# USING INTERACTION PROTOCOLS IN DISTRIBUTED CONSTRUCTION PROCESSES

Santtu Toivonen, Tapio Pitkäranta
*VTT Information Technology*
*P.O.Box 1203, FIN-02044 VTT, FINLAND*

Heikki Helin
*TeliaSonera Finland*
*P.O.Box 970, FIN-00051 Sonera, FINLAND*

Jung Ung Min
*Stanford University, CIFE, Terman Engineering Center*
*Stanford, CA 94305-4020, USA*

Keywords: Interaction protocols, software agents, adaptability, construction process.

Abstract: We present an interaction protocol based approach for facilitating distributed construction processes. In our approach, software agents represent various participants of a construction project. Examples of such are contractor, subcontractor, and supplier. These agents are supposed to communicate according to predefined interaction protocols. Should an agent be unaware of some protocol needed in the process, it benefits from adopting it. We approach this problem with interaction protocol descriptions serialized in a commonly agreed upon format and design our agents so that they can adapt to the descriptions. We present a scenario in the field of construction industry, where the project participants do not know in advance how to communicate with each other. However, by adapting to the interaction protocol descriptions provided by the respective parties they are eventually able to interact.

## 1 INTRODUCTION

Internet's emerging era is that of Web Services. Like HTML pages of the current web, also Web Services are distributed across the Internet and addressable with URIs. The most evident difference is the inclusion of intelligent computer programs in addition to human beings as the consumers of web content. This calls for providing material in the web with formal and machine-understandable descriptions, i.e., creating the Semantic Web.

This paper discusses a case of business-to-business collaboration in the domain of construction industry. More specifically, we outline a framework for intelligent software agents representing various participants of a construction project to collaborate. Collaboration is based on interaction protocol descriptions provided by the participants. In order to enable dynamic collaboration, the descriptions conform to mutually agreed upon interaction protocol ontology we introduced in (Toivonen and Helin, 2004).

Motivation for the chosen case stems from the contractor's need to keep track of the progress of a project it has initiated. This presupposes direct interaction between the contractor's expeditor and other participants. The expeditor agent, if unaware beforehand how to communicate with a participant, can download interaction protocol descriptions provided by the participant and adapt its behavior accordingly.

In addition to the above motivation to facilitate information flows between construction process participants, we also have a technological objective. Our intention is to investigate software agents adapting according to interaction protocol descriptions. Even though the sample scenario is focused to a particular phase of a construction process, our approach is suitable to virtually any distributed application with independent parties interacting via software agents. However, expediting process as a real world problem provides us with interesting proof-of-concept material.

The rest of the paper is organized as follows: In the next section we describe a scenario for applying our approach in certain parts of a construction process. Section 3 summarizes the interaction protocol



Toivonen S., Pitkäranta T., Helin H. and Ung Min J. (2004).
USING INTERACTION PROTOCOLS IN DISTRIBUTED CONSTRUCTION PROCESSES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 344-349
DOI: 10.5220/0002599503440349

ontology, how to describe interaction protocols based on it using RDF, and how to adapt to the descriptions. Finally, Section 4 presents some concluding remarks.

# 2 DISTRIBUTED CONSTRUCTION PROCESS

## 2.1 Sample Scenario

Supply chain management in the construction industry can at a general level be divided into two separate processes: procurement and expediting (Williams, 1995). This paper deals with the expediting process. Traditionally, performing an expediting process means paying visits to the premises of project participants, making phone calls, or exchanging emails for checking the status. There are previous attempts, such as the one described in (Kim et al., 2000), for applying software agents in construction processes. A distinctive property in our approach is that we automate parts of the expediting process by applying interaction protocols to the work performed by the contractor's software agent that acts as an expeditor.

Typically the contractor wants to keep track of the project it has initiated, and therefore expects to receive status reports of the project from the subcontractor(s) regularly. That helps the contractor to identify and avoid possible risks involved in the project such as schedule delays and cost overruns (Barrie and Paulson, 1992). However, often the contractor cannot rely solely on the information the subcontractor(s) provide in the status reports. A subcontractor can either intentionally conceal that the project is behind schedule, or some important piece of information might unintentionally be out of date in a status report.

For verifying the information in a status report, the contractor's expeditor contacts the other participants directly. Thereby, in addition to interacting with the subcontractor, we add functionalities for the expeditor agent to interact with the rest. Figure 1 depicts the states of the expediting process from the expeditor's point of view. The expeditor acts somewhat as a detective trying to dig up information from various sources and incorporate it into a complete picture.

With many project participants involved, direct interaction between the contractor and the rest can also help to avoid problem amplification. Consider a case where a supplier delivers raw material to a manufacturer, who is supposed to fabricate bulk products from that material and deliver them to another manufacturer. That manufacturer, in turn, is supposed to build a more refined product from them and again deliver it to yet another participant. Suppose further, that a problem of some kind happens at an early stage of the process—say the delivery of raw material goes to a wrong address. Unless there is frequent direct interaction between the contractor and all these participants, reacting to this misfortune most likely takes longer time. Direct interaction, instead, can fasten the information exchange. That, in turn, can help in avoiding the "bullwhip effect" (Lee et al., 1997) and the project will likely not fall as much behind the schedule.

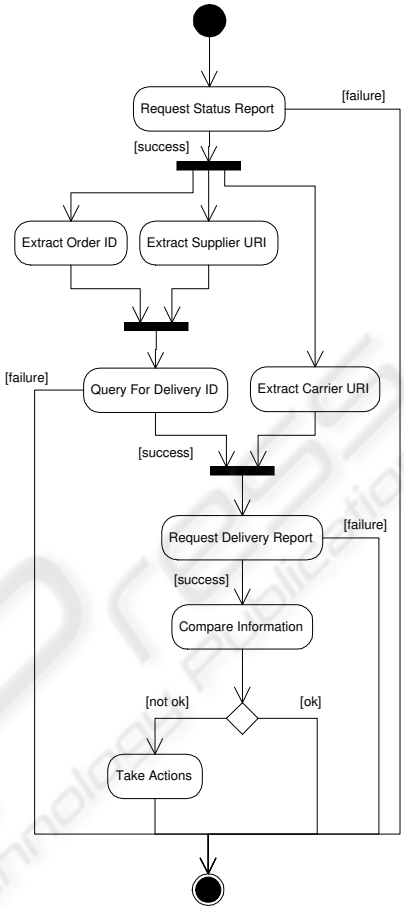


Figure 1: Progression of the expediting process from the contractor's point of view

In our scenario, the expeditor knows initially only how to interact with the subcontractor, and therefore it has to learn how to do so with other participants. We approach this problem by defining interaction protocol descriptions, and design the expeditor agent so that it can process the descriptions, and adapt its behavior accordingly.

## 2.2 Interaction Protocol Based Collaboration

We model the interaction between the contractor and the subcontractor using FIPA Request (Foundation for

Intelligent Physical Agents, 2000c) interaction protocol. The contractor requests the subcontractor to send the status report. The subcontractor then tries to perform the requested action. In the successful case the subcontractor informs the contractor that the action is performed. Should something go wrong, the subcontractor sends a failure message instead.

Assume the subcontractor has ordered a steel delivery from the supplier with the help of an external carrier, but the delivery has not yet arrived. It nevertheless includes the order in the status report before sending it to the contractor. In our scenario the contractor wants to keep track of the whole process. Thereby it contacts the supplier and the carrier to check the status of the order and the delivery. The contractor first requests the subcontractor to send the status report. By examining it as depicted in Figure 1, it finds out the addresses (URIs) of the agents representing supplier and carrier, as well as the ID of the steel order. Note that this examination, i.e., how the contractor extracts the desired information from the status report, is not in the focus of our current research, and it could be performed by the software agent itself, or by a human being. In any case, after retrieving this information the contractor queries the supplier for the ID of the steel delivery with the order ID as the parameter. The supplier provides the contractor with it, and the contractor uses it next as a parameter for requesting a delivery report from the carrier.

Since the contractor has not collaborated with the supplier and the carrier before, being able to do so at this time presupposes adaptability. In the next section we present interaction protocol descriptions in more detail. We start with the interaction protocol ontology, then move on to the actual descriptions, and after that discuss how the contractor's expeditor agent can process the information found in the descriptions and adapt its behavior accordingly.

# 3 DESCRIBING INTERACTION PROTOCOLS WITH SEMANTIC WEB TECHNOLOGIES

## 3.1 Interaction Protocol Ontology

Interaction protocols specify ordered sets of messages to be exchanged between conversating agents. We describe the interaction protocols using RDF (Lassila and Swick, 1999), a language designed for describing any resources in the Semantic Web. As mentioned, interaction protocol descriptions conform to an interaction protocol ontology (Toivonen and Helin, 2004). Core concepts of that ontology are depicted in Figure 2.

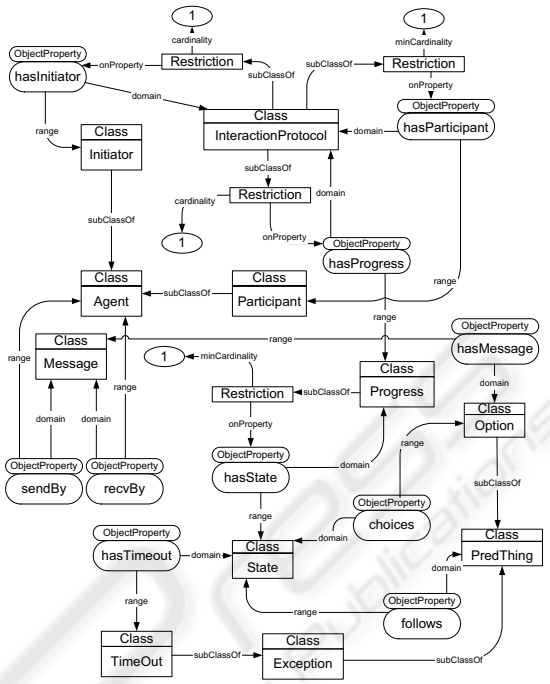


Figure 2: Core concepts of the interaction protocol ontology

Basically all interaction protocols have one initiator and one or more participants. Progress of the protocol is defined with states following each other. More specifically, states have options to choose from, and the chosen option determines which state to shift to. Options can have messages associated with them that are sent between the participating agents.

## 3.2 Describing Interaction Protocols

Among other interactions, the supplier knows how to respond to queries for selected things. In our scenario, the contractor asks the supplier for the ID of the steel delivery using the order ID found in the status report as a parameter. In order to do so, it adapts to FIPA Query (Foundation for Intelligent Physical Agents, 2000b) interaction protocol. This serialization is provided by the supplier. The following excerpts of the serialization do not capture the entire protocol, but some pivotal points. Using our ontology, description of the interaction protocol class is as follows:

```
<rdf:Description rdf:about="&this;queryDelivId">
  <rdf:type rdf:resource="&ip;IP" />
  <ip:hasInitiator
   rdf:resource="&this;contractor" />
  <ip:hasParticipant
   rdf:resource="&this;supplier" />
  <ip:hasProgress
```

```
    rdf:resource="&this;queryProgress" />
</rdf:Description>
```

The abbreviation `&ip;` refers to the `ip`-namespace containing the interaction protocol ontology. The interaction protocol class `IP` contains properties called `hasInitiator`, `hasParticipant`, and `hasProgress`, which are in this serialization given values denoting to RDF-descriptions appearing in the same file. The agents taking part in the protocol are defined in the following way:

```
<rdf:Description rdf:about="&this;contractor">
  <rdf:type rdf:resource="&ip;Initiator" />
</rdf:Description>

<rdf:Description rdf:about="&this;supplier">
  <rdf:type rdf:resource="&ip;Participant" />
</rdf:Description>
```

The agents have to comprehend the interaction protocol ontology in order for the adaptation of the individual protocol descriptions to take place. For example, the agents have to understand the basic difference between the `Initiator` and the `Participant`, i.e., that it is the `Initiator` that starts the protocol by sending the first message. Definition of the `queryProgress` class is serialized as follows:

```
<rdf:Description
 rdf:about="&this;queryProgress">
  <rdf:type rdf:resource="&ip;Progress" />
  <ip:hasState rdf:resource="&this;start" />
  <ip:hasState
   rdf:resource="&this;replySupplier" />
  <ip:hasState rdf:resource="&this;end" />
</rdf:Description>
```

Of the three states of this protocol, `start` and `end` are found in every protocol instance and knowledge of their semantics is presupposed of the agents adapting to the protocol descriptions. The agent has to know, for example, that the first state of the protocol is `start`. Below are serialized two of these three states, namely `start` and `replySupplier`.

```
<rdf:Description rdf:about="&this;start">
  <rdf:type rdf:resource="&ip;State" />
  <ip:choices rdf:resource="sendDelivquery" />
</rdf:Description>

<rdf:Description
 rdf:about="&this;replySupplier">
  <rdf:type rdf:resource="&ip;State" />
  <ip:choices
   rdf:resource="&this;sendSupplInform" />
  <ip:choices
   rdf:resource="&this;sendSupplFailure" />
</rdf:Description>
```

Every state has one or more options. These are denoted by `ip:choices` tags. The only option of `start` is `sendDelivQuery`, while `replySupplier` has two options. The actual choice between these depends on the success of the supplier's ability to answer to the query. Moving on, the options have their own serializations, which contain the possible messages to be sent, as well as the target states for the protocol to shift to. Below are two options, namely `sendDelivQuery` resulting from the activation of the `start` state and `sendSupplInform` resulting from the positive response by the supplier to the initial query.

```
<rdf:Description
 rdf:about="&this;sendDelivQuery">
  <rdf:type rdf:resource="&ip;Option" />
  <ip:hasMessage
   rdf:resource="&this;delivqueryRef" />
  <ip:follows
   rdf:resource="&this;replySupplier" />
</rdf:Description>

<rdf:Description
 rdf:about="&this;sendSupplInform">
  <rdf:type rdf:resource="&ip;Option" />
  <ip:hasMessage
   rdf:resource="&this;supplInform" />
  <ip:follows rdf:resource="&this;end" />
</rdf:Description>
```

The target state is denoted by the `ip:follows` tag. For example, `sendDelivQuery` has the above-serialized `replySupplier` as its target state. It also entertains one message, namely `delivQueryRef`, which is presented below. `sendSupplInform`, instead, has `end` as its target state, and `supplInform` as the message associated with it. The message serializations contain the information on who sends and who receives the messages, as well as references to the possible message contents.

```
<rdf:Description
 rdf:about="&this;delivqueryRef">
  <rdf:type rdf:resource="&fipa;queryRef" />
  <ip:sendBy rdf:resource="&this;contractor" />
  <ip:recvBy rdf:resource="&this;supplier" />
  <fipa:hasContent
   rdf:resource="&this;queryContent" />
</rdf:Description>

<rdf:Description rdf:about="&this;supplFailure">
  <rdf:type rdf:resource="&fipa;Failure" />
  <ip:sendBy rdf:resource="&this;supplier" />
  <ip:recvBy rdf:resource="&this;contractor" />
</rdf:Description>
```

The above message definitions contain the senders and the receivers of the messages, denoted by the respective `sendBy` and `recvBy` tags. Additionally, a new namespace called `fipa` is introduced. It denotes a location containing the FIPA extension to the interaction protocol ontology, which is described in detail in (Toivonen and Helin, 2004). Among other things specific to FIPA ACL (Foundation for Intelligent Physical Agents, 2000a), the extension allows

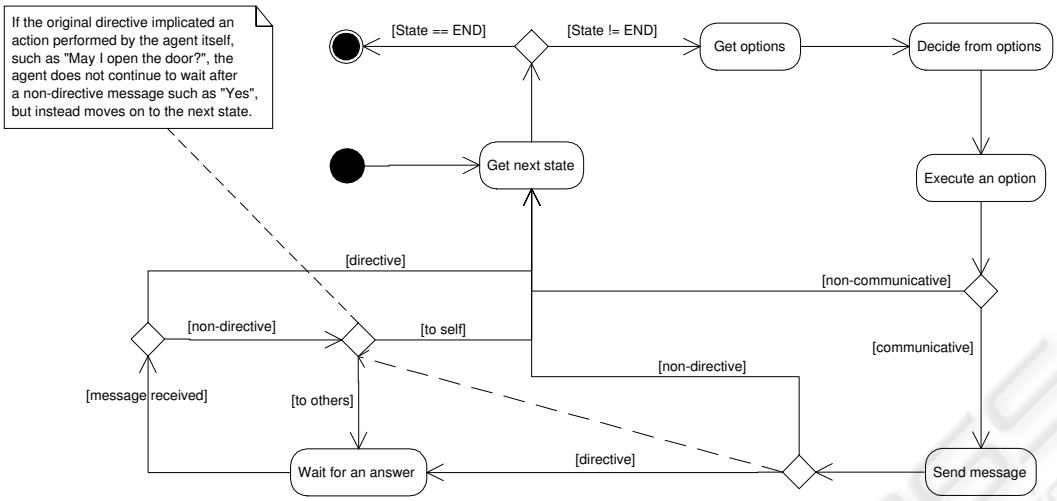


Figure 3: Flow of a protocol from the initiator's point of view

expressing message contents. Below is serialized the `queryContent` class, which expresses the thing to be queried, in this case the ID of the delivery report.

```
<rdf:Description rdf:about="&this;queryContent">
  <rdf:type rdf:resource="&fipa;Content" />
  <fipa:expression>
    String
      ORDER_ID = "http://www.sscompany.com/"
                 +"orderReports#steelOrd123";
    String query = "SELECT ?x"
      +" WHERE (?y, <rdf:type>,"
      +" <report:orderReport>),"
      +" (?y, <report:deliveryReport>, ?x)"
      +" AND ( ?y eq <" +ORDER_ID +">)"
      +" USING " +nameSpaces;
  </fipa:expression>
</rdf:Description>
```

The message content is expressed inside the `fipa:expression` tag. This phrase expresses the RDQL (Seaborne, 2002) query for a delivery ID (?x) with the order ID (?y) as the parameter. Order ID, as depicted in Figure 1, is found in the initial status report received from the subcontractor. Here we assume that the agents share a supply chain reports ontology, denoted by the `report` namespace, which captures the general structures of order and delivery reports. For example, the contractor knows that the order report can contain information about the delivery and thereby has reason to perform the above query.

With the delivery ID that the contractor receives as a reply to the above query it proceeds to request the delivery report from the carrier. The protocol is described in the similar manner as the query and not included here. Eventually, once the contractor has received the delivery report, it can compare the information in it with the information in the original status report received from the subcontractor. Should the two reports contain conflicting information, it can take appropriate actions (see Figure 1). Software agents could take part in performing the comparison, for example by notifying their owner about mismatches between the reports. However, considering these details is outside the scope of this paper.

## 3.3 Adapting to Interaction Protocol Descriptions

The only agent adapting to the interaction protocol descriptions in the above scenario is the contractor's expeditor. Other agents of the scenario have knowledge of the interaction protocols beforehand. Figure 3 depicts how a protocol progresses. Knowledge of start and end states is assumed to be known by every agent—also by the adaptable expeditor—as mentioned above. Being the initiator of the protocol, the expeditor first searches for the start state. From that state it extracts different options. Based on its current goals, it chooses one option and executes it. Note that agents' goals having effect on the decision are excluded from interaction protocol descriptions.

Executing an option that does not entail message sending is called a non-communicative option. A communicative option, instead, entails sending a message. If such message is directive, the sender of that message expects a reply to that message. Additionally, with directive messages the sender aims to get the receiver to do something, such as perform an action or answer a question (with the exception of directives "directed" to oneself). As an example consider FIPA Request; after sending the `request` message the initiator waits for an answer to it from the partici-

pant before the protocol shifts to the next state. For a non-directive message, instead, there are no answers and the protocol moves immediately to the next state. Examples are all the other messages in FIPA Request, namely `refuse`, `agree`, `failure`, and `inform`.

Unlike `agree`, all other non-directive messages in FIPA Request protocol are followed by the end state. Instead, `agree` is followed by the one where (after trying to perform the requested action and based on its results) the participant sends the appropriate message to the initiator. Note that excluded from Figure 3 are canceling and exception handling mechanisms. An agent participating in a protocol has the possibility of canceling the protocol at any point. Also, exceptions can arise at any point of the protocol.

Contractor's expeditor agent is able to adapt its behavior to conform to the interaction protocols described and provided by other process participants by combining the following information: Knowledge of the interaction protocol ontology; knowledge of the flow of a protocol as depicted in Figure 3; knowledge of utilized domain ontologies such as the abovementioned supply chain reports ontology.

# 4 CONCLUSION

In this paper we considered applying adaptable agents in an expediting process of a distributed construction project. In our scenario a software agent representing a contractor acted as an expeditor. It had the intention of contacting other project participants directly for verifying information in a status report received earlier from the subcontractor.

Since the expeditor agent did not know in advance how to interact with other project participants, it adapted to interaction protocol descriptions provided by them. These interaction protocol descriptions were serialized in RDF, and followed an interaction protocol ontology. Such shared ontology specifying the general structure of conversations between the agents brings flexibility in multi-agent systems. So long as the agents are aware of the ontology, modifying existing protocols or launching new ones is straightforward. Note that our intention was not to contentually solve problems related to expediting processes, but instead to present an application area for agents adapting to interaction protocol descriptions.

Our future work around the area includes further developing functionalities of the agents. Progress of an interaction protocol could be more dependent on the contents of the messages than it is at the moment. The agents could also have the functionality of composing protocol descriptions themselves and storing them in RDF. At the moment the descriptions are hand-written by humans. In addition, we are planning on applying software agents adapting to interaction protocol descriptions in wireless networks. In such networks, should all the client devices entertain an agent capable of adapting to interaction protocol descriptions, the client devices could inform the server about their capabilities (screen sizes, memory capacities, etc.), connection types (WLAN, GPRS, Bluetooth, etc.), as well as user profiles and contextual details. The server side agent could provide them with appropriate interaction protocol descriptions for connecting with the services.

# REFERENCES


Barrie, D. and Paulson, B. C. (1992). *Professional Construction Management: including C.M., design-construct, and general contracting*. McGraw-Hill, Inc, New York.

Foundation for Intelligent Physical Agents (2000a). *FIPA ACL Message Structure Specification*. Geneva, Switzerland. Specification number XC00061.

Foundation for Intelligent Physical Agents (2000b). *FIPA Query Interaction Protocol Specification*. Geneva, Switzerland. Specification number XC00027.

Foundation for Intelligent Physical Agents (2000c). *FIPA Request Interaction Protocol Specification*. Geneva, Switzerland. Specification number XC00026.

Kim, K. et al. (2000). Compensatory negotiation for agent-based project schedule coordination. In *Proceedings of The Fourth International Conference on Multiagent Systems (ICMAS-2000)*, pages 405–406. IEEE Computer Society Press.

Lassila, O. and Swick, R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C. Available at: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Lee, H. et al. (1997). The bullwhip effect in supply chains. *Sloan Management Review*, 38(3):93–102.

Seaborne, A. (2002). Jena Tutorial: A Programmer's Introduction to RDQL. Available at: http://www.hpl.hp.com/semweb/doc/tutorial/RDQL/.

Toivonen, S. and Helin, H. (2004). Representing interaction protocols in DAML. In van Elst, L., Dignum, V., and Abecker, A., editors, *Agent-Mediated Knowledge Management: Selected Papers from AAAI 2003 Spring Symposium*, volume 2926 of *Lecture Notes in Artificial Intelligence*, pages 310–321, Berlin, Germany. Springer.

Williams, G. (1995). Fast track pros and cons: Considerations for industrial projects. *Journal of Management In Engineering*, 11(5):24–32.