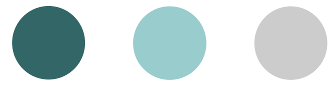


Generating and Supporting Dynamic Heterogeneous MAS

Glenn Jayaputera
Arkady Zaslavsky
Seng Loke
{Glenn.Jayaputera, Arkady.Zaslavsky,}@infotech.monash.edu.au
S.Loke@Latrobe.edu.au

Background

- In general MASs exhibit the following properties
  - The agents within software systems have fixed functionality assigned at design phase
  - The number of agents in the system is fixed
- Issues related to developing and running MAS:
  - Agent expertise requirement: agent communication languages, coordination, autonomy, etc.
  - Lacking of run-time support.
  - Uncertainty with the number of agents in the MASs.

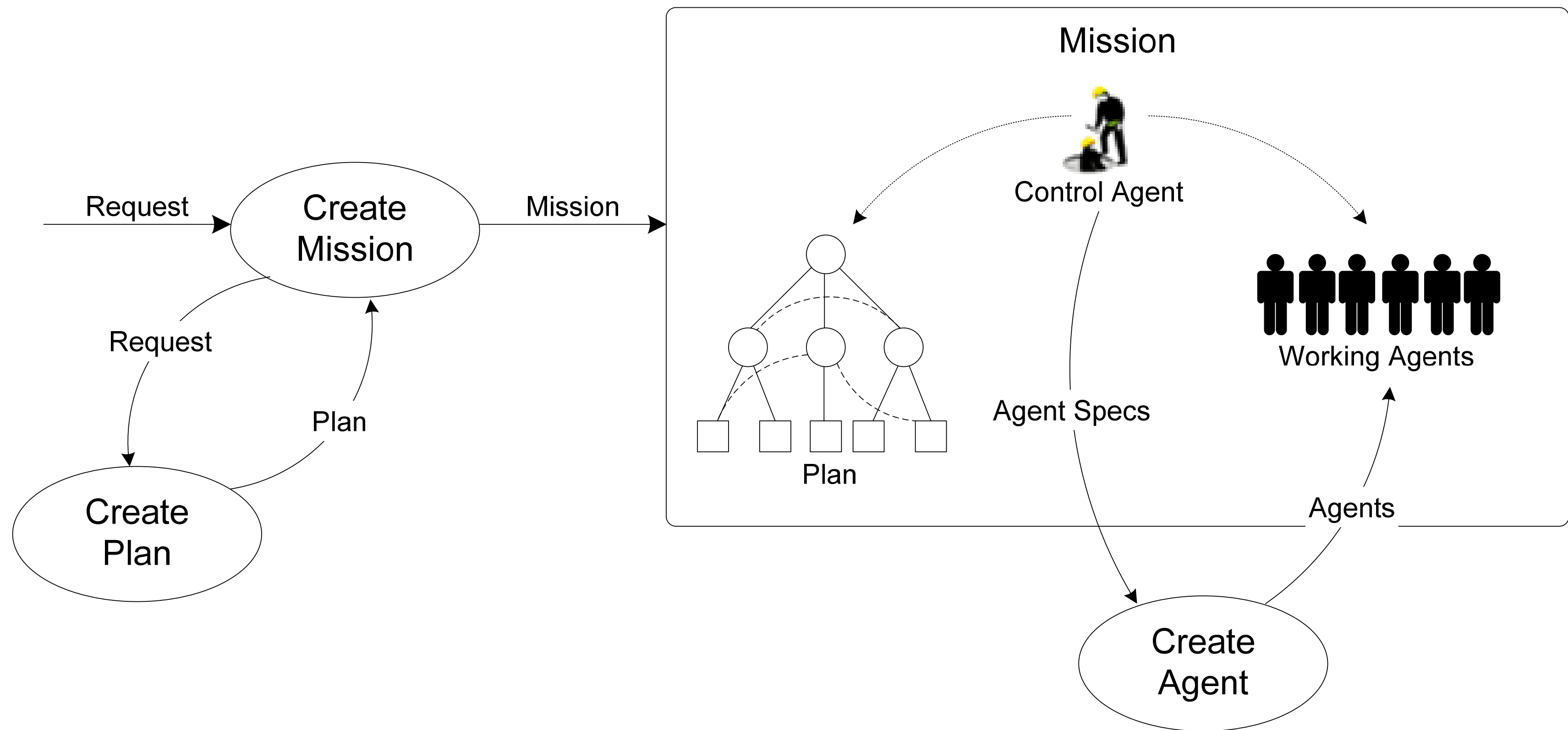
Motivation

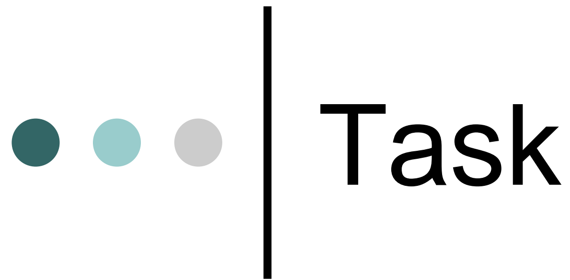
- MASs have to be flexible and dynamic
  - Agents are created on-demand based on a given plan
  - Agents are assigned one or more tasks at run-time.
- Run-time support is an important element of any MASs.

Objectives

- *Adding a new level of abstraction which focuses on the whole application (MAS) rather than individual agents.*
- Emphasizing automatic generation of agents when needed (on-demand).
- Providing run-time support framework and mechanisms:
  - Execution Management Mechanism.
  - Plan Modification Mechanism.

Proposed System's Conceptual view





Task

A task is defined as a tuple of the form (u,N,y,s,o),

$$u \in U, y \in Y, s \in S, o \in O$$

where:

U is a set of unique IDs which are strings,

N is set of locations at which the tasks must be carried out,

Y is a set of task types, i.e. Y = *{Primitive, Compound}*,

S is a set of task statuses, i.e.

S = *{Completed, Pending, InProgress, Failed, Aborted, Assigned}*,

O is a set *of functions (which represent the application logics) that a task will execute*

Link

A link is defined as a tuple of the form

$$(t_i, t_j, q),\ i, j \in \mathbb{Z}^+, q \in Q, i \neq j, t_j, t_j \in T$$

where:

Q type of the relationship, and

$$Q = \{Includes,\ DependsOn\},$$

T is a set of tasks as defined previously.

Task Decomposition Graph (TDG)

- A plan is represented in a DAG (Directed Acyclic Graph)-based structure called Task Decomposition Graph (TDG).
- TDG is a pair of sets of tasks and links, hence TDG=(T, L), with the following properties:

Property 1

$$\forall (t_i, t_j, q) \in L, f_{type}(t_i) \neq Primitive$$

Property 2

$$\forall (t_i, t_j, q) \in L, \text{ if } q = Includes \text{ then } f_{type}(t_i) = Compound$$

Property 3

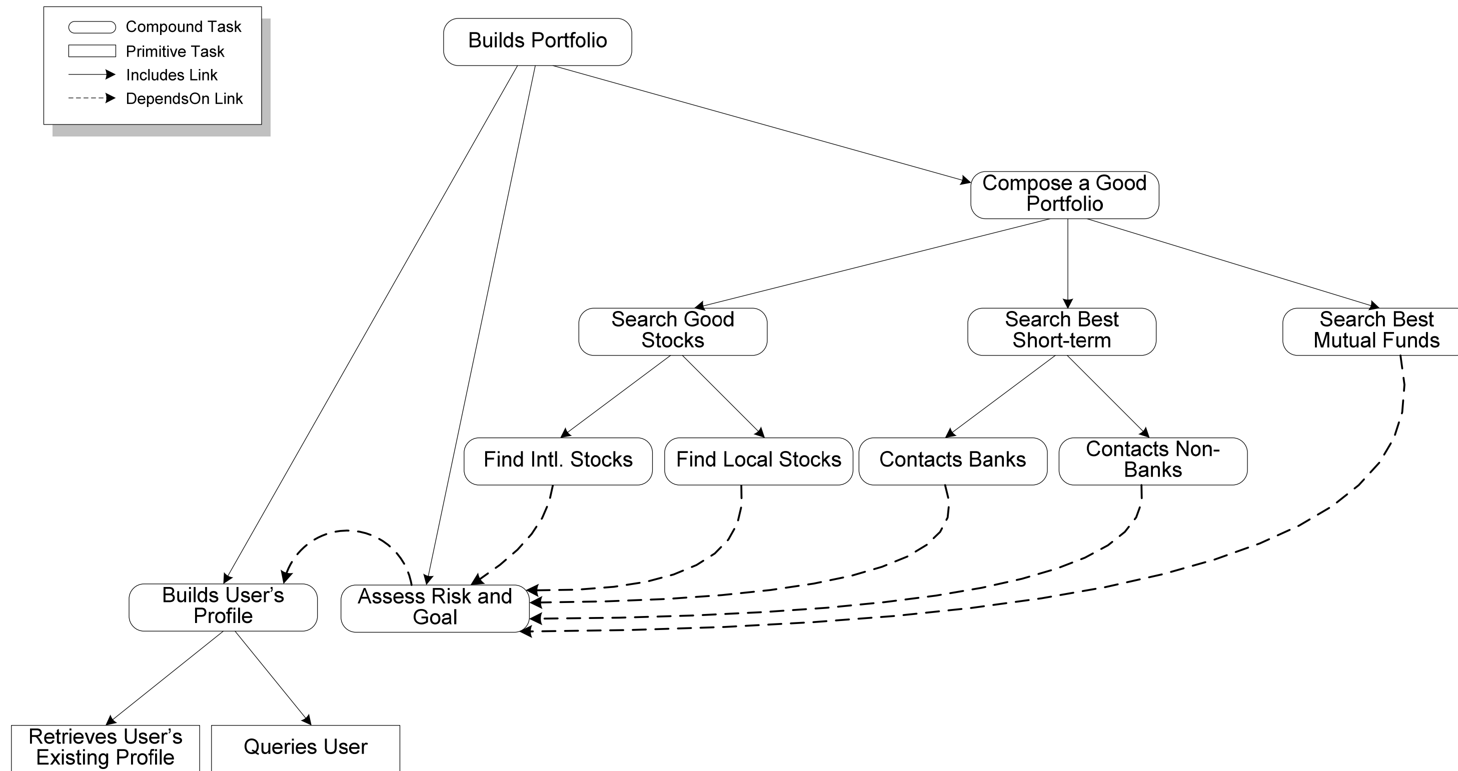
$$\forall (t_i, t_j, q) \in L, \text{ if } f_{type}(t_j) = Compound \text{ then } \exists (t_j, t_k, q') \in L$$

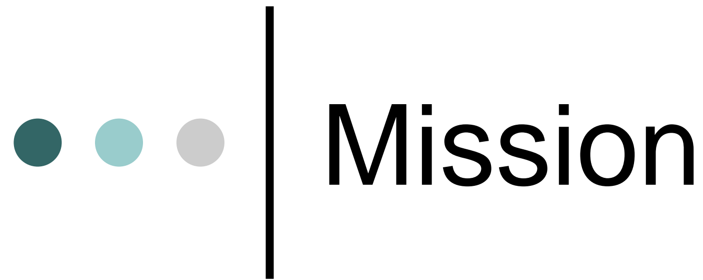
where

$$t_i, t_j, t_k \in T,\ q, q' \in Q \text{ and } i, j, k \in \mathbb{Z}^+, i \neq j \neq k$$

$f_{type} : T \rightarrow Y$ is a function that returns the type of a given task

TDG – Cont’d





Mission

A mission is a tuple of the form (o, P, A, Z), where:

o	is the objective/goal of the mission, a string,
P	is a set of plans/TDGs,
A	is a set of agents (mobile and/or stationary), and
Z	is a set of mission states.

Mission Execution

- Mission is executed by executing the embedded plan.
- Plan is executed by executing the tasks specified in the plan.
- Plan's structure is dynamic and evolving as the mission is executed.

$$PE : p_0 \xrightarrow{fx_0} p_1 \xrightarrow{fx_1} p_3 \xrightarrow{fx_2} \ldots \xrightarrow{fx_n} p_n$$

where

$P = \{p_0, p_1, ..., p_n\}$ is a set of plans

$F = \{fx_0, fx_1, ..., fx_n\}$ is a set of operations applicable to P

Mission Execution (cont'd)

$f_{setTaskStatus}(p, t, s')$	Set/change the task's status
$f_{abortTask}(p, t)$	Abort a task
$f_{prunePlan}(p)$	Prune/clean-up the plan
$f_{mutateTasks}(p, \mu_T)$	Mutate/change tasks' type
$f_{executeNaive}(m)$	Non cost-efficient mission execution
$f_{executeOptimized}(m)$	Cost-efficient mission execution
$f_{modPlan}(p, \Delta)$	Plan modification

Set Task Status Operation Algorithm

1 $f_{setTaskStatus}(p, t, s')$
2 {
3 // Recall $p = (T, L)$
4 if $(s \neq s')$ then
5 {
6 $t_x := (u, N, y, s', o)$ // Create a new task t_x
7 // Create a set of "to be removed" links
8 $\delta_L^- := \{(t_i, t_j, q) \in L \mid t_i = t \vee t_j = t\}$
9 // Create a set of "to be added" links
10 $\delta_L^+ := \{(t_x, t_j, q) \mid (t, t_j, q) \in \delta_L^-\} \cup \{(t_i, t_x, q) \mid (t_i, t, q) \in \delta_L^-\}$
11 $T' := T \setminus \{t\} \cup \{t_x\}$
12 $L' := L \setminus \delta_L^- \cup \delta_L^+$
13 }
14 return (T', L')
15 } // End of $f_{setTaskStatus}(p, t, s')$

Abort Task Operation Algorithm

1 $f_{abortTask}(p,t)$
2 {
3   // Check to see if t has any dependents. Recall $p=(T,L)$
4     $DLSet := \{(t_i,t_j,q) \in L \mid t_j = t \wedge q = DependsOn\}$
5   if $(DLSet \neq \varnothing)$ then
6   ; //Stop traversing and do nothing
7   else
8   {
9   // Get all sub-tasks of task t
10         $ILSet := \{(t_i,t_j,q) \in L \mid t_i = t \wedge q = Includes\}$
11   // For each of these subtasks we will abort it recursively
12         $\forall (t_i,t_j,q) \in ILSet$ do
13             $f_{abortTask}(t_j);$
14         $f_{setTaskStatus}(p,t,Aborted);$
15   }
16 }

Mission Execution (cont'd)

Plan Pruning Operation Algorithm

1 $f_{prunePlan}(p)$
2 {
3 // Get all the aborted tasks from the current plan
4 $T_\delta = \{t \in T \mid f_{status}(t) = Aborted\}$
5 if $(T_\delta \neq \emptyset)$ then
6 {
7 // If there are task to be removed then get all the links
8 // associated with these tasks
9 $L_\delta = \{(t_i, t_j, q) \in L \mid t_i \in T_\delta \vee t_j \in T_\delta\}$
10 // Then remove those tasks and links from the plan
11 $T := T \setminus T_\delta$
12 $L := L \setminus L_\delta$
13 }
14 return (p)
15 }

Mission Execution (cont'd)

Task Mutation Operation Algorithm

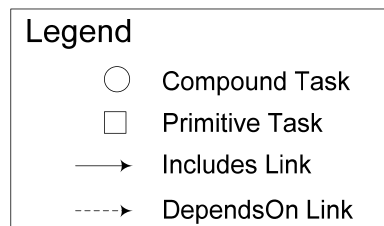
1 $f_{mutateTasks}(p, \mu_T)$
2 // Recall $p = (T, L)$ and
3 $// \mu_T = \{t_i \in T \mid f_{type}(t_i) = Compound \wedge (\forall (t_i, t_j, q), f_{status}(t_j) = Completed)\}$
4 {
5 // Get all the outgoing links from all the tasks in μ_T to any tasks
6 $L_1 := \{(t_i, t_j, q) \in L \mid t_i \in \mu_T\}$
7 // Get all the primitive tasks that the links in L_1 are pointing to
8 $\delta_T^- := \{t_j \mid (t_i, t_j, q) \in L_1\}$
9 // Create similar tasks to all tasks in μ_T but with primitive type
10 $\delta_T^+ := \{(u, N, Primitive, s, o) \mid (u, N, Compound, s, o) \in \mu_T\}$
11 // Get all the incoming links to all the tasks in μ_T
12 $\varphi_L^- := \{(t_i, t_j, q) \mid t_j \in \mu_T\}$
13 // For all the links in φ_L^- create their counterparts. However,
14 // instead of pointing to the tasks in μ_T make them point to
15 // the tasks in δ_T^+
16 $\varphi_L^+ := \{(t_i, t_x, q) \mid (t_i, t_j, q) \in \varphi_L^-, t_x \in \delta_T^+\}$
17 //
18 // Now perform the mutation
19 $T := ((T \setminus \delta_T^-) \setminus \mu_T) \cup \delta_T^+$
20 $L := ((L \setminus L_1) \setminus \varphi_L^-) \cup \varphi_L^+$
21 return (p)
22 } // End Of $f_{mutateTasks}(p, \mu_T)$

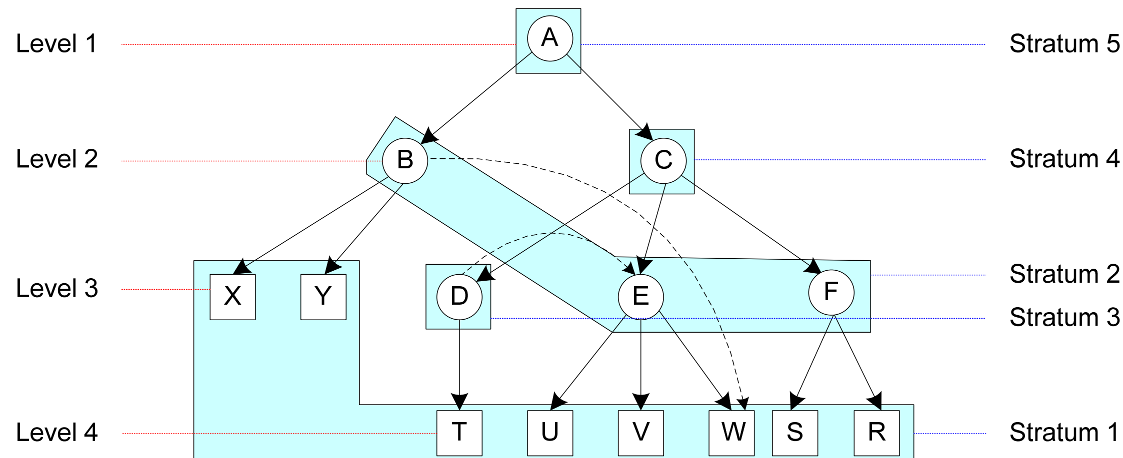
Mission Execution (cont'd)

Non Cost-Efficient Mission Execution

- Tasks in a plan are executed by strata fashion.
- A stratum is defined as a set of primitive tasks whose status is pending.

$$st = \{t \in T \mid f_{type}(t) = Primitive \wedge f_{status}(t) = Pending\}$$





Mission Execution (cont’d)

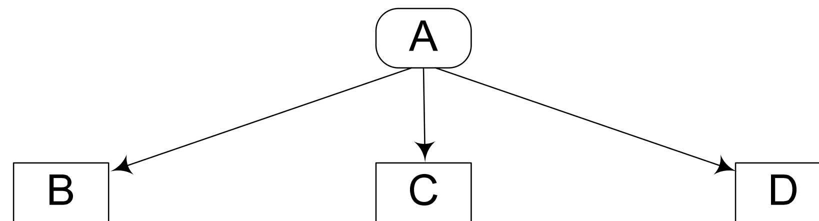
Non Cost-Efficient Mission Execution Algorithm

1 $f_{executeNaive}(m)$
2 {
3   // Recall that $p = (T, L)$
4   $p := f_{getPlan}(m)$
5   // While the mission has not been accomplished and not aborted
6   while $(!f_{isAccomplished}(m) \wedge \, !f_{isAborted}(m))$
7   {
8   // Create a stratum
9     $st := \{t \in T \mid f_{type}(t) = Primitive \wedge f_{status}(t) = Pending\}$
10   // For each task the the stratum
11     $\forall t \in st$ do
12   {
13   // Create an agent for a task
14       $anAgent := f_{createAgent}(t)$
15   // Change the status of the task to become "Assigned"
16       $f_{setTaskStatus}(p, t, Assigned)$
17   // Send an "Execute" command to the agent to perform the task given
18       $f_{sendCommand}(anAgent,$ CMD_EXECUTE)
19   // Record the agent details into AgentActivityTable
20       $f_{putAAT}(anAgent)$
21   }
22   // Go to sleep mode until awakened
23     $f_{sleep}()$
24   }
25 } // End Of $f_{executeNaive}()$

Mission Execution (cont'd)

Cost-Efficient Mission Execution Algorithm

- Creating an agent for each task in a plan is not cost-efficient because the next stratum cannot be executed until the current stratum is completed.



- Assume that the elapsed time of tasks B, C and D are w_1, w2 and w3. Also assume that w1+w2 < w3.
- Task A cannot be executed until task D is completed. Hence creating 3 agents for tasks B, C and D is not efficient.
- A better approach is to create an agent for tasks B and C, and another agent for task D.

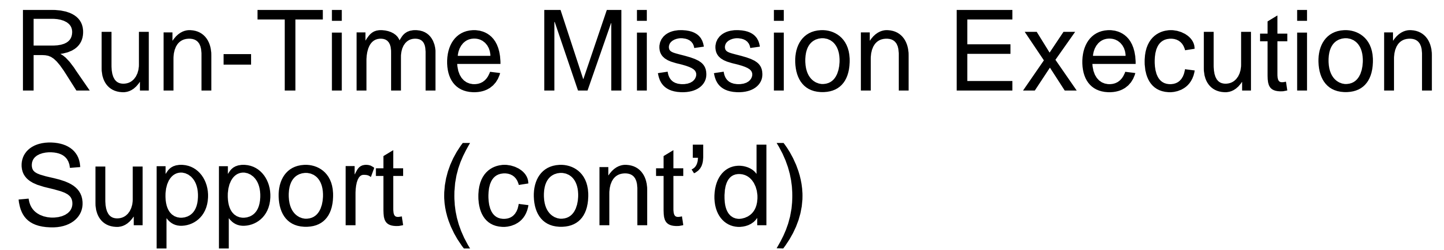
Mission Execution (cont'd)

Cost-Efficient Mission Execution Algorithm

```
1  f_executeOptimized(m)   {
2    p := f_getPlan(m)                                   // Recall that p = (T, L)
3    // While the mission has not been accomplished and not aborted
4    while (!f_isAccomplished(m) ∧ !f_isAborted(m))   {
5      st := {t ∈ T | f_type(t) = Primitive ∧ f_status(t) = Pending}      // Create a stratum
6      st_sorted := f_sort(f_quickSort, f_elapsed, st)   // Create a sorted stratum based on the sorting function supplied
7      // Do the bin-packing routine on the sorted stratum
8      X := f_FFD(st_sorted)                             // X is a set of task clusters
9      ∀x ∈ X do   {                                     // For each task clusters in X do
10       anAgent := f_createAgent(x)                     // Create an agent for a task cluster
11       ∀t ∈ x do                                       // Change the status of the tasks to become "Assigned"
12         f_setTaskStatus(p, t, Assigned)
13       // Send an "Execute" command to the agent to perform the task given
14       f_sendCommand(anAgent, CMD_EXECUTE)
15       // Record the agent details in the AgentActivityTable
16       f_putAAT(anAgent)
17     }
18     // Go to sleep mode until awakened
19     f_sleep()                                         // Time out must be set for f_elapsed(t), where t ∈ x_1
20   }
21 }  // End Of f_executeOptimized()
```

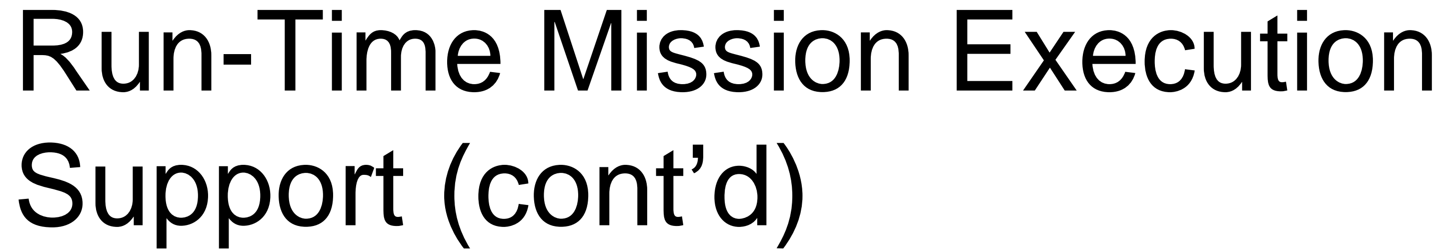
Run-Time Mission Execution Support

- Tasks may or may not produce results/output when executed. Subsequently, these results may or may not be needed by other tasks/agents. Based on this fact, we design a placeholder for agents to exchange data. We call this placeholder Mission Data Space (MDS).
- MDS is defined as a set of tuples of the form (u,r,ts) where:
  - u is a task UID,
  - r is the result produced by u,
  - ts is the timestamp at which r was produced.



Run-Time Mission Execution Support (cont'd)

- It is important to capture and preserve the mission execution history.
- The benefits of capturing and preserving the mission execution history are:
  - the ability to suspend and resume the mission execution at the same or different locations.
  - the valuable knowledge for the mission planner to use in its learning process so that it can generate better plans in the future.



Run-Time Mission Execution Support (cont'd)

Let $E = \{e_1, e_2, e_3, ..., e_k\}, k \in \mathbb{Z}^+$ be the set of events that can trigger transitions, and $MS = \{ms_1, ms_2, ..., ms_l\}, l \in \mathbb{Z}^+$ be a set of mission states.

A mission execution ME is a sequence of interleaved mission states and events such that:

$$ME : ms_1 \xrightarrow{e_1} ms_2 \xrightarrow{e_2} ms_3 \xrightarrow{e_3} \ldots \xrightarrow{e_k} ms_l$$

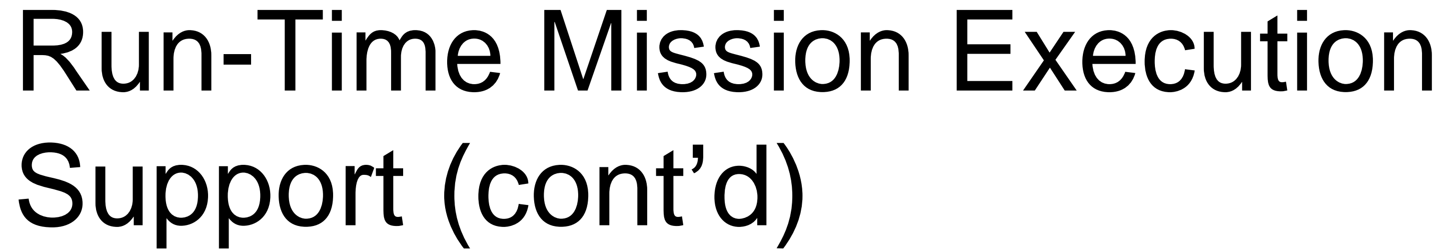
A mission state is a snapshot of the state and data of the mission, and is defined as a tuple of the form (p, mds, e, ts) where:

p is the current plan,

mds is the snapshot of the mission data space,

e is the event that caused the transition, $e \in E$ and $E = \{Suspend, Stop, Resume, Start, ChangePlan\}$ is a set of discrete events that can trigger a transition, and finally

ts is the timestamp at which the mission state is generated.



- Failures may occur during a mission execution even though the plan produced by the planner is the best plan at the time.
- Failures are hard to avoid. Hence, MASs must be adaptable to changes in the environment in order to minimize failures, and hence not to abandon the mission prematurely.
- Our approach to such a challenge is to support run-time plan modifications.

Given a plan $p = (T, L)$, let

Δ be a set of tasks and links that are to be added or removed from p,

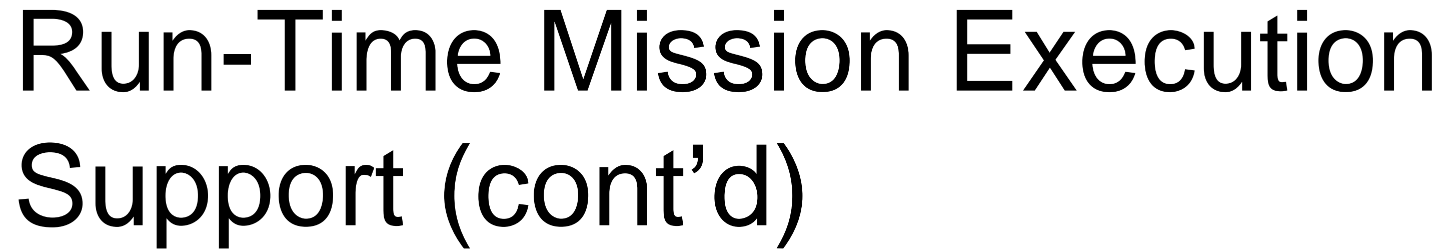
$\delta_T = \{t \mid t \in \Delta\}$ be a set of tasks to be added to or removed from p,

$\delta_L = \{(t_i, t_j, q) \mid (t_i, t_j, q) \in \Delta\}$ be a set of links to be added to or removed from p,

then $f_{modPlan}(p, \Delta)$ is an operation that modifies plan p with the changes stated in Δ according to the following transition rule:

$$(T, L) \xrightarrow{f_{modPlan}(p,\Delta)} ((T \cup \delta_T) \setminus (T \cap \delta_T), (L \cup \delta_L) \setminus (L \cap \delta_L))$$

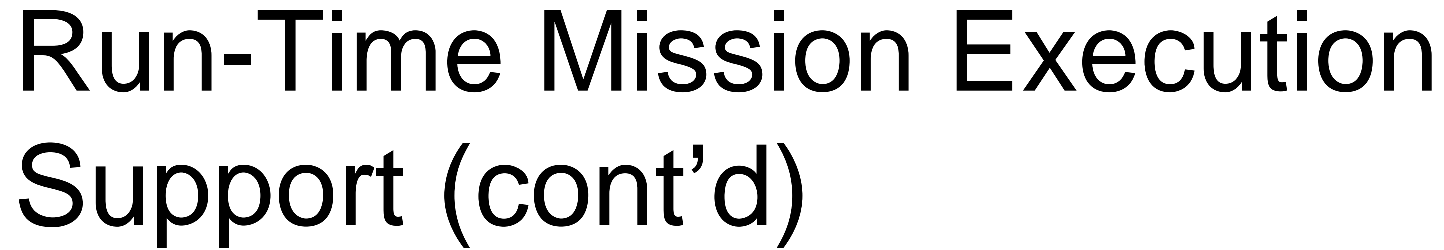
where $i, j \in \mathbb{Z}^+, q \in Q$



Run-Time Mission Execution Support (cont'd)

Plan Modification Operation Algorithm

1 $f_{modPlan}(p, \Delta)$
2 {
3 // Recall $p = (T, L)$
4 $\delta_T = \{t \mid t \in \Delta\}$
5 $\delta_L = \{(t_i, t_j, q) \mid (t_i, t_j, q) \in \Delta\}$
6 // Only perform the operation is Δ is valid and not empty
7 if $((\delta_T \neq \emptyset) \wedge (\delta_L \neq \emptyset))$ then
8 {
9 $T = (T \cup \delta_T) \setminus (T \cap \delta_T)$
10 $L = (L \cup \delta_L) \setminus (L \cap \delta_L)$
11 }
12 return (p)
13 }



- In order to address issues such as missing or unreachable agents during the mission execution, a table that maintains records of the agents' locations and activities is used. This table is called *AgentActivityTable*.

AgentActivityTable is a tuple of the form (a, n, ε) where:

$a \in A$ is an agent and A is a set of agents,

$n \in N$ is the location of the agent and N is a set of locations,

ε is the task execution detail and is defined as a tuple of the form $(t, ts^{\alpha}, ts^{\omega})$ where:

- $t \in T$ is the task being executed, and T is a set of tasks,
- ts^{α} is the time stamp when the agent starts the task execution, and
- ts^{ω} is the time stamp when the agent finishes the task execution.

eHermes

Mission Object Model

Agent
AgentID
Role
MCA
Worker
Type
Stationary
Mobile

WorkOn
Source
Dest

TDG
Task
Location
Type
Compound
Primitive
Codes
Lib
TaskID
Status
Completed
Pending
InProgress
Failed
Aborted
Assigned

Link
Type
Includes
DependsOn
LinkID

At
Location
IP Adress
Port No

Is_A
At
Is_A
Is_A
Is_A

AgentActivityTable
Agent
Location
Execute
Task
Start
Finish

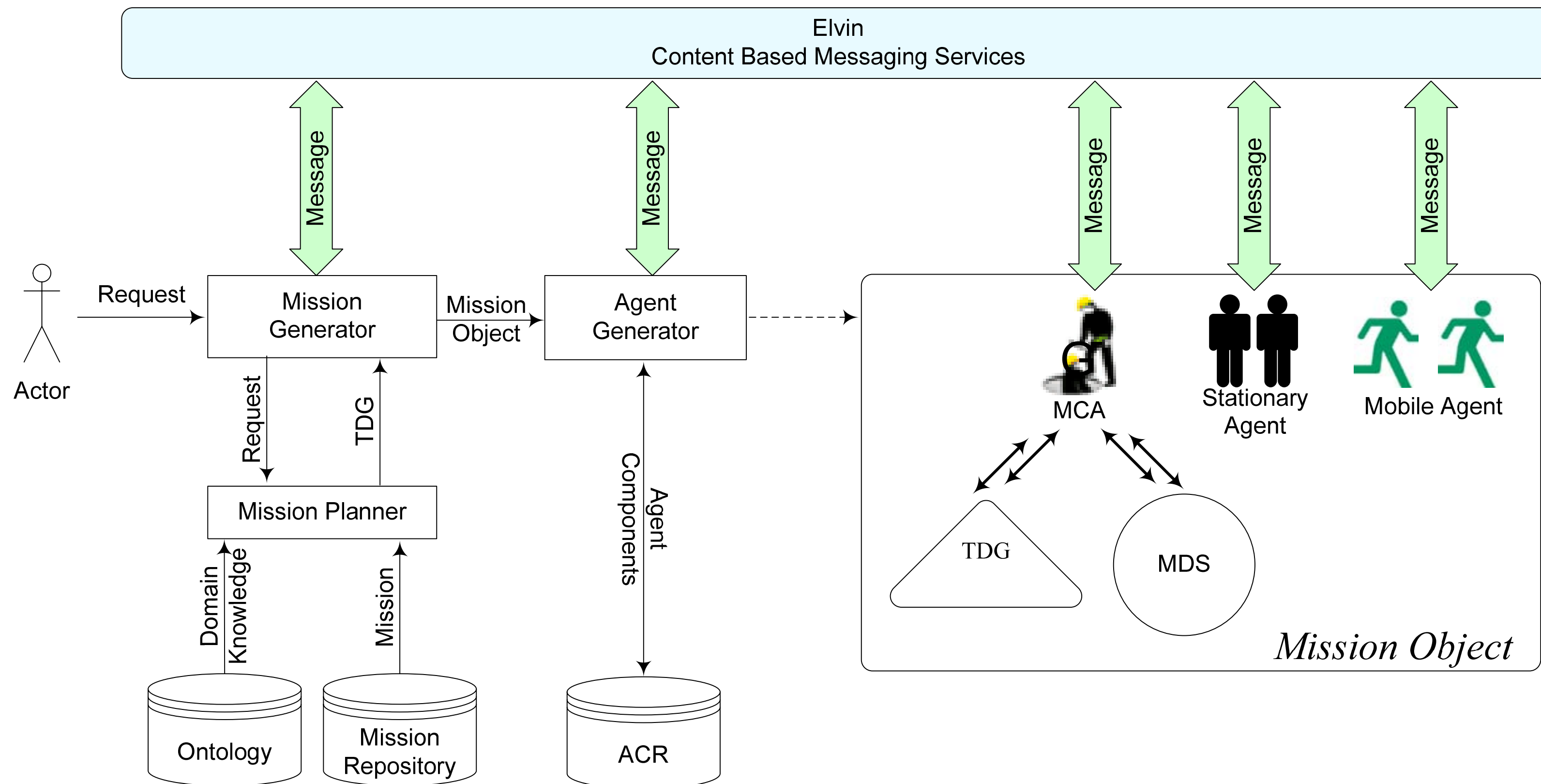
Mission State
MDS
Task
Time Stamp
Result
Plan
Delta
δ+
δ−
Current
Time Stamp
Event
Start
Stop
ChangePlan
Suspend
Resume

Is_A
Is_A
Is_A

Time Stamp
Day
Month
Year
Hour
Min
Sec

eHermes (cont'd)

System Architecture



eHermes (cont'd)

System Architecture

- *Mission Generator*. Mission Generator is the component that is responsible for generating mission objects.
- *Mission Planner*. The Mission Planner produces plans in the form of the TDG.
- *Agent Generator*. The Agent Generator is the component which is responsible for generating agents at run-time.
- *Elvin*. eHermes' agents communicate via short and content-based messages, and Elvin provides communication infrastructure for eHermes

eHermes (cont'd)

Car Purchasing Scenario

Compound Task
Primitive Task
Includes Link
DependsOn Link

Car Purchasing

Get Finance

Select Finance

Get Personal Loan

Get Home Loan

Select Lease

Get Operating Lease

Get Novated Lease

Buy Car

Select Car

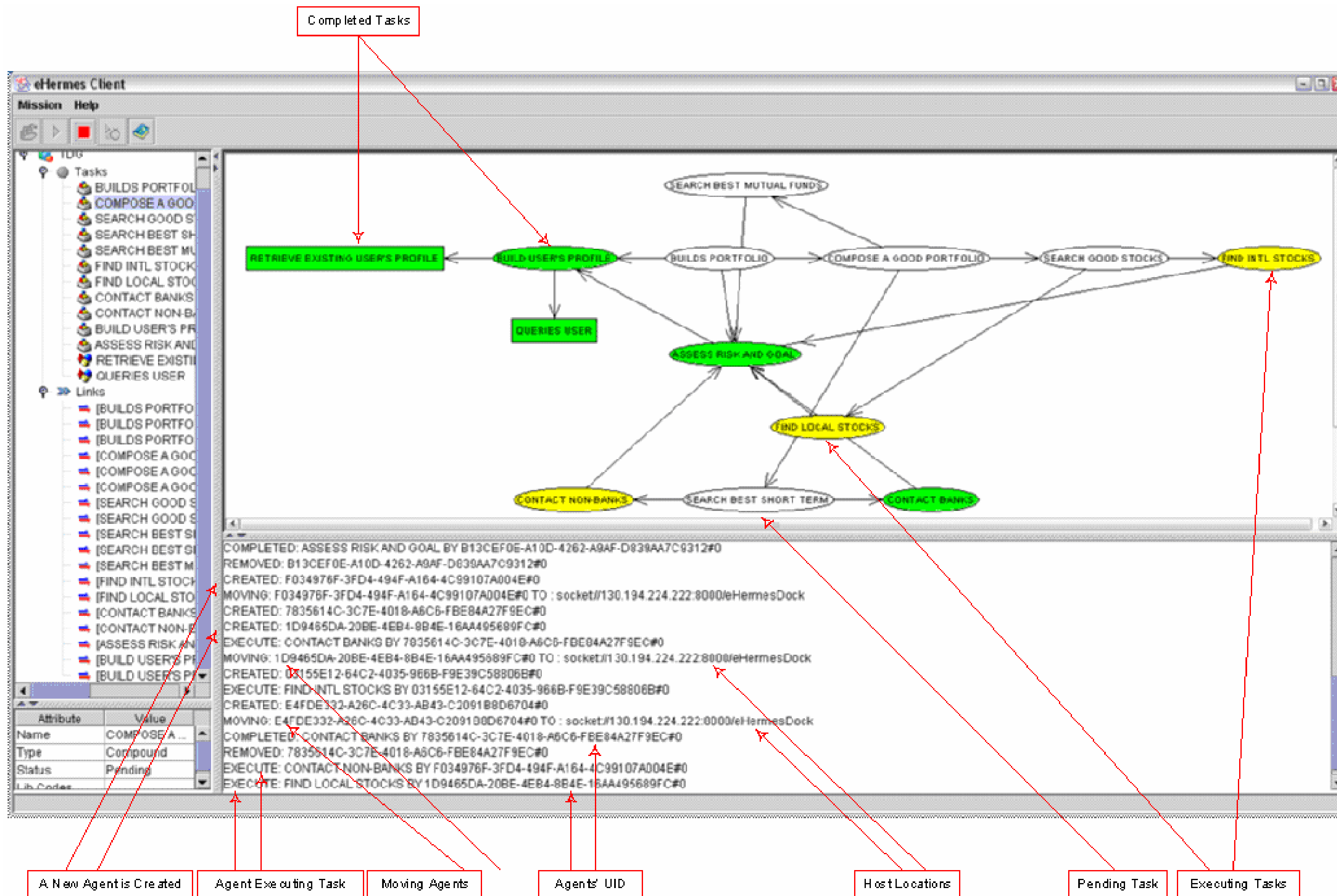
Find Used Cars

Find New Cars

Get Car Specs

eHermes (cont'd)

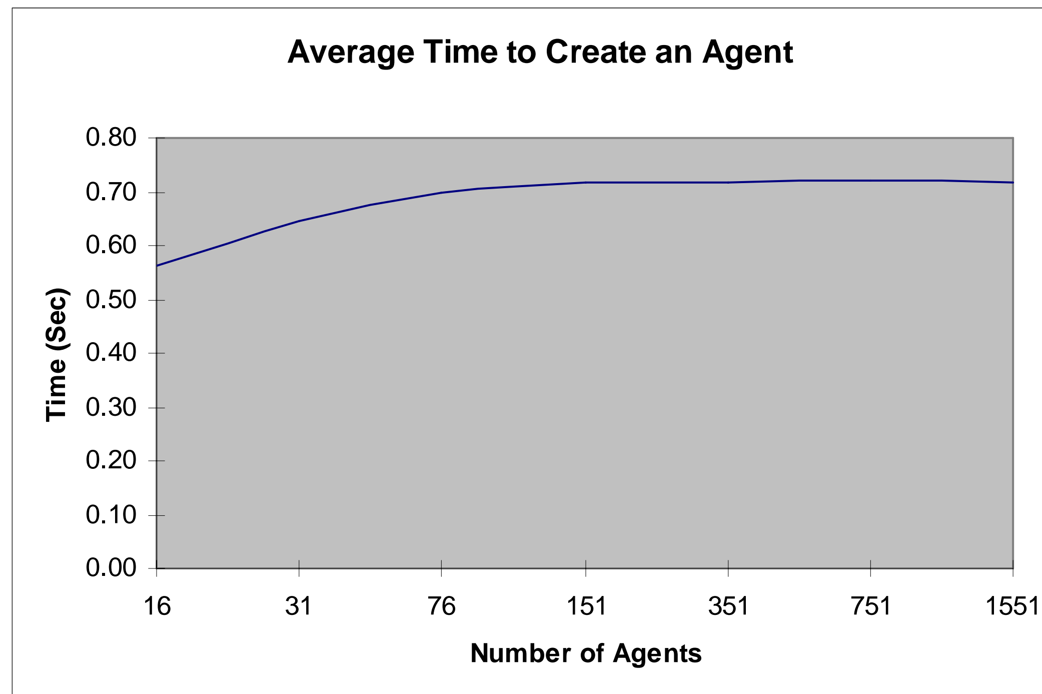
Car Purchasing Scenario



eHermes (cont'd)

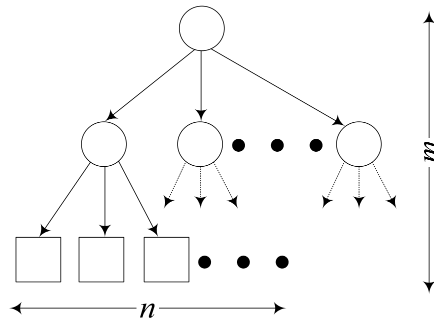
Agent Generation Test

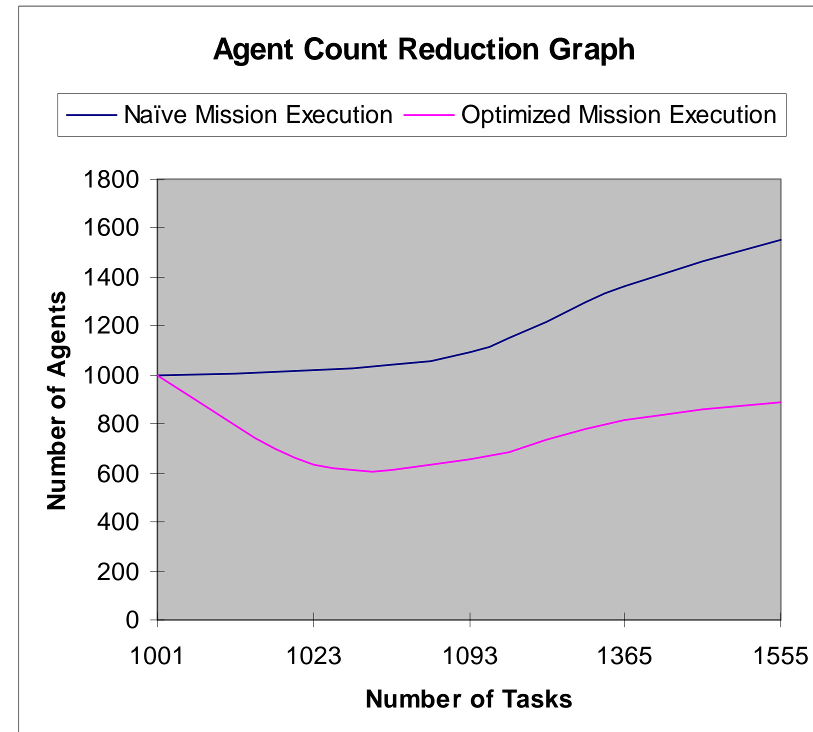
Test No	No Of Tasks	No Of Agents	Mission Execution Elapsed Time (Sec)	Average Time to Create an Agent (Sec)	Performance Decrease (%)
1	16	16	9	0.5625	0.00
2	31	31	20	0.6452	14.70
3	76	76	53	0.6974	23.98
4	151	151	108	0.7152	27.15
5	351	351	251	0.7151	27.13
6	751	751	541	0.7204	28.07
7	1551	1551	1111	0.7163	27.34



eHermes (cont'd)

Agent Count Reduction Test

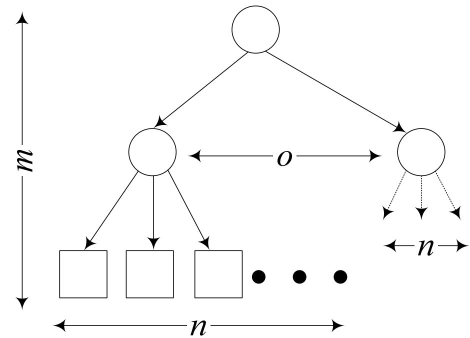




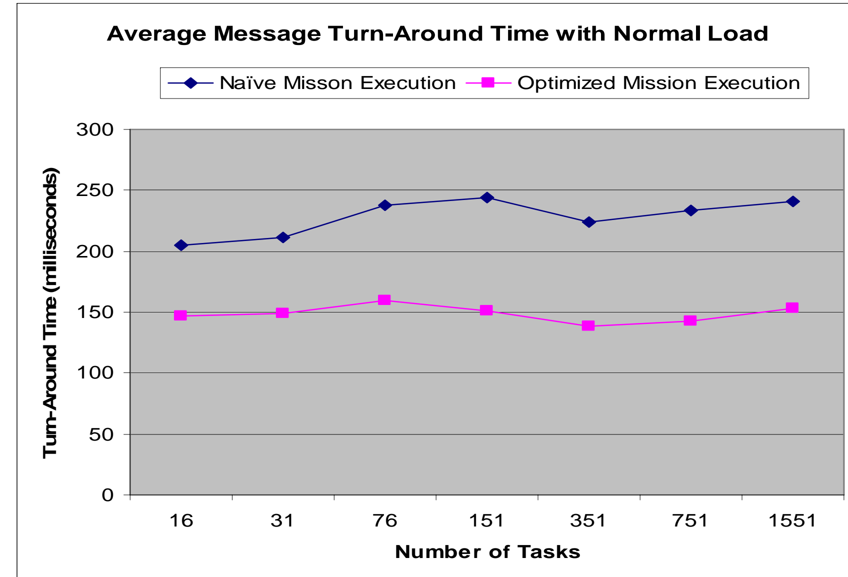
Test No	Plan Structure Parameters	No Of Tasks	Agent Count - Naïve Mission Execution	Agent Count - Optimized Mission Execution	Agent Count Reduction (%)
1	m=1000, n=1	1001	1001	1001	0.00
2	m=9, n=2	1023	1023	633	38.12
3	m=6, n=3	1093	1093	657	39.89
4	m=5, n=4	1365	1365	814	40.37
5	m=4, n=6	1555	1555	889	42.83

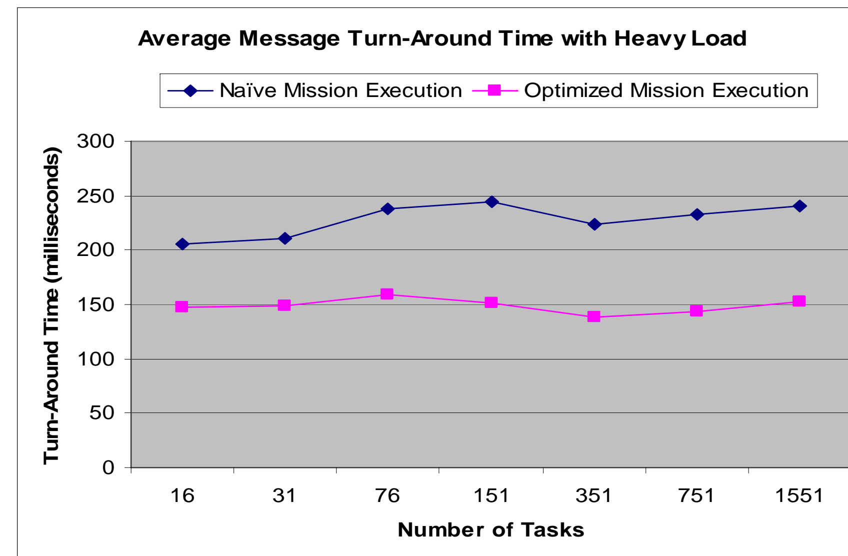
eHermes (cont'd)

MCA Workload Test



Test No	Plan Structure Parameters	No Of Tasks	Normal Load (mSec)		Heavy Load (mSec)	
			Naïve	Optimized	Naïve	Optimized
1	o=5, n=2,l=3	16	154	104	205	147
2	o=10, n=2,l=3	31	175	103	211	149
3	o=25, n=2,l=3	76	155	112	238	159
4	o=50, n=2,l=3	151	158	113	244	151
5	o=50, n=2,l=4	351	173	110	224	138
6	o=50, n=2,l=5	751	165	103	233	143
7	o=50, n=2,l=6	1551	159	111	241	153



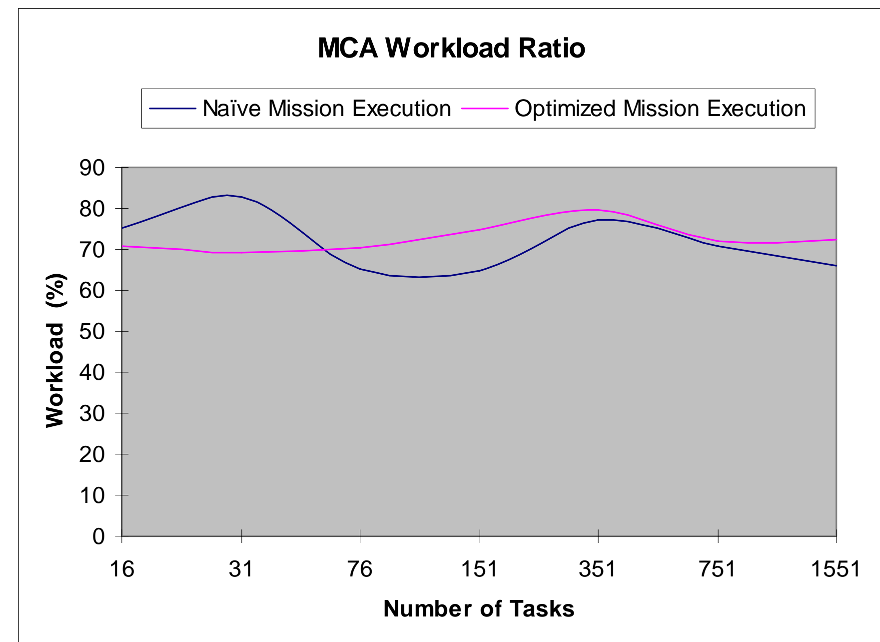


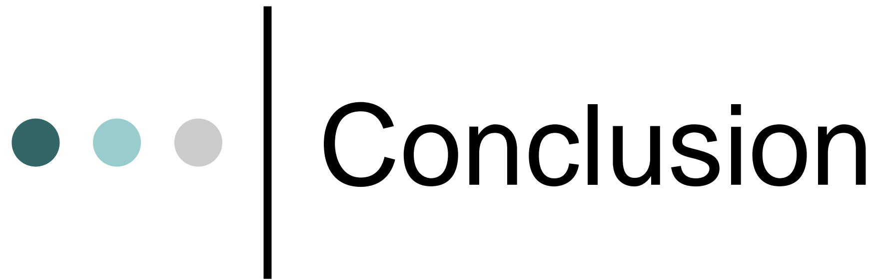
eHermes (cont'd)

MCA Workload Test

No of Tasks	Naïve Mission Execution		MCA Average Workload (%)
	Normal Load	Heavy Load	
16	154	205	75.12
31	175	211	82.94
76	155	238	65.13
151	158	244	64.75
351	173	224	77.23
751	165	233	70.82
1551	159	241	65.98

No of Tasks	Optimized Mission Execution		MCA Average Workload (%)
	Normal Load	HeavyLoad	
16	104	147	70.75
31	103	149	69.13
76	112	159	70.44
151	113	151	74.83
351	110	138	79.71
751	103	143	72.03
1551	111	153	72.55



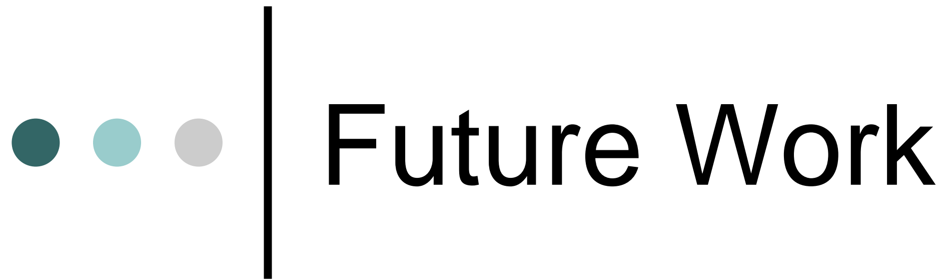


Conclusion

- *The notion of on-demand, just-in-time and run-time agent generation*. This notion allows agents to be generated when they are need, assigned one or more tasks, and autonomously self terminate themselves once the tasks have been completed.
- *The new level of abstraction called the mission*. This new level of abstraction provides a starting point where dynamic MASs can be composed at run-time. The formal definition of the mission and operations applicable to the mission as well as their formal semantic are specified.
- *The formal definition of the plan/TDG (Task Decomposition Graph)* and *the two strategies (i.e. naïve and optimized) to execute the plan* are defined, developed and analyzed. These two strategies execute the tasks in the plan in parallel and by strata.

Conclusion (cont'd)

- *The notion of run-time mission execution support*. Such a notion allows the mission's plan to be modified at run-time without stopping the MAS, as well as to move the mission to other locations. The operation to modify the plan has been formally specified and presented.
- *The formal mission object model.* This formal object model is used as the base for the eHermes system implementation.
- *The overall architecture of the eHermes system*, the prototype system built based on the proposed theory, concepts and algorithms



Future Work

- The current design requires all the operations on the MDS to be conducted through the MCA. When much information needs to be exchanged between the agents, this approach can give unnecessary load to the MCA. In order to address this issue, agents should be allowed to read/write data directly from/to the MDS.
- Collaboration between the worker agents must be increased. The current system design does not take agent collaboration into account.
- Currently the TDG is used for representing the overall mission plan, and not for individual agents. Further research and study might need to be conducted to determine the possibility of having two-level planning abstractions. The top level represents the mission plan, whilst the lower level represents the individual agent's plan.
- The number of agents required to carry out a mission can be further reduced by reusing agents that have completed their tasks. In order to achieve that, the MCA must be able to send new tasks to the agents on the fly.