Just in Time Mobile Agent Generation and Management

G. Jayaputera, S. W. Loke, A. Zaslavsky
School of Computer Science and Software Engineering, Monash University
900 Dandenong Road, Caulfield East, Victoria 3145, Australia
+61 3 9903 2787
{glenn.jayaputera, seng.loke, arkady.zaslavsky}@infotech.monash.edu.au

ABSTRACT

This paper presents a new and innovative approach called mission-based just in time agent generation. The approach allows agents to be constructed on the fly, at run-time and just when they are needed. This is a completely different approach to the traditional way of creating MAS in that agents are constructed at design time. We present the theoretical work behind the notion as well as an experimental result and future work.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent Agents.*

General Terms

Algorithms, Design, Theory.

Keywords

TDG, Mission, eHermes, just in time, on demand, dynamic MAS.

1. INTRODUCTION

In this paper, we present an innovative and novel approach in building dynamic MAS. Different from the traditional approach where the agent developers pay a lot of attentions on concerns such as how each agent might communicate or cooperate with another, how it might negotiate or coordinate, learn, etc., this new approach allows the developers to very much concentrate on task or problem level semantic such as the business logic for the whole application. Thus our approach adds a new level of abstraction focusing on the purpose of the application instead of the process of creating the agents for the application.

We refer our approach to as the *mission-based just in time agent generation*. This approach facilitates the ability of the system to generate a set of agents (mobile or stationary) on-demand. The on-demand notion here is referring to the timing of the agent generation; hence it means the agents are generated just when they are needed to perform some actions. Naturally they are removed from the system when their services are no longer required. Therefore, our approach guarantees that there are no "jobless" agents but only the useful ones are kept in the system.

2. MISSION: THE FORMAL MODEL

Software agents are constructed because they need to perform some actions; that is they have a *mission* to accomplish. Therefore, mission is one of the corner stones of our vision. Various quantities of agents where each of them has its own specialty are constructed depending of the mission in hand. A mission is formally modeled as a tuple of the form $M = (g, P, A, Z)$, where g denotes goal of the mission; in string, P denotes a set of plans, A denotes a set of mobile and/or stationary agents working on the mission and finally, and Z denotes a set of mission states.

A plan $p \in P$ is as a DAG-based structure called *TDG (Task Decomposition Graph)* and is defined as a pair *TDG=(T,L)*, where *T* is a set of tasks and *L* is a set of links between the tasks.

A Task is defined as a tuple of the form *(u,n,y,s,o)*, where $u \in U$; *U* is a set of unique IDs, $n \in N$; *N* is a set of locations at which tasks are executed, $y \in Y$; *Y={primitive, compound}* denotes the task type, $s \in S$; *S = {completed, pending, inprogress, failed, aborted, assigned}* denotes the task status, and finally $o \in O$; *O* is a set of functions and/or logic calculations that must be performed.

A link is defined as a tuple of the form (t_i, t_j, q) ; $i, j \in \square^+, t \in T, q \in Q$ and *Q={includes, dependson}* is the link attributes. The *includes* attribute is used to capture the inclusion relationship between tasks. The *dependson* attribute is used to describe the dependency between tasks. In TDG, primitive tasks are atomic tasks and cannot depend on any other tasks

Given a function $f_{status} : T \to S$ which returns the status of the given tasks, a mission is called *accomplished* when all the tasks are completed, hence: $\forall t \in T$ s.t. $f_{status}(t) = completed$.

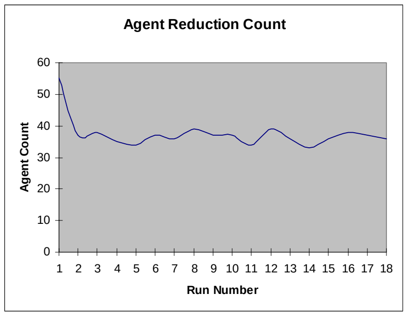
A mission state is defined as a tuple of the form *(V,R,e)* where $V \subseteq T$, *R* is a set task states and data defined as tuple of the form $(t, d, w), t \in V, d = date, w = value$ and finally, $e \in E$; *E* defined as *E={start, stop, resume, suspend, modify}* is a set of events that caused the transition. When a mission is executed, the system actually executes the tasks specified in that plan. However, the system does not execute the tasks directly, rather through a set of agents. The number of agents generated depends on the number of tasks that needs executed at the time. Tasks are executed in the stratum by stratum fashion. Given a function $f_{type} : T \to Y$ which returns the type of a task, a stratum *ST* is defined as: $ST = \{t \mid t \in T, f_{type}(t) = primitive \wedge f_{status}(t) = pending\}$.

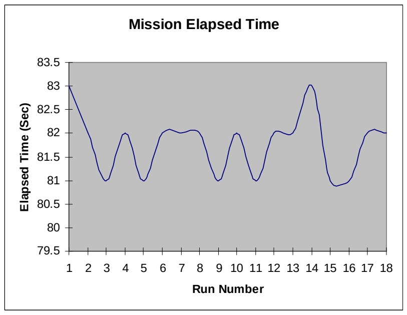
A compound task is changed into a primitive task when if all its direct sub tasks are completed. Once it becomes a primitive task then it can be executed in the next of execution cycle. Agents are not generated for each task in a stratum because it is not cost efficient. Instead, an optimization is carried out where the system tries to reduce the number of agents generated without compromising the mission execution's elapsed time. Let $f_{\cos t}(t) = f_{\cos t}(createAgent(t)) + f_{\cos t}(execute(t))$ be the function that calculates the cost for executing a task. For simplicity we assume that the cost for executing a task is constant ignoring factors such as the host locality and computing power. Therefore, optimization is achieved by reducing the cost of creating the agents. The cost of executing a mission is then defined as: $\sum_{i=1}^{|T|} f_{\cos t}(t_i), \forall t_i \in T$.

A critical time (CT) of a stratum ST is the defined at the minimum amount of time required to accomplish all the tasks in ST. If the tasks in ST are executed concurrently and given $f_\Theta(t) \in \square^+, t \in T$; a function that returns the elapsed time of task t, then the critical time of ST is defined as $CT_{ST} = \max\{ f_\Theta(t_i) \mid t_i \in ST, i \in \square^+ \}$. Executing a mission can be regarded as executing the strata in the total order fashion, hence ST_i can only be executed once ST_{i-1} is completed. Given such condition and the fact that the strata are determined at run-time, the mission execution optimization must be conducted at the strata level and dynamically at run-time. Optimization is performed by allocating as many tasks as possible to an agent providing the elapsed time of a stratum does not exceed its critical time. This type of optimization is classified as an off-line bin-packing problem. Finding an optimal solution to this class of problem has been proven to be an NP-Hard problem and hence we use a near optimal solution; FDD (First-Fit Decreasing) [1]. The tasks in each stratum are sorted in a decreasing order based on their f_Θ value. The size of the bin is then $f_\Theta(t_1)$. The tasks are then separated into clusters where the total value of f_Θ in each cluster is less or equal to $f_\Theta(t_1)$, that is $\sum_{i=2}^{n} f_\Theta(t_i) \le f_\Theta(t_1)$.

3. EXPERIMENTAL RESULT

One of the experimentations we have conducted is the test to see how well the optimization strategy works. Our concern is whether the reduction of the number of agents working in the mission affects the elapsed time of the mission itself. In this test, the system is given a mission that comprises of 55 tasks. The system runs the mission 18 times. In the 1st run, the optimization is turned off while in the subsequent runs it is turned on. In each run we measure the mission execution elapsed time and number of agents generated. From the graph in Figure 1, it shows that the number of agents is successfully reduced from 55 agents down to the minimum of 13 agents in run no 14. The graph in Figure 2 shows the elapsed times of each run. From this graph it can be seen that the reduction of the number of agents does not affect the mission completion time. These graphs have show us that the optimization strategy via cost/benefit analysis that we used in the system has achieved its goal, that is, reducing the number of agents working in the mission with not too much affect on the mission elapsed time.





4. CONCLUSION AND FUTURE WORK

We have presented an innovative and novel notion we called mission-based just in time agent generation. Moving away from the traditional style of constructing agents, our approach emphasizes on the timing of agent generation. That is, only generate the agents just when they are needed and remove them when their service is no longer needed.

Some of the benefits of this approach are: (i) *the dynamic creation of MAS*. Now MAS do not need to be constructed at design time, rather dynamically created at run-time, (ii) *right amount of agents with appropriate type of skill sets*. Agents in the MAS are guaranteed to have the skill sets needed to achieve the mission, (iii*) an ability to stop, suspend, resume and move a mission to any host*, (iv) *adaptivity and robustness*. Since the agents can be dynamically created then incapable agents can be replaced at run-time with the capable ones.

Future work includes: (i) task reallocation enhancement where agents are not destroyed once they have completed their assignment, rather reused. These agents are given a new set of tasks with a new set of skills needed to complete the new tasks; (ii) since the TDG can be potentially large then there is a need to be able to divide it into sub-TDGs. These sub-TDGs will be managed by local agents who will report to one who control the overall mission execution. Furthermore, sub-TDGs can be transferred and executed remotely.

ACKNOWLEDGEMENT

Support from the ARC Linkage Grant LPO211384 and Microsoft Research Asia is thankfully acknowledged.

5. REFERENCES

[1] H. Kellerer, U. Pferschy and D. Pisinger, *Knapsack Problems*, Springer, 2004.