

# Development of Electronic Acquisition Model for Project Scheduling (e-AMPS) Using Java-XML

Will Y. Lin

*Ph. D. Candidate, Dept. of Civil Engrg., National Taiwan University,  
Taipei 10617, Taiwan, E-mail: will@ce.ntu.edu.tw*

**Abstract:** During the construction phase, participants in a multi-contract project acquire external real-time scheduling information from other involved parties and use this to make appropriate decisions in regard to project control. There are two major obstacles to project participants gaining efficient access to external information in a distributed data environment: (1) the variety of data structures that project members may use, and (2) lack of an automatic mechanism for data acquisition. Based on the ontology defined by eXtensible markup language Schema (XML Schema) and an automatic mechanism called Message Transfer Chain (MTC), an Electronic Acquisition Model for Project Scheduling (e-AMPS) centralized in an information agent, Message Agent (MA), was developed. Each participant equips a Message Agent as his unique information window to automatically acquire external information and provide other participants with scheduling information as well. The ultimate goal of this study is to build an automatic communication environment for multi-contract projects to solve the abovementioned difficulties, and thus achieve effective communication among project participants.

**Keywords:** Internet; WWW; Project Management; Intelligent Agent; Information Integration

## Introduction

Most scheduling theories take into account a variety of situations, such as weather, site layouts and so on, according to available information while scheduling to produce a “perfect” schedule, which seems to forecast the future very well. However, due to a large number of construction uncertainties before the project starts, such as material shortages and interference between two tasks of different subcontractors, it is common that the initial schedule has such a variance with the real condition of construction that some planned tasks cannot be carried out accordingly. Recent planning-related research, such as Lean construction suggests that schedules should be updated adequately and constantly after the construction starts, according to the real-time engineering information available to keep themselves concurrent and useful. In most multi-contract projects, however, it’s common for 80-90% of the tasks to be performed by subcontractors such that scheduling for these projects is a cooperative task which requires many project members to take part in. In order to realize the continuous scheduling suggested by Lean construction under this circumstance,

it’s necessary for these subcontractors to “dynamically communicate” together.

Communication in construction industry during construction phase is extremely complex. In terms of information technology, communication can be simplified as the exchange and reuse of information or messages between two independent parties. In this sense, to automate the communication among construction project members implies to automate the exchange and reuse of information or messages. The exchange and reuse of engineering information have been an issue in the field of automation in construction since information technologies were first introduced in 70’s. Much research and related applications have also been developed to achieving all kinds of automation in communication. However, there still are two major obstacles to automate the continuous and collaborative scheduling for multi-contract projects: (1) the variety of data structures for scheduling that project members may use, and (2) the lack of an automatic mechanism for data acquisition in such a multi-user workplace for most multi-contract projects.

Based on the ontology defined by the eXtensible markup language Schema for Scheduling (XSS), the Data Acquisition Language for Scheduling (DALs), the Hierarchy Searching Algorithm (HSA), and an automatic mechanism called Message Transfer Chain (MTC), an Electronic Acquisition Model for Project Scheduling (e-AMPS) centralized in an information agent, Message Agent (MA), was developed. Each participant equips a Message Agent as his unique information window to automatically acquire external information and provide other participants with scheduling information as well. The ultimate goal of this study is to build an agent-based communication environment for multi-contract projects to solve the above-mentioned difficulties in automating communication in a multi-user workplace, and thus realize continuous and collaborative scheduling.

### Architecture of e-AMPS

To solve the difficulties involved in sharing scheduling information among project participants in a data-distributed environment, an agent-based communication environment called Electronic Acquisition Model for Project Scheduling (e-AMPS) has been developed. The model is centralized in an information agent called Message Agent. Basically, Message Agent is a computer program that deals with all messaging tasks involved in automatic communication, and will be introduced in the following sections. Each participant in the same project, named a *Host* or *Contact Node* in the following paragraphs, equips a Message Agent as a unique information window so that Message Agents in the same project can automatically communicate with each other. In this section, we introduce the basic framework of e-AMPS and the functions of Message Agent in order to give an overall picture of the proposed concepts. Figure 1 illustrates the complete architecture of e-AMPS. The complete automatic communication consists of two different levels of replying to the imported requests: Data-retrieving level and Decision-making level. In this paper, we only focus on the Data-retrieving level. However, the components within the Decision-making level are also addressed to some extent in this section to help draw a more complete picture

of our model. The complete framework of e-AMPS consists of five major components: Ontology base, Message Agent (MA), Open Data Repository, DALs-speaking Decision Support Systems for Scheduling, and Message Queues [1].

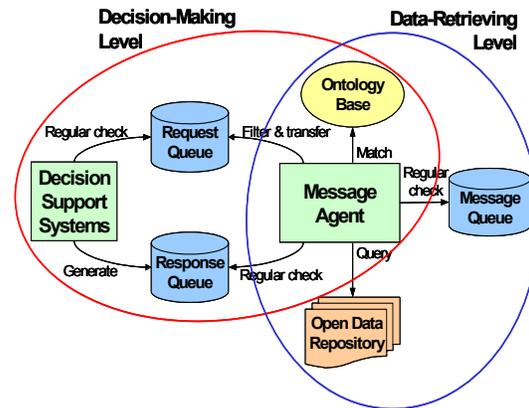


Figure 1: Main Framework of e-AMPS

- (1) **Ontology base:** This stores all ontology on scheduling in terms of data schema, called XSS, the syntax of which is adopted from XML schema in our study. The ontology here is defined as “a specification of a conceptualization, or a description of the concepts and relationships that can exist for an agent or a community of agents.”
- (2) **Open Data Repository:** This contains scheduling information with standard data structure defined by the ontology (XSS), whose data structure is shown in Figure 2, and is in XML syntax [1]. It’s basically a file folder that contains all scheduling information files of standard formats. There are two kinds of scheduling information files for each e-AMPS: Schedule File (schedule.xml) and Contract File (contract.xml).
- (3) **Message Agent:** This deals with all manipulation of incoming and outgoing messages following the communication mechanism built by e-AMPS concepts. It communicates with other Message Agent mounted on other contact nodes, and also with its local decision support systems through the mapping table.
- (4) **DALs-speaking Decision Support Systems for Scheduling:** They are built by the host, independently from the Message Agent. They have independent decision support models to

generate specific decisions toward certain fields. Most important of all, these decision support systems all recognize the DALS and use it to request for information from other project participants as their input data [1].

(5) Message Queues: Message Queue, physically an open access file folder, contains all messages (requests or responses) from other Message Agents. Each Message Agent will access its Message Queue regularly and automatically to react according to the messages [1].

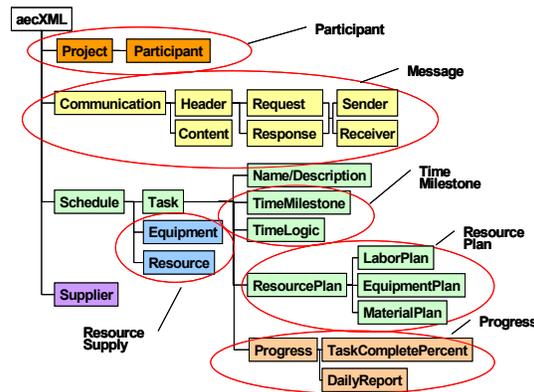


Figure 2: Data Structure of XSS

The utilization of e-AMPS can be divided into three stages:

(1) Installation: Every practitioner in the construction industry, i.e. Owner, A/E, contractor, and supplier, equips a Message Agent that is implemented by Java, regardless of platform to be used, and therefore a communication environment is then built where Message Agents in the same project can automatically communicate with each other. When installing the system, a practitioner is prompted to specify the local file folders for the Ontology Base that contains the data schema file, the Open Data Repository that contains the scheduling files, and the Message Queue that contains the message file. The folder for Message Queue should also be associated with a Uniform Resource Indicator (URI), which can be openly accessed by Message Agents of other project participants.

(2) Contract signing: Once the practitioner signs a contract with another party, the scheduling information is then prepared according to the data schema, and is deposited in the Open Data Repository, where the

Message Agent can access and make inquiries about the shared scheduling information.

(3) Contract execution: While the contract is being carried out, the Message Agent checks the Message Queue regularly and deals with all messaging tasks automatically according to the message it receives, such as sending the requests originating from the Host or passing the responses sent by other Message Agents to another Message Agent.

The detailed design of the Message Agent can be referred in another paper [1].

### Implementation of Message Agent 1.0

Java™ language is a rich environment for XML programming since there have been more XML-specific resources available in Java than in any other programming language. There are two major reasons why Java meets XML programming. The first is their shared reliance on the Internet. XML was designed to be straightforwardly usable on the Internet, while Java was designed to be used over the Internet. Java works well in a distributed environment, allowing users and programs to share information easily, while XML provides a tool for distributing and storing that information. The other reason is their shared use of hierarchical structures. Java's object-orientation and XML's fundamental use of nested hierarchies is a suitable match of combination. Programmers can easily develop tree structures with Java that match the structures of an XML document, making it easy to convert XML files into instantly usable data in Java application or applet. Due to the above reasons, this research uses Java 2 as the developing language. The Java™ API for XML processing has been added to the Java 2 Platform. It provides basic support for processing XML documents through a standardized set of Java Platform APIs, and other network-specific programming facilities suitable for the implementation of e-AMPS and Message Agent. Several programming features are addressed first, which are multi-thread processing, parsing with a validating mode using XML Schema, and the use of Remote Method Invocation (RMI).

(1) Multi-thread processing: A thread — something called an execution context or a lightweight process — is a single sequential

flow of control within a program. A single thread process means that a process has a beginning, a sequence, and an end and at any given time during the runtime of the thread, there is a single point of execution. On the other hand, multiple threads mean that there are more than one single thread running at the same time and performing different tasks within a program. Since carrying out various manipulations of a message, the Message Agent is implemented with multiple threads and thus different manipulations of a message are able to proceed independently and smoothly.

(2) Validating documents using XML Schema: There are two types of XML parsers, divided by different function levels: validating parser and non-validating parser. There are also two methods to validate an XML document: using Document Type Definition (DTD) or using Schema. An XML document is valid if it has an associated DTD or Schema, and if the document compiles with the constraints expressed in it. A DTD defines the data structure of an XML document. It specifies the order in which tags occur, what the tags are, and how many tags are allowed. A DTD provides a uniform format for defining the structure and markup of an XML document. Unlike DTDs, however, XML Schemas adhere to the XML specification and provide better support for XML namespaces and more data types. It is also a recommendation of the W3C. Schemas provide a more flexible means for defining the structure, content, and semantics of XML than DTDs. In many areas of application, DTD is replaced with XML Schema nowadays although DTDs had been widely adopted for years.

Due to the above-mentioned advantages of XML Schemas, the Message Agent adopts a validating parser using XML Schema.

(3) Use of Remote Method Invocation (RMI): Since several major manipulations of a message are involved in passing an XML-based message from a local Message Agent to remote Message Agents, an approach of file transferring from one host to another is required by the Message Agent. Although the protocol File Transfer Protocol (FTP) or other message transfer methods such as SOAP is a possible way to be applied to this end, the Message Agent 1.0 adopts a remote access

mechanism provided by Java called Java Remote Method Invocation (RMI) since RMI allows an object running in one Java Virtual Machine (VM) easily to invoke methods on an object running in another Java VM. RMI provides for remote communication between programs written in the Java programming language.

Figure 3 is the flowchart of starting up and stopping Message Agent, which is the main stream of the whole program. Since the process of dealing with messages undertaken by the Message Agent is a routine task with a given running period, the main stream starts at arousing a thread called *MainLoop()*. *MainLoop()* then triggers three child threads: *MessagingTask\_AppendingMessage()*, *MessagingTask\_CheckingMessage()*, and *MessagingTask\_DispatchingOutboxMessage()*.

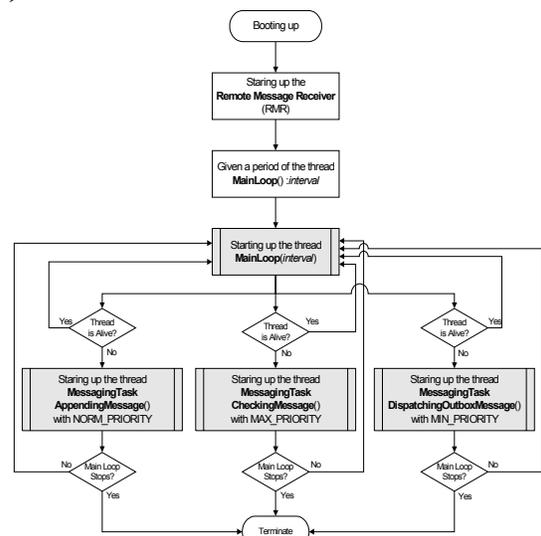


Figure 3: Flowchart of Running Message Agent

In three child threads of the thread *MainLoop()*, whether the thread lifecycle of last execution is finished or not would be examined first. If the thread is still “alive”, the new thread will not be triggered. Thread priority is set mainly according to the average running time spent. The thread that spends the longest time averagely gets highest priority. Figure 4 illustrates the core objects of Message Agent version 1.0 and their relationships one another. The class *MainFrame* is the visual user interface that initiates the root class *MessageAgent* of the whole program. Under

the root class, there are thread class *MainLoop* with three child threads, five major message processor/manipulation classes, and a **RMI** class *RemoteReceiver* that implements the interface *FileReceiver*.

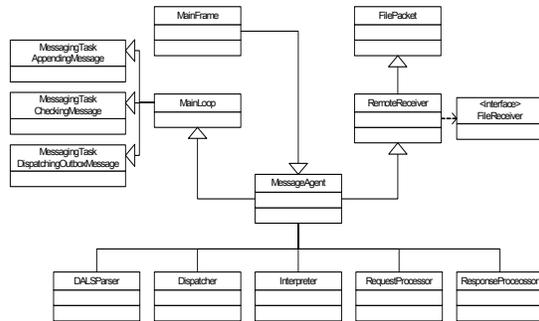


Figure 4: The Relationship between Objects

**Scenario**

A hypothetical design-build project is made up to illustrate more fully the concept of e-AMPS and the effect of the Message Agent. The milestone network and bar chart of the project are shown in Figure 5.

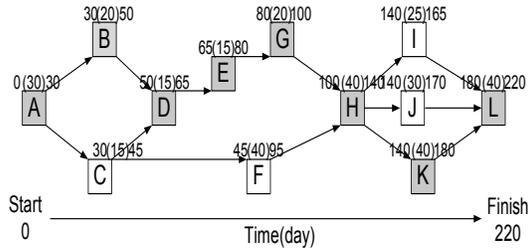


Figure 5: The Project Network of the Example

There are 12 packages enclosed in this project, undertaken by a general contractor and his 11 sub-participants, from P1 to P11. Figure 6 shows summary bar charts of all sub-participants under the cooperation structure of the example project. The entire project starts on Jan 1<sup>st</sup>, 2002 and finishes on Aug 8<sup>th</sup> in the same year using a calendar of 7-workingday a week due to simplify the complexity of the example.

In following paragraphs, a scenario is made up respectively associated with a typical communication behaviors for scheduling: requesting for progress data. The whole communication cycle, from the original request to the terminal responses, is recorded and represented as well as some important

facts and results are extracted to emphasize the effect of e-AMPS.

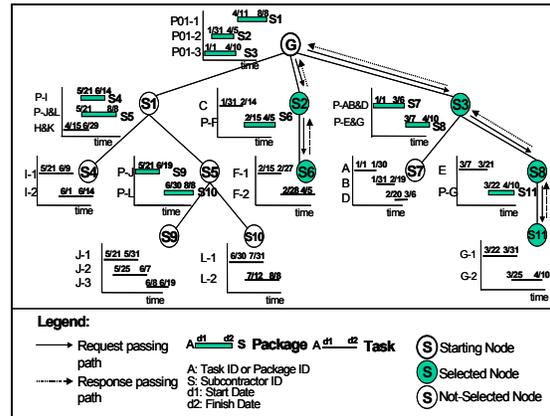


Figure 6: Message Passing Path of the Scenario

On Mar 10<sup>th</sup> 2002, the general contractor originates a request for the progress information of all tasks whose duration overlaps a period between Mar 15<sup>th</sup> and Mar 25<sup>th</sup>, about two weeks before the **Task H&K** of participant P1 starts. The original request without the header created by a pre-programmed process at the general contractor’s site is deposited in the Message Queue, and waits for his Message Agent to dispatch it. The Message Agent of the general contractor detects this request and automatically performs the HSA. Since having no upper messengers, Message Agent of the general contractor decides to dispatch the request to two of its lower messengers, S2 and S3, since the packages undertaken by S2 and S3 meet time and scope constraints.

The request to S2 is bypassed to S2’s lower, S6, due to Package P-F undertaken by S6 meets the constraints specified by the request. Upon receiving the bypassed request from S2, S6’s Message Agent perform the query transformation and generate a response sending back to S2, flowed by another bypassing by S2 back to the general contractor.

Figures from Figure 7 to Figure 9 show the sequences of message manipulations by the Message Agents of participants involved in this scenario, G, S2, and S6, respectively. At the site of G in this scenario, G’s Message Agent passes two requests (requestId: 3945 & requested: 8379) to two of his lowers, S2 (URI:

140.112.10.31) and S3 (URI: 140.112.10.32), respectively. 35 seconds later, it receives the first one response (responseId: **3884**) from S3 in which the original replier is S8, according to the message log at the site of G. 10 more seconds later, it receives the second responses (responseId: **3389**) from S2, in which the response is generated by S6. 6 more seconds later, it receives the last response again from S3, in which the original response is generated by S11 and is bypassed through S8 and S3 in turns.

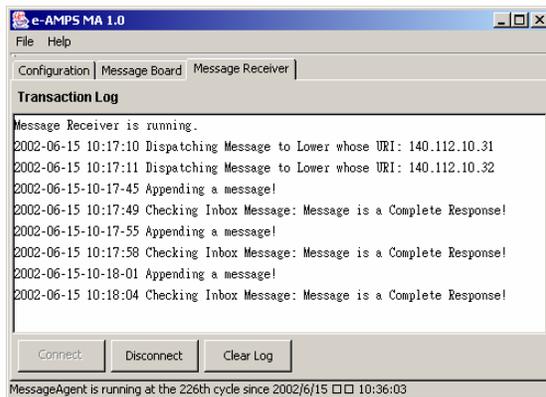


Figure 7: Message Manipulation at the site of G (URI: 140.112.10.16)

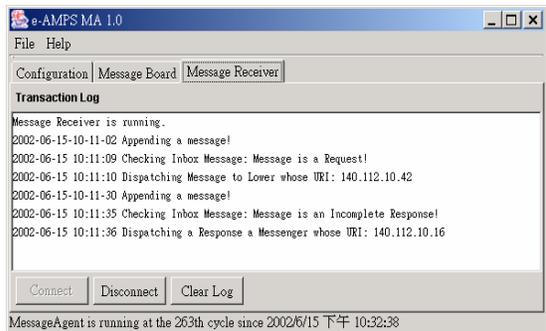


Figure 8: Message Manipulation at the site of S2 (URI: 140.112.10.31)

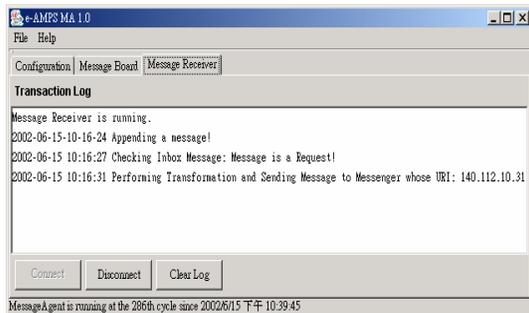


Figure 9: Message Manipulation at the site of S6 (URI: 140.112.10.42)

## Conclusion

Sharing of project scheduling information among subcontractors is useful for predicting potential delays and taking any necessary precautions. However, there are two major obstacles to multi-contract project participants accessing the external information they need efficiently: (1) the variety of data structures that project members may use, and (2) lack of an automatic mechanism for automatic data acquisition. An agent-based communication environment called Electronic Acquisition Model for Project Scheduling (e-AMPS) is developed to solve the abovementioned shortcomings. Message Agent was implemented using Java 2 and tested in IBM PC with Windows 2000 OS. The testing and system performance have been evaluated with positive results.

## Acknowledgement

The authors would like to acknowledge the National Science Council, Taiwan, for financially supporting this work under contract No. NSC-90-2211-E-002-074.

## Reference

1. W. Y. Lin, "Development of Electronic Acquisition Model for Project Scheduling (e-AMPS) Using Java-XML" PHD Dissertation, Department of Civil Engineering, National Taiwan University, 2002.