Using Transactional Workflow Ontology in Agent Cooperation

Jarmo Korhonen, Lasse Pajunen, and Juha Puustjärvi
SoberIT
{jarmo.korhonen, lasse.pajunen, juha.puustjarvi}@hut.fi

Abstract

In this paper, we introduce a transactional workflow ontology that can be used to extend cooperative agent systems. Multi-agent systems generally use messaging protocols to implement the system, but the protocols are hardwired in the agents. We identified this inflexibility problem when we implemented a prototype agent system BriefsAgents that used DAML+OIL-based ontology for data transmission between agents. Ontology-based workflow description enables the agents to use a workflow engine to run their internal state, instead of hard-wiring the state model in the implementation. In addition, agents can communicate their workflow to other agents, and implement reliable transaction control. The proposed workflow ontology uses similar principles for transactions as our WorkMan system, but different transaction models can be used.

1. Introduction

MULTI-AGENT systems (MAS) are composed of collaborative agents in a shared platform that together implement more complex functionality. Interaction between agents is similar to object oriented methodology, but the agents have a greater degree of autonomy. A software agent in MAS can for example initiate a communication with other agent without explicit outside command to do so. Communication between agents is formalized in a set of protocols, which the agents must implement to be able to cooperate. The problem with this approach is that system architecture is not very flexible.

We implemented a prototype multi-agent system BriefsAgents that used DAML+OIL ontology to communicate data between agents. Basically, the system has two types of agents: query agents and data agents. The database was generated from information gathered in BRIEFS project [1]; each data agent managed one concept in the BRIEFS ontology. The query agents used one or more data agents to make queries. We used FIPA-OS, a reference implementation of the FIPA standard as the agent platform, and used Java language to actually implement the agent-based system.

The protocols we used in BriefsAgents were straightforward; there were no negotiations and all protocols were between only two agents at a time. We noticed that changing the protocols was difficult, as the agents became increasingly dependent on each other. This lack of flexibility can be corrected by using a workflow description to run the agents. In addition, ontology enables us to communicate this workflow between agents.

BriefsAgents did few changes to the data. More complex systems require better support for reliable transactions. We have also implemented a prototype system called WorkMan [9] that combines workflows and transactions using standard database constraints. We are researching how this approach can be combined with multi-agent systems like in the BriefsAgents prototype. We also research how workflow models presented for web services can be adopted by MAS agents. Current workflow description languages and MAS lack support for transactions.

In this paper, we propose a transactional workflow ontology that can be used in MAS. Ontology here is a taxonomy of concepts and a set of relations between concepts. The rest of the paper is organized as follows. In Section 2, we summarize related work on similar fields. In Section 3, we present a framework for MAS using workflow ontology. In Section 4, we describe the basic structure of the workflow ontology. Finally, in Section 5, we use the ontology to describe an example workflow with transactions.

2. Related work

2.1. Multi-Agent Systems

Several independent industrial and research groups have started standardization of multi-agent technology. There has been effort by OMG for making agent-based systems using CORBA IDL definitions [3] and KAoS group from agent-oriented system perspective. Recently, MAS standardization has taken place in Foundation for Intelligent Physical Agents (FIPA) [2]. FIPA standard describes agent communication mechanisms and an agent platform structure. This paper uses standard MAS architecture and web services without any changes.

There has also been a European project on using workflow systems and MAS together [10]. Several agent-based workflow management systems have been presented, e.g. in [16]. This paper differs in that the ontology is intended for use with web service-based systems.
2.2. Workflow

Workflows have been researched for a long time, especially to automate business processes. There has also been a standardization effort by Workflow Management Coalition (WfMC)[4].

DAML-S [5] is a transactional workflow ontology that is related to semantic web services. It is intended for use with web services; there are also other web service workflow specifications, e.g. WSFL [6] from IBM. Web service workflow specifications are relevant because web services and semantic web provide service infrastructure for software agents. We present in this paper a way to add advanced transaction models to the workflow models presented in [5] and [6]. The ontology in section 4 itself is similar.

2.3. Transactions

Distributed transactions have been combined with workflows in [7]. The workflow ontology is open to different advanced transaction models (ATMs) and it supports for example saga [8], option [9], and no transaction models.

3. Multi-agent framework with workflow ontology

Workflow ontology can be used in MAS for two purposes: to drive the agent internal states and to direct different protocols between agents. As in WSFL, workflow ontology can also be used to aggregate existing agent services to form more complex agents.

Agents have a set of internal states. Implementation defines agent’s internal states. Workflow ontology needs to use this structure; therefore, agent implementation needs a workflow engine to run its state machine. In Fig. 1, we present an extended agent framework where a Workflow Management Agent (WMA) facilitates agents communicating their workflow to other agents. This also extends the BriefsAgents’ agents that only had the internal state engine.

![Agent Framework Diagram]

Fig. 1. Workflow in agent framework

WMA has purposes similar to Directory Facilitator (DF) agent in FIPA standard [2] platform or UDDI [14] registry in web services. Agents register their workflow in WMA and are able to query workflow specification for other agents. This enables agents for example to restrict their cooperation to agents that conform to the same transaction model.

An agent has internal structure and internal state model. In this paper, we suppose a structure that conforms to FIPA specifications. We extend that structure with standard implementations of a workflow engine and web service interface. Both of these use a set of parameters to direct their operation; the actual implementation should not vary from agent to agent. A state engine is the part of agent that currently handles both internal states and external communications. It contains for example the method implementations of the actual services the agent provides.

Workflow engine has five tasks:
- register the agents workflow on WMA
- read and execute the workflow
- combine the workflows of other agents; e.g. if there is a two-phase operation, both must be present in the overall workflow
- handle communication between agents and pass the messages to the state engine where appropriate
- call web services where directed by the workflow

Web service interface is a separate part of agent, because it needs to integrate with web services that are separate from agent platform. There are different models to integrate with web services; here we concentrate on IBM web services stack and DAML-S semantic web services model. The web service interface has three tasks
- find web services requested by the workflow engine
- integrate with web service workflow
- call web services when the workflow engine requests it

The workflow engine handles transactions. The agent uses a single transaction mode at a time, even though it may support different transaction models when using other services or agents. Workflow consists of one transaction that must succeed or fail entirely. The operations taken in case of success or failure depend on the transaction model.

Transaction requirements from different agents are combined by the workflow engine. The combination requires that three things are known for each agent: what messages it needs to get the work done and what to do in case of success and failure. All are not required; for example in option model [9] failure can be handled by not calling the option.

Integrating web service workflows is handled by the web service interface. There is no transaction support in either DAML-S or WSFL. It is currently not possible to find transactional operations automatically, without explicit transaction support. With our transactional workflow ontology, web services can describe their transactional properties; this is an extension to WSFL.

4. Transactional Workflow ontology

In this section, we present a simple ontology to describe the
Workflow. The intent is to show how the transactions can be added to other ontology (e.g. DAML-S).

Workflow consists of tasks and links between tasks. Links may split workflow into different optional or concurrent paths and later join them, meaning that not all tasks are necessarily successful or done at all. A workflow is a single transaction, which will either succeed or fail entirely. If transaction fails, all tasks must be aborted. How this is achieved depends on the used transaction model.

Workflow engine combines agent states, communication between agents, and advanced transaction models to one master workflow, which is a sum of all tasks and transactions. This is similar to WorkMan, which uses database constraints to achieve the overall consistency for the workflow. Master workflow is a hierarchical model in two ways: separate workflows can be combined in a master workflow as a two-level hierarchy, or a task may itself have a workflow as a recursive hierarchy. For example, one task can call a web service that uses WSFL description to combine several lower-level services.

Agent communication can be modeled in AUML [11]. AUML has explicit presentation of possible agent internal states. Workflow ontology models only states of protocols, because the focus is on transactions. Implementation details are handled at the state engine, which gets instructions from the workflow engine.

4.1. Concepts

Workflow ontology has two levels; semantic level and implementation level. The ontology has three semantic level concepts: task, control link, and data link. Fig. 2 presents the basic structure of the workflow ontology. This ontology is intended to work with web service workflow description languages, e.g. WSFL and DAML-S, and uses similar concepts where applicable.

The task types are send message, receive message, request-response, solicit-response, start transaction, end transaction, and notify error. First four of the task types in Table I are based on WSDL port types [12]. This defines how the particular task will behave. For example, if the task type is receive message, that task will not proceed unless the agent gets a message from another agent.

**Table 1. Task types**

<table>
<thead>
<tr>
<th>Task type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send message</td>
<td>Send a message to an agent, no response</td>
</tr>
<tr>
<td>Receive message</td>
<td>Wait for a message from another agent</td>
</tr>
<tr>
<td>Request-response</td>
<td>Wait for a message and send a response</td>
</tr>
<tr>
<td>Solicit-response</td>
<td>Send a message and wait for response</td>
</tr>
<tr>
<td>Start transaction</td>
<td>Workflow engine transaction control</td>
</tr>
<tr>
<td>End transaction</td>
<td>Workflow engine transaction control</td>
</tr>
<tr>
<td>Notify error</td>
<td>Send an error message to other agents</td>
</tr>
</tbody>
</table>

Transaction type defines how that task should be used in case of success and failure. For example, if transaction type is option, the commit operation will be called after the workflow has succeeded. In case of failure, it is not called. ACID transaction model will use both commit and abort operations, depending on the result. If the task is optional, its failure will not affect the success of the whole workflow.

Protocol type property states what kind of messaging will be used. Possible values include FIPA messaging [2] and SOAP [13].

Note that different transaction models in the tasks do not need to be scheduled by the developer. By stating the transaction type and the needed operations, developer can let the workflow engine do the ordering of the tasks needed only for transaction management.

Control link is a semantic level concept based on WSFL; it does not have corresponding implementation outside the workflow engine. Control links specify the execution order for the tasks. The control link types are go, fork, join, start, end, loop, commit transaction, and abort transaction. Start and end link are special cases that tell the workflow engine which task to run first and when it should stop. Start collects all workflow and transaction descriptions and combines them to a single master workflow. The master workflow is then used by the workflow engine to drive the agent. End transaction task is always followed by commit transaction and abort transaction control links; those link types can not be used anywhere else.

Join is a synchronizing control link that must be preceded by a fork. Fork does not require a join unless the concurrent flows must be synchronized.

Each control link can have a precondition that is evaluated to see if link is currently enabled or disabled. The condition is a Boolean expression. For example, the expression could be that exactly one out of two concurrent tasks must have been successful. Loop links must have a condition. Commit and abort can not have a condition, as they already have an implicit condition of successful/failed transaction.
Data links are used to describe how information flows during workflow. A data link specifies that its source task passes some named data to the workflow engine. The workflow engine will in turn pass that named data to the target tasks. Data links are based on WSFL.

Data links may have a data translation to describe how information is formatted during link transition. These translations are implementation level, and workflow engine will call agents internal operations to do the translation. The translation can be for example a XSL template that is applied to some XML data to make it uniform before sending it to the target task.

4.2. Transactions

There are two tasks and two control links that are special to transactions. Start transaction task and end transaction task define the boundaries. Commit transaction and abort transaction links follow after end transaction. In normal case, none of these are explicitly stated in the workflow description, as they can be automatically placed at the start and end of the workflow.

Start transaction task is usually the first task, with a start control link. This task gathers transaction types from the master workflow and workflow engine starts to monitor the success or failure of tasks.

End transaction task decides whether we succeeded or not. It must always be after start transaction task. It uses the information from the master workflow created in start control link, so there is an implicit data link between these two tasks. There is also two implicit control links after the end transaction task, commit transaction and abort transaction.

All normal task and link types are executed regardless of transaction, in the order specified by the control links. Before start transaction task all tasks are optional. Optional tasks do not affect the selection of commit or abort transaction link after the end transaction task. Tasks state their transaction type and commit and abort operations. The workflow engine takes care of calling the respective operations after end transaction task. The tasks report workflow engine their success or failure.

There is one exception when developer needs to explicitly take care of transaction processing. If the order in which the commit/abort operations are executed is significant, and it differs from the order of the tasks themselves, the transaction type is set to no transaction. The developer must then describe the full ordering of the tasks; including commit and abort tasks for after commit and abort transaction links.

5. Example of using ontology

This section uses our workflow ontology in a real life problem, a business trip reservation. There is one agent doing the reservations, and three agents for the services.

The user wants flight tickets, hotel room, and optional car reservation. These use different transaction models; e.g. flight ticket agent is operating in ACID model. Fig. 3 shows the workflow for flight ticket reservation agent. It offers three operations: order, commit order, and abort order. The workflow description contains only task Order Ticket, with the other operations as properties.

![Fig. 3. Flight ticket service workflow](image)

Hotel room reservation uses option model. The agent has two sequential operations that are used when reserving rooms. Abort operation is not mandatory, as the option would automatically expire.

Car reservation operates on saga model. Therefore, there is only one method needed if everything goes well. The abort method is used if the transaction fails. Fig. 4 presents a combined workflow where tasks are defined in the workflow description to be parallel.

![Fig. 4. A complete service workflow](image)

In this workflow, hotel reservation, ordering flight tickets, and booking a car are done concurrently without any specific order. They are executed as a transaction. The join operation is used to synchronize the concurrent tasks. After completing the transaction in the end transaction task, the appropriate commit or abort operations are called. In this example they are done in sequence, but the workflow generation tool could
also design a workflow model where they are executed in parallel.

In DAML-S, this workflow would have to be explicitly stated. In this ontology, the hotel, flight and car services would describe their transactional operations; the workflow engine would then be able to automatically to generate the operations after success or failure.

6. Conclusion

We have shown how workflow ontology could be used to drive agents in a multi-agent system. There are clear advantages to this approach:

- agent functionality can be modeled and driven in a familiar way, instead of defining new models such as AUML
- protocols need not be known beforehand, as agents can communicate their workflow when needed
- transactions are a proven method for reliability and can add robustness to MAS as well

This approach can also be integrated to web services that have their own workflow models. The lack of transaction support will likely be corrected in later versions of DAML-S and WSFL.

This workflow ontology can be used in a FIPA standard agent platform without modifications. We shall define a RDF serialization for this workflow ontology and test it on some agent system.

This ontology can be useful when there are tools that support it. Most importantly, it will need a workflow engine that can use it.

References


