Performance Evaluation of Dynamically Assembled Multiagent Systems
Glenn Jayaputera, Seng Loke, Arkady Zaslavsky
CSSE, Monash University, Australia

Abstract
Most methodologies for developing software agents focus at design phase. The proposed notion of
dynamically generating multiagent systems (MASs) from user-defined specifications (called "missions" in our
approach) is somewhat unorthodox and introduces a new paradigm in MASs development and usage, concentrating
on run-time phase. In the dynamic MASs agents are created on-demand at run-time with different
functionalities depending on the objectives of the mission. In this paper, we propose and demonstrate a way for
evaluating MASs at run-time. Performance evaluation of eHermes - the dynamic MAS generator/constructor - is
then presented and analyzed.

1. Introduction
The introduction of a generic assembly and execution shell for multiagent systems in [6] is the first stepping
stone towards a new and promising notion of dynamic MAS. Our concept of having generic and dynamically
constructed MASs came from our observation that current MASs are too specific to particular problem domains
because the agents working in such systems are too specialized. In other words, those agents are good in what
they are designed for but demonstrate little flexibility and adaptability. While such systems might be adequate for
their purposes, they do not fit well with real world environments where conditions can change unpredictably.
Hence, such traditional fixed behavior MASs are becoming less desired.

Building MASs are certainly more complicated than building legacy software systems because now the
developers have to pay attention to additional aspects such as the agent communication language, intelligence,
coordination, security, learning, coalition, negotiation, mobility and so on. Tools to help those developers to
reduce the development complexity and time are springing up, for example, ZEUS [9], JADE [1] and PAUL [2].
While such tools have certainly achieved their goals in helping the developers, they do not make the output (i.e.
the MAS themselves) adaptable. Hence, when the system cannot tolerate the changes in the environment, then that
system must be stopped, reprogrammed and/or modified, recompiled, and re-executed.

One of the proposals to address this challenge is to have intelligent agents with abundance capabilities and are
able to perform organizational structuring to exchange roles within the community [3]. Our approach to this
challenge is to have MASs that can change their behaviors quickly at run-time, or a way of quickly (re-)generating a
MAS (or parts thereof) according to the needs of the moment and on-demand.

In order to generate an agent, one must have an objective/goal for that agent. Hence, our hypothesis is that
in order to dynamically generate MASs at run-time the overall purpose (or mission) of those systems must be
concisely expressed. That is, the agents are not generated arbitrarily but in accordance with the mission specified.

In our model, a multiagent system does not have any prior knowledge about what to solve at the start but when
given a mission, it is able to work out a means to achieve the goal of that mission efficiently. Thus, the number of
agents constructed and their functionalities will depend on the complexity of the given mission.

In this paper, we present the formal model of our mission model and the results of the performance
evaluation tests on eHermes, our prototype system. The rest of the paper is organized as follows. Section 2
presents the formal model of the mission concept. Section 3 briefly explains the conceptual view of eHermes and
discusses its major components. Section 4 presents the performance evaluation that we conducted. Finally, we
conclude in Section 5 by presenting future work.

The notions of mission-based MAS and just-in-time agent generation have been proposed in [4]. General
concept and discussion on those notions can be found in those papers. In this section, we briefly present the formal
model as background for readers.

In our model, a mission is defined as a tuple of the form \( M = (g, P, A, Z) \) where \( g \) denotes the goal of the
mission (a string), \( P \) denotes a set of plans, \( A \) denotes a set of agents working in the mission, and \( Z \) denotes a set of
mission states.

A plan is represented in a DAG-based structured called \( TDG \) (Task Decomposition Graph). The TDG is defined
as a pair \((T, L)\) where \( T \) is a set of tasks that must be executed and \( L \) is a set of links between those tasks.
A task is defined as a tuple of the form \((u, n, y, s, o)\) where: \(u \in U\), where \(U\) is a set of unique IDs, \(n \in N\), where \(N\) is a set of locations at which tasks are executed, \(y \in Y\), where \(Y=\{\text{primitive}, \text{compound}\}\) denotes the task’s types, \(s \in S\), where \(S=\{\text{completed}, \text{pending}, \text{inprogress}, \text{failed}, \text{aborted}, \text{assigned}\}\) denotes a set of task status, and finally \(o \in O\), where \(O\) is a set of functions and/or logic calculations embedded in the task.

Links in the TDG represent the relationships between tasks. A link is defined as a tuple of the form \((t_i, t_j, q)\) s.t. \(t_i, t_j \in T\), \(q \in Q\) and \(Q=\{\text{includes}, \text{dependson}\}\). The \text{includes} attribute is used to capture the inclusion relationship between tasks. The \text{dependson} attribute is used to describe the dependency between tasks. Hence, a link \((t_i, t_j, \text{dependson})\) means that \(t_j\) depends on \(t_i\), as such that \(t_i\) must be completed before \(t_j\) can start. Since a TDG can be potentially large in size, then a precaution must be taken in order to prevent the system from performing exhaustive graph search every time it scans the graph looking for the next task to be executed. Our approach to this issue is as follows: (a) only execute the primitive tasks, (b) compound tasks are converted into primitive tasks once all their direct sub-tasks have been completed, and finally (c) disallow any dependencies between primitive tasks.

Given \(f_{\text{status}}: T \rightarrow S\), a function that returns the status of a task, a mission is accomplished when all the tasks are completed. Hence, \(\forall t \in T \ s.t. f_{\text{status}}(t) = \text{completed}\).

When a mission is executed, the system actually executes the tasks specified in the current plan of that mission. Indeed, the task execution is actually conducted in the strata fashion. A stratum is defined as \(\{t \in T : f_{\text{type}}(t) = \text{primitive} \land f_{\text{status}}(t) = \text{pending}\}\), that is, a set of pending primitive tasks.

Through the mission states, our model allows a mission to be suspended and resumed, as well as moved to a different host/location at any time. The execution can be continued at the new location. A mission state is defined as a tuple of the form \((V, R, e)\) where \(V \subseteq T\), \(R\) is a set of states and data of the tasks in \(V\) and \(e\) is the event that caused the transition. \(r \in R\) is defined as a tuple of the form \((t, d, w)\) where \(t \in V\), \(d = \text{date}\), \(w = \text{value}\). \(E\) is a set of possible events and defined as \(E=\{\text{start}, \text{stop}, \text{suspend}, \text{resume}, \text{modify}\}\) and \(e \in E\).

Creating one agent for each task in a plan is certainly a non-optimized approach to execute the plan. A better approach is to create as few agents as possible with minimal impact to the mission’s critical time. A critical time of a mission is defined as the minimum amount of time required to accomplish the mission. Minimizing the number of agents to work on a mission means we are minimizing the cost of executing that mission. The cost of executing a mission is defined as \(\sum_{t \in T} f_{\text{cost}}(t), \forall t \in T\) and the cost of executing a task is \(f_{\text{cost}}(t) = f_{\text{cost}}(\text{createAgent}(t)) + f_{\text{cost}}(\text{exec}(t))\). In our model, we assume that the cost for executing a task is constant at any host, ignoring the locality and computing power of that host. Furthermore, we ignore any cost that is associated with agents’ mobility. Hence, given such assumption, the only parameter that one can fine-tune to reduce the overall cost is the cost of creating the agents.

Recall that the plan is executed by strata and hence logically, the agent count reduction should be carried out at each stratum. The tasks in each stratum are executed in parallel. Given \(f_0(t) \in \mathbb{Z}, t \in T\), a function that returns the execution elapsed time of task \(t\), then the critical time of a stratum \(ST\) is defined as \(\max\{f_0(t_i) : t_i \in ST, i \in \mathbb{Z}\}\). That is, the minimum amount of time required to complete all the tasks in stratum \(ST\) is equal to the maximum elapsed time of the tasks in \(ST\). Therefore, by allocating as many tasks (from the stratum \(ST\)) to each agent, and provided the elapsed time of these agents does not exceed the critical time of stratum \(ST\), then the number of agents can be reduced.

Optimizing task allocation to the agents is an off-line bin packing problem and hence finding the optimal solutions to this problem would be NP-Hard. However, there are a number of near optimal solutions which have been proposed and the best one is FFD (First-Fit Decreasing) [8]. Using FFD algorithm, the tasks in each stratum are sorted in decreasing order according to their \(f_0\) values. Hence, the task that needs the longest time to complete will be the first and the shortest will be the last. The size of the bin is then the value of \(f_0\) of the 1st task (\(f_0(t_1)\)). The tasks from this stratum are then grouped into clusters where the sum of \(f_0\) of each clusters is less than or equal to \(f_0(t_1)\). Given \(n\) as total number of items to be bin-packed, FFD can be easily implemented in \(O(n \log n)\) time [8].

3. eHermes: The Dynamic MAS Generator

eHermes is the prototype tool being developed as the proof of concept for generating dynamic MASs. The conceptual view of eHermes has been presented in [6] and its architecture can be found in [5], hence we do not intend to present rigorous discussion in this paper.

eHermes has four major components, they are:

- **Mission Generator**, the component for generating the mission objects.
- **Mission Planner**, the component for generating the plan.
• Agent Generator, the component for assembling agents on demand.
• Elvin [10], the communication backbone for all the agents in eHermes.

A mission is produced once a request is received by eHermes, and the request can come from either human users or other MASs. When constructing a mission, the Mission Generator uses the Mission Planner to generate the best possible initial plan. Finding the best planning algorithm is beyond the scope of this project and we assume that the planner can generate a plan within a reasonable time. The Mission Planner does not always generate new plans for every request, but can reuse some of the existing plans if available. Once the mission object is ready, a group of agents (stationary or mobile) are generated on the fly to carry out the mission according to the plan(s) embedded in the object itself. The agents are assembled by the module called Agent Generator. Components of these agents come from a repository called Agent Component Repository (ACR).

A special agent called Mission Control Agent (MCA) is created for each running mission. MCA is in charge in monitoring the mission execution. This includes tasks such as controlling when the agent must be created, coordinating and orchestrating the task allocation, and mediating between the user and the system.

eHermes’ agents communicate with each other using content-based short messages as opposed to speech act dialogues. This is because of the mobility nature of these agents. When these agents are on the “not-so-reliable” wireless network, to having “quick conversations” are desirable. Secondly, given the master-slave organization structure of the agents, there is little need to have lengthy conversations such as negotiation or bargaining type of conversations. Elvin is used as the backbone for eHermes communication sub-system because it is fast and can be used directly due to its Java classes. Other content based messaging system can be used if available.

eHermes is being implemented on top of Grasshopper agent toolkit [7], which is a Java™ based mobile agent toolkit. Grasshopper was chosen to provide the underlying mobile agent infrastructure while we focus on the implementation of our concepts.

4. Experimentation

One of the experiments we have conducted is the experiment on the optimization strategy we use. We investigate how such strategy performs on different missions with different plans (and hence their complexities). The metric we use to measure the plan’s complexity is the number of tasks specified in the plan. The plans’ complexity is specified by two parameters, $m$ and $n$. Parameter $m$ is used to set the depth of the TDG while parameter $n$ is used to set fan-out factor of each compound task in the TDG. Hence, when the value of $n=1$ then the TDG becomes a sequence of tasks (for $m>1$). In this experiment, each agent is assigned with a task to perform a data retrieval and computational logic. Table 1 presents the result of the experiment.

<table>
<thead>
<tr>
<th>Test #</th>
<th>Plan Details</th>
<th>No of Agents Generated</th>
<th>Agent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1024</td>
<td>1</td>
<td>1025</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2</td>
<td>1023</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>1093</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1365</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1555</td>
</tr>
</tbody>
</table>

Table 1: Optimization Test Results

As expected, the results on Table 1 shows that the optimization strategy does not give any impact when the plan structure is linear ($m=1$, $n=1$). This is because the tasks can only be carried out sequentially. However, when the value of $n$ is increased we see that the number of agents generated is reduced by around 42%. We also see a direct relationship between the fan-out factor and agent count reduction, in that when the value of $n$ is increased the percentage of agent reduction is also increased. This pattern is shown in Figure 1.

While the test on the optimization strategy is encouraging, however, it is important to measure the mission elapsed time as well. For the approach to be successful it must not affect the mission elapsed time too much when the number of agents is reduced. Figure 2 shows the comparison of the mission elapsed time with

![Agent Reduction Graph](image1)

![Mission Elapsed Time Comparison Graph](image2)
and without optimization. It shows that the optimization strategy does not affect the mission elapsed time at all.

5. Conclusion and Future Work

We presented our concept of the on-demand agent generation methodology. This methodology is the pivot idea to achieving dynamic MASs. We have presented the formal model the mission and it’s plan structure. We have discussed our strategy to execute the plan and our optimization approach in reducing the number of agents generated for a given mission. The benefits of this approach are:

1. **Agents are determined by the run-time needs of a mission.** Hence, the agents’ functionality and their quantity always match the given mission.

2. **Flexibility, adaptability and robustness.** Agents can be regenerated as the environment changes. The failure of the mission execution is also minimized because those not so capable agents can be replaced at any time. Hence, our approach is robust.

3. **A new abstraction in MASs.** Our approach provides a separation between the goal and the tools to achieve the goal. The tools are the agents themselves, and tools are disposable items. This view contrasts with the traditional MAS view where such separation does not exist – the overall mission of the MAS is emphasized and explicitly represented in our view.

From our experiments, the following conclusions are drawn:

- The cost/benefit analysis works well in this system. It shown that the number of agents generated is reduced as long as the plan structure is not linear and fans out. The result also shows a trend that the extent of the reduction increases as the number of subtasks of the compound tasks increases.

- The test also shows that the mission elapsed time does not increase as the number of agents working for the mission decreases.

Future work for this project includes:

- Dynamic load distribution. It is necessary to investigate a possibility to reduce the MCA load by creating clones and assigning parts of the plan to these clones. This strategy is useful especially when part of the plan must be executed remotely. Further investigation is needed to proof the benefit of doing so and subsequently when and how to split up a plan into sub-plans.

- Currently, the optimization process only considers tasks within the stratum boundary (horizontal boundary). However, further investigation and study is needed to find out if better results can be obtained when the vertical boundary is considered.

- All the tests conducted so far are carried out on a LAN with typical network loads and hence the results might be biased toward the university network environment. Therefore, further tests must be conducted on the real world LAN and WAN to see if both results have the same pattern or not.

Acknowledgements

Support for this project from Australian Research Council (ARC) Linkage Project Scheme and Microsoft Research Asia Grant LP0211384 is thankfully acknowledged.

References


