information. The document will be identified by the unique key we have defined in step four.

Now is the time to build the agents. We need an agent platform to build and run agents. We have chosen JATLite as the agent platform.

JATLite is a set of Java packages that facilitates the agent framework development using the Java language. It provides basic communication tools and templates based upon TCP/IP. It especially facilitates the development of agents that exchange KQML messages. It also provides a special Agent Message Router (AMR) functionality. The AMR allows any registered agent to send messages to any other registered agent by making a single socket connection to the AMR; messages are forwarded without the sending agent having to know the receiving agents' physical address. The AMR buffers all messages, like an email server, so that messages are not lost due to network transient problems. This also allows the individual agent to go down or logout and return for its messages at a future time.

EXAMPLE OF THE METHODOLOGY

In this section, we will present an example of the design methodology that we presented in the last secton. One system we considered is the Course drop/add form flow system .

Example: Course drop/add form flow system

Requirement

The student will submit details of the course to be added or dropped by accessing the form via the internet. The student must be registered and logged in the system prior to the submission. The details of the student information will be stored in the database. The details of the course to be added or dropped will also be stored in the database. The message will be sent to the adviser about the submission of the form. The adviser will also be a registered user of the system. The adviser will access the document and accept or reject the request made. Depending on the action taken by the adviser, the message will be sent to the registrar's office or back to the student. If the adviser accepts the request, the registrar will get the message to make the request final.

From the use case diagram, we get the following tasks:

1. Submit or start the document,
2. Register/login,
3. Send message,

4. Process the document,
5. Store document as well as the user information into the database, and
6. Finalize the request.

Step 2: Identify entities

We recognized tasks in the first step. Now we will look at the entities in the drop/add document flow system. Entities in the system are nothing but the actors in the use case diagram. If we look at the use case diagram, we get the following actors (entities) in the system:

1. Student,
2. Adviser,
3. Registrar, and
4. Database.

Step 3: Identify agents and their tasks

Communication between agents is made using KQML. The main focus in this step is to determine the information wrapped in the message content to retrieve document data from the database. The document itself does not flow from agent to agent; rather, the document is stored in a central database. The unique document identifier is sent in the message from agent to agent. Agents retrieve the document from the database using this unique id. Thus, it is a very crucial part of the design to define the unique id. As we discussed in the methodology, the designer is free to choose his/her own unique id. We also give some guidelines to choosing the ID, later.

In the drop/add agent-based document flow system, we chose a combined key for the unique identifier, and it is the student agent id (NDSU id) plus the timestamp of the document submission.

RESULTS

The protocol for designing agent-based document flow systems was successfully implemented. We presented a design methodology which is very simple and flexible in nature. We walked through the steps in the methodology to produce working versions of two experimental systems.

One of the experiments was the drop/add course form flow system that is used for NDSU students to drop/add courses. We successfully mapped the generic items in the design methodology with the system-specific items. We made a working version of the system walking through steps in the methodology.

The other example we considered was the flow system of the graduate student application form to participate in commencement. This system is used by the NDSU graduate

students who want to participate in commencement. We successfully plotted the generic items in the design methodology with the commencement flow system.

Thus, we claim that our protocol is simple and easily adaptable. The protocol worked for a specific domain of the document flow system. We tested it with a single-form-based document flow system, and it worked very well. We want to extend our view that it will work for more complicated office systems.

Though the method we presented may suffice for the experimental systems, it may not be adequate for a critical document flow or workflow systems. The generic items in the architecture presented should be mapped properly with the specific system to get the best result.

Another limitation of the system is to define the proper unique identifier to map the agents with the database. We recommended using a combined key of agent id plus timestamp to the database for this purpose. The agent platform, JATLite, which we used, has a limitation of producing an erroneous timestamp. The designer should address this problem when choosing a unique identifier. For example, instead of using timestamp, the designer can choose a suitable hashing algorithm specific to the needs.

CONCLUSION AND FUTURE WORKS

In this study, we introduced and developed a simple and flexible architecture for designing an agent-based document flow system. It addresses the issues pertaining to agent-based document flow or workflow systems in a generic manner while allowing system-specific alterations. It can be easily adapted by many offices. It also represents a low cost in terms of design and development time.

We worked on a single-form-based document flow system. This kind of flow system has an initiator and a curator of the document. If the problem to be solved is similar to the examples we worked, then the document flow designer should be able to quickly convert the problem into a working version. We presented generic flow architecture as well as the methodology. A designer will just replace those generic items in the architecture with the system-specific items.

Although the protocol will work well for a single-form-based office system, there is room to further enhance the protocol for multi-form-based complicated office systems. There is room for further enhancement of the system in the area of intelligibility of the agent. Human interactions can be reduced by making the agents more intelligible.

Improvement can also be made in the area of modularity of the system by separating tasks/activities from the agents. One way of doing that is by employing a separate task/activity server, and mapping the tasks with the corresponding agents. The system can also be made more interoperable by engaging a universal data format, XML (Extended Markup Language).

BIBLIOGRAPHY

[1] Ting Cai, Peter A. Gloor, and Saurab Nog. "Dartflow: A Workflow Management System on the Web Using Transportable Agents." Technical Report TR96-283, Dept. of Computer Science, Dartmouth College, 1996.

[2] Workflow Management Coalition. "Introduction to the Workflow Management Coalition." http://www.wfmc.org/about.htm, September 2003.

[3] Paul A. Buhler and Jose M. Vidal. "Semantic Web Services as Agent Behaviors." In Agentcities: Challenges in Open Agent Environments, Springer-Verlag, 2003, pp. 25-31.

[4] H.S. Nwana. "Software Agents: An Overview." The Knowledge Engineering Review, 11(3), 1996, pp. 205-244.

[5] AgentBuilder. "When should I use Agents." http://www.agentbuilder.com, September 2003.

[6] Clarence A. Ellis and Gary J. Nutt. "Office Information Systems and Computer Science." Computing Surveys, 12(1), March 1980, pp. 27-60.

[7] V. Neelakantpillai. "An Object-oriented Protocol for Rapid Prototyping of Document Flow Systems." M.S. Thesis, North Dakota State University, Fargo, 2000.

[8] P. Mambrey and M. Robinson. "Understanding the Role of Documents in a Hierarchical Flow of Work." In Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, AZ, 1997, pp. 119-127.

[9] R. Paturu. "A Flexible Protocol to Design Document Flow Systems for Rapid Prototyping." M.S. Thesis, North Dakota State University, Fargo, 1998.

[10] Paul Buhler and José M. Vidal. "Towards Adaptive Workflow Enactment Using Multiagent Systems." Information Technology and Management Journal, 2003, to appear.

[11] Z. Maamar, N. Troudi, and P. Rostal. "Software Agents for Workflow Support." The Journal of Conceptual Modeling, 12(1), February 2000.

[12] J. Meng, S. Helal, and S. Su. “An Ad-Hoc Workflow System Architecture Based on Mobile Agents and Rule-Based Processing.” In Proceedings of the International Conference on Parallel and Distributed Computing Techniques and Applications, Las Vegas, NV, June 2000.

[13] A. Dogac, et al. “A Workflow System through Cooperating Agents for Control and Document Flow over Internet.” In Proceedings of the 7th International Conference on Cooperative Information Systems, Eilat, Israel, September 6-8, 2000.

[14] IBM. “WebSphere MQ Workflow.” http://www-3.ibm.com/software/integration/wmqwf, September 2003.

[15] Plexus Software. “FloWare.” http://www.plx.com/products/floware/floware.html, September 2003.

[16] H. Jeon, et al. “JATLite: A Java Agent Infrastructure with Message Routing.” IEEE Internet Computing, 4(2), 2000, pp. 87-96.

[17] T. Finin, et al. “KQML as an Agent Communication Language.” In Proceedings of the 3rd International Conference on Information and Knowledge Management, ACM Press, Gaithersburg, MD, November 1994.