A Multi Agent System for MS Windows using Matlab-enabled Agents

Th.M. Hupkens[a] R.Thaens[b]

[a] Royal Netherlands Naval College, Combat Systems Department,
P.O. Box 10000, 1780 CA Den Helder, The Netherlands

[b] Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands

Abstract

A method is described to program the properties of agents of a multi agent system by using Matlab instructions or Matlab m-files. Apart from the obvious advantage that all Matlab functionality is immediately available, there is the added advantage that all debugging capabilities that Matlab provides can be used. Also it becomes possible to change values of the parameters during a multi agent session. As an example a simple task scheduler was built. Since the MS-Windows messaging system is not very well suited to exchange large amounts of data between agents, a memory-mapped file is used to improve the performance of the system.

1. Introduction

Intelligent software agents provide a very flexible manner to solve various problems. Unfortunately, most existing development tools for multi-agent systems require much experience in Java programming. For our research we need a fast prototyping system for the behavior of the agents. Furthermore, for reasons of speed and safety, we want to run the system on a single computer. For these reasons we have built a multi agent framework based on the MS-Windows messaging system. The actual intelligent behavior of the agents can be programmed in Matlab. This is particularly important for research purposes, because it provides almost unlimited flexibility. The researcher is able to run several tests using different m-files, he can change parameter values during run-time and, best of all, all Matlab functionality is immediately available, including the functionality of toolboxes such as the “Symbolic Math” toolbox.

The communication between the agents in the current implementation is hard-coded in a high level language (Delphi-Pascal), but in future

implementations, the system will be changed such that this aspect of the multi-agent system can also be coded in Matlab or specified by means of a text file.

Unfortunately the MS Windows messaging system is not very well suited for creating multi agent systems, because only messages of very limited size can be sent. So messages should be used as notification messages only and not for data transport. Data transfer can be realized in a simple, fast and reliable way by means of memory-mapped files, which can be seen as common memory between different applications.

2. Agent systems based on Windows messages

Every MS-Windows message has a unique message number. For user-defined messages that communicate between separate applications, this number must be obtained from the operating system, because only then this number is guaranteed to be unique during a Windows session. This prevents conflicts with messages broadcasted by other applications. Apart from the limited capabilities of the messaging system, it is a simple system and relatively fast, compared to for instance protocols that are often used in multi-agent systems [1].

3. Memory mapped files

MS-Windows does not allow different applications to access the same memory directly. However, it is possible to create a (temporarily) file of arbitrary size and map the contents of this file to memory. All application can write to or read its content, assuming the file is created with read and write permission. Applications obtain a pointer to this mapped file, so reading and writing data is as easy as reading or writing data from any ordinary variable. In the current application, the implementation details are defined in a separate unit, in such a way that programs that use this unit do not have to be aware of any implementation details, but they see the shared memory as an ordinary, structured variable. Sharing data this way is very fast and simple [2].

4. Matlab as an automation server

On MS-Windows platforms, Matlab can act as an automation server [3]. Automation allows one application (the controller) to control another application (the server). Matlab commands can be executed, and data can be exchanged between the Matlab workspace and the application's memory[1]. Matlab can be launched from an application in two essentially different ways:

1. Only one session of Matlab is opened; further programs connect to this session. The Matlab global workspace is available to all applications that connect to Matlab. This is very useful for exchanging data.

2. For every application a separate Matlab session is started. There is no common workspace. This is useful if one needs Matlab for sophisticated calculations, where only the calling agent needs the result of the calculations.

In both cases the Matlab command window can be either hidden or visible. If visible, the user may change variables, pause the process, change m-files online et cetera.

5. Implementation details

Some implementation details:

- The programs were coded in Delphi-Pascal (version 6). The messaging part of the program and the Matlab calls are equally simple to code in Visual Basic (memory mapped files are more difficult, because VB does not support pointers). In C or C++, connection to Matlab is a bit trickier because the type "Variant", which is needed, is not a standard C++ type.
- In order to get results from the Matlab global workspace, a temporary variable is needed. This variable could in theory interfere with running m-files. To avoid this, a strange (hopefully unique) name is used. This name is cleared immediately after use.

[1] Matlab can also make use of Distributed COM for cross network communication, but this is not of interest for our purposes, because we restrict ourselves to applications that run on one system.

All agents can be instructed to use a Matlab m-file or to use the default (hard-coded) strategy. Only if any of the agents wants to use Matlab, all agents place the necessary information in the Matlab common workspace. The name of the m-file (if any) and the task specification can be specified either during run-time (for debugging or demonstration purposes) or via a configuration text file for batch processing.

6. A task scheduler based on the multiple agent system

The primary goal of the multi-function phased array radar (MFR) onboard naval ships is to provide a complete, accurate and reliable air picture to the operator. In addition, the MFR can assist in cueing onboard sensors and weapon systems when engaging targets. In order to do so, the MFR needs to revisit objects already identified and in track, and also has to search for new objects. The radar beam can switch almost instantaneously between two directions in space. The 'track' and 'search' tasks are therefore virtually decoupled and can be scheduled consecutively to, but independent from, other tasks. The following paragraphs present an initial design of a scheduler based on a multi agent system (see Thaens [4]).

6.1. Description of the scheduling problem

A task $T_i, i \in \{1,2,...,N\}$ is defined as:

T_i = {

D = duration

t_b = earliest start time

t_f = latest completion time

P = priority of this task, $0 < P < 1$

},

with $D < t_f - t_b$.

The measure P is an indication of the importance of this particular task. In case of radar scheduling, P can be derived from an operational risk analysis based on operational parameters like the identity of the object, flight profile, Rules of Engagement in force, et cetera.

The individual agents are relatively simple and do not necessarily have to know about the existence of the entire set of other agents. This

simple architecture only caters for a mapping of tasks on available timeslots, but does not include any feedback on developments in the operational scenario or on constraints laid down by the technical performance of the radar. Currently for every agent the initial priority is given at the moment of creation. In reality however, the priority of a specific task is scenario dependent, which requires the agent to be cognizant of its environment.

6.2. A basic Task Scheduler

The presented radar scheduler is a basic initial design. In order to demonstrate the use of agents for planning tasks on a timeline, it suffices to use a blackboard model that contains two timelines. Every task in the task set is instantiated by an agent, which only knows the particularities of its task (T_i). In addition to these task agents, an additional evaluator agent is introduced. The evaluator agent evaluates the proposed timelines with regard to the utility of that particular timeline. The utility may be defined for instance as:

$$U = \sum \delta_i P_i, \text{where } \begin{cases} \delta_i = 0 \text{ if task not in planning} \\ \delta_i = 1 \text{ if task in planning} \end{cases}.$$

Initially the Actual Time Line (ATL) is empty and an agent is selected from the list of available agents. This list contains all agents that have not been able to insert their task into the most actual timeline. Based on the task it represents, the agent attempts to place the task in an available space on the timeline. In doing so, the agent creates a so-called Proposed Time Line (PTL). If the agent is finished, the PTL is put on the blackboard next to the ATL. If the PTL yields a higher utility than the ATL, the evaluator agent copies the PTL to the ATL, else the ATL remains the same. If the evaluator accepts the PTL, the proposing agent is removed from the agent list. In case an agent is not successful in finding a vacant position on the ATL, it can decide to remove an already planned agent from the ATL and insert its task instead. If the evaluator agent accepts the PTL in that case, the removed agent is added to the list of agents again, so in turn it may try to get back into the ATL on another position. This simple mechanism only requires passive agents who wait in turn to propose a new

timeline. There is no communication between the agents nor will the agents attempt to cooperate in order to have the overall utility maximized. As a result we have seen situations in which sub-optimal utility values have been achieved without the system being able to overcome this disadvantage. An elegant way of reducing the risk of sub-optimization is through increasing the level of cooperation between the agents. Cooperating agents need to be able to exchange information with other agents when a conflict between them occurs. Instead of letting every individual agent propose a modified timeline on a blackboard, the agents in conflict could evaluate more complex changes to the timeline, which involve an even larger group of agents. For further reading on coordination and agent negotiation see [5].

6.3. The Task Scheduler using Matlab-enabled agents

If desired, the user of the MAS can specify a different Matlab m-file for every agent that is started (see the Appendix for example m-files). The function takes one parameter, the logical number of the agent. The calling agent provides this number. The function should output a single value (in case of the evaluator agent it is the utility of the current proposed time line, in case of a task agent it is the proposed starting point on the time line). The name of the function is not important. If any of the agents uses Matlab, all agents will specify their task in the Matlab common workspace. We performed a number of tests using Matlab-enabled agents. Several different strategies for the task agents were successfully tested (e.g. the agents try to place their task such that at maximum one other task is replaced by their task), as well as a utility function that favors short processing times. During testing it appeared very useful that the parameters could be changed during runtime.

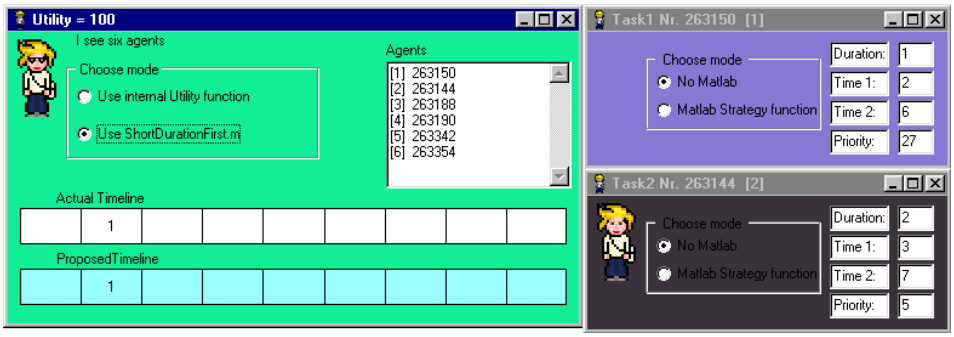

Figure 1. Left: the evaluator agent. Right: above an inactive task agent; below the currently active agent.

7. Conclusions

Matlab-enabled agents are very well suited for use in intelligent multi-agent systems, especially in a research environment. The researcher does not need to program in Java, but has all Matlab functionality directly at his disposal. The system was successfully tested for several agent behaviors.

References

[1] E. Cortese, F. Quarta, G. Vitaglione and P. Vrba. Scalability and Performance of the JADE Message Transport System. Analysis of suitability for Holonic Manufacturing Systems, *Exp – in search of innovation*, Volume 3, n3, September 2003

[2] Hernán Moraldo. Faster File Access With File Mapping, *http://www.flipcode.com/tutorials/tut_filemapping.shtml*

[3] Matlab technical documentation. See e.g. *www.ece.osu.edu/matlab/techdoc/ matlab_external/ch07cl17.html*

[4] R. Thaens. Sensor scheduling using Intelligent Agents. *Proceedings of the 7th International Conference on Information Fusion*, pp. 190-197, July 2004.

[5] S. Kraus. Strategic Negotiation in Multiagent Environments. *MIT Press*, USA, ISBN 0-262-11264-7, 2001.

Appendix

In this Appendix some very simple m-files are shown to illustrate the simplicity of the method. In reality these function can be as complicated as needed for the task at hand. Note that it is possible to hide the contents of m-files (algorithms as well as variable names, parameter values et cetera) by replacing the m-files by preparsed code (using the Matlab-function *pcode*); the resulting binary p-files will be used by the agents instead of the corresponding m-files.

```
function X = CalculateUtility(i)
%% Example file of a utility function
%% This function favors short processing times
%% i specifies the logical number of the agent
%% Use with Evaluator Agent only; do NOT use with Task Agents

global Task
X = round(100/Task(i).Duration)
```

Figure A.1 Example of a Matlab m-file to calculate the Utility

```
function X = IgnoreFinalTime(Nr)
%% Example file for strategy of a Task Agent
%% This Agent neglects t_b
%% Nr specifies the logical number of this agent (you need not know
%% this number; it is generated by the Agent program automatically)
global Task ActualTimeLine
t0 = -1
for t = Task(Nr).t_b : size(ActualTimeLine, 2) - Task(Nr).Duration + 1
   t0 = t;
   for d = t : t + Task(Nr).Duration - 1
      if ActualTimeLine(d) > 0 t0 = -1; break; end;
   end;
   if t0 > 0 break; end;  %% ready: empty place found
end;
if t0 < 0 t0 = Task(Nr).t_b; end; %% no place available
X = t0;
```

Figure A.2 Example of a Matlab m-file to specify a certain strategy of a task agent