
AIMMS Multi Agent and Web Services User's Guide - Introduction

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 1

Introduction

In this chapter you will find a brief introduction to the basic concepts of agents. This introduction is then followed by some possible motivations for using agents. The last part of this chapter introduces you to the way agents have been implemented in AIMMS.

This chapter

1.1 Basic agent concepts

An *agent* is a software program that independently performs its task on behalf of its user. As software programs all perform tasks on behalf of users, the distinguishing feature is to be found in the word “independently”. Even though contact between an agent and its user is possible, the typical agent operates autonomously.

Agents

A *multi-agent system* is a community of agents that communicate together to accomplish individual and/or common goals. These agents are sometimes referred to as intelligent agents or decision agents to reflect their decision-making role within the community. A *multi-agent application* is a specific application of a multi-agent system to accomplish a specified set of individual and/or common goals.

A multi-agent system and application

In addition to being *autonomous* and *intelligent*, as indicated above, agents are also *reactive* and *proactive*. They are reactive in that they respond to other agents. They are proactive in that they work to accomplish a goal.

Agent characteristics

Agents communicate by sending messages. Even though messages can be very complex structures in certain multi-agent languages, in this chapter it suffices to think of messages as a medium to transfer data in the form of numbers and strings from one agent to another. Since agents can be proactive and reactive, messages can either be initiated or be responded to. Messages play a central role in the overall design of a multi-agent system.

Agent communication

Agents in a multi-agent system do not necessarily reside on the same host computer. As shown in Figure 1.1, agents can be distributed over several computers, and each agent has its own associated message queue. Two-way communication is indicated through the words “request” and “response”, and both requests and responses are implemented through messages. Whenever an agent is idle, it will read the next message on its message queue, and react accordingly. Such communication between agents takes place asynchronously, because it is not known when an agent will be available to react to a particular message.

Communication network

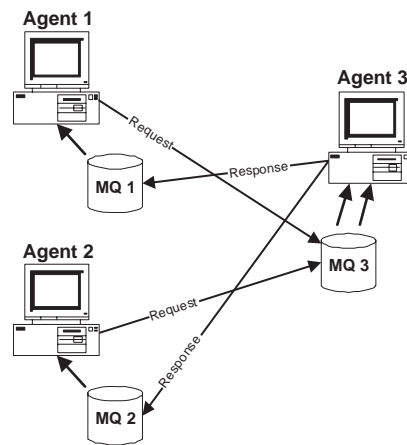


Figure 1.1: Communication between agents

From AIMMS version 3.6 onwards, a multi-agent community can also be deployed as a *web service*. This means that a special type of agent is placed in the community, the so-called *AIMMS web service handler*. This special agent can be called by means of *web service requests* by external applications. This technology enables a multi-agent community to be accessed by third-party software from anywhere on the Internet.

Web services

There are applications in which the communication between agents has to be synchronized in order to be meaningful, such as a market clearing agent that wants to receive all bids before clearing the market. Typically, synchronization facilities are introduced through special message types or wait mechanisms. This topic will be considered in the later chapters.

Synchronization

The computing paradigm underlying multi-agent systems is that of *parallel computing* and *non-deterministic computing*: agents can reside on different processors, and the order of execution is not predetermined due to the asynchronous communication between agents.

Computing paradigm

1.2 Examples of motivation

In this section are some indications of the types of problems that lend themselves to being analyzed and solved by agent technology. For the purpose of brevity, the problems are presented in a somewhat abstract and introductory format.

This section

A purchasing agent is a software program that independently checks inventories at regular moments, determines order quantities, requests price information from suppliers, selects a supplier on the basis of several criteria, places orders, traces deliveries, and updates the inventory data when goods have arrived. This is a typical example of an individual and proactive decision agent.

A purchasing agent

In an auction run by autonomous agents there may be bidding agents and a market clearing agent. The bidding agents supply their bid each round in accordance with well-specified market rules. The market clearing agent informs the bidding agents each round of how the market is cleared. The market rules also determine when the auction is over. Such an auction is a typical example of a multi-agent system containing all agent characteristics.

Auctions

In an advanced plant control system there will be a hierarchy of agents. At the lowest level there are multiple agents observing, registering, and analyzing plant measurements. At a higher level there are coordinating agents that gather status information from agents below them in the hierarchy. Such a control system is a typical example of distributed computing with asynchronous decision making.

Control systems

The computational time required in several solution approaches, in the areas of simulation and optimization, can be improved through course-grain parallelization. Worker agents can be introduced to execute work that can be done in parallel. Typical examples are Monte Carlo simulations, second stage computations in two-stage stochastic programming, partitioned branch-and-bound computations, and submodels in multi-commodity networks following the allocation of arc capacity to commodities. In such multi-agent applications it is common to introduce a dispatcher agent to keep track of idle worker agents. The dispatcher will then ensure that all computational work is divided effectively among all the workers.

Algorithms

Assume that a large decision problem, such as an extensive supply-chain planning model, can be divided up into distinct decision phases. In addition, assume that each phase has its own model, algorithm and parameter settings, and that there are separate constraints across the various phases. Then a distributed solution approach may be appropriate for obtaining a good-quality solution to the overall problem.

*Distributed
problem solving
...*

The basic concept is to have multiple agents solving phases using the different models, algorithms and parameter settings, and letting the output of one phase be the input to the next. The evolving tree of solutions can be observed by special agents to verify global constraints, and to give feedback to try alternative parameter settings based on their observations. These special agents can also eliminate partial solutions based on their poor overall quality. Humans can also watch the functioning of such a multi-agent application, and can also provide input to steer the overall approach.

*... evaluating
many
alternatives*

By using many agents at once, and letting each agent run on its own processor, it may be possible to find one or more good alternative solutions to a very large problem in a matter of hours. The same approach, but without the use of agents, could take weeks or even months before the same overall solution quality is obtained.

*... using
massive
parallelization*

1.3 Agent-related concepts in AIMMS

In this section you will find a brief introduction to the agent concepts that are part of the AIMMS multi-agent technology. This introduction is meant to provide a basic understanding. Further details are to be found in the multi-agent reference provided in Chapter 3.

This section

1.3.1 Agents in AIMMS

It may ease understanding to view an AIMMS agent as an AIMMS session running an instance of a specific AIMMS project. When there are multiple such agents, their corresponding AIMMS sessions will be running on one or more computers within a local area network, and their corresponding project files will be stored at a specific network location.

AIMMS agents

In a multi-agent system it is quite natural to distinguish between several types of agents. The types are indicated through the *role* that each agent plays in the overall underlying application. In the previous section we encountered some examples of roles such as purchasing agent, bidding agent, market clearing agent, worker agent, and dispatcher agent. Agent roles play a crucial part

Agent roles

in constructing and understanding multi-agent systems as implemented in AIMMS.

In AIMMS, messages make up the vocabulary to be used between agents. An AIMMS agent can send a message to another agent by calling a send procedure with arguments as illustrated in Figure 1.2. The first two arguments are always the names of the sending and the receiving agents. The remaining arguments are the data items to be transferred. In this example, the two data items are input arguments.

AIMMS
messages...

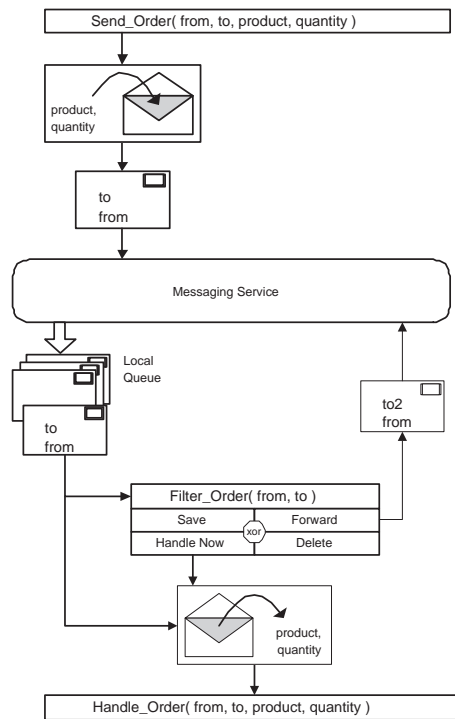


Figure 1.2: Messages with only input arguments

Once an AIMMS agent makes a call to a send message, the underlying AIMMS technology packs the input data to be sent to the addressee. Once sent, the data package will arrive in the message queue of the addressee. As soon as the receiving agent is idle, the underlying AIMMS technology takes the first message from the queue, unpacks the data and calls the handling procedure that corresponds to the particular message.

... en route

In AIMMS, a message handler is a procedure with the same data arguments as the corresponding send procedure. The underlying AIMMS technology ensures that the incoming data are passed to the arguments of the message handler. Inside the body of the message handler the available data are processed. Typically there are one or more statements that transfer the incoming local data into global identifiers.

AIMMS message handler

In addition to input arguments, an AIMMS message can also have output arguments, i.e. data items with argument property Output or InOut. In such a situation, values must be sent from the receiving agent back to the sending agent. Inside the body of the handler procedure all the output-related arguments have to receive a value through assignment statements. As soon as the handler procedure has been executed, the corresponding return values are automatically sent back to the original sender of the message without requiring any further action by either the sending or the receiving agent.

Return arguments ...

The sending agent has to wait for the arrival of the return values. The waiting time is not known in advance, and is dependent upon the performance of both the receiving agent and the underlying network. Therefore, any send procedure with one or more output arguments must always have a time-out argument. Execution will resume as soon as return values arrive, provided this is within the specified time-out period.

... require a time-out

If at the end of the time-out period return values have not arrived, execution will resume but all computations will be based on previous values of the output arguments. One should therefore always check the special return value of the send procedure, and use an error handler when this return value indicates that no new data have been passed into the output arguments.

... and an error check

If you want to avoid a sending agent waiting for the arrival of return values, you should design your multi-agent application such that all messages have only input data. Whenever a particular message requires a response from the receiving agent, then this response can instead be defined as a separate message to be sent by the receiving agent. A typical example is when the sending agent has to send the same message to several individual agents. In this case, all the messages can be sent at once. The requested responses will eventually arrive at a later time, and will be handled on a first-in-first-out basis by the appropriate handler procedure of the sending agent. In this way, the data transfer is fully asynchronous.

Without return arguments ...

If the underlying application (for example, an auction) requires the sending agent to collect one or more responses before continuing execution, this can be implemented by using the special waiting functions offered by AIMMS. With these functions one can block the current execution of the sending agent until

... response can subsequently be synchronized

one or more specified messages are received and handled. Naturally, these synchronization functions require a time-out value that determines the maximum period that such a function will wait for the responding message(s).

There are agents, such as forwarding agents, that do not need to unpack the data of incoming messages. For such agents you can specify that the incoming message should first go to an AIMMS message filter. Such a filter is also a procedure, and allows the agent to either delete, save, or forward the message. If, for any reason, the message is not deleted, saved, or forwarded, inside the filter, then the data will be unpacked and the message automatically passed on to the corresponding message handler.

AIMMS message filter

You, as a designer of a multi-agent application, can specify whether or not a particular message is a priority message. Such a message is not only handled as soon as the receiving AIMMS agent is idle, it is also handled between execution statements. This requires care on your part to ensure that a priority message does not have any negative side effects when handled between execution statements. All priority messages are placed before the regular (non-priority) messages on the message queue. Priority messages are handled in sequence on a first-come-first-served basis.

Priority messages

1.4 Agent construction support

AIMMS provides several facilities for designing and running a multi-agent system using the concepts described in this chapter. The design facilities consist of two extensive dialog boxes: one for the overall designer of a multi-agent application, and one for the designer of a specific agent with its own particular agent role.

AIMMS agent design technology for ...

If you are the overall designer of a multi-agent application to be written in AIMMS, you will need to specify all the necessary ingredients such as agent roles and agent messages, which, together, are also referred to as the *community setup*. For each role you must indicate which messages can be sent or received, and for each message you must indicate its from-agents and its to-agents. In addition, you must specify the permitted priority level of each message. AIMMS offers a special dialog box to support you in this design task.

... community setup

If you are the designer of an individual agent in AIMMS, you need to select its specific role from the roles in the community setup. In accordance with this role, you then need to specify the send procedures for all messages to be sent, and the message handlers for messages to be received. Again, AIMMS offers a dialog box to support you in this design task.

... and agent setup

Other facilities to assist in running a multi-agent system in AIMMS consist of being able to log on to an existing network of agents, to start a remote agent, and to log off. In addition, there is a whole layer inside AIMMS, not visible from within the model, that ensures that the AIMMS agent functionality works properly across the local area network.

Other facilities