Patterns of Delegate MAS

Tom Holvoet, Danny Weyns
DistriNet labs
Department of Computer Science
K.U.Leuven
Email: {Tom.Holvoet,Danny.Weyns}@cs.kuleuven.be

Paul Valckenaers
Center for Industrial Management
Department of Mechanics
K.U.Leuven
Email: Paul.Valckenaers@mech.kuleuven.be

Abstract

Delegate MAS has been proposed and investigated as an integrated coordination technique for so-called self-organising coordination-and-control applications. Delegate MAS consist of three types of light weight, ant-like agents that assist domain agents in their coordination tasks - the types are exploration, intention and feasibility ants. The technique is especially suitable for distributed applications in large-scale, dynamic systems.

Literature shows that, for various application domains, solution approaches based on self-organisation have been proposed that have several similarities to delegate MAS, yet are not identical.

In this paper, we specify three reusable solution patterns for coordination in distributed, large-scale, dynamic systems. To motivate the patterns, we first visit several solution approaches from various domains that bear resemblance with respect to coordination. We then identify common application characteristics as well as common technical challenges that underlie the approaches. We describe recurring solution techniques, and consolidate these in three solution patterns. The patterns are 'smart messages', 'delegate MAS', and 'delegate ant MAS' which rely on stigmergic ant agents. Identifying the patterns fosters reuse of particularly useful coordination techniques, and can serve as a catalyst for new or altered approaches.

1. Introduction

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

Literature is flooded with technical descriptions and experiments of self-organising systems in different application domains. Networking, middleware, distributed optimisation, 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 decentralised systems. Research on 'delegate MAS' for manufacturing control and traffic management [1], [2], Polyagents for military applications and manufacturing control [3], mobile agents for internet applications such as e-commerce or tourist assistance [4], [5], bio-inspired network routing [6], [7] and bio-inspired distributed middleware management [8]. Several of these works are inspired by ant colony optimisation techniques [9]. 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.

In this paper, we visit four approaches, and we identify common application characteristics in Section 2. Section 3 describes common technical challenges that motivate the approaches, and lists recurring solution techniques. We consolidate the most notable recurring solution techniques in three solution patterns, described in Section 4. The patterns foster reuse of particularly useful coordination techniques, and can server as a catalyst for new or altered approaches. The paper concludes in Section 5.

2. Related Approaches

Our research studies engineering of large-scale self-organising systems, with a particular focus on environment-centric coordination mechanisms and their integration within local decision components. In [1], we propose 'delegate MAS'. Delegate MAS is a coordination mechanism that is inspired by ant behaviour and the concepts of BDI-based agents. During our research, we observed approaches for self-organising 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.

2.1. 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-organising 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 [10], 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 behaviour 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 behaviour 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 behaviour to a dedicated behaviour 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 [11], [12]) 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 [1].

2.2. Bio-Inspired Distributed Middleware Management for Stream Processing Systems

In their recent paper [8], Lakshmanan and Strom report on a decentralised, 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 centralised server which has up-to-date global knowledge and which could calculate an optimal allocation of tasks to nodes is unfeasible. Therefore, **a decentralised 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 optimisation techniques [9], [13]. 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 [8].

2.3. AntHocNet

AntHocNet [7] 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, maximising 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, centralised 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 optimisation metaheuristic [9], and in particular extends AntNet [14], 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 neighbours, 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 [7].

2.4. 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 [3] is a decentralised, 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 behaviours 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 behaviour 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 behaviour 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 [3], [15].

3. Recurring technical challenges and solutions

The approaches as described above each have shown their usefulness for particular application instances. Similarities between the approaches are apparent. We extract two notable recurring technical challenges that underlie and motivate the approaches. Next, we identify four recurring solution techniques.

3.1. Recurring technical challenges

The application domains that were addressed by the different approaches have in common that they are large scale, both in number of entities as well as in physical distribution, and need to operate in quite dynamic environments - both external causes as well as obstruction between the entities themselves require an adaptive approach. Operating in such conditions requires considering two essential technical challenges: (1) how to bring relevant global information locally, and how to disseminate local information globally; and (2) how to avoid instability in the presence of multiple autonomous decision makers, whose decisions affect each other. We discuss both challenges below.

Global-to-local / local-to-global dissemination of information. The core application entities need to make decisions in order to successfully perform their own task - a data producer needs to find a suitable route to a destination, or a suitable allocation of computational tasks in intermediate nodes, orders in a manufactory need to decide on what resources to use, etc. Any application entity needs to make decisions that will directly affect its own performance. To make an 'optimal' decision, the entity must have all relevant information. Relevant information includes information about the environment topology, the current state of the environment and its resources (network nodes, machines,...), future state of the environment and its resources, and the state information that results from intended behaviour of other application entities. In fact, any core application entity requires gathering information about a large number of entities. Getting this information is not a trivial task, since the application entity needs to find out for itself which other entities it needs information from. Moreover, the dynamic nature of the applications make this a complex endeavour.

Stability. Coordination of entities in a large-scale and complex environment is not easy. Dynamics in the environment make that decisions need revision. Due to changes, new opportunities may become available, while decisions that could have been nearly optimal at some point, may become useless somewhat later. Adaptability is obviously necessary. However, some caution is in order. Changing a decision by one entity may directly effect the quality of potential decisions by other entities, and a waterfall effect may lead the overall system into instability. Mechanisms allowing temporal inertia in behaviour are necessary to keep stability of the system manageable.

3.2. Recurring technical solutions

In the visited approaches, there are several similarities in the way that the technical challenges are tackled. We list the most notable recurring solutions.

Smart messages. In literature, one can find a variety of approaches for coping with the large-scale of a system. Hierarchical solutions compensate for the scale by adding one or more levels of hierarchy. Any entity belongs to a hierarchical level, and if it requires information from outside of its own level, a parent node is used to get and possibly filter this information. Hierarchies can be defined based on physical location boundaries or organisational boundaries (e.g. [16], [17]). Although appropriate for some cases, hierarchies tend to break for very large and very dynamic systems. Complexity due to dynamically managing the hierarchies is one obvious burden.

Other approaches rely on pure self-organisation in the sense that entities will be allowed to observe, communicate and act only in their direct vicinity - these systems rely on suitable overall behaviour to emerge from these local interactions. To improve the overall behaviour, local decision makers can become local experts and can learn patterns of behaviour and how to react locally. Although pure self-organisation is suitable in some application instances, it makes a large compromise with respect to the optimality of the solution.

Both hierarchical and purely self-organising approaches recognise the complexity of gathering relevant global information, making compromises with regard to complexity of managing the information gathering, and to optimality.

The approaches described in Section 2 take a different approach. Rather than taking full responsibility of information gathering via direct communication, the core application entities delegate a substantial part of this process to a separate mechanism.

The mechanism that is used can be referred to as 'smart messages'. Instead of directly communicating with all parties that may have relevant information for a particular entity, and manage the complex information gathering process, the entities create an autonomous, mobile message. The smart message contains both behaviour and state. The behaviour of a smart message is executed at every node, determines how it will interact with entities at a node. It also defines

its mobile behaviour, i.e. it decides on the node(s) to move to next. A smart message can aggregate state from every node. Smart messages can autonomously traverse relevant parts of the environment and interact with nodes.

These light-weight, autonomous mobile entities are used to autonomously scout for relevant global information, and bring relevant information along the path back to their initiator. Or they take local information and disseminate the information to locations in the environment for which the information is considered relevant.

Inertia. To avoid thrashing in behaviour and decision making, mechanisms of inertia need to be adopted. Inertia mechanisms allow to restrain changes in order to have a more stable overall system.

Adopting inertia mechanisms is a delicate balance, however. A system that is too inert will not be able to change its behaviour when changes in the environment occur, potentially leading to poor overall performance. Insufficient inertia leads to unstable systems that may spend more time adapting than meeting system requirements.

We see three mechanisms of inertia in the approaches as discussed in Section 2. A first mechanism allows data to get outdated. A time stamp is associated with data, and if the time stamp is not renewed, the data are considered to be outdated and are discarded. The mechanism is quite well-known, among others in the form of leases in Jini [18]. A second mechanism allows information to diminish over time. A typical example can be found in digital pheromones. Pheromones are value data that periodically diminish in value. Pheromones can be updated through a pheromone update rule, e.g. as in the following ACO-inspired update rule.

$$\tau'_{i,j} = (1 - \rho).\tau_{i,j} + \Delta\tau_{i,j}$$

$\tau_{i,j}$ denotes the pheromone value on edge (i, j). $\Delta\tau_{i,j}$ represents the new value that denote the latest valuation of the edge. Typically, a parameter, e.g. ρ ($0 <= \rho <= 1$), determines to what extent the new, updated value depends on the old value vis--vis the new value. A low value for ρ leads to more inertia in the value, a value for ρ closer to 1 allows the value to change faster towards new data values.

A third mechanism ensures inertia in the behaviour of individual agents. In a BDI-based agent architecture, agents explore different optional behaviours, and then choose a particular behaviour as their intention. A suitable commitment strategy defines the inertia of the agent with respect to the intended behaviour. A blindly committed agent, for example, will not reconsider its behaviour. A cautious agent will more often revise its intention by re-exploring and evaluating different options. Although only one approach explicitly refers to BDI [11], all approaches include an inertia mechanism within the behaviour description of the main application entities.

Ant colony optimisation. The approaches all found inspiration in ant colony optimisation (ACO) techniques to some extent. While the pheromone mechanism mentioned earlier can be used in isolation, ant colony optimisation combines this mechanism with a particular overall policy on how to use and interpret pheromones. In particular, ACO-inspired pheromone values that are being dropped in the environment explicitly represent the quality of a particular part of a solution in an overall solution. One example is pheromone that is associated with a particular edge in a graph-structured environment, to locally represent the quality of the link in an overall solution. The pheromone update value (i.e. $\Delta\tau_{i,j}$) represents the quality of this edge in one particular solution that has been explored. The pheromone value will affect the probabilistic choice of entities exploring solutions.

Symbiotic simulation. A symbiotic simulation is a simulation that emphasises its relation with the physical system [19], [20]. A symbiotic simulation system involves a feedback to the physical system that is meant to control the physical system. The simulations perform "what-if" experiments, driven by data collected from the physical system under control. In turn, the results from the what-if experiments performed by the simulator can be used to control the dynamic behaviour of the physical system. The physical system benefits from its symbiosis with the simulation system from decisions made in near real-time. Decision making is based on the outcome of what-if experiments which involve the simulation of several scenarios, each representing a different decision alternative. This technique allows to predict future system state and behaviour.

This technique is used by exploration ants in the delegate MAS approach, which ask what-if questions to the resource agents on the nodes that they pass, and by scouting ants in [8], which query a node's queueing model to get a prediction, and by ghost agents in the Polyagents approach which simulate behaviours.

4. Smart messages, Delegate MAS and ant agents

The literature study revealed recurring solution techniques for typical technical challenges. In this section, we consolidate these observations by describing three (interdependent) architectural solution patterns. For each pattern, we provide a detailed description. Then we indicate how the patterns match the approaches described in Section 2.

By naming and relating the individual patterns, we are able to provide a clear meaning to the terms. An ant agent is differentiated from a smart message, and the term delegate MAS is given a clear definition. We describe the patterns along a typical template for patterns, including a description of the context, problem and motivation, forces, solution. Examples and known uses are listed in Section 4.4.

4.1. Smart message

Context

This problem that is addressed by this pattern arises in the context of applications in large-scale, dynamic distributed systems. Application-level software entities need to interact with other distributed entities for coordination, synchronisation, information gathering. The underlying, distributed communication environment is a (dynamic) graph topology, where nodes have connections to neighbouring nodes, possibly with varying quality of service.

Problem / Motivation

In general, a problem occurs when multiple and complex interactions are required between one entity (called E) and other entities. Under various circumstances, it is not possible or not desirable to have simple direct interactions with entity E. A situation where this is not possible occurs if the exact identification or location to other entities is not known, or a route to other entities is not known due to the dynamics of the environment. Sending messages back and forth can be troublesome if a route between the entities is not guaranteed.

A situation where this is not desirable is a sequence of interactions with various entities that are physically related. For example the entity E needs to interact with an entity on node n_1 first, then with an entity on node n_2 that is a neighbour of n_1, then with an entity on node n_3 that is a neighbour of n_2, and so on. In this situation, entity E would play a centralistic role if all interactions would occur via direct message passing between E and entities on nodes n_i.

Forces

The dynamic nature of the targeted systems constitutes a challenge. A solution needs to cope with the dynamic topological and quality of service characteristics.

Communication between distant nodes should be limited. The overhead of such communication can be substantial, esp. in dynamic or unstable network conditions.

An entity that needs to manage all its interactions in uncertain network conditions is a complex task. A solution that could support alleviating or managing this complexity.

Solution

The solution identifies a 'smart message' as the core building block - see Figure 1. A smart message is a self-contained entity that is comprised of state and behaviour, and retains information about the entity that is responsible for the message (e.g. the sender). The smart message autonomously moves in the environment and interacts with nodes. The behaviour typically includes

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

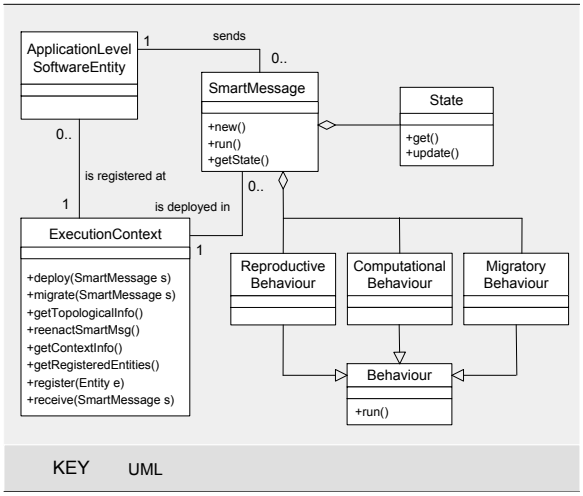

Fig. 1. Smart message

A smart message is deployed in an execution context, i.e. an environment in which the message can perform its behaviour 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 neighbouring node,
- receiving and consequently re-enacting smart messages (i.e. triggering their behaviour execution), and
- querying for context information, including for topological information (e.g. providing lists of neighbouring nodes) and for other registered entities (e.g. providing references to agents that are active at the node).

4.2. Delegate MAS

A smart message is a single, mobile unit. Smart messages can effectively be used in a conglomerate of smart messages, collectively executing a particular task or role. A delegate MAS represents such a managed conglomerate.

Context

Similarly as for smart messages, the problem that is addressed by this pattern arises in the context of applications in large-scale, dynamic distributed systems. Application-level software entities need to interact repeatedly over a period of time with other distributed entities, for coordination, synchronisation, information gathering. The underlying, distributed communication environment is a (dynamic) graph topology, where nodes have connections to neighbouring nodes, possibly with varying quality of service.

Problem / Motivation

The repeated interactions with various entities need to be managed appropriately. This includes the specification of individual interactions, the frequency of interactions, the aggregation and processing of results from interactions.

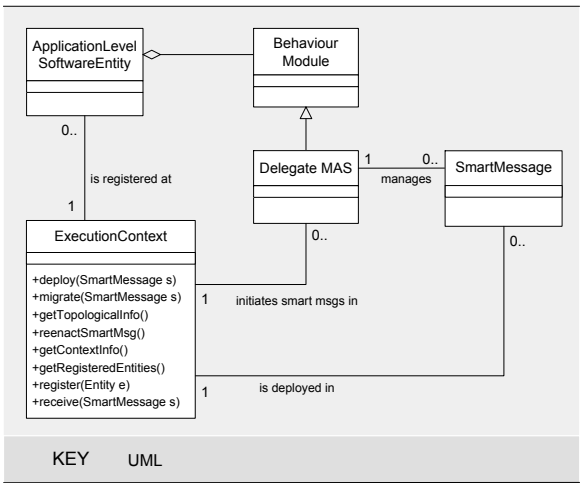

Fig. 2. Delegate MAS

Forces

A set of related interactions, collectively pursuing an objective of an application-level entity, needs to be managed in order to ensure coherence of their behaviour while avoiding unnecessary overhead. Related interactions should be performed coherently, according to a shared policy. For example, one entity that needs to interact with five distant nodes in order to find out which of the nodes can be reached the fastest, must ensure that the individual interactions with the nodes rely on the same objectives and evaluation criteria, and the information that the interactions produce must be interpreted in a coherent manner.

Communication between distant nodes should be limited. The overhead of such communication can be substantial, esp. in dynamic or unstable network conditions. For example, interactions to continuously monitor a path between two nodes should not flood the network.

Individual interactions must account for the dynamic nature of the targeted systems.

Solution

A delegate MAS (see Figure 2) is a *behaviour module* [21], i.e. a well-defined behaviour that an agent can perform to reach a particular objective or task[1]. An agent's behaviour consists of selecting and executing behaviour modules, possibly in a concurrent manner. The agent itself manages the activation and deactivation of behaviour modules, as well as the coordination between behaviour modules. A behaviour module is monitored and controlled by an agent, and fulfils a well-defined objective or task on behalf of the agent.

A delegate MAS is a behaviour module that uses *smart messages* to fulfil 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 (parameterised) behaviour and initial 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 behaviour 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).

1. Other terms that are strongly related to behaviour modules are capabilities, skills, or plans.

4.3. Delegate ant MAS

A delegate ant MAS is a delegate MAS that is specifically targeted for implementing distributed, ACO-like behaviour.

Context

The problem that is addressed by this pattern arises in the context of applications in large-scale, dynamic distributed systems, where application-level software entities need to find a solution for reaching their objectives. A solution for one entity basically consists of finding suitable routes through the graph-structured network, and task allocation.

Problem / Motivation

Finding a solution includes multiple and complex interactions with typically many entities. The entities with which to interact are connected sequentially. A solution needs to be explored, and information about the quality of the explored solution should be shared. An application-level entity can base its decision on this shared information about the quality of a solution.

Forces

A solution needs to cope with the dynamic topological and quality of service characteristics.

Communication between distant nodes should be limited. The overhead of such communication can be substantial, esp. in dynamic or unstable network conditions.

Finding a solution for all entities in the system is complex, since individual solutions affect one another. Individual solutions should be coordinated.

Solution

Delegate ant MAS is an ACO-inspired refinement of delegate MAS, see Figure 3. A delegate ant MAS manages *ant agents* instead of smart messages. We define an ant agent as a smart message of which the behaviour and state is directly related to ant colony optimisation 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 which it reports back. The migratory behaviour of an ant agent is based on a *probabilistic rule* that takes into account pheromone values that are associated with connections to neighbouring 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:

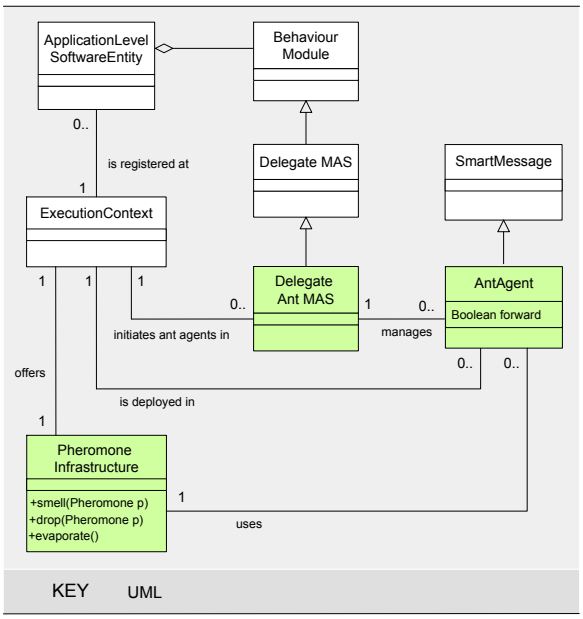

Fig. 3. Delegate ant MAS

- *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 behaviour* - the behaviour (incl. computational, migratory and reproductive behaviour) of a typical ant agent distinguishes between two phases: a forward behaviour, and a backward behaviour; the forward behaviour aims to explore the environment for a solution (potentially updating pheromone values already), the backward behaviour 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.

4.4. The patterns in action

Although originally, the 'delegate MAS approach' was said to be inspired by ant-like behaviour, the study of other approaches and the identification of the patterns above revealed that there is a difference between exploration, intention and feasibility ants on the one hand side, and the ACO based approaches on the other hand.

This observation brings two insights. First, the approach as described in [1] is not ACO-based. This argues for not using the term 'ant agent' to refer to the three types of delegate MAS. The term 'smart messages' is more appropriate. Smart feasibility messages are sent out to drop information at every node it comes across. Smart exploration messages perform a symbiotic simulation on every node it passes. A smart intention message knows a path, and performs a reservation at the nodes in this path. Time stamped data is dropped by various smart messages, which is discarded if not renewed. An ACO-based mechanism would include a valuation of a solution to be translated in a pheromone value in the environment, and a probabilistic path selection rule. This is not present in these three types of delegate MAS.

Second, the distinction between smart messages and ant agents brings inspiration. Additional types of delegate MAS can be defined and assist in the approach, making better use of ACO-based techniques. We give two examples. A smart feasibility message drops 'road signs' in the environment - information that smart exploration messages can use to find a suitable route. A feasibility *ant agent*, as part of a delegate ant MAS, could additionally drop a valuation of an edge, corresponding to the quality of a route to a particular destination. Similarly, exploration *ant agents* could, in their backward phase, drop a valuation of the quality of the explored path in the form of pheromone on edges. This information could be used by other exploration ant agents. Both examples could lead to more performant solutions for coordination and control applications. This is a matter for future research.

These insights will stimulate the re-engineering of delegate MAS based applications, and will inspire further developments of C&C applications (e.g. for pickup-and-delivery problems (PDPs) or inland shipping management). The systems will explicitly make use of the patterns in their design.

With respect to the other three related approached, we briefly indicate the relation between the approaches and the patterns described above. The bio-inspired distributed middleware management for stream processing systems combines two types of smart message delegate MASs with one type of ant agent delegate MAS. Scouting ants include symbiotic simulation in their behavior. In AntHocNet, reactive on-demand-routing uses a delegate MAS of ant agents - routing ant agents explore a path to a destination, and drop pheromones on their way back. Proactive routing is achieved by a delegate MAS of smart messages, diffusing local pheromone information in the neighbourhood, possibly influencing the reactive routing ants. Polyagents consist of coarse grain agents (represented by avatars) that issue ghosts to explore multiple behaviours. Ghost agents are typically ant agents, managed by the avatar.

5. Conclusion

"Mature engineering disciplines are characterised by reference materials that give engineers access to the fields systematic knowledge. Cataloguing architectural patterns is a first step in this direction." (From *'The golden age of software architecture' by Clements and Shaw*, [22])

Large-scale and dynamic, decentralised 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, decentralised 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 decentralised power grid management. Decentralised power producers and consumers need to coordinate to avoid peeks in power production and maximising the use of green energy. Another family of applications is pickup-and-delivery problems (PDP [23]). 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

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 researchers of the FWO-Vlaanderen.

References

[1] T. Holvoet and P. Valckenaers, "Exploiting the environment for coordinating agent intentions," in *E4MAS*, ser. Lecture Notes in Computer Science, D. Weyns, H. V. D. Parunak, and F. Michel, Eds., vol. 4389. Springer, 2006, pp. 51–66.

[2] D. Weyns, T. Holvoet, and A. Helleboogh, "Anticipatory vehicle routing using delegate multi-agent systems," in *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*. IEEE, October 2007, pp. 87–93.

[3] 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*, ser. Lecture Notes in Computer Science, vol. 4442. Springer, 2007.

[4] R. Kowalczyk, J. P. Müller, H. Tianfield, and R. Unland, Eds., *Agent Technologies, Infrastructures, Tools, and Applications for E-Services, NODe 2002 Agent-Related Workshops, Erfurt, Germany, October 7-10, 2002. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2592. Springer, 2003.

[5] A. Carzaniga, G. Picco, and G. Vigna, "Designing distributed applications with mobile code paradigms," in *ICSE '97: Proceedings of the 19th international conference on Software engineering*. New York, NY, USA: ACM, 1997, pp. 22–32.

[6] G. D. Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[7] G. D. Caro, F. Ducatelle, and L. Gambardella, "Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks," *European Transactions on Telecommunications*, vol. 16, pp. 443–455, 2005.

[8] G. Lakshmanan and R. Strom, "Biologically-inspired distributed middleware management for stream processing systems," in *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2008, pp. 223–242.

[9] M. Dorigo and G. D. Di Caro, *The Ant Colony Optimization Meta-Heuristic*. McGraw-Hill, 1999, pp. 11–32.

[10] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: Prosa," *Computers in Industry*, vol. 37, no. 3, pp. 255–276, 1998.

[11] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484.

[12] M. E. Bratman, *Intentions, Plans, and Practical Reason*. Cambridge, MA, USA: Harvard, 1987.

[13] M. Dorigo, G. D. Caro, and L. Gambardella, "Ant algorithms for discrete optimization," *Artif. Life*, vol. 5, no. 2, pp. 137–172, 1999.

[14] G. D. Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[15] V. Parunak, S. Brueckner, D. Weyns, T. Holvoet, P. Verstraete, and P. Valckenaers, "E pluribus unum: Polyagent and delegate mas architectures," in *Eighth International Workshop on Multi-Agent-Based Simulation*, ser. Lecture notes in computer science, vol. 5003, 2007, pp. 36–51.

[16] K. Fischer, J. P. Müller, M. Pischel, and D. Schier, "A model for cooperative transportation scheduling," in *ICMAS*, V. R. Lesser and L. Gasser, Eds. The MIT Press, 1995, pp. 109–116.

[17] M. Mes, M. van der Heijden, and A. van Harten, "Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems," *European Journal of Operational Research*, vol. 181, no. 1, pp. 59–75, 2007.

[18] W. K. Edwards, *Core JINI*. Prentice Hall Professional Technical Reference, 2000, foreword By-Joy,, Bill and Foreword By-Murphy,, Brian.

[19] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Symbiotic simulation systems: An extended definition motivated by symbiosis in biology," in *PADS '08: Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 109–116.

[20] R. Fujimoto, W. Lunceford, E. Page, and A. Uhrmacher, "Grand challenges for modeling and simulation," Dagstuhl, Seminar Report 350, 25.08.2002-30.08.2002 (02351), 2002.

[21] P. Maes, "Situated agents can have goals," in *Designing Autonomous Agents*, P. Maes, Ed. MIT Press, 1990, pp. 49–70.

[22] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Softw.*, vol. 23, no. 2, pp. 31–39, 2006.

[23] M. W. P. Savelsbergh, "The general pickup and delivery problem," *Transportation Science*, vol. 29, no. 1, pp. 17–29, 1995.