

Swarming Polyagents Executing Hierarchical Task Networks

Sven Brueckner, Theodore C. Belding, Robert Bisson, Elizabeth Downs, H.V.D. Parunak
Vector Research Center, a Division of TechTeam Government Solutions, Inc.
3250 Green Court, Suite 250, Ann Arbor, MI 48105
{sven.brueckner, ted.belding, robert.bisson, liz.downs, van.parunak}@newvectors.net

Abstract

Swarming agents often operate in benign geographic topologies that let them explore alternative trajectories with minor variations that the agent dynamics then amplify for improved performance. In this paper we demonstrate the deployment of swarming agents in the non-metric and discontinuous topology of a process graph. We align our research with traditional AI approaches and focus on Hierarchical Task Network (HTN) descriptions of constraints and preferences in the execution of abstract methods by a group of real-world entities. In particular, we adapt the TAEMS representation to place a greater emphasis on the mediation of method-execution through shared resources and collectively achieved quality (stigmergic coordination). The paper presents our polyagent model and experiments that demonstrate the scalability of the system and the ability of our agents to achieve optimal entity coordination.

Keywords: polyagents, swarming, scheduling, prediction, self-organization

1 Introduction

A domain for which higher levels of cognition are often considered necessary is coordination in the execution of complex tasks. For example, tests and treatments on a hospital patient can be represented in a treatment plan, which has both internal coordination relationships (some tests must be done in a particular order or within certain time limits) and external coordination relationships between treatment plans (only one MRI machine exists; certain ancillary hospital units prefer to run similar tests in batches to reduce set-up times, etc.) [4]. Another example is the on-line coordination of pre-planned activities in dynamic environments such as military, law-enforcement, or disaster planning scenarios [3]. Several law-enforcement units may wish to surprise suspects at different locations nearly simultaneously so they cannot warn each other. Besides coordinating the surprise itself, some units may require equipment or information whose delivery time is not known in advance. The structure of such tasks can be represented as a graph, specifically, a Hierarchical Task Network or HTN.

All of these types of scenarios have been typically approached by building systems where complex agents have an internal representation of their own plans (and how they relate to the plans of other agents). Examples include

CSC agents [6], or unrolling each agent's view of the HTN into a Markov decision process over which MDP techniques can be applied [7], or translating it into a Simple Temporal Network and applying STN techniques [10].

We take a radically different approach. Rather than putting the HTN inside of *complex agents*, we put *swarming polyagents* inside of the HTN. Coordination is achieved, not by conventional inter-agent dialogs based on each agent's individual analysis of the HTN, but by means of interactions among the agents mediated by the structure of the HTN itself. This paper demonstrates this approach by showing how swarming polyagents can operate on an HTN. Specifically, we work with a dialect of the TAEMS task language [5] that emphasizes the importance of resources, both real and virtual, in coordination (thus resource-TAEMS or rTAEMS). What sets this model apart from other (self-organizing) scheduling and execution approaches is that it includes in its reasoning semantic representations of method-execution preferences that require the coordination of multiple entities.

Section 2 introduces the polyagent modeling construct and summarizes the rTAEMS dialect. Section 3 presents in detail the specific polyagent model where activity coordination emerges from swarming interaction on an rTAEMS graph. Section 4 reports on experiments that demonstrate the effectiveness of this approach. Section 5 outlines future work that expands the approach and concludes this paper.

2 Background

In this section we first summarize the rTAEMS HTN modeling approach, introducing the key terms that define the topology in which the swarming agents operate. Then we briefly introduce our polyagent modeling construct, which uses swarms of simple agents that project specific aspects of the system state into the future for informed decision making.

2.1 Resource TAEMS (rTAEMS)

A hierarchical task network (HTN) is a collection of events, together with two kinds of relations among them: a hierarchical structure relating tasks to their subtasks, and other relations constraining the order of execution among the tasks. In this paper we focus on the Resource-TAEMS (rTAEMS) dialect of TAEMS as a specific instance of an HTN formalism [1, 5]. For a more detailed

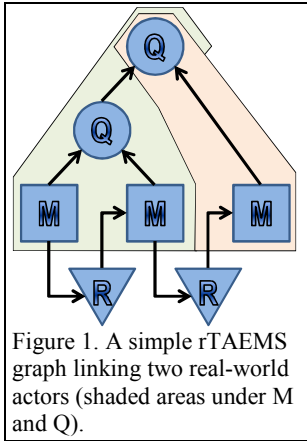


Figure 1. A simple rTAEMS graph linking two real-world actors (shaded areas under M and Q).

introduction and motivation of rTAEMS we refer the reader to [8].

Figure 1 shows a simple example of an rTAEMS graph. The circles (“Q”) are tasks and subtasks that may be associated with one or many actors in the real world (two shaded areas), and can be subdivided into lower-level activities. Since they also serve as the hosts of the quality accumulation process, we call

these rTAEMS nodes “**quality**” nodes. The rectangles (“M”) are “**method**” nodes, which are the lowest level of activity. Each method is associated with a single actor and provides a statistical representation of the execution behavior of this activity (e.g., duration, deadlines). Finally, the triangles (“R”) are the “**resource**” nodes that are emphasized in the rTAEMS dialect over the traditional TAEMS specification.

The primary purpose of the rTAEMS graph is to coordinate the activities of various actors to *maximize overall quality* achievement while *adhering to any method ordering and timing constraints*. The ordering constraints are imposed by the R nodes in the graph. Those nodes carry a non-negative abstract resource level. Methods that start execution consume a given amount of resources from R nodes that have incoming links to the M node. Methods can only start if the resource levels on the incoming R nodes are sufficient for consumption of the specified amounts. When methods complete, they produce a given amount of resources on R nodes that have incoming links from the M node. The actual amount of resources consumed and produced is defined as static annotations to the R-to-M (consuming) and M-to-R (producing) links. Timing constraints associated with a particular M node may further limit the time window in which the method may be started.

When methods complete, they produce also a given amount of quality for Q nodes that have incoming links from the M node. Similar to R nodes, Q nodes carry a non-negative abstract quality level. In addition, any Q node defines a quality accumulation function (QAF) that combines the quality levels on all incoming links (M and Q nodes) into this node’s quality level. That quality level is then used as an input to the QAF at the node’s parent, and so on. The current quality achieved by the actors is defined as the current quality level at the root of the Q node hierarchy.

2.2 Polyagents Modeling Framework

For a more detailed introduction to polyagents, we refer the reader to [9]. The “poly” in “polyagent” reflects the fact that each relevant domain entity is represented by multiple agents: a single avatar and multiple ghosts, combining structured self-organizing swarms (ghosts) that explore large search spaces with classical reasoning approaches (optional) in the avatar. Avatars and ghosts differ in four ways.

Multiplicity: Each entity has only one avatar, but may have multiple ghosts existing concurrently.

Scope: An avatar persists as long as the entity it represents. Ghosts are transient. They are continually generated by an avatar at a specified rate, and they die off after a specified period or upon some specified event.

Reasoning: The avatar may use complex symbolic reasoning, and may communicate directly with other avatars. Ghosts are stigmergic, or ant-like. They independently explore alternative paths and coordinate their actions only indirectly, through changes that they make to a shared computational environment. The most common mechanism for ghost interactions with each other and with the avatars of other entities is through digital pheromones, scalar variables that the agents deposit and sense in the environment. As a result, ghost reasoning is a simple and rapid numerical computation over their behavioral model and the pheromone strengths in their vicinity. Traditionally, the topology of the space over which the ghosts swarm is a representation of the geo-spatial aspects of the domain. In this paper, we are demonstrating swarming on HTN graph representations.

Responsibility: The avatar’s responsibility is to maintain a model of the domain entity and predict and possibly control its behavior. To that end, it generates and tunes a stream of ghosts, whose mission is to evaluate alternative actions and possible interactions. The emergent result of the ghosts’ reasoning can then be used to bias or guide the avatar’s actions.

3 The rTAEMS Polyagents

The various polyagents in our model are coupled through external state variables and temporal pheromone fields that facilitate indirect information exchanges. The pheromone fields are either a probabilistic projection of the state variables into the future (e.g., projected levels at resource nodes or quality nodes), or they encode additional coordinating information required to generate schedules that are correct (enablement) and optimized (quality, deadlines). In our polyagent model, we maintain pheromone fields across the entire graph indexed by a positive temporal offset (future) relative to the current real-world time (avatar time). As ghosts execute their behavioral model, they move through this index of fields

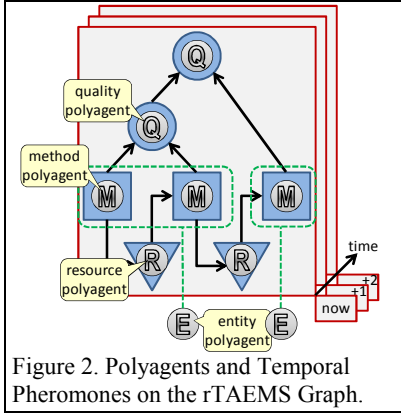


Figure 2. Polyagents and Temporal Pheromones on the rTAEMS Graph.

estimate of one or more external state variable. In particular, if the external state variable is discrete (e.g., resource level), then the ghost state is discrete as well. 2) The ghost samples pheromone fields at its current temporal location and turns pheromone concentrations into probabilities over state variables. Using their random number generator, the ghost then samples these probabilities to postulate the occurrence of particular events that may change its internal state. 3) Based on these events, the ghost changes its internal state, emulating the change of external variables. 4) Finally, the ghost deposits pheromones at its current temporal location, affecting the event probabilities that other ghosts perceive.

Thus, a ghost emulates a possible evolution of a set of external state variables over time and adds this forecast to the probabilistic representation of the state variable in the temporal pheromone field. This coupling of polyagents through actual or projected state variables allows us to discuss the operation of the polyagent model from the perspective of the information flow among variables first (section 3.1), before explaining the specific behavior of the individual agents (section 3.2).

Our polyagent model distinguishes “infrastructure” and “execution” polyagents. The purpose of the infrastructure polyagents is to provide guiding information for the execution polyagents, who in turn construct (ghosts) and execute (avatars) a particular method schedule. The *infrastructure polyagents* represent individual nodes in the rTAEMS graph and their behavior depends on their node type. Thus we distinguish “resource” polyagents, “quality” polyagents, and “method” polyagents. The *execution polyagents*’ associated with the rTAEMS graph is less localized. They model the behavior of real-world entities that may execute certain methods in the graph. In the current implementation, any M node is associated with one particular “entity” polyagent (Figure 2).

3.1 Abstract Information Flows

We describe the information flows that maintain a correct and optimized schedule for the execution avatars.

from the current time into the future, changing their temporal location.

The manipulation of the pheromone fields by the polyagents’ ghosts always follows the same pattern: 1) A ghost carries an internal state that reflects their own

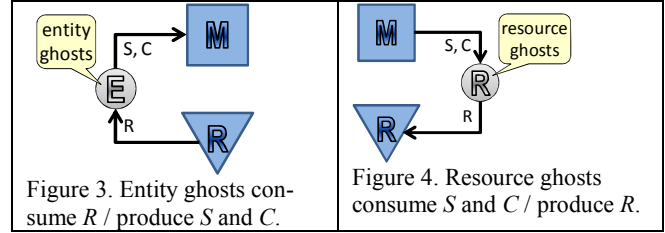


Figure 3. Entity ghosts consume R / produce S and C.

Figure 4. Resource ghosts consume S and C / produce R.

3.1.1 Correct Schedules

The entity ghosts decide when to start and complete a method and accordingly, they deposit temporal “starting” (S) and “completing” (C) pheromones of the selected M node. For correct schedules, the decision whether to start a method depends on the availability of resources consumed by the method. These resource levels are derived from the “resource” (R) pheromone concentrations at the resource nodes linked to the method. Thus, entity ghosts consume R and produce S and C moving through time (Figure 3).

The resource ghosts model the evolution of the level of their R node. As a resource ghost moves through time, it maintains its discrete estimate of the resource level and it deposits this amount of R pheromones. It modifies its estimate by postulating starting and completing events for those methods that consume from or produce to its R node. It postulates these events from the observation of the S and C pheromone fields of those methods. Thus the resource ghosts consume S and C and produce R (Figure 4).

Figure 5 shows that entity ghosts affect the behavior of resource ghosts (by starting and completing methods) while resource ghosts in turn affect the behavior of entity ghosts (by estimating the resulting resource levels). Thus, entity and resource ghosts form a stigmergic feedback loop that results in the emergence of correct schedules where methods are only executed if sufficient resources are available for their consumption.

3.1.2 Optimized Schedules

The stigmergic interaction of entity and resource ghosts in Figure 5 produces correct schedules that are not optimized according to the quality accumulation defined by the Q nodes of the rTAEMS graph. Also, these schedules do not include optimizations that allow high-value methods with early deadlines to be executed on time.

The quality ghosts estimate the evolution of the quality level at their associated Q node. That level changes when a method’s completion adds quality to the node, or when child Q nodes change their levels and change the outcome of the quality accumulation function (QAF).

Like resource ghosts, quality ghosts observe the S and C phe-

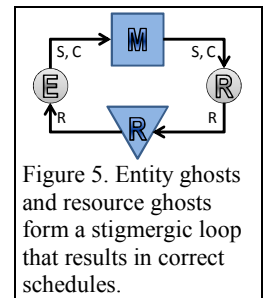
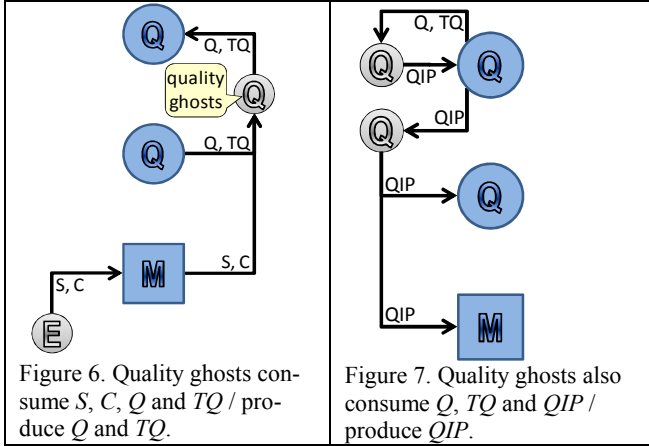


Figure 5. Entity ghosts and resource ghosts form a stigmergic loop that results in correct schedules.



romone levels on those M nodes that provide quality to their node and postulate starting and completing events. Completing events increase the level of quality in the ghost. Quality ghosts also observe the “quality” (Q) pheromone in their node’s Q children and estimate their current projected quality level. The estimated quality levels provided by associated M and Q nodes are the QAF inputs. The QAF result becomes the ghost’s new quality level and it also determines the amount of Q pheromone that the ghost deposits. Thus, quality ghosts consume S , C , and Q and produce Q pheromones (Figure 6).

The infrastructure polyagents on the Q nodes collectively maintain an estimate of the likely evolution of the quality levels based on the projected execution of methods. To guide the selection of enabled methods, we need to compare the quality of the projected schedule with the total quality that could be achieved. We extend the behavior of the quality ghosts to consider the maximum achievable quality of their M node children and apply the QAFs.

The maximum achievable quality of a method depends on whether the method was already executed or not. If a quality ghost considers a method completed, then the achievable quality is the quality produced. Otherwise, it is the quality that the method is projected to achieve (zero in the case of a missed deadline). The quality ghosts consume S , C and Q to produce “total quality” (TQ) pheromone deposits (Figure 6).

From the calculation of the achieved and total quality profile at the Q nodes, we compute the *quality improvement potential* that remains at the M nodes. This calculation starts at the root of the quality hierarchy, where the quality ghosts deposit a “quality improvement potential” (QIP) pheromone equal to the difference of the Q and TQ values. Quality ghosts on all nodes of the hierarchy (including the root) take their local QIP value and distribute it to their children according to their respective QAF. For instance, in a SUM QAF, the QIP deposits are proportional to the children’s TQ contributions. Thus, quality ghosts consume Q and TQ at the root and QIP on all nodes and produce QIP at Q and M nodes (Figure 7).

The concentrations of QIP pheromones on M nodes optimize schedules for high quality. But, QIP alone results in greedy schedules as it does not account for method deadlines. Therefore, method ghosts take the local QIP estimate and combine it with the remaining time to the deadline of their method to compute and deposit the method’s “urgency” (U) pheromone.

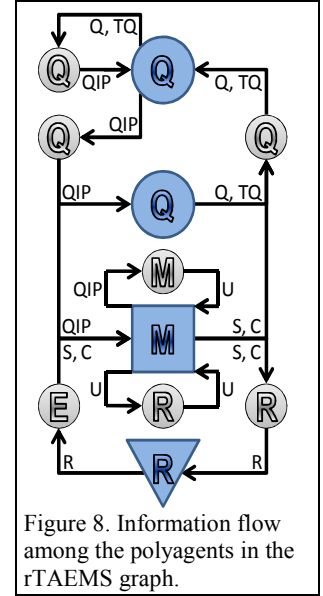
Finally, we want to induce schedules that execute even low- QIP /late-deadline methods early if they lead to the enablement of high- QIP /early-deadline methods. Thus, we extend the resource ghosts to consume U from their consuming methods and deposit U proportionally to their providing methods if their projected resource level is insufficient to enable their consuming methods.

To produce schedules that are correct in regards to enablement and optimized in regards to quality achievement and deadline adherence, entity ghosts need to consider the resource levels at the enabling resource nodes (R) as well as the urgency levels at the method nodes that belong to their entity polyagent (U). Figure 8 shows the entire information flow among the infrastructure and execution polyagents within the topology of the rTAEMS graph.

There are two ways that coordinating information flows from the future into the decision process of the entity polyagent. Implicitly, the QIP calculation assumes the eventual execution of methods in the total quality estimate. Explicitly, the urgency calculation assesses upcoming deadlines and the propagation of urgency by the method and ghost agents move that measure further upstream.

3.2 Specific Agents

Now we discuss the operation of the various polyagents in detail. We start with the ghost logic that maintains a schedule forecast for the near future and then describe its execution by the avatars. We present the ghost and avatar operation in sequence, but in reality those two agent types operate in parallel at different time scales (many ghost cycles between any two avatar cycles). First, we discuss how entity and resource ghosts form a correct schedule. Then we include the remaining ghost types to maintain optimized schedules.



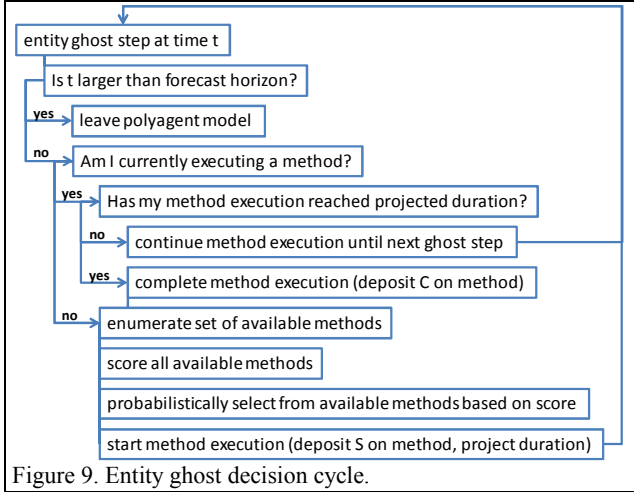


Figure 9. Entity ghost decision cycle.

3.2.1 Correct Schedules

Correct schedules emerge in the stigmergic interaction between swarms of entity ghosts and resource ghosts.

3.2.1.1 Entity Ghosts

An entity polyagent represents a particular real-world actor capable of executing a given set of methods. These methods are modeled as M nodes in our rTAEMS graph. The ghosts maintained by the entity avatar establish a correct and optimized schedule in collaboration with ghost swarms from other entity polyagents and supported by the ghosts of the infrastructure polyagents.

Following the general polyagent modeling paradigm, the entity avatar continuously creates entity ghosts at a fixed rate. Upon creation, entity ghosts copy relevant aspects of the current avatar state into their own state and are placed on the temporal location that corresponds to the current real-world time of the avatars. Then, with each ghost decision cycle, the ghost advances one discrete time step into the future until it reaches the model's forecast horizon. There it ceases to exist.

The initial state of an entity ghost comprises the execution history and the current execution state (what, if any, method is being executed now) of its avatar. With each decision cycle, the entity ghost advances this state by choosing to execute methods from its set of M nodes.

Figure 9 shows the basic decision cycle for an entity ghost. It first checks whether it has passed the forecast horizon. If not, then the ghost asserts whether it is currently in the process of executing a method. In that case, the ghost needs to decide whether it should consider the method completed or whether it should continue executing the method until its next decision cycle based on an internal duration counter set at the start of the method. If the counter reaches zero, the ghost deposits a unit amount of C pheromone at the method's node in the field indexed with its current time t .

To select a new method, the ghost iterates over all M nodes of its entity and assesses their current availability.

The ghost considers a method available, if it has not been executed before by either its avatar or by itself (c.f. re-entrant methods in Future Research 4.1). Furthermore, the availability of a method also depends on its enablement by R nodes. To determine method enablement, the ghost samples the R pheromone on each providing R node and probabilistically estimates its current resource level. This determination is made under the assumption that the pheromone level is in steady state based on regular deposits by resource ghosts (c.f. [2] for detailed analysis of pheromone dynamics). The method is considered enabled, if the sampled resource levels for all enabling resources are above their respective minimum enablement threshold.

If the resulting set of available methods is empty, the entity ghost just pauses for this decision cycle. Otherwise, it selects a method from that set with uniform probability. Marking the start of the method, the ghost deposits a unit amount of S pheromone on the M node and initializes its method duration counter by sampling the method duration distribution from the node's configuration.

3.2.1.2 Resource Ghosts

We assign a resource polyagent to each R node in the graph. The resource ghosts collectively estimate the evolution of the level of their R node from the current actual level to the model's forecast horizon. This collective estimate is reflected in the R pheromone concentrations on the node, which are sampled by the entity ghosts to decide on a method's enablement state.

A resource ghost carries a discrete resource level. As the ghost is created by its resource avatar, this value is set to the current actual resource level. Also at initialization, the resource ghost determines for each producing or consuming method, whether this method is currently being executed by an entity avatar.

Figure 10 shows the basic decision cycle for a resource ghost, beginning with the check for the model's forecast horizon. The ghost iterates over all providing M nodes. For each such node, it samples S and C pheromones levels, and estimates the probability that this method is starting or completing at this time. Depending on

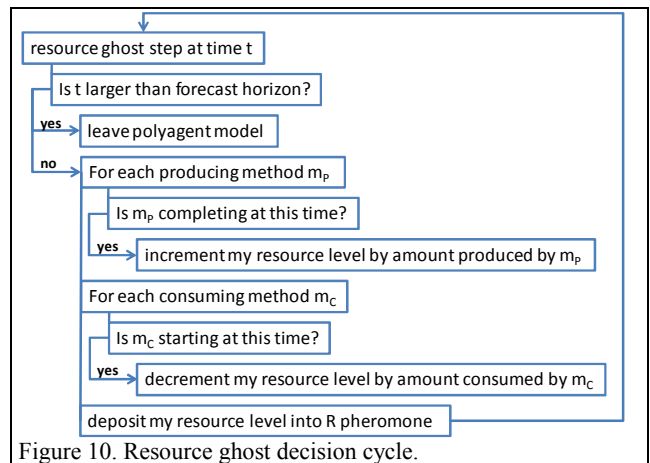


Figure 10. Resource ghost decision cycle.

whether the ghost expects the method to start or complete next, the ghost either uses a probability derived from S or C to postulate these starting or completing events. If the ghost postulates a completing event for a producing method, it increments its internal resource level by the amount produced by that method.

After handling possible increments, the resource ghost considers all consuming methods and again postulates starting and completing events. Here the ghost decrements its resource level upon starting events by the amount consumed by the respective method.

It is important to note that the resource level of a particular ghost is permitted to drop to negative values even though the actual resource level of the node never falls below zero. Prohibiting these forbidden states would otherwise artificially constrain the statistical variations that the current method execution patterns may produce.

The ghost completes its decision cycle by depositing R pheromones equal to its new resource level onto its node into the field indexed with the ghost's current time.

3.2.2 Optimized Schedules

The stigmergic interaction between entity and resource ghosts leads to the emergence of correct schedules. The feedback loop between the execution decisions of the entity ghosts and the resource level estimates produced by the resource ghosts ensures that only those methods are executed that have a high likelihood that sufficient resources for consumption are available. In fact, the entire space of correct schedules is accessible as any correct execution choices may be explored.

Now we show how the infrastructure ghosts process quality accumulation and deadline information to provide additional guidance for the entity ghosts to select optimized schedules from the correct ones.

3.2.2.1 Quality Ghosts

The hierarchy of Q nodes with M nodes at the leaves specifies the accumulation of method-produced quality up to the root of the rTAEMS graph. The quality at the root is the overall performance measure applied to the team of actors whose actions and their interdependencies are modeled. The quality ghosts estimate the evolution of quality levels at each Q node to the forecast horizon and use this estimate to guide the entity ghosts to execution decisions that have the highest potential to improve the root-level quality.

Quality ghosts are very similar to resource ghosts. They carry a level measure for their node (quality instead of resource), and this level is initialized from the current level at their node. If the Q node receives direct contributions from M nodes, the quality ghosts postulate starting and completing events for these methods from the S and C pheromones and increase their internal quality level for any completed providing method.

In contrast to a resource ghost, the internal level of a quality ghost is not just determined by the quality produc-

tion of associated methods. Instead, the ghost's Q node may also have other Q nodes as children. There the ghost probabilistically derives the currently predicted quality level from their Q pheromones. The ghost uses the estimated method quality contributions and the child Q node levels as input into its QAF. The QAF result determines the current quality level of the ghost and the ghost deposits Q pheromones of this amount into its Q node.

As discussed in section 3.1.2, quality ghosts also estimate the accumulation of total quality that can be achieved at their respective node. The mechanism for the creation of TQ pheromone fields is very similar to the Q pheromone generation. For TQ the quality ghost tracks the execution of the providing methods through the S and C pheromones but postulates total quality production as long as the method has not passed its deadline without being completed. Using its node's QAF, the quality ghost combines achievable quality of its providing methods with probabilistically determined TQ levels of any child Q node. It deposits the resulting total quality level as TQ pheromones on its node.

With the Q and TQ fields established by the quality ghosts, we now have sufficient information about the possible improvement of the overall root-level quality. The quality ghosts of the root node deposit the quality QIP pheromone as the current difference between their Q and TQ levels. All temporal QIP fields below the root node are maintained by the quality ghosts of the parent node. These ghosts sample the QIP pheromone at their own node and distribute that value as QIP deposits to their children according to the local QAF. For instance, if the QAF at the parent is a SUM or a MAX, then the parent's QIP is distributed to the children proportionally to the difference between Q and TQ at the respective child. Thus child nodes that still offer the largest gain in quality get assigned the largest quality improvement potential. Other QAFs, like for instance a MIN, trigger an inverse proportional distribution of the parent's QIP .

The distribution of parent QIP to child nodes of a Q node does not distinguish between Q node and M node children, distributing the root QIP to down individual methods. Thus, the combined operation of all quality ghosts maintains a quality improvement potential profile starting at the current avatar time out to the forecast horizon, identifying which methods (if enabled and executed) may provide the largest gains for root-level quality.

3.2.2.2 Method Ghosts

In scenarios without deadlines, QIP alone would be sufficient to guide the entity ghosts towards an optimal schedule. In this case, entity ghosts may just greedily pick methods that offer the largest QIP and fill in any remaining smaller quality gains later. But if these low-gain methods are associated with a deadline, then they should be executed earlier than high-gain methods with later deadlines. Therefore it is necessary to combine the QIP infor-

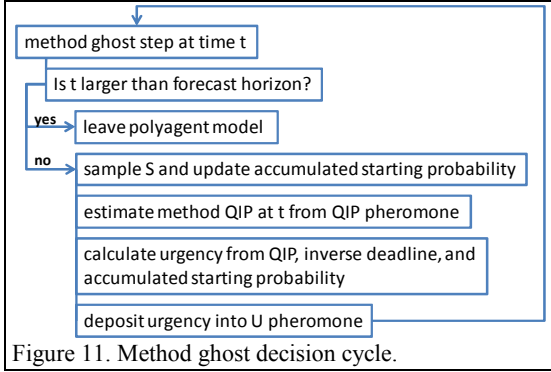


Figure 11. Method ghost decision cycle.

mation of a method with any deadline associated with the method to determine the urgency with which the method should be selected if it is enabled.

The method ghosts perform this simple calculation. Their internal state accumulates the likelihood that their method has been started at or before their current ghost time. If the entity avatar already started the method, then the ghost’s starting estimate is 100% right from its initialization. Otherwise it accumulates starting probabilities sampled from the S pheromone field in each method ghost step.

In each decision cycle (Figure 11) the method ghost samples its local QIP level and divides it by the time that remains until the deadline or the forecast horizon (whichever comes first). It then multiplies this value with $1 -$ accumulated starting probability – the likelihood that the method is not yet started. The resulting urgency measure grows with increasing quality improvement potential, with an approaching deadline, and with decreasing starting probability. The method ghosts deposit this urgency value into the U pheromone field.

3.2.2.3 Resource Ghosts

One final step in the urgency calculation is necessary to ensure that methods with high indigenous urgency (close to deadline, high QIP , low starting probability) get enabled by upstream methods in time even if these predecessor methods themselves have low indigenous urgency. In effect, we want to selectively “propagate” urgency upstream along the method enablement relationships expressed by the R nodes between them.

We extend the behavior of the resource ghosts beyond our initial description. After completing the operations associated with the creation of correct schedules, a resource ghost sums up the urgency of consuming methods that currently have insufficient enablement by their providing resources. Each such urgency value is weighted with the resource level that the respective method would consume from the ghost’s resource. The resulting “resource urgency” value is then distributed (U pheromone deposits) proportionally among all providing methods according to the level of resource they would be contributing. The temporal index of these deposits is offset by

the duration of these methods, increasing the urgency to start these methods in time.

3.2.2.4 Entity Ghosts

In the previous sections we showed how the ghosts of the infrastructure polyagents create a rich information environment across space (rTAEMS graph) and time (up to the forecast horizon) based on global quality improvement potential and method deadlines resulting in localized urgency fields at the method nodes. Now we discuss how the entity ghosts take this information into account when making their execution decisions.

The summary of the entity ghost decision cycle in Figure 9 already includes the necessary steps: “score all available methods” and “probabilistically select method based on score”. To create correct schedules it was sufficient to select among the available methods randomly. With method urgency information available, the entity ghost scores its available methods by their U pheromone concentrations. Thus, methods with higher urgency have a higher likelihood of being executed.

Methods that have an urgency of zero will not be executed since they neither contribute additional quality themselves nor enable other methods that may provide quality. Thus it is possible that even though methods are enabled for a particular entity ghost, the ghost may still decide not to execute anything. This is a desirable behavior as it allows entity polyagents to ignore unproductive methods.

3.2.3 Executing Schedules

We had mentioned before, that in the general polyagent model, the avatar may be the host of complex (cognitive) reasoning process about its entity. For the rTAEMS model presented in this paper, such complex reasoning is not necessary, because already the infrastructure and execution ghosts collectively maintain a correct and optimized schedule in the distribution of S and C pheromones on method nodes. Thus, all necessary reasoning about which (if any) methods should be actually executed next by the entity avatars is performed by the swarming ghosts of the system. All an **entity avatar** has to do now in its decision logic is to exploit the guidance that is generated by the exploration of the information landscape by its ghosts.

The entity avatar (Figure 13) executes similar decision logic as an entity ghost (Figure 9) in regards to its overall execution behavior. If, in a particular decision step, it is already executing a method, it decreases its method duration counter and completes the method if the counter reaches zero. If the avatar is ready to select a new method for execution, establishes a set of available methods. It is up to the entity avatar to ensure that its execution remains

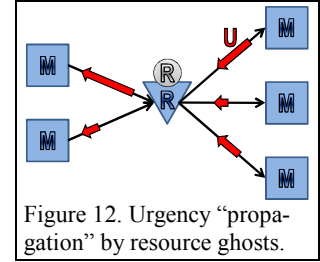


Figure 12. Urgency “propagation” by resource ghosts.

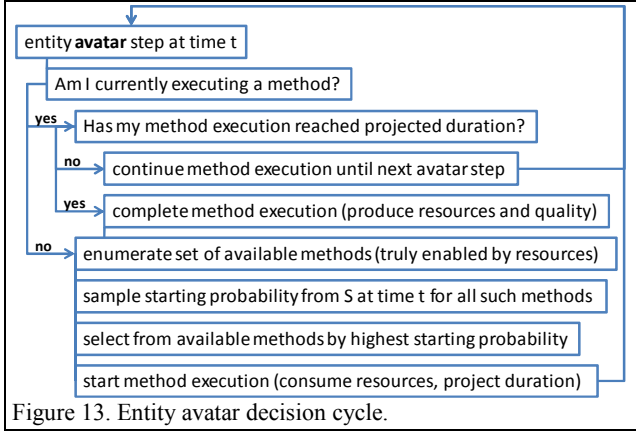


Figure 13. Entity avatar decision cycle.

correct in regards to *actual* enablement and deadlines of its methods. Its ghosts used R pheromones to estimate enablement and though it is unlikely that this estimate is wrong at the beginning of the forecast window (close to the actual system state), entity avatars still have to enumerate their set of available methods based on actual resource levels at the enabling nodes and exclude those methods that it actually executed before or that have run out of time.

From the set of available methods, the entity avatar selects the method that has the highest starting likelihood (from the S pheromone) at the first ghost cycle. If there is more than one such method, the avatar has no further guidance and selects among the maximum likelihood methods randomly. If the highest starting probability is still below a configurable threshold, then the avatar does not select any method for execution and pauses instead until its next cycle.

In the forecasting component of the polyagent model, entity ghosts do not consume or produce resources directly. Instead, resource ghosts observe the starting and completing probabilities of their associated methods and maintain the resource-level forecast in the R pheromone. Conversely, the execution by the entity avatars constitutes the real and irreversible start and completion of methods. Thus, as an avatar starts a method, it actually consumes resources (decrements resource levels), and when it completes a method, it actually produces resources (increments resource levels) and quality (increments quality levels). As a consequence, the decision process of **resource avatars** is empty.

While the resource avatars are impoverished in their behavior, **quality avatars** still play an important role. They have to track the actual production of quality by the entity avatars and recursively roll those up to the root node of the quality hierarchy. This roll-up provides an immediate picture of the currently achieved quality by the polyagent system.

Finally, **method avatars** simply register the execution of their method by an entity avatar to establish the correct initial state of their ghosts.

4 Evaluation Experiments

We report on experiments with our rTAEMS polyagent system. First we discuss capability experiments on artificially constructed graphs that highlight particular challenges. Then we report on benchmark tests against traditional TAEMS approaches.

4.1 Capability Experiments

Below is a small sample of the various experiments that we performed to test agent interaction, fine tune agent parameters, and analyze the scalability of the algorithm by increasing the number of avatars and methods.

4.1.1 Stepped Deadline Graph (Single Entity Avatar)

The graph contains ten methods M_{1-10} with a common duration of one, producing a quality of one each (no quality preferences). Method M_i has a deadline at $t=i$ (stepped deadlines). The optimal sequence of method execution by an avatar is highly constrained by the deadlines. If any one method is executed out of sequence, a loss of quality is observed.

Without quality preferences, the entity ghosts provide sufficient guidance for the avatars to execute the stepped graph flawlessly (see screenshot in Figure 14). But we find that differences in quality production may lead to greedy, out-of-sequence execution of higher-value methods. A tuning parameter that balances the impact of deadlines with the preferences expressed by quality production suppresses this greedy behavior to a point. But in the case of the stepped graph, the modeler who constructed the fully deadline-constrained graph should not have added any conflicting quality preferences.

4.1.2 Quality/Deadline balance Graph (4 Entity Avatars)

As observed in the previous section, higher quality offered by one method may overwhelm the urgency to execute another to meet its (or its dependants) deadline. This series of experiments demonstrates the existence of a balance point where quality greed overwhelms deadline constraints.

We specify a graph with 4 avatars $\{A, B, C, D\}$ and their respective two methods $\{M[A-D]_1, M[A-D]_2\}$. Methods $M[A-D]_1$ are all assigned a deadline of 1, while

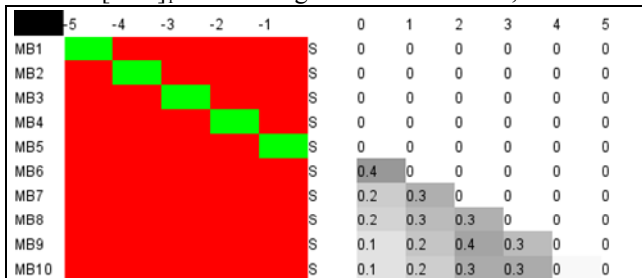


Figure 14. Correct execution of the Stepped Deadline Graph methods (rows) by the avatars (left) predicted execution (right).

Table 1. cTAEMS Benchmark Characteristics and Results.

<i>Characteristics</i>			
Name	#Entities	#Methods	#Nodes
MayOptNLE4	6	12	28
MayOptContingent3	3	18	43
<i>Results</i>			
Name	Optimal Quality	Mean Quality	Std. Dev. Quality
MayOptNLE4	65	65	0
MayOptContingent3	66.5	66.5	0

methods $M[A-D]_2$ are assigned a deadline of 2. The quality produced by the method was set as follows: $M[A-D]_1=1$, $MA_2=1$, $MB_2=2$, $MC_2=3$, and $MD_2=4$.

The optimal sequence of method execution for any avatar is M_1 before M_2 . Without the aforementioned (4.1.1) tuning parameter, the system is able to execute this optimal sequence only for avatar A. The other avatars were led by their ghosts to execute their second method ($M[B-D]_2$) first because of the quality improvement potential they were offering. Thus, the deadlines of $M[B-D]_1$ were reached and their quality contributions lost.

The tuning parameter affects the contribution of QIP to the urgency of a method and thus balances the competing optimization goals of maximizing quality and meeting deadlines. Repeating the experiment with a decreased QIP impact, avatar B now also executes MB_1 before MB_2 , but $M[C-D]_1$ still expire.

These experiments highlight that the designer of rTAEMS graphs must not only adhere to the correct syntax of the graph, but must also be aware of the emergent dynamics of the polyagent system that resolves competing optimization goals.

4.1.3 Stepped Deadline Graph (Scaling Tests)

To explore the scalability of our polyagent approach, we first increase the number of stepped methods (see 4.1.1) executed by a single entity avatar to 500 (M_{1-500} , $dM_i=i$) and then increase the number of avatars that are associated with the stepped methods to ten (alternating avatar-method association $A_1=\{M_1, M_{11}, M_{21}, \dots\}$, $A_2=\{M_2, M_{12}, M_{22}, \dots\}$). In all cases, our polyagents were able to produce the optimal (stepped) execution sequence, naturally with increasing computational cost (linear with #methods, #avatars). It is worthwhile to point out that all polyagent interactions in our model are local on the rTAEMS graph. Thus, the distribution of this system over many computational hosts for a distributed group of coordinating entities is very straight forward.

4.2 Benchmark Experiments

In order to compare our polyagents' performance with related work, we experimented with two cTAEMS task networks taken from the May, 2007 evaluation trials of the DARPA COORDINATORS program (K. Decker, personal communication). The task networks used in these trials were intended to be simple enough to be solv-

able by an optimal cTAEMS algorithm, while still being complicated enough to evaluate and compare the performance of non-optimal, heuristic solvers.

The two networks chosen were "MayOptNLE4" and "MayOptContingent3" (Table 1). We converted the original cTAEMS networks into equivalent rTAEMS networks, preserving semantics and resulting quality. Accounting for the probabilistic nature of our approach, we executed 25 replications of each network with a different random seed. The results in Table 1 show that our approach achieved optimal results.

5 Conclusion and Outlook

Typically, swarming and even polyagent applications place their agents in shared computational environments with geographic topologies. These metric and reasonably continuous spaces provide the agents with sufficient space to explore alternative trajectories with minor variations where the non-linear dynamics of the agent system amplifies these variations when they offer improvements to the system's performance. In this paper we demonstrate how swarming agents may be deployed on the non-metric and discontinuous topology of a process graph, using the metric and continuous temporal domain and the distribution of numeric resource and quality levels as the source for those minor variations that are essential to the adaptiveness of self-organizing algorithms.

We align our research with traditional Artificial Intelligence approaches and focus on Hierarchical Task Network (HTN) descriptions of the constraints and preferences in the execution of abstract methods by a group of entities. In particular, we adapt the TAEMS representation for HTNs to place a greater emphasis on the mediation of method-execution through shared resources and collectively achieved quality (stigmergic coordination). On the rTAEMS graph representation of methods that are enabled by the availability of resources produced by other methods and whose execution produces quality that is aggregated to a system-level quality achievement, we place "infrastructure" polyagents on each node that project the evolution of the state of their node forward in time. The entities that are using the rTAEMS graph to coordinate their activity are also represented by polyagents, driving through their projected and actual execution of methods the evolution of the infrastructure agents.

In this paper we discussed in detail the population-level dynamics of the complex polyagent system on rTAEMS, specified the decision logic of the agents that make up the swarming component of the polyagents ("ghosts"), and reported on capability and benchmark experiments. We were able to show that our polyagent approach to scheduling and execution of HTNs is capable of achieving optimal performance while offering the ability to dynamically reschedule to adapt to changing environments and to distribute the process among mul-

multiple hosts associated with the coordinating entities. (Due to the stochastic nature of the algorithm, optimal performance cannot be guaranteed on more complicated problems.)

The project that funded this research activity has come to an end. But in a related project, we are now extending the rTAEMS polyagent system in several directions. We are expanding the applicability of the rTAEMS process model by supporting the execution of a method more than once (re-entrant methods) and potentially by different entities (shared methods). Also, to model opposing “sides” among the entities for instance in war-games, we allow rTAEMS graphs to have more than one quality root.

The main extension comes from the specialization of the methods. In the rTAEMS version reported in this paper, a method is characterized by abstract attributes such as its duration or deadline and we assume that the method always concludes successfully (producing quality). The specialized method nodes will include a detailed execution model that simulates the execution of the method in a geo-spatial model. From that simulation, we derive dynamically the expected duration of that method and the amount of quality it produces. Thus, methods could have varying duration and success depending on the spatial context in which they are executed.

Acknowledgements. This research was conducted with the support of the office of Naval Research (Contract # N00014-06-1-0467). The results presented do not necessarily reflect the opinion of the sponsor. The authors are grateful to Prof. Keith Decker for extensive discussions and suggestions on this research.

6 References

[1] M. Boddy, B. Horling, J. Phelps, R. P. Goldman, R. Vincent, A. C. Long, B. Kohout, and R. Maheswaran. C_TAEMS Language Specification, Version 2.02. DAR-PA, Arlington, VA, 2006.

[2] S. Brueckner. Return from the Ant: Synthetic Ecosystems for Manufacturing Control. Dr.rer.nat. Thesis at Humboldt University Berlin, Department of Computer Science, 2000.

[3] W. Chen and K. Decker. The Analysis of Coordination in an Information System Application – Emergency Medical Services. In P. Bresciani, P. Giorgini, B. Henderson-Sellers, and M. Winiko, Editors, Agent-Oriented Information Systems, vol. 3508, LNAI, pages 36-51. Springer, New York, NY, 2005.

[4] K. Decker and J. Li. Coordinating mutually exclusive resources using GPGP. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(2):133-158, 2000.

[5] B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey. The TAEMS White Paper. 2004.

[6] R. T. Maheswaran, P. Szekely, M. Becker, S. Fitzpatrick, G. Gati, J. Jin, R. Neches, N. Noori, C. Rogers, R. Sanchez, K. Smyth, and C. Vanbuskirk. Predictability & Criticality Metrics for Coordination in Complex Environments. In *Proceedings of Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, Estoril, PT, 2008.

[7] D. J. Musliner, E. H. Durfee, JianhuiWu, D. A. Dolgov, R. P. Goldman, and M. S. Boddy. Coordinated Plan Management Using Multi-agent MDPs. In *Proceedings of AAAI Spring Symposium on Distributed Plan and Schedule Management*, Palo Alto, CA, 2006.

[8] H. V. D. Parunak, T. Belding, R. Bisson, S. Brueckner, E. Downs, R. Hilscher, and K. Decker. Stigmergic Modeling of Hierarchical Task Networks. In *Proceedings of the Tenth International Workshop on Multi-Agent-Based Simulation (MABS 2009, at AAMAS 2009)*, Budapest, Hungary, (forthcoming), 2009.

[9] H. V. D. Parunak and S. Brueckner. Concurrent Modeling of Alternative Worlds with Polyagents. In *Proceedings of the Seventh International Workshop on Multi-Agent-Based Simulation (MABS06, at AAMAS06)*, Hakodate, Japan, Springer, 2006.

[10] S. F. Smith, A. Gallagher, T. Zimmerman, L. Barbulescu, and Z. Rubinstein. Distributed Management of Flexible Times Schedules. In *Proceedings of Sixth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007)*, Honolulu, Hawaii, pages 472-479, 2007.