Delegate MAS Patterns for Large-Scale Distributed Coordination and Control Applications

Tom Holvoet
DistriNet labs
Dept. of Computer Science,
KULeuven, Belgium
Tom.Holvoet@cs.kuleuven.be

Danny Weyns
DistriNet labs
Dept. of Computer Science,
KULeuven, Belgium
Danny.Weyns@cs.kuleuven.be

Paul Valckenaers
Center for Industrial Mgt
Dept. of Mechanics,
KULeuven, Belgium
Paul.Valckenaers@mech.kuleuven.be

ABSTRACT

This paper documents patterns that support coordination in large-scale, dynamic, distributed systems. The patterns are related, and are linked to existing patterns as documented in literature (e.g. by Yaridor and Lange [AL98], Kendall e.a. [KKPS98]). This paper refines the patterns as described in [HWV09].

The patterns are DISCIPLINED FLOOD, PROPERTINERARY, SMART MESSAGE, DELEGATE MAS, and DELEGATE ANT MAS. Identifying the patterns fosters reuse of particularly useful coordination techniques, and can serve as a catalyst for new or altered approaches.

1. AUDIENCE

The patterns in this document are particularly intended for researchers as well as practitioners who study and develop large-scale decentralized systems - including decentralized control systems, internet applications. The reader should be familiar with typical issues and challenges in developing distributed systems, and be acquainted with elementary terminology of agents and multi-agent systems (MAS, [Woo09]), and the discrete optimization metaheuristic called Ant Colony Optimization [DDC99].

2. MOTIVATION

The complexity of large-scale distributed applications has motivated several researchers and practitioners to study and develop self-organizing systems. A self-organizing system is typically composed of a large number of components that interact and cooperatively reach the system objectives. The global behavior of the system emerges from local interactions. Engineering self-organizing systems is, however, known to be quite a challenge.

Literature is flooded with technical descriptions and experiments of self-organizing systems in different application domains. Networking, middleware, distributed optimization, distributed simulation, logistics management, peer-to-peer systems, ... These systems are more often than not the result of quite smart engineering that combines clever technical solution ideas with expert domain knowledge that can be exploited to obtain performant and flexible systems.

One example is literature that reports on particular coordination mechanisms for large-scale decentralized systems. Research on ‘delegate MAS’ for manufacturing control and traffic management [HV06, WHH07], Polyagents for military applications and manufacturing control [PB07], mobile agents for internet applications such as e-commerce or tourist assistance [KMTU03, CPV97], bio-inspired network routing [CD98, CDG05] and bio-inspired distributed middleware management [LS08]. Several of these works are inspired by ant colony optimization techniques [DDC99]. Some focus on the integration of coordination in e.g. a BDI-based agent architecture (Beliefs-Desires-Intentions). Some include symbiotic simulation. Within their respective application domains, these approaches yield valuable solutions by combining clever ideas that relate to coordination mechanisms. In our research, we observe that these approaches bear similarities in their key solution techniques.

For clever solution techniques to prosper beyond a single system and become reusable assets for other software engineers in building other applications, they can be described in a more generic, reusable fashion. Patterns are one of the most appreciated instruments for reuse in software engineering. Patterns emerge from frequent use and experience. They identify a generic problem, a suitable generic solution scheme including solution quality characteristics. As such, engineers can reuse proven solutions, and can be inspired by them in solving similar problems.

3. SETTING THE SCENE: ANTICIPATORY VEHICLE ROUTING

A concrete example of a large-scale, dynamic distributed control system can help set the scene for the patterns in this paper, although without going in too much technical details on how the problem can be solved. The example is vehicle routing coordination in traffic.

Everyone is familiar with traffic and - unfortunately - with traffic jams. Traffic jams are not only a source of huge economic loss with substantial influence on environmental deterioration, they are considered to extremely inconvenient for road users. Often, however, traffic jams could be avoided altogether. Traffic jams are often a consequence of poor usage of available resources. If all road users would have relevant and accurate information (including forecast traffic density) of alternative routes towards their destination(s), they could make a better choice, reducing traffic jam waiting time and travel time.

Typical satellite navigation (satnav) devices can help people find the fastest route to their destinations - at least, under the assumption that there is no one else out there. More advanced satnav devices are able to make use of *current* traffic information, and may suggest a detour if a road has been blocked. Anticipatory vehicle routing takes routing one step further, by taking into account traffic forecasts in calculating the travel time for alternative routes. Statistical approaches to anticipatory vehicle routing are useful for traffic flows with regular patterns. In case of disturbances (e.g. road blocks due to accidents or road works), statistical approaches do not yield satisfactory results. At best, road users should be able to take into account the actual intentions of other users. If a road user would need to go from point A to point B, and knows that there are three alternative routes, it would be interesting to know whether or not other road users intent to massively use a part of one of the routes from A to B. The driver could take this into account and choose for another route.

It is clear that route guidance is a coordination problem that is huge in terms of underlying infrastructure as well as in number of actors involved, it is submitted to various sources of dynamics - road blocks, road infrastructure constraints, road users entering and leaving the system continuously, and many more.

A centralized approach to individual-based, anticipatory vehicle routing is hence infeasible for obvious reasons. For a decentralized approach, we will assume[1] vehicles to be equipped with electronic devices with computational capacity, which moreover are able to communicate wirelessly with other devices in their vicinity. Additionally, we assume road infrastructure elements (such as roads, cross roads) to be equipped with computation and communication devices as well.

While a solution for anticipatory vehicle routing is sought to reduce traffic jams and deal with traffic disturbances, the overhead that the solution can create is strictly bound by the computational and bandwidth capacity of the devices in the vehicles and road infrastructure.

The decentralized architecture as discussed in [WHH07] utilizes the devices in vehicles to assist in routing their respective drivers by coordinating with other devices of both vehicles and road infrastructure, i.e. traffic resources.

First, every vehicle - that is, their respective devices - would need to communicate with other vehicles and resources in order to (1) explore different routes towards its destination(s), (2) find out about the intended road usage of the other vehicles, and (3) to disseminate its own intentions. Clearly, sending direct messages to every individual vehicle and resource element is impossible. On the other hand, broadcasting or flooding the distributed environment for this purpose is unacceptable due to the tremendous overhead that such messaging policy would create. A vehicle, however, is interested in spreading a message is a 'disciplined way'; that is, the message can be provided with specific cloning behaviour that makes the message spread to those elements in the system that can be relevant for this message. If a vehicle wants to go from A to B, there is no point in sending a message towards a location C that is not part of any route between A and B.

Second, people may need to visit a number of destinations instead of just one. In some cases, the order of destinations can be fixed, in other cases the order is irrelevant, and yet other situations could prescribe other ordering relations between destinations. Hence, a message that is sent by a vehicle as discussed above may need to take this 'itinerary information' along, in order to judge upon the suitability of cloning itself in a particular direction.

In fact, vehicles could issue 'smart messages', which incorporate a disciplined flooding policy and itinerary information with mobile code and state to be executed in the nodes that the messages arrive at. For instance, a message that arrives at a node that represents a crossroad, could communicate with the device that manages this crossroad to find out about future reservations. As such, a smart message is a light-weight, mobile agent, and is sometimes referred to as ant-like agents or just ants.

Entities such as vehicles that would make use of such smart messages would need to repeat sending out these messages to make sure that the information that is either gathered from the environment or send to the environment is up to date. The component within the behaviour of these entities that manages sending smart messages is called a 'delegate MAS'. The vehicle 'delegates' behaviour to this management component, which manages the smart messages, i.e. which manages a set of mobile agents.

The approach to anticipatory vehicle routing as documented in [WHH07] defines vehicles that each use two instances of delegate MAS, each with their own types of smart messages. Figure 1 aims to visualise the approach. First, the 'exploration delegate MAS' of a vehicle is responsible for exploring different routes towards the destination(s) of the vehicle. Smart exploration messages - or exploration ants - are sent out over different feasible routes, and interact with the resource entities (i.e. the road infrastructure elements) through *what-if* scenarios. This allows the smart message to be informed about future traffic load at the time that its vehicle could arrive at this element. The infrastructure element would e.g.‘ if the vehicle would arrive here at time t, it would take the vehicle about Δt to pass through this element. The smart message clones towards the infrastructure element devices that represent elements that are physically connected to the current infrastructure element, and that could be a hop in the route towards the desination(s) of the vehicle. When it reached its goal, the smart message reports back to its vehicle. The vehicle collects such information and maintains one route as its intended route (e.g.

[1]The assumptions are made for the sake of setting the scene for the patterns, not because the authors are convinced that these are realistic assumptions today.

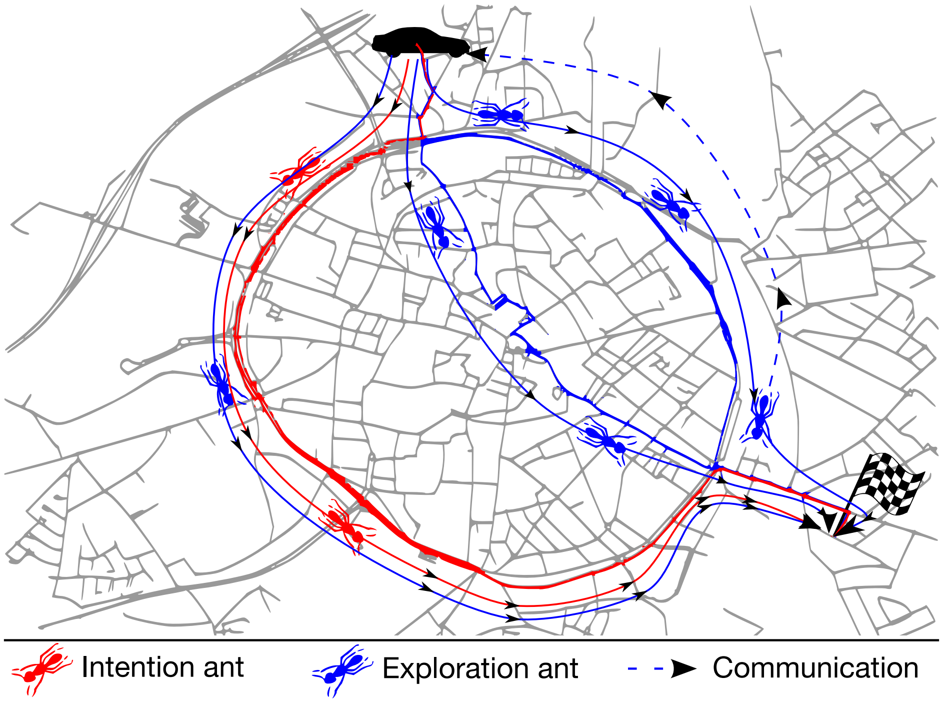

Figure 1: Ants exploring the environment of the city of Leuven (Belgium). Exploration ants, depicted in blue, explore the environment and aggregating forecast information. Based on this information, the vehicle chooses a route it intents to follow and informs the road infrastructure elements along this path using intention ants, depicted in red.

the fastest route to the destination(s)).

Second, the 'intention delegate MAS' of a vehicle is responsible for disseminating its own intended route to the infrastructure elements along this route. Smart intention messages - or intention ants - are equipped with information about the intended route and travel along the infrastructure to inform the fact that the vehicle intends to be at this resource at that particular point in time, and make reservations. The reservations are volatile and need to be refreshed regularly; reservations that are not refreshed are removed after some time. The intention smart messages report back with the exact information of what has been reserved. The vehicle can then decide to stick to this intended route or rather explore for a better alternative.

This case is one example of a broader domain of so-called 'coordination and control applications'.

4. COMMON CONTEXT AND FORCES

All patterns described in this document are related to a **common context**. The context consists of large-scale distributed systems for which applications must be developed. Applications that are considered are so-called 'coordination and control applications'. Examples of applications are traffic management, manufacturing control, distributed process management. Due to some non-functional requirements (e.g. scalability or robustness), centralized solutions are considered infeasible. Instead, the architects develop decentralized software architectures, where decentralized software entities (or agents) need to communicate and collaborate to coordinate their activities and collectively control the distributed system. Software entities that are deployed on network nodes can execute behavior and communicate with entities on other nodes in their vicinity.

The context dictates common **forces** to which solutions must adhere. First, the *large-scale* of the system makes it particularly challenging - one needs to ensure that all entities acquire relevant information to base their decisions upon, and vice versa, one needs to ensure that local information that is relevant for other entities is properly disseminated. Second, the applications are deployed on networked computer systems with limited *capacity* in terms of computational resources and bandwidth. Communication overhead should be limited, in number and size of messages. The additional computational complexity that decentralization brings should be limited as well. Third, these systems are susceptible to various sources of *dynamism* (including network topology changes, node or communication failures, software failures). Dynamism should not be considered as exceptional situations, but rather as 'business as usual'.

Patterns that have been documented before, and which are particularly useful in this context with the above forces, can be found in Kendall e.a. [KKPS98], Aridor and Lange [AL98]. In particular Aridor and Lange identified three classes of patterns for designing mobile agents. Traveling patterns support routing and quality of service behavior of mobile agents, task patterns are concerned with the breakdown of tasks and the way they are allocated to (mobile) agents, interaction patterns support designing how (mobile) agents interact. The layered agent pattern as described by Kendall e.a. [KKPS98] proposes a generic agent architecture based on well-defined layers.

These patterns assist developing decentralized applications using mobile agents, yet are not (and not meant to be) complete. The patterns described in this document make a useful addition to literature for the targeted application domain.

5. THE PATTERNS

The patterns that are proposed in this paper constitute several pieces that can be used to make a single puzzle. The patterns describe generic solutions for agent interaction and coordination. While the individual patterns are illustrated using scenarios from traffic management, more elaborate descriptions of known usage of the patterns are addressed in Section 6.

5.1 Disciplined flood

The first pattern describes a mechanism that a software entity can use for sending messages in a large-scale environment.

Problem / Motivation

How can a software entity send a message to other entities in the distributed environment, of which it may not know their locations, yet avoid unnecessarily flooding the environment with communication messages?

In the traffic case discussed earlier, a vehicle could need to send a message towards potential destinations, e.g. to find out whether there is a package at any of these locations that it could pickup. The vehicle does not have information about routes for the messages towards the destinations. Flooding the environment with messages is infeasible.

Solution

Software entities issue messages that encapsulate specific cloning behavior. The messages may be considered MESSENGER AGENTS, as presented in [AL98], that are enhanced with behavior logic: the messages will match information that is available at the current node (in particular, information on outbound nodes) with information about their own target. The matching process yields a probability value for each outbound node. The probability that a message is cloned towards an outbound node is determined by this value.

In the example of traffic coordination, vehicles can send such messages to their destination(s). The nodes that represent road infrastructure elements can maintain road signs (or routing tables) that indicate whether particular outbound nodes can lead to the target, and if so, provide an estimate of the distance. The messages' cloning behaviour uses this information in calculating the probability of cloning towards an outbound node.

Consequences

This solution allows to steer the flooding behavior of messengers based on local information. Depending on the quality of the local information, the disciplined flood allows to reduce usage of communication bandwidth. Using this patterns requires maintaining good quality of the local information related to the messengers' objectives. Care needs to be taken that messages do not clone for ever. Moreover, the message may not reach its destination altogether, because of the dynamic nature of the context (links may be broken, messages may be lost). This is not necessarily harmful, yet important for the message sender to be aware of - an autonomous software entity - or agent - is by definition unsure whether its action are or are not fulfilled [Woo09].

Related patterns

This pattern provides a generic solution to limited cloning behavior of flooding messages. Related to this pattern is, as mentioned, the MESSENGER AGENT pattern [AL98]. A MESSENGER AGENT allows software entities (or mobile agents) to asynchronously communicate with other entities while moving on (in terms of mobility and behavior). The DISCIPLINED FLOOD solution adds cloning and does not aim to reach one particular destination, but rather has its own objective to fulfill - i.e. reach any destination with a particular property.

5.2 Propertinerary

The ITINERARY pattern is a well-known pattern in mobile agent [AL98]. It is a so-called traveling pattern, that is concerned with routing a mobile agent among multiple destinations. An ITINERARY maintains a list of destinations and defines a routing scheme based on this list.

The PROPERTINERARY is a variation of the ITINERARY pattern. The pattern maintains a list of *properties* of destinations rather than destinations themselves, and defines a routing scheme based on this list.

Problem / Motivation

How can a mobile agent, that is sent out on behalf of another software entity, self-route in order to fulfill its objective if this objective is to visit a sequence of locations with particular properties?

In the example of the traffic case, a software entity representing the driver of a vehicle can issue a mobile agent to explore various routes that bring it along the school of the children, any grocery shop, and any post office, where going to the school must be visited first, but there is no preferred order for the other two activities (running errands and buying stamps).

Solution

A mobile agent is endowed with routing behavior that is based on an 'itinerary of properties' (we call this PROPERTINERARY). The itinerary of properties defines sequences of location properties. The itinerary can either be represented as a list, but could as well be a (directed or undirected) graph structure.

The routing behavior is guided by (1) an itinerary of properties, and (2) that agent state, that represents the locations (and their properties) visited so far. When the mobile agents needs to decide upon the next node to move to, it will consider which neighbouring nodes have properties that would match a next element in the itinerary, or that would bring the agent to a location that is closer to a location that matches the next property in the sequence. The locations themselves need to make their properties available for inspection, and could disseminate their properties to their neighbours. If the agent cannot find information that may help in deciding upon the next node, it either randomly visits its neighbouring nodes or dies off (e.g. using a maximum hop count).

Consequences

The propertinerary solution makes the concrete objective of a mobile agent explicit, and uses this to steer its routing in the distributed system. This solution does not ensure efficient achievement of its objective - it mainly provides a design structure that allows matching such an objective with local information.

Related patterns

The MESSENGER PATTERN [AL98] self-routes a message to one destination. The ITINERARY pattern [KKPS98] describes routing behavior based on a set of destinations that a mobile agent needs to visit. Instead of explicit destinations, the propertinerary describes sequences of properties of locations that need to be visited.

5.3 Smart message

A smart message structures the behaviour of a mobile agent and incorporates its reproductive (or cloning) behaviour, migratory (or self-routing) behaviour, and computational behaviour.

Problem / Motivation

How can a distributed entity, such as an agent, engage in complex interactions with (sequences of) various other entities in the large-scale and dynamic environment?

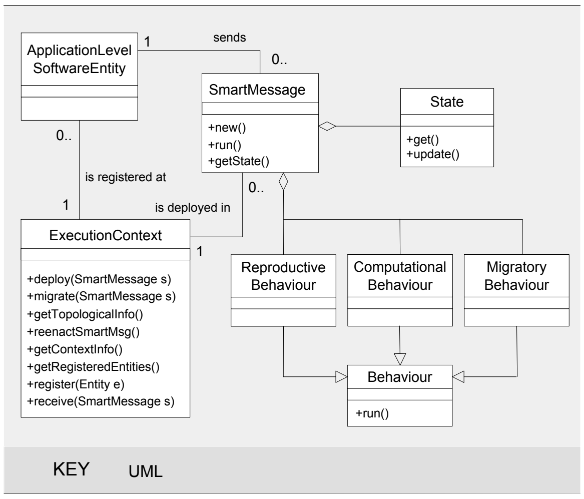

Figure 2: Smart message

In the example of traffic management, how can a vehicle explore various paths towards its destination(s), and for each path interact with the infrastructure elements in this path in order to find out about traffic forecasts? By aggregating traffic information about all elements in a route, one can assess the quality (e.g. travel time) of the route.

Solution

One can use a self-contained, mobile entity that is comprised of state and behavior, and that retains information about the entity that is responsible for the message (e.g. the sender) - see Figure 2. Such an entity is called a SMART MESSAGE. A SMART MESSAGE autonomously moves in the environment and interacts with nodes.

The behavior of a smart message includes:

- querying context information, and possibly interacting with local software entities,
- executing local computation that complies with the messages' objective,
- exhibiting migratory behavior that decides which node to move to next, and
- executing reproductive behavior which allows new smart messages or clones of itself to be created and spawned in the execution context.

A smart message is deployed in an execution context, i.e. an environment in which the message can perform its behavior and update its state, and which is statically linked to a node. The execution context offers services for

- creating new smart messages,
- migrating a smart message to a neighboring node,

- receiving and consequently re-enacting smart messages (i.e. triggering their behavior execution), and
- querying for context information, including for topological information (e.g. providing lists of neighboring nodes) and for other registered entities (e.g. providing references to agents that are active at the node).

In the traffic management example, vehicles can send out a smart message to explore the quality of various journeys that the vehicle could make. It can use a disciplined flood to limit communication overhead in its search, combined with a propertinerary of several destinations it aims to visit. In order to assess the time it would take the vehicle itself to perform the journey, the smart message can interrogate the nodes it passes to find out about their forecast traffic density. The smart message accummulates this information until it was able to explore an entire journey. At that time, the smart message changes its reproductive, migratory and computational behaviour such that it can resend itself to the vehicle that created the message in the first place.

Consequences

A smart message is a valuable instrument for software entities for communicating with other entities in a large-scale and uknown environment. Care should be taken that the reproductive behaviour of the smart message does not cause flooding (e.g. using a disciplined flood) or eternal traveling behaviour. Due to the limited communication capacity, the state and mobile code must be limited in size. Due to limited computational capacities of nodes, the behaviour must be of limited complexity.

Related patterns

Although there is no such thing as a 'mobile agent pattern', a smart message is an autonomous and mobile software entity and could be considered to be a mobile agent. As a mobile agent, the smart message routes itself and performs its behavior in order to reach its objective. Two aspects support presenting SMART MESSAGEs as a new pattern. First, the light-weight nature is crucial. Smart messages may not have a complex agent architecture, yet consist of three basic behavioral components. Second, smart messages have a particular subordinate role compared to high-level software entities - smart messages make a generic solution to communication and interaction between high-level software entities in large-scale distributed applications.

A SMART MESSAGE can make use of DISCIPLINED FLOOD to define its reproductive and migratory behaviour. It can use PROPERINERARY if visiting a sequence of locations (based on properties) is part of its objective.

The MESSENGER and ITINERARY patterns [AL98] describes an autonomous message that self-routes towards one ore more destination. Smart messages are self-contained messages endowed with their own objectives.

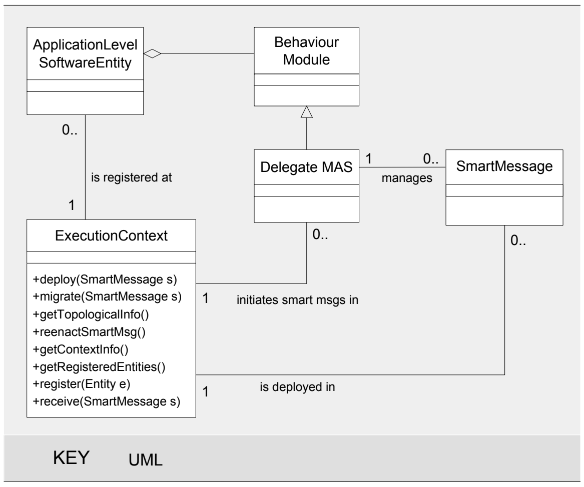

Figure 3: Delegate MAS

5.4 Delegate MAS

A SMART MESSAGE is a single, mobile unit. SMART MESSAGEs can effectively be used in a conglomerate, collectively executing a particular task or role on behalf of a software entity. Such a conglomerate needs to be managed properly - e.g. smart messages need to be sent out regularly, and the results that are collected by smart messages need to be processed and provided to the entity. A DELEGATE MAS represents such a managed conglomerate.

Problem / Motivation

How can a software entity make efficient and effective use of SMART MESSAGEs, without becoming too complex itself?

Solution

The presented solution is to delegate the management of the communication and interaction to a separate module that manages smart messages. A DELEGATE MAS (see Figure 3) is a *behavior module* [Mae90], i.e. a well-defined behavior that an agent can perform to reach a particular objective or task[2]. An agent's behavior consists of selecting and executing behavior modules, possibly in a concurrent manner. The agent itself manages the activation and deactivation of behavior modules, as well as the coordination between behavior modules. A behavior module is monitored and controlled by an agent, and fulfills a well-defined objective or task on behalf of the agent.

A delegate MAS is a behavior module that uses SMART MESSAGEs to fulfill its objective or task. As such, a delegate MAS is in charge of the management of the smart messages, and encapsulates a policy for creating smart messages (including a policy about timing and frequency of creating messages) with their own suitable (parameterized) behavior and ini-

[2]Other terms that are strongly related to behavior modules are capabilities, skills, or plans.

tial state, and spawning the messages through the execution context of the node. Additionally, the delegate MAS module collects the results that smart messages report back. The results are processed to meet the expectations that the agent has of this behavior module. Processed results are forwarded to the coarse grain agent for further interpretation (e.g. via a shared data space or an event-listener mechanism).

Consequences

This solution allows coarse-grain application-level software entities to delegate part of their communication and interaction behavior to a separate module. This supports managing the complexity. Using this pattern requires careful consideration of which aspects of the decision making can be fully delegated to this behavior module and which aspects must remain controlled by the application-level entity itself.

5.5 Delegate ant MAS

A delegate ant MAS is a delegate MAS that is specifically targeted for implementing distributed, ACO-like behavior. ACO stands for the ant colony optimization meta-heuristic [DDC99].

Problem / Motivation

How can software entities in large-scale distributed and dynamic environments, cooperatively build routing information.

The problem that is addressed by this pattern arises when application-level software entities need to find suitable routes through the graph-structured network *cooperatively*, hence yielding to more up-to-date routing information. As an example, assume vehicles that need to perform tasks (e.g. movement of goods, such as pickup and delivery services) needing to *route* towards pickup and delivery locations. Vehicles need to cooperate to improve the accuracy of the routing information.

Solution

While ACO is a discrete optimization meta-heuristic, the fundamental ideas of ACO can be applied for decentralized cooperative routing.

Delegate ant MAS is an ACO-inspired refinement of delegate MAS, see Figure 4. A delegate ant MAS manages *ant agents* instead of smart messages. We define an ant agent as a smart message of which the behavior and state is directly related to ant colony optimization techniques. The objective of an individual ant agent is to traverse the environment in search for a solution, on behalf of its delegate ant MAS to

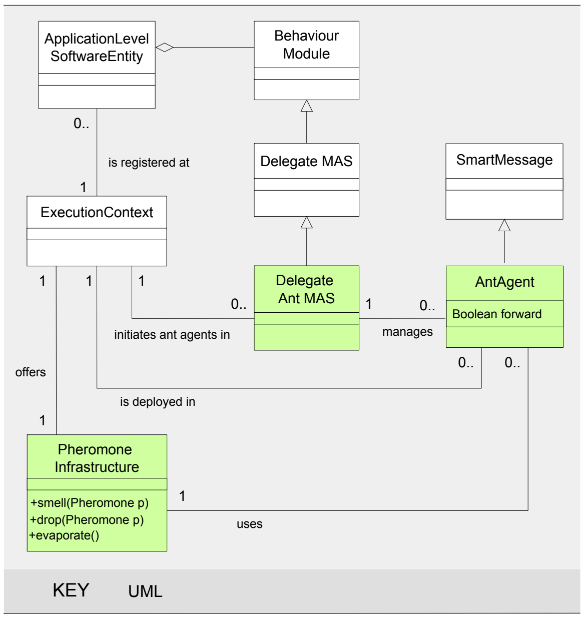

Figure 4: Delegate ant MAS

which it reports back. The migratory behavior of an ant agent is based on a *probabilistic rule* that takes into account pheromone values that are associated with connections to neighboring nodes. An ant agent influences the pheromone values on nodes by a valuation of a connection as part of an overall solution.

As such, an ant agent requires the following refinement compared to SMART MESSAGES:

- *pheromone infrastructure* at every node - specific infrastructure holds the pheromone data; the infrastructure updates the pheromone data based on the valuation of ant agents, and possibly based on an automatic evaporation function; the pheromone infrastructure is offered to ant agents as a service that is accessible via the execution context;

- *forward-backward behavior* - the behavior (incl. computational, migratory and reproductive behavior) of a typical ant agent distinguishes between two phases: a forward behavior, and a backward behavior; the forward behavior aims to explore the environment for a solution (potentially updating pheromone values already), the backward behavior traces back to the responsible entity, updating pheromone values at the different nodes in its path; the phase that an ant agent is in, is stored in the ant agent's state.

In case of the traffic example, smart ant messages can drop routing information that they have collected on behalf of their vehicle, and make it available - through the pheromone infrastructure. Other ant messages can benefit from this information, yielding more accurate routing information.

Consequences

Delegate ant MAS combine delegate MAS with ACO-based principles: the quality of solutions that have been explored by smart ant messages are represented in the environment for other explorers to exploit. Practical experience shows, however, that delegate ant MAS requires a disciplined approach to parameter setting: many parameters are involved (evaporation rate, quality computation, frequency of exploration, and so on), which either need to be fine-tuned off-line (e.g. using genetic algorithms or similar approaches) or need to be monitored and adapted at run-time, i.e. depending on the circumstances at hand.

6. KNOWN USES

Our research studies engineering of large-scale self-organizing systems, with a particular focus on environment-centric coordination mechanisms and their integration within local decision components. In [HV06], we propose 'delegate MAS'. Delegate MAS refers to a coordination mechanism that is inspired by ant behavior and the concepts of BDI-based agents. During our research, we observed approaches for self-organizing system that have interesting similarities to our work. We present these approaches below. For each approach, we indicate (1) the application area for which it has been proposed, (2) key characteristics of the problem that is addressed, and (3) the basic elements and philosophy of the approach. Finally, we indicate how the approaches relate to the patterns described earlier in this document.

6.1 Delegate MAS

The coordination mechanism called 'delegate MAS' is the result of research on the application of multi-agent system (MAS) technology in **manufacturing control**. In particular, we investigated self-organizing solutions for controlling mobile units - which enable partially fabricated goods to move through a manufactory - in a graph structured network of machines and conveyor belts. The approach is based on the PROSA reference architecture [BWV+98], which identifies the core domain agents that mainly represent the mobile units and the resources. Our experience in using the approach for manufacturing control led us to identify a class of applications which we call **'coordination-and-control (C&C) applications'**. C&C applications are applications in which software controls entities in an underlying physical environment. Entities include fixed (non-mobile) resources, capable of performing particular operations, as well as mobile entities which can move in the environment. The purpose of a C&C application is to execute "tasks". Executing a task requires moving through the environment and performing operations by using resources. The environment itself is highly dynamic. Resources may crash, new resources may be added, connections between resources may be added, lost, or their characteristics (e.g. throughput, speed) may change. Members of this family of coordination and control applications include, next to manufacturing control, traffic control and web service coordination, but also supply chain management and multi-modal logistics.

Problem characteristics include the large scale, dynamics, uncertainty, and going concerns as the system objective. The approach needs to cope with large scale of the system, both in terms of number of agents (vehicles, orders, tasks) and physical distribution. Dynamics are intrinsic to any realistic control system, hence the software needs to be designed to cope with uncertainty w.r.t. the perceptions of the world and the results of actuation. The approach must aim for a continuous strive for performance in the presence of dynamics. As such, C&C systems are never single-shot applications but going concern applications.

In **the approach**, the domain agents need to coordinate their behavior for satisfying the (functional and non-functional) system requirements. Here we distinguish *task agents* which manage and control the mobile units, and *resource agents* which manage and control the static resources. Direct communication and negotiation protocols are an obvious approach for coordination, yet lead to very complex agent behavior due to the large number of protocols an agent can be involved in, and dynamics.

Delegate MAS alleviates this complexity by delegating part of the coordination behavior to a dedicated behavior module. Delegate MAS is, to some degree, inspired by food foraging in ant colonies. Ants autonomously explore their environment, and drop pheromones in their environment to indicate the presence of food. Pheromones act as stimuli for other ants. Pheromone trails evaporate over time, and eventually disappear if not reinforced.

We exploit these principles in our approach and define three types of light-weight agents (called 'ant agents'), which each represent a different delegate MAS. *Feasibility ants* are issued regularly by resource agents to autonomously travel the environment and distribute information about the paths that they have followed - i.e. leaving road signs towards their respective resource agents. *Exploration ants* are sent regularly by task agents to autonomously explore the environment for paths that its task agent could follow. On every node, an exploration ant interrogates the resource agent at that node to find out about a potential scheduling of its task agent. When a path is found, the exploration ants report back to their issuing task agent. The task agent weighs the different alternative paths, and decides upon one path as its intention. *Intention ants* are regularly sent out to disseminate this information and make reservations at every node on the intended path. The terminology makes the link to BDI-based agents (Beliefs-Desires-Intentions, see [RG91, Bra87]) obvious. Exploration ants inform the task agent about possible options, after which a task agent selects an option as its intention.

A more detailed description of the approach can be found in [HV06].

From this description, it can be seen that the exploration ants are smart messages which incorporate the disciplined flood and propertinerary. The set of exploration ants is managed using delegate MAS. Feasibility ants are a second type of delegate MAS that consists of smart messages that exhibit the behavior of a disciplined flood. Intention ants are managed by a delegate MAS, and make specific smart messages, which adopt an itinerary approach [AL98] - i.e. they are equipped with a set of locations they need to visit.

6.2 Bio-Inspired Distributed Middleware Management for Stream Processing Systems

In their recent paper [LS08], Lakshmanan and Strom report on a decentralized, ant-inspired algorithm for placing (graphs of) **stream processing tasks** onto a distributed network of machines. Stream processing systems support applications such as processing financial market data or sensor network data. Stream data sources produce large volumes of data at high and variable rates. Performance can be enhanced by dynamically placing stream processing tasks on strategic nodes in the network.

The management application must aim at placing the stream processing tasks on network nodes, taking into account diverse problem characteristics. A first characteristic is dynamics. The flow graph can change (a new query or consumer of stream processing results), network topology and quality characteristics can change, data production rates may vary. Then, the source and consumer nodes can be geographically dispersed in a large network. Third, next to finding 'optimal paths between producers of streaming data, over processing nodes, to consumers', it must be ensured that nodes in the path still comply to the expectations: nodes in the path must still have sufficient computational capacity to meet the process operators of the query without adversely affecting the performance of queries whose operators are already deployed on these nodes.

In this application, a centralized server which has up-to-date global knowledge and which could calculate an optimal allocation of tasks to nodes is unfeasible. Therefore, **a decentralized solution** is proposed and evaluated. In particular, producers, nodes and consumers coordinate the task placement, not by direct communication protocols, but via autonomous, ant-like entities. Also here, three types of ant agents are proposed. Routing ants are created by data producer sites, and explore paths to query consumers. When they reached their destination, they report back to the producer site, leaving a pheromone trail along the path they have followed. The amount of pheromone is proportional to the quality of the path for the query. This is closely related to ant colony optimization techniques [DDC99, DCG99]. Scouting ants exploit the pheromone information about routes to the destination and perform hypothetical placement of tasks along the path. Much like exploration ants in the delegate MAS approach, scouting ants explore different solutions for the problem from the point of view of one domain agent (a producer here, and a task agent in the delegate MAS approach). A queueing model at intermediate nodes is used to estimate the effects of task placement at this node. Scouting ants report back to the producer node when a path with hypothetical task placement has been explored. When a producer has reports about several scouting ants, enforcement ants (somewhat similar to intention ants) make actual arrangements along one particular path. The approach also allows to take into account multiple producers for one query. Join points are then identified - join points are nodes in the network where data from several producers is merged. The placement algorithm is recursively executed from each of he producers to this join point.

A more detailed description of the approach can be found in [LS08].

The resemblance of the approach with the DELEGATE MAS approach is striking. The pattern usage is hence similar, except that the scouting ants are to be considered as a 'delegate ant MAS', since ACO-inspired techniques are used explicitly to assist routing.

6.3 AntHocNet

AntHocNet [CDG05] addresses the problem of **routing in mobile ad hoc networks** (MANETs). In typical MANET applications, data sources need to send data packages to destinations. Nodes however are mobile and communicate over wireless connections.

Routing in MANETs is particularly difficult due to intrinsic dynamics. Nodes are mobile, therefore the network topology changes continuously, and therefore paths between nodes have a limited lifetime. Overall, maximizing network performance is a key objective. Bandwidth usage is limited and variable, as the wireless communication medium is shared and interference may occur. Scale is another quality criterion - routing must support networks that consist of large numbers of physically distributed nodes. Load balancing should avoid network congestion - information about multiple paths from a source to a destination is desirable.

Due to the nature of MANETs, centralized control of routing is obviously not an option. In a MANET there are no designated routers. **Every node** is able to execute **routing functionality**. AntHocNet aims to combine reactive and proactive routing. Reactive routing denotes an approach that corresponds to routing-on-demand. Routing information is gathered when a new data transfer is initiated or an existing path fails. Proactive routing ensures that routing is available at all times. AntHocNet is to a large extent based on the ant colony optimization metaheuristic [DDC99], and in particular extends AntNet [CD98], a routing algorithm for wired networks. On demand of a node, reactive forward ants explore the network for paths to a particular destination. These ants use existing routing information if available. If not, they perform a local broadcast. When the destination is reached, the ant becomes a reactive backward ant which updates pheromone tables along its path back to its source. Pheromone table entries represent the quality of a hop to a next node for reaching the destination. Proactively, nodes diffuse information they have on destination nodes to their neighbors, called pheromone diffusion. Source nodes periodically send out proactive forward ants which follow the diffused pheromone, in order to find other paths to the destination. This potentially yields multiple path to the destination, which can be used concurrently for stochastic data routing.

A more detailed description of the approach can be found in [CDG05].

Reactive forward ants form a DELEGATE ANT MAS of smart ant messages that collaboratively ensure up to date routing information. The pheromone difffusion mechanism corresponds to a DELEGATE MAS mechanism with DISCIPLINED FLOOD.

6.4 Polyagents

The term polyagent refers to an agent-based design approach that has been validated in diverse application areas, including manufacturing control and military applications such as unmanned vehicle routing and commander control. As one example, unmanned vehicle control is an application that aims to find and maintain a suitable path for an unmanned vehicle towards a destination. A path corresponds to a route in the environment that evades hostile regions.

A battle field environment contains uncertainty and is rapidly changing. Efficiency in controlling the vehicles, robustness with respect to message loss, and adaptability to a changing environment are the main architectural drivers for this application.

Polyagents [PB07] is a decentralized, agent-based approach. It starts from the basic principle that domain entities can be represented by multiple agents rather than a single agent. A polyagent consists of one avatar, that is linked to the entity itself, and multiple ghost agents that explore alternative behaviors of the avatar. Ghost agents are typically computationally simple agents that interact via digital pheromone fields. Ghosts explore the environment and probabilistically choose their actions, based on the locally available pheromones. Ghost agents can optionally drop pheromones themselves. For the unmanned vehicle routing application, ghosts imitate ant behavior by exploring a path to the destination of the vehicle, avoiding threats. Ghosts drop 'nest pheromone' while going outbound, and 'target pheromone' on their way back to the avatar. As the ghost agents report back to the avatar that sent them, the avatar can base its own decision on the experience that the ghosts had in the environment. We note that the decision of an avatar is not explicitly communicated through the environment. Besides the fact that this is undesirable in a battle field environment, polyagents are mainly instruments to assist an avatar in making its decision based on the current state of the environment, and not for coordination of future movements of several avatars. In general, the approach does not dictate or constrain how and for what purpose ghost agents can be sent out. The application of commander control illustrates the use of ghosts as explorers of possible future states of the avatar itself. This application is an example of the use of polyagents in adversarial or purely competitive systems. In this case, polyagents can only roam local representations of observed behavior of other agents, they do not interact with these other avatars. In this paper, we will focus on the use of polyagents to roam and interact with other agents in a cooperative setting.

More details on polyagents can be found in [PB07, PBW$^+$07].

Polyagents make a form of DELEGATE ANT MAS where ghost agents roam the environment using DISCIPLINED FLOOD, and drop pheromone information to indicate probabilities of future behavior trajectories.

7. CONCLUSION

"Mature engineering disciplines are characterized by reference materials that give engineers access to the fieldÕs systematic knowledge. Cataloguing architectural patterns is a first step in this direction." (From *'The golden age of software architecture' by Clements and Shaw*, [SC06])

Large-scale and dynamic, decentralized applications are particularly hard to engineer. Several solutions have been described in literature, yet it is not easy to consolidate the solutions into reusable assets. One important reason is that a solution technique makes only sense in a particular application if applying the technique is studied in detail and a positive evaluation results from this study. However, reusable solution patterns serve as inspiration rather than as instantiatable templates. In this paper, we have overviewed various approaches in the area of large-scale and dynamic, decentralized solutions. We identified recurring technical challenges that each of the approaches address, as well as recurring solution techniques. We attempt to consolidate these findings in a limited number of clearly defined solution patterns: smart messages, delegate MAS and delegate ant MAS.

Many challenges can be distinguished. On the agenda for future work is providing an in-depth, formal definition of the patterns. This allows to more rigourously define the patterns, which is necessary for unambiguously applying the patterns in an application.

Another challenge is the engineering or re-engineering of non-trivial applications, inspired by the solution patterns presented in this paper. One application that we intend to study in the near future is decentralized power grid management. Decentralized power producers and consumers need to coordinate to avoid peeks in power production and maximizing the use of green energy. Another family of applications is pickup-and-delivery problems (PDP [Sav95]). The patterns identified in this paper have been an interesting source of inspiration in our first experiments in both areas. The patterns stimulate to consider solutions beyond pure ACO-based techniques, or beyond the original delegate MAS approach.

Acknowledgment

We thank our EuroPLOP shepherd for his useful feedback and hints.

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven. Danny Weyns is a post-doctoral researcher of the FWO-Vlaanderen.

8. REFERENCES

[AL98] Yariv Aridor and Danny B. Lange. Agent design patterns: elements of agent application design. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 108–115, New York, NY, USA, 1998. ACM.

[Bra87] Michael E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard, Cambridge, MA, USA, 1987.

[BWV$^+$98] Hendrik Van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37(3):255–276, 1998.

[CD98] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[CDG05] Gianni Di Caro, Frederick Ducatelle, and Luca M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16:443–455, 2005.

[CPV97] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 22–32, New York, NY, USA, 1997. ACM.

[DCG99] Marco Dorigo, Gianni Di Caro, and Luca Gambardella. Ant algorithms for discrete optimization. *Artif. Life*, 5(2):137–172, 1999.

[DDC99] Marco Dorigo and Gianni D. Di Caro. *The Ant Colony Optimization Meta-Heuristic*, pages 11–32. McGraw-Hill, 1999.

[HV06] Tom Holvoet and Paul Valckenaers. Exploiting the environment for coordinating agent intentions. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *E4MAS*, volume 4389 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2006.

[HWV09] Tom Holvoet, Danny Weyns, and Paul Valckenaers. Patterns of delegate mas. In *Self-Adaptive and Self-Organizing Systems*, pages 1–9. IEEE Computer Society, September 2009.

[KKPS98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 92–99, New York, NY, USA, 1998. ACM.

[KMTU03] Ryszard Kowalczyk, Jörg P. Müller, Huaglory Tianfield, and Rainer Unland, editors. *Agent Technologies, Infrastructures, Tools, and Applications for E-Services, NODe 2002 Agent-Related Workshops, Erfurt, Germany, October 7-10, 2002. Revised Papers*, volume 2592 of *Lecture Notes in Computer Science*. Springer, 2003.

[LS08] Geetika T. Lakshmanan and Rob E. Strom. Biologically-inspired distributed middleware management for stream processing systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 223–242, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[Mae90] Patti Maes. Situated agents can have goals. In Patti Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[PB07] V. Parunak and S. Brueckner. Concurrent modeling of alternative worlds with polyagents. In *Multi-Agent-Based Simulation VII, International Workshop, MABS 2006, Hakodate, Japan, May 8, 2006, Revised and Invited Papers*, volume 4442 of *Lecture Notes in Computer Science*. Springer, 2007.

[PBW$^+$07] H. Van Dyke Parunak, Sven Brueckner, Danny Weyns, Tom Holvoet, Paul Verstraete, and Paul Valckenaers. E pluribus unum: Polyagent and delegate mas architectures. In *Eighth International Workshop on Multi-Agent-Based Simulation*, volume 5003 of *Lecture notes in computer science*, pages 36–51, 2007.

[RG91] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[Sav95] Martin W. P. Savelsbergh. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[SC06] M. Shaw and P. Clements. The golden age of software architecture. *IEEE Softw.*, 23(2):31–39, 2006.

[WHH07] Danny Weyns, Tom Holvoet, and Alexander Helleboogh. Anticipatory vehicle routing using delegate multi-agent systems. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE,*, pages 87–93. IEEE, October 2007.

[Woo09] Michael Wooldridge. *An Introduction to Multiagent Systems*. Wiley, Chichester, UK, 2. edition, 2009.