Presented at the Eighth International Workshop on Multi-Agent-Based Simulation (MABS07) at AAMAS 2007, Honolulu, Hawaii, 15 May 2007. Forthcoming in the Proceedings.

E Pluribus Unum: Polyagent and Delegate MAS Architectures

H. Van Dyke Parunak[1], Sven Brueckner[1], Danny Weyns[2], Tom Holvoet[2], Paul Verstraete[2], Paul Valckenaers[2]

[1] NewVectors LLC, 3520 Green Court, Suite 250, Ann Arbor, MI 48105 USA
{van.parunak, sven.brueckner}@newvectors.net
[2] Katholieke Universiteit Leuven, 3001 Leuven, Belgium
{danny.weyns, tom.holvoet}@cs.kuleuven.be
{paul.verstraete, paul.valckenaers}@mech.kuleuven.be

Abstract. For the past few years, our research groups have independently been developing systems in which a multi-agent system (typically of lightweight agents) provides some functionality in service of a higher-level system, and often of a higher-level agent in that system. This paper compares our approaches to develop a more generic architecture of which our individual approaches are special cases. We summarize our existing systems, describe this architecture and the characteristics of problems for which it is attractive, and outline an agenda for further research in this area.

1 Introduction

Great ideas often occur to several researchers at the same time. At AAMAS06 and its workshops, NewVectors (NV) reported a modeling construct that represents a single domain entity with multiple agents, a “polyagent” [13, 14]. Katholieke Universiteit Leuven (KUL) described how individual agents in a manufacturing system could delegate certain tasks to a swarm of ant-like agents, a “delegate MAS” [5, 6].

The use of multiple agents to model a single agent is not new, but typically each of the multiple agents has a distinct function, which will be lost if that agent is eliminated. An example of this functional decomposition is the CODAGE system developed at the Laboratoire d’Informatique of the Université de Paris [8]. What sets our systems apart is that they use multiple agents *with the same function* to explore some combinatorial space through which the single agent must move—a planning space, or a space of possible futures, or a space of alternative decisions. The multiple agents conduct concurrent agent-based simulations to guide the decisions of the single agent. The number of agents modulates the performance and efficiency of the system, but not the functionality that is achieved.

The Latin phrase in our title applies to our topic in two ways. First, each of our systems uses a swarm of agents to provide a unified function, producing “one out of many” within the setting of a single application. Second, in seeking to unify our technical vision, we wish to produce “one (higher-level architecture) out of many” (or at least two) previously independent approaches. Realizing this second unification will enable us to pursue a common research agenda, develop common tools, and leverage off of one another's results.

Section 2 reviews our two approaches, with examples of how they have been applied in practice, and compares their motivations and foci. Section 3 discusses design considerations. Section 4 looks at future work. Section 5 concludes.

2 Polyagents and Delegate MAS

We begin by reviewing the two independent systems that motivate this analysis, and exploring their complementarities.

2.1 Polyagents at NewVectors

The Polyagent Model.—Agent-based modeling conventionally associates a software agent with each entity in the domain. For example, an entity might be a soldier in a military domain, or a vehicle in a traffic model, or a person in a social simulation. A polyagent represents each domain entity with multiple agents: a single avatar that links it to the entity, and a swarm of ghosts that explore its alternative behaviors. Figure 1 shows the conceptual architecture of a polyagent.

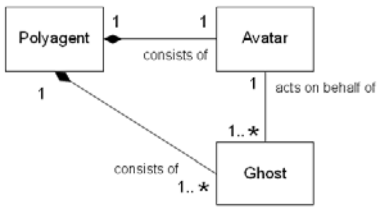

Fig. 1. A polyagent represents a domain entity with one avatar and multiple ghosts.

The avatar persists as long as its entity is active, and maintains its entity's state. It may use sophisticated reasoning. Each avatar generates a stream of ghosts. Ghosts die after a fixed period of time or after some defined event. Each avatar controls the rate it generates ghosts, and typically has several concurrent ghosts. The ghosts are the "multiple agents with the same function" mentioned in our introduction.

Ghosts explore alternative behaviors for their avatar. In the applications constructed by NewVectors researchers, they are computationally simple, and interact through a digital pheromone field, a vector of scalars that depends on both location and time. Each ghost chooses its actions stochastically based on a weighted function of nearby pheromones, and optionally deposits its own pheromone. A ghost's "program" is the vector of weights.

The main benefit of representing a single domain entity with multiple agents is to multiply the number of interactions that a single run of the system can explore. Instead of one trajectory for each avatar, we now have one trajectory for each ghost. If each avatar has k concurrent ghosts, we explore k trajectories concurrently, leading to an increase in the number of interactions being explored at each step of an n-avatar system from n to k^n [13]. The avatar can base its decisions within a single run of the system on the multiple possible futures explored by its ghosts. In effect, the ghosts form an agent-based model that supports the decisions of the higher-level avatar agent.

The avatar can

- modulate the number of its ghosts, their rate of generation, and the distribution of their parameters to control the exploration of alternative futures;
- evolve them to learn the best parameters for a given situation;
- review their behavior to estimate its own future experience.

NewVectors researchers have applied the polyagent model to three distinct domains: factory scheduling, robotic routing, and combat prediction.

Factory Scheduling.—Our earliest application of polyagents did real-time job-shop scheduling [1] with three types of agents: processing resources, parts, and policy agents. Avatars of processing resources with different capabilities and capacities and avatars of parts with changing processing needs (due to rework) coordinate to optimize material flow through a complex, high-volume manufacturing transport system. Only part avatars deploy ghosts. Policy agents and resource (machine) avatars are traditional single agents, whose loads the ghosts explore in order to choose assignments for the parts.

Robotic Routing.—Robotic vehicles must continuously replan their paths, as their knowledge of the environment changes due to limited sensor range and environmental change. In military applications, vehicles must navigate dynamically changing sets of targets and threats. Ants solve a similar problem in forming paths between nests and food sources [9]. Ants searching for food deposit nest pheromone while climbing the food pheromone gradient left by successful foragers. Ants who find food deposit food pheromone while climbing the nest pheromone gradient left by outbound ants. The pheromone fields collapse into a path as the ants interact. We have emulated this behavior to route robotic aircraft [16]. The agent controlling the robot sends out a stream of ghosts that execute the ant path planning algorithm in real time. The ghosts deposit nest pheromone as they move from the robot to seek out targets while avoiding threats, and target pheromone after they have found a target and are making their way home. Positive reinforcement among ghosts through their target pheromone leads to the emergence of high-density target pheromone paths that guide the robot.

Combat Prediction.—A commander in urban combat may have observations of the recent behavior of the adversary, and want to extrapolate these to predict future behavior. We use polyagents to evolve a model of the internal personality of each real-world entity

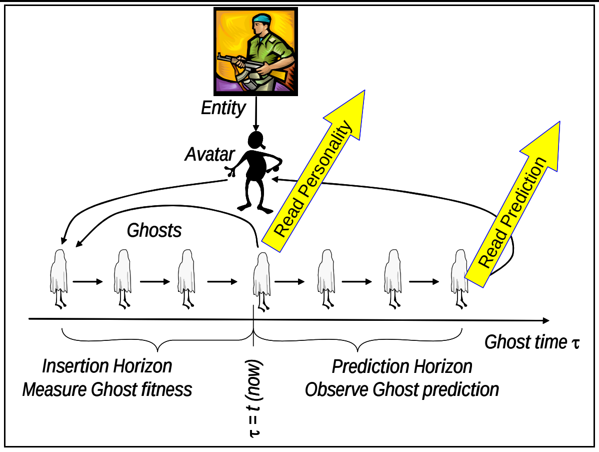

Fig. 2. Each avatar generates a stream of ghosts that sample the personality space of its entity. They evolve against the entity's recent observed behavior, and the fittest ghosts run into the future to generate predictions.

and predict its future behavior [11]. Figure 2 shows the process. Ghosts live on a timeline of discrete pages indexed by τ (distinct from real time t) that begins in the past and runs into the future. The avatar inserts ghosts at the insertion horizon (say $\tau - t = -30$, the state of the world 30 minutes ago), sampling each ghost's parameters to explore alternative personalities of its entity.

The avatars record pheromones representing the observed state of the world on each page between the insertion horizon and $\tau = t$. The inserted ghosts interact with this past state. Their fitness depends not just on their own actions, but also on the behaviors of the rest of the population, which is also evolving. τ advances faster than real time, so eventually $\tau = t$, when the avatar compares each ghost with its entity's actual state.

The fittest ghosts have three functions.

1. Their personality estimates the personality of the corresponding entity.
2. They breed, and their offspring reenter at the insertion horizon.
3. They run into the future, exploring possible futures of the battle that the avatar analyzes to predict enemy behavior and recommend friendly behavior. In the future, the pheromone field is generated by other ghosts rather than avatars. Thus it integrates the various futures that the system is considering, and each ghost interacts with this composite view of other entities.

While many of the applications summarized in this paper deal with robotic systems, this application is important in showing the relevance of polyagents to social systems. In the past, multiple ghosts can be evolved against an entity's outward behavior to discover the underlying personality (including emotion). In the future, the ability of multiple ghosts from different avatars to interact with one another lets us explore multiple possible social interactions efficiently in a single run of the system.

2.2 Delegate MAS at KUL

Delegate MAS Model.—A delegate MAS is a modeling construct that consists of a swarm of lightweight agents (ant agents) that provide a service for a higher level agent (the issuing agent) to support this agent in fulfilling its functions (Figure 3).

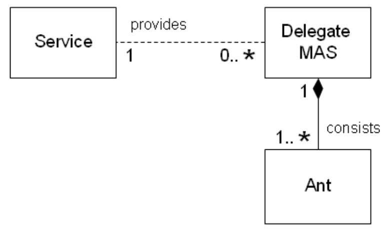

Fig. 3. A delegate MAS is a swarm of ant agents, and can provide various kinds of services.

The issuing agent, representing a domain entity, may simultaneously have several delegate MAS, each rendering a specific service, and it may use a combination of delegate MAS to handle a single one of its concerns. For example in resource allocation problems, two distinct delegate MAS often prove useful: a swarm of exploration ants that seek out possible routings among resources on behalf of a task, and a swarm of intention ants that communicate a task's likely routing back to the resources. The issuing agent controls the number of ant agents, their program, and their parameter settings. The number is bounded at any instant. Each ant agent may only perform a bounded computational effort within its bounded lifetime and has a bounded footprint (memory). In other words, a delegate MAS is (computationally) efficient by design. However, the ‘program’ of an ant agent is not constrained otherwise.

The ants in a delegate MAS deposit, observe, and modify information (pheromone) in the virtual counterpart of the real world. This information can be any kind of data structure; it is not limited to vectors of scalars. Moreover, the environment in which the information is deposited may transform this information. For instance, bookings made by intention ants are inserted into a resource agent’s planning scheme. All pheromone information has an expiration time (evaporation).

Finally and most importantly, a delegate MAS delegates in two manners. First, the issuing agent assigns a responsibility to the delegate MAS. Second, the ant agents delegate to the environment in which they travel and evolve. For instance, exploration ants query resource agents about expected processing times, processing results, transportation times, etc. Intention ants delegate the local scheduling to the resource agents. Exploring ants use product agents to evaluate routing options. This extreme usage of delegation enables a delegate MAS to cope with a dynamic, heterogeneous and unpredictable world. Its design nowhere assumes that data structures suffice to capture the diversity of the problem domain.

Real-time Resource Allocation and Task Execution.— Delegate MAS have been developed for applications that perform real-time resource allocation and that supervise the execution of the tasks requiring these resources. The resources and tasks are diverse and heterogeneous. Furthermore, competitive performance requires resource allocation and task execution to account for the specific nature of both the resources and the tasks, and their interactions. Moreover, the system must be able to deal with multiple allocations and task execution steps ahead in time. The applicability of delegate MAS outside this domain remains an open issue.

Until now, Manufacturing Execution Systems (MES) constitute the main application area for delegate MAS research. The first application targeted by the research addressed car body painting. The design of this pioneering implementation has been improved in subsequent developments, addressing a confection flow shop, a machine tool shop and a heat treatment facility respectively [18, 23]. Other application areas that have been explored are railway systems, traffic control and supply networks [3, 20]. We briefly discuss two example applications.

Manufacturing Execution Systems.**—** In the MES prototypes, the issuing agents are PROSA (Product-Resource-Order-Staff Agent) [22] agents . All PROSA agents have counterparts in reality, which facilitates integration and consistency (indeed, reality is fully integrated and consistent). The main PROSA agents in the MES are:

- Resource agents reflecting the actual factory. They offer a structure in cyber space on which other agents can virtually travel through the factory.
- Order agents reflecting manufacturing tasks.
- Product agents reflecting task/product types.

Both resource agent and order agents issue delegate MAS. A single agent may have several delegate MAS. Each delegate MAS has its own responsibility.

Resource agents use a delegate MAS to make their services known throughout the manufacturing system. Ant agents collect information about the processing capabilities of resources while virtually traveling through the factory. These *feasibility ants*

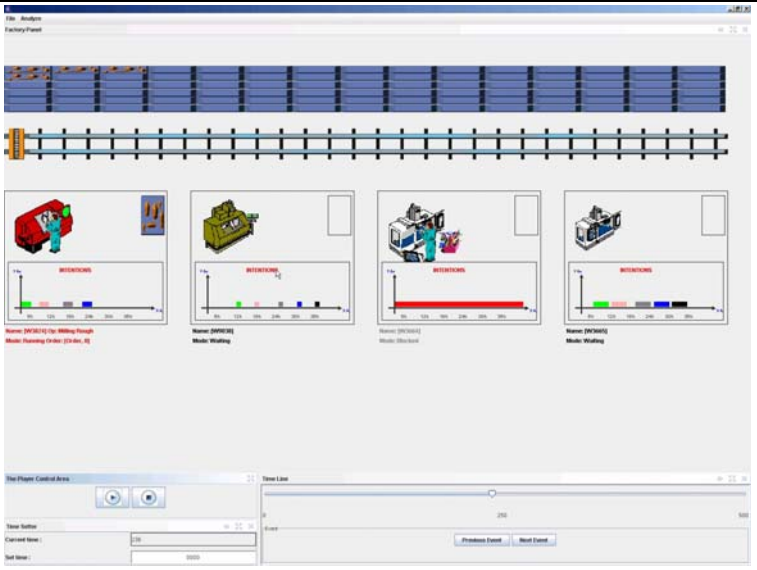
Fig. 4. Intention ants notify resource agents about the intentions of their respective order agent to occupy resources at a specific time slots in the near future. Resource agents use this information to self-schedule.

deposit this information (pheromone) at positions in cyber space that correspond to routing opportunities.

Each order agent is an issuing agent for a delegate MAS in which *exploring ants* scout for suitable task execution scenarios. In addition, each order agent is an issuing agent for a second delegate MAS that informs the resource agents of its intentions: *Intention ants* regularly reconfirm bookings for slots at resources (Figure 4). Specific manufacturing execution systems employ additional delegate MAS to deliver case-specific services [21].

Traffic Control System.— We have applied delegate MAS in an experimental traffic control system that proactively tries to predict and avoid road congestion [7]. Each car in the system is represented by a task agent, while road segments and crossroads are represented by resource agents. Task agents use resource agents to book a certain road in advance trying to avoid road congestion. Three types of delegate MAS are used: (i) resource agents issue feasibility ants to gather information about the underlying environment (which roads lead to which destinations). (ii) task agents issue exploration ants to gather information about the costs of possible routes; (iii) task agents issue intention ants to book the best possible route. A booking must be refreshed regularly to maintain the reservation.

We have applied the delegate MAS approach to the Leonard crossroad, a well-known Belgian congestion point between the Brussels Ring and the E411 Motorway (Figure 5). Tests for a realistic morning peak scenario show a reduction of 26% of congestion time for an increase of only 4% of extra traveled distance.

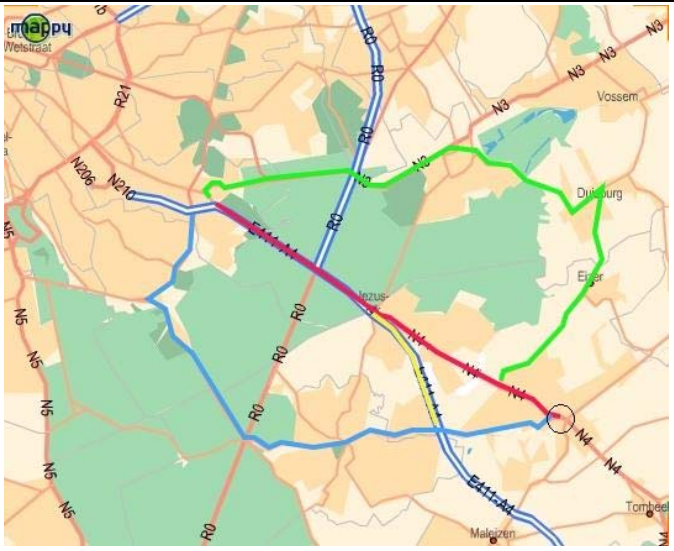

Fig. 5. A car at the circle (at the bottom right) has three possible routes: straight ahead, left, and straight on and then right. In the current situation, the car follows the route straight ahead, which has the minimal cost.

2.3 Complementarities

The Polyagent and Delegate MAS models were developed independently of one another, and thus have differing objectives.

The main insight in the Polyagent model is that multiple representatives of a single agent can be used to explore alternatives for that agent. Thus it emphasizes the relation between the single persistent avatar and the multiple transient ghosts.

The main insight in the Delegate MAS model is that a swarm of agents can perform some service in support of a larger system. A delegate MAS does not include an avatar, though it is typically used to support the reasoning of a single agent. Thus a delegate MAS can be viewed as a part of a polyagent, the swarm of ghosts that is associated with an avatar.

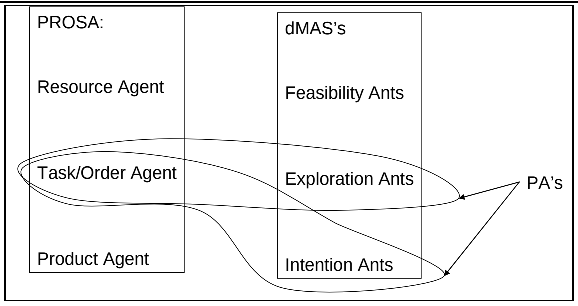

Fig. 6. A delegate MAS is a swarm of functionally homogeneous agents that explore multiple alternatives concurrently. A Polyagent uses a delegate MAS to explore alternatives for a single, usually more complex, agent, the Avatar.

Figure 6 makes this orthogonal relationship explicit.

The left-hand side of the picture shows the agent types in a conventional MAS, the PROSA architecture for manufacturing control [2]. The right hand shows the three delegate MAS that KUL has developed to support a PROSA system. Feasibility Ants explore connected sequences of resources, Exploration Ants estimate the quality of a particular sequence of resources on behalf of a task agent, and Intention Ants propagate the task agent's current intentions back to the resources so that they can schedule their availability.

This figure contains two polyagents (or a single polyagent with two types of ghosts), and three different relations between conventional agents and delegate MAS.

The Exploration Ants and Intention Ants both work on behalf of the Task Agent, which therefore constitutes an Avatar under the definition of a polyagent.

For Feasibility Ants two alternative designs can be considered. In [6], Feasibility Ants do not represent a single resource agent, but construct virtual routes through the network of resource agents. Thus they support the system as a whole, but are not part of a polyagent. An alternative implementation is possible in which each Resource agent sends out its own Feasibility Ants to deposit a quantitative pheromone on upstream resources. The relative strength of this pheromone would encode the relative distance of the node from the issuing resource, and thus enable Exploration Ants to assess the sequence of resources available from a given node. In this alternative implementation, a resource and its feasibility ants would constitute a polyagent.

Product Agents do not use the services of any delegate MAS, either directly or indirectly. They show that delegate MAS may be applied to part of a MAS, while other parts function using conventional MAS mechanisms.

To discuss these families of systems together, we need to establish some common vocabulary, which at points may differ from the vocabulary used in our previous papers. We propose (Figure 7):

A **delegate MAS** is a swarm of agents that provide some service for a higher-level agent system.

A **Ghost Agent** is one agent in a delegate MAS (where the term "ant" was used in the original Delegate MAS papers).

A **Polyagent** is the combination of a high-level agent with one or more delegate MAS. The recognition that a single polyagent can include several delegate MAS is an extension of the original polyagent model. All of the ghosts in a single delegate MAS have the same function, but the different delegate MAS in a single polyagent support different functions.

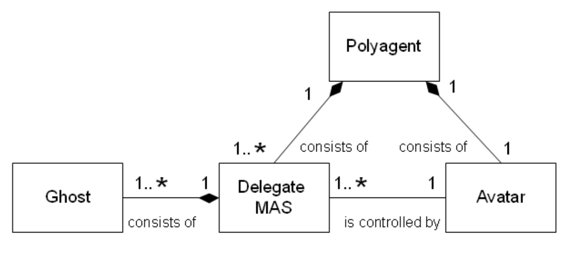

Fig. 7. An EPU system may use several Polyagents. Each Polyagent is the combination of an Avatar and one or more delegate MAS. Each delegate MAS may render a specific service for the Avatar, and the Avatar may use a combination of delegate MAS to handle a single one of its concerns.

An **Avatar** is the agent paired with a one or more delegate MAS in a polyagent.

An **Entity** is something in the domain that is represented by an agent.

An **EPU system** is any system that draws on the constellation of ideas that we bring together in this paper. The acronym EPU recalls our motto, *e pluribus unum*.

When we need to refer to the characteristics of specific systems that one or the other of our teams has previously constructed, we will designate them by NV (NewVectors) or KUL (Katholieke Universiteit Leuven).

Integrating these two models yields clear benefits to both of our teams.

From the dMAS perspective, the concepts of Avatar and Polyagent in an EPU system provide explicit architectural constructs for designing systems.

From the Polyagent perspective, the delegate MAS construct encapsulates a swarm of ghosts; this providing a clean approach to associate different types of ghosts (in terms of different delegate MAS) with a single Avatar/Polyagent

3 Design Considerations

As developers of real-world applications, we want to distill our experience into engineering guidelines for future exploitation of EPU systems. In this section we develop an integrated list of the domain characteristics for their application, which reflect similarities between our respective systems. The differences between our systems reveal the variability along which EPU systems can be developed.

3.1 Domain Characteristics

Some of the domain characteristics for applying EPU systems are shared with other multi-agent systems. Other characteristics are peculiar to our approach.

3.1.1 Common Domain Characteristics

Dynamism.—Many domains are in constant change, and a MAS to manage them must be able to consider alternatives rapidly in order to adapt to this change. Using the parallelism of a delegate MAS is one way to support this dynamism.

Locality of Decision and Action.—Agent systems naturally lend themselves to domains in which the primary information sources and the loci of action available to an agent are localized in some topology. This characteristic is especially valuable for EPU systems. The idea of using many representatives (the ghosts) to explore alternatives concurrently requires that the ghosts execute extremely efficiently, and both of our approaches use environmentally-based coordination via digital pheromones to enable light-weight ghosts. Such techniques are most effective when there is a strong correlation between an agent's location and its information and potential actions.

Going Concerns.—Systems can conveniently be divided into problem solvers (typically activating a tightly coupled community of agents to reach an achievement goal, at which point the system can shut down) and going concerns (using a more loosely coupled set of agents to support a maintenance goal that requires constant attention) [10]. The ability of EPU systems to deal with dynamism makes them particularly valuable for handling going concerns.

3.1.2 Specific Domain Characteristics for EPU Systems

Temporal Constraints.—The delegate MAS must run fast enough to be of service to the real-world system it is supporting. This requirement usually means that the ghosts must be able to more faster than the avatar. Otherwise, one might just as well let the avatar do the search.[1] One broad class of systems that satisfies this condition is systems dealing with the movement of physical entities. Physical constraints usually slow the movement of such entities so that ghosts, which can move at cyber-speed, can explore alternative trajectories faster than real time.

Space of Multiple Options.—The replication of ghosts in an EPU system has the purpose of exploring alternatives concurrently. Such techniques are more useful as the problem space presents higher levels of combinatorial complexity.

Simulation-Friendly.—The ghosts must be able to simulate the problem domain efficiently. This requirement is supported by two further characteristics:

- **Simulation Models**.—Ghosts need efficient simulation models or other mechanisms to evaluate single options quickly, at least to a rough level of accuracy.
- **Constrained Problem Space**.—If a system is highly constrained, its behavior may be relatively insensitive to details of individual agent actions, permitting the use of simplified models. We call this characteristic "universality" [17]. Such constraints may arise in two ways. First, the static structure of the environment may reduce the options that agents can follow. Second, the dynamics of the system may exhibit a few large basins of attraction leading to large equivalence classes in the space of possible solutions.

3.2 Variability in Applying EPU Systems

Contrasts between our systems reveal a number of degrees of freedom that can be exploited in engineering an EPU system for a particular application.

Locus of Functionality.— KUL provides a plug-in architecture that allows the behavior of ghosts to be modulated by injecting additional functionality, while NV's implementations tend to have monolithic ghosts.

How Smart is a Ghost?—Previous NV applications tend to use stigmergic agents, but this is not a requirement for application of EPU system's. The KUL plug-in architecture can use any computational method, so long as it respects the temporal constraints outlined in Section 3.1.

Ghost Interaction.—Past NV applications of polyagents tend to take advantage of interactions among ghosts of the same avatar, mediated environmentally. For example, path planning depends on positive feedback among ghosts representing the same entity. In this approach, the ghosts in a polyagent not only *retrieve* knowledge from the environment, but actually *generate* new knowledge, reducing the decision-making

[1] This restriction is not strictly true. Even if ghosts only move at the same speed as their avatars, an EPU system may still be of some value on physically parallel hardware, enabling multiple alternatives to be evaluated in parallel with one another.

load in the avatar. KUL ghosts could behave this way, but currently do not. The result is to place more responsibility for decision-making on the avatar. However, KUL's ghosts do interact with those of other types, in that exploration ghosts read symbolic pheromones written by feasibility ghosts.

Writing to the Environment.—Closely related to ghost interaction is the question of whether or not ghosts can change the state of the environmental nodes that they visit. Three alternatives are available.

1. Ghosts can read from the environment but not write to it. This is the approach taken by the exploration ghosts in the KUL factory control system.
2. Ghosts can leave information in the environment for use by other ghosts, either of the same types or of different types. The first approach is used in NV's path-planning application, where it enables the generation of information (a routing) by positive feedback among the ghosts. The second is used in KUL's factory control system, where exploration ghosts read the accessibility information left by feasibility ghosts.
3. Ghosts can leave information for consumption by the environment, as do the intention ghosts in KUL's system.

Ghost Speciation.—In KUL's factory control system, a single task agent has two separate delegate MAS, one for exploration and the other for propagating intentions. NV's applications have had a single type of ghosts for each avatar, and Brueckner's factory architecture [1] used pheromones deposited by a task agent's ghosts to estimate the level of intention that the task agent has for a given resource.

Time Management.—Frequently, the space of alternatives over which ghosts are searching extends over time, and ghosts need some way to distinguish different future times from one another. One alternative, used by KUL and in Brueckner's early work, associates a timeline with each entity in the system that the ghosts may encounter, an approach we call *entity priority*. The other, used in NV's more recent systems, maintains a book of pheromone pages, each for a successive moment in time, and all entities are represented on each page. We call this approach *time priority*. The two alternatives pose an interesting trade-off. In an entity-priority system, a ghost on an entity's agent can easily compare the state of that entity at different times, but to compare different entities at the same time, it must move from one to the other. With time priority, a ghost can efficiently see the effect of multiple entities at a moment in time (to the degree that their pheromone fields overlap), but to see the state of one entity at different times, it must move through time.

These two models have evolved naturally in the domains in which KUL and NV have developed their systems. Two factors have motivated the respective decisions: the kinds of entities involved, and the nature of the reasoning to be done.

In most factory settings, resource agents and task agents behave in space-wise local but time-wise spread-out ways. This distinction makes it natural that timelines are local to agents representing application entities (resources, tasks). Indeed, the manufacturing machines are independent of one another, so entity priority enables faster selection of candidate time slots without significant sacrifice. In the combat modeling addressed in NV's latest systems, the most important entities are all combatants, and whatever Red does to Blue, Blue may also consider doing to Red. So the asymmetry that makes it feasible to manage time on one type of entity and not the other in the factory setting is not available. In addition, in combat, it is more important to understand how entities are interacting with one another than it is to examine a single entity's evolution over time, so time priority is preferable.

4 Future Work

Up to this point, EPU systems have been driven by the needs of specific problems. As we refine the approach into a reusable architectural approach, several issues require further investigation. We record them here as a roadmap for our own activity, and to invite other researchers to join us in extending this powerful approach.

Architectural Patterns for EPU Systems—The conceptual architecture of EPU systems described in Section 2.3 (Figure 7) represents a generic architectural pattern to develop agent systems for domains that satisfy the characteristics described in Section 3.1. The conceptual architecture describes the essential architectural elements (avatars, delegate MAS, ...) and relation types together with a set of constraints on how it may be used to build EPU's.

From experience with particular classes of EPU systems, we have derived concrete instances of the general architectural pattern. Figure 8 shows one recurring pattern.

This architectural pattern consists of two specific Avatar types: Task and Resource Avatars that extend the general Avatar type. Three specialized delegate MAS provide BDI-like functionality through the environment to the Task Avatar:

- **Feasibility delegate MAS** are managed by Resource Avatars and build up beliefs about the environment (e.g., feasible production paths through a factory or a map in a traffic environment)
- **Exploration delegate** MAS are managed by Task Avatars and build up desires (e.g., useful paths through the factory to produce a particular product or useful routes in a traffic application)

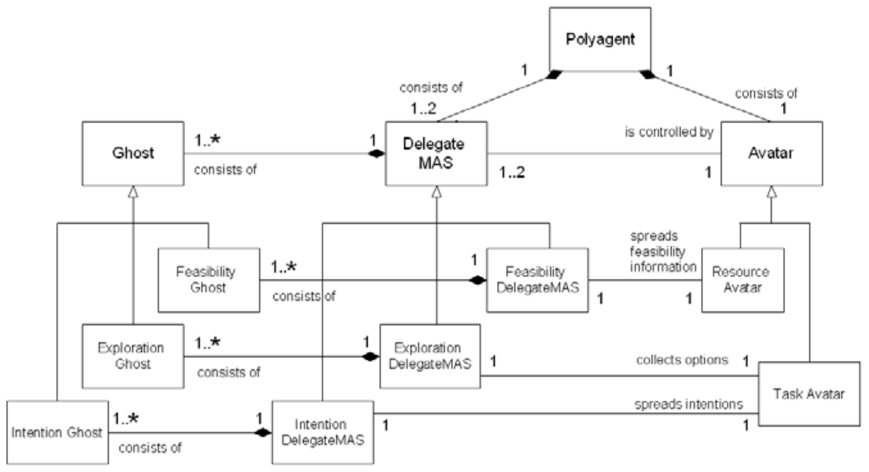

Fig. 8. Resource Avatars issue Feasibility Delegate MAS and Task Avatars issue Exploration and Intention Delegate MAS to obtain BDI-like functionality through the environment.

- **Intention delegate** MAS are managed by Task Avatars and build up intentions (e.g., paths of booked resources to produce a product or selected routes in a traffic application)

This architectural pattern is one specific instance of the general EPU pattern. Such an architectural pattern provides a reusable asset for engineers to build new EPU applications. An interesting venue for future work is to derive other architectural patterns for EPU systems.

Dynamics.—Any system with multiple interacting nonlinear components has the potential for complex dynamics that may either support or compromise the purpose for which the system has been constructed. EPU systems multiply the problem by embedding many MAS within a single system. Thus special attention must be paid to issues such as convergence, stability, and catastrophic discontinuities in behavior.

When an EPU system is used to reason into the future, one particular dynamic concern is of special interest. Nonlinear systems can exhibit chaotic behavior that causes the trajectories emanating from nearby points in state space to diverge arbitrarily far from one another, making long-range prediction impossible. Short-range prediction is still possible, and by continuously generating short-term predictions, one can reliably move ahead, as we have shown elsewhere [15]. But it is important to estimate just how far ahead predictions are meaningful, and at what point (the "prediction horizon") they become no better than random noise. We discuss some preliminary steps to studying this problem elsewhere [12]. Much remains to be done in enabling individual avatars to estimate how far into the future they should let their ghosts search, thus improving both their efficiency and the accuracy of the information that they produce.

Mechanism Design.—Current EPU systems are closed, with a single developer who can ensure honest behavior on the part of avatars and ghosts. We expect the technique to become more widespread and enter the general toolbox of MAS developers, a prospect enhanced by the modular plug-in architecture being developed at KUL. EPU systems may have components developed by different parties, and in this case we need to give attention to the possibility that a delegate MAS may conduct deception on behalf of its avatar (or be deceived by the ghosts of another avatar). This problem is a generalization of the problem of deception and reliability in MAS in general, and techniques for mechanism design need to be adapted to this setting.

One promising approach is the use of reputation maintenance mechanisms. The attractiveness of EPU systems to going concerns (discussed in Section 3.1.1) means that avatars (or the organizations that issue them) can be expected to appear repeatedly, and other entities can learn their reliability. For example, in the KUL factory system, if task agents from a particular source regularly renege on an unusually high percentage of their reservations, it would be natural for resource agents to discount further reservations from those agents to avoid being undersubscribed. In this example, an agent's reputation transforms its statements into an expectation based on the statement, with an associated level of uncertainty.[19]**.**

Guarantees vs. Adaptability.—The large space of alternatives that a delegate MAS can explore can make a system more adaptable, but also less predictable. In many commercial settings, users require firm guarantees, at least on lower-bound behavior. Providing such guarantees requires ways to balance the exploration of the system against fixed behaviors that may be less adaptable but more predictable.[4].

Integration with Legacy Systems.—Rarely will an EPU system completely supplant an existing system. It is more likely to be applied to part of the system, to give some advantage such as improving performance, adapting more rapidly to changes, or reducing variance. The techniques for achieving such integration are a rich field for study. In some cases, it may be possible to run the EPU system alongside the existing system and use its outputs selectively to adjust the legacy system. In other cases, one may embed components of the legacy system into the EPU system to provide specific functions, which must then interact with the functions being provided by the EPU system. Both approaches (and others) invite investigation.

Interference.—The need for ghosts to run efficiently makes pheromone-based coordination attractive for delegate MAS, but leads to a problem. As ghosts explore alternative futures, they may deposit pheromones along different paths. Sometimes both paths are reasonable, but in other cases they are mutually exclusive, and it can be difficult to distinguish the two cases. More generally, the problem is that pheromones can accumulate and decay, but cannot interfere with one another, and over time the pheromone space can become muddy. As a result, when many futures are being explored, the space becomes muddy. If competing options could interfere, one could cancel out the other, avoiding the muddiness.

Autonomic Capabilities.—Delegate MAS are a promising approach to addressing the self-X capabilities required for autonomic computing: self-monitoring, self-adjusting, self-healing, and so forth. In this case, the ghosts' focus is inward, on the components of the system, rather than outward toward the application domain. Developing idioms and mechanisms for these functions is an important research objective, and will greatly increase the potential for open EPU systems.

5 Conclusion

Traditionally, agents in a MAS are mapped to domain entities either one-to-one, or according to a functional decomposition. Sometimes it is advantageous to assign multiple agents with the same function to a single domain entity, in order to explore alternatives concurrently. When the single agent is controlling a physical system, its multiple representatives constitute a set of interacting agent-based simulations exploring the combinatorial space through which the single agent moves. Such an approach yields an EPU system. Between our two research teams, we have constructed EPU systems in several domains, including manufacturing control in several industries, supply chains, traffic control, robotic routing, and combat prediction. By examining our techniques together, we have been able to identify both the conditions under which such an approach is useful, and a number of design choices that are available to engineers who wish to exploit this technique for future applications. The approach opens up a range of interesting and important questions for further research.

6 References

[1] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Dr.rer.nat. Thesis at Humboldt University Berlin, Department of Computer Science, 2000. http://dochost.rz.hu-berlin.de/dissertationen/brueckner-sven-2000-06-21/PDF/Brueckner.pdf.

[2] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference Architecture for Holonic Manufacturing Systems: PROSA. *Computers In Industry*, 37(3):255-276, 1998. http://www.irit.fr/TFGSO/DOCS/TFG2/TFGIISO_Valckenaers.pdf.

[3] K. DeSwert, P. Valckenaers, B. Saint German, P. Verstraete, K. Hadeli, and H. Van Brussel. Coordination and Control for Railroad Networks Inspired by Manufacturing Control. In *Proceedings of IEEE 2006 Workshop on Distributed Intelligent Systems*, pages 201-206, IEEE, 2006. http://doi.ieeecomputersociety.org/10.1109/DIS.2006.21.

[4] K. Hadeli. *Bio-Inspired Multi-Agent Manufacturing Control System with Social Behaviour*. Thesis at Katholieke Universiteit Leuven, Department of Department of Mechanical Engineering, 2006.

[5] T. Holvoet and P. Valckenaers. Beliefs, desires and intentions through the environment. In *Proceedings of 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Hakodate, Japan, pages 1052-1054, 2006.

[6] T. Holvoet and P. Valckenaers. Exploiting the Environment for Coordinating Agent Intentions. In *Proceedings of Third International Workshop on Environments for Multi-Agent Systems (E4MAS06)*, Hakodate, Japan, Springer, 2006.

[7] E. Huard, D. Gorissen, and T. Holvoet. Applying delegate multi-agent sytems in a traffic control system. CW 467, Katholieke Universiteit Leuven, Leuven, Belgium, 2006. http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW467.abs.html.

[8] J.-D. Kant and S. Thiriot. Modeling one Human Decision Maker with a MultiAgent System: the CODAGE Approach. In *Proceedings of Fifth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS06)*, Hakodate, Japan, ACM, 2006.

[9] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997. http://www.newvectors.net/staff/parunakv/gotoant.pdf.

[10] H. V. D. Parunak. From Chaos to Commerce: Practical Issues and Research Opportunities in the Nonlinear Dynamics of Decentralized Manufacturing Systems. In *Proceedings of Second International Workshop on Intelligent Manufacturing Systems*, Leuven, Belgium, pages k15-k25, Katholieke Universiteit Leuven, 1999. http://www.newvectors.net/staff/parunakv/ims99.pdf.

[11] H. V. D. Parunak. Real-Time Agent Characterization and Prediction. In *Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Industrial Track*, Honolulu, Hawaii, ACM, 2007. http://www.newvectors.net/staff/parunakv/AAMAS07Fitting.pdf

[12] H. V. D. Parunak, T. Belding, and S. Brueckner. Prediction Horizons in Polyagent Models. In *Proceedings of Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07)*, Honolulu, HI, pages (submitted), 2007.

[13] 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. http://www.newvectors.net/staff/parunakv/MABS06Polyagents.pdf.

[14] H. V. D. Parunak and S. Brueckner. Modeling Uncertain Domains with Polyagents. In *Proceedings of International Joint Conference on Autonomous Agents and Multi-*

Agent Systems (AAMAS'06), Hakodate, Japan, ACM, 2006. http://www.newvectors.net/staff/parunakv/AAMAS06Polyagents.pdf.

[15] H. V. D. Parunak, S. Brueckner, R. Matthews, J. Sauter, and S. Brophy. Real-Time Evolutionary Agent Characterization and Prediction. In *Proceedings of Social Agents: Results and Prospects (Agent 2006)*, Chicago, IL, Argonne National Laboratory, 2006.

[16] H. V. D. Parunak, S. Brueckner, and J. Sauter. Digital Pheromones for Coordination of Unmanned Vehicles. In *Proceedings of Workshop on Environments for Multi-Agent Systems (E4MAS 2004)*, New York, NY, pages 246-263, Springer, 2004. http://www.newvectors.net/staff/parunakv/E4MAS04_UAVCoordination.pdf.

[17] H. V. D. Parunak, S. Brueckner, and R. Savit. Universality in Multi-Agent Systems. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, New York, NY, pages 930-937, ACM, 2004. http://www.newvectors.net/staff/parunakv/AAMAS04Universality.pdf.

[18] P. Peeters, H. Van Brussel, P. Valckenaers, J. Wyns, L. Bongaerts, M. Kollingbaum, and T. Heikkila. Pheromone based emergent shop floor control system for flexible flow shops *Artificial Intelligence in Engineering*, 15(4 (Oct)):343-352, 2001.

[19] B. Saint Germain, P. Valckenaers, P. Verstraete, O. Bochmann, and H. Van Brussel. Supply Network Control, An Engineering Perspective. 2004.

[20] B. Saint Germain, P. Valckenaers, P. Verstraete, and H. Van Brussel. Resource coordination in supply networks. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 1972-1978, 2004.

[21] P. Valckenaers, K. Hadeli, B. Saint Germain, P. Verstraete, and H. Van Brussel. Emergent short-term forecasting through ant colony engineering in coordination and control systems. *Advanced Engineering Informatics*, 20(3 (July)):261-278, 2006.

[22] P. Valckenaers and H. Van Brussel. Holonic manufacturing execution systems *CIRP Annals of Manufacturing Technology*, 54(1):427-432, 2005.

[23] C. Zamfirescu, P. Valckenaers, H. Van Brussel, and B. Saint Germain. A case study for modular plant control In, *Holonic and Multi-Agent Systems for Manufacturing*, vol. LNAI 2744, pages 268-279. Springer, 2003.