CHAPTER 5

Applying Delegate MAS Patterns in Designing Solutions for Dynamic Pickup and Delivery Problems

Shaza Hanif, Tom Holvoet

Department of Computer Science, KU Leuven

5.1 Introduction

Pickup and Delivery Problems (PDPs) have received considerable research attention mainly due to an increase in motorization, urbanization, and globalization. In PDPs, goods (objects or people) have to be transported from the origins to the destinations while complying with a set of constraints. With the difference in constraints and business requirements, more than 30 major variants of PDP have been identified and studied (Parragh et al., 2008).

Many approaches have been proposed and studied to deal with these variants. Some of them use combinatorial optimization approaches to calculate the best schedule in a centralized way. Such approaches require accurate global information to be managed at a centralized location. In practice, PDPs are dynamic; transportation requests appear nondeterministically; road conditions and the number of available trucks may change at runtime. This dynamism makes it very hard for the traditional one-shot planning approaches to provide an efficient and up-to-date schedule while maintaining global information for the whole system. Additionally, the real-world PDPs are large scale; a typical PDP involves hundreds of trucks and thousands of transportation requests. This means the centralized solution has to confront the problem of scalability. Due to both issues, decentralized solutions have gained more and more momentum in recent years.

Multi-agent systems (MAS) are extensively used in building decentralized solutions (Wooldridge, 2002). In these solutions, problem entities – such as trucks and packages – are modeled as autonomous and collaborative agents. Because of their decentralized nature, MAS-based solutions tend to not need centralized data/control and can provide opportunities for scalable solutions. However, engineering an MAS-based solution for PDP is known to be quite challenging. This is due to the fact that typically such decentralized solutions consist of a large number of agents that interact and cooperatively reach the system objectives.

Designing and implementing the interactions and coordination mechanisms between the agents on such a large scale, in the context of a dynamic system, is a complex task.

Many decentralized coordination mechanisms are proposed for tackling the agent interaction and coordination problem to build a decentralized MAS. These include digital pheromones (Brueckner, 2000), gradient fields (Mamei et al., 2004), market-based control (Clearwater, 1996), and tokens (Xu et al., 2005). These mechanisms are typically the result of smart engineering that combines good technical ideas with expert domain knowledge. Although such mechanisms can be exploited to design flexible solutions, a fundamental problem of these mechanisms is the lack of guidance on how to systematically choose, use, and adapt such mechanisms for a new application setting – such as a new PDP variant. Currently, all the existing knowledge and the best practices for the coordination mechanisms is spread over hundreds of papers without a clearly structured and directly usable description of the mechanisms.

Lack of this knowledge makes the design of an MAS-based solution very complex and costly (Wolf and Holvoet, 2007). In order to enable these coordination mechanisms to be used beyond a single application, they should be described in a generic, reusable fashion. Patterns are one of the most appreciated instruments for reuse in software engineering. As pointed out by Shaw and Clements (2006) "Mature engineering disciplines are characterized by reference materials that give engineers access to the fields systematic knowledge. Cataloguing architectural patterns is a first step in this direction". Patterns emerge from frequent use and experience. They identify a generic problem and provide a suitable generic solution scheme. Pattern-based design is increasingly used to construct the complex agent interaction behaviors in MAS (Aridor and Lange, 1998; Oluyomi et al., 2007; Wolf and Holvoet, 2007).

Delegate MAS patterns (Wolf and Holvoet, 2007; Holvoet et al., 2009) are a collection of interaction patterns proposed as a result of research for so-called "coordination-and-control" applications (Holvoet and Valckenaers, 2007). These patterns use the Separation of Concerns (Dijkstra, 1982) design principle by delegating part of the coordination behavior to a dedicated behavior module. They can effectively alleviate the design and implementation complexity of agent interactions. For simplicity, we use the term "Delegate MAS Patterns" to refer to this set of patterns.

In this chapter, rather than discussing and evaluating one or more MAS-based solutions to PDP variants, we illustrate how clearly defined patterns can help build such solutions. To that end, after a brief description of patterns, we first performed a core architectural breakdown of PDP to specify the elements that can be reused for building solutions of multiple PDP variants. We also identify generic challenges in designing the decentralized coordination for such MAS-based solutions. Later, the delegate MAS patterns are applied to design the agent interactions for two distinct variants of PDPs, namely the "personal rapid transportation" (Tuts, 2010) and the "hierarchical PDP" (Claes et al., 2010). We implemented our solutions in the simulation tool – MAS-DisCoSim (Gompel et al., 2010) developed in our previous work.

Although these two PDPs exhibit distinguished features and different business requirements, our experience shows that the Delegate MAS pattern can be used to effectively construct decentralized coordination mechanisms for both. This enables developers to focus on the development of the core functionality of an agent, directly related to the business requirements of a specific PDP variant. Thus, agent design and implementation complexity can be significantly reduced. This chapter focuses on the pattern-based software design. The solutions themselves for the two PDP variants can be found in our previous work (Claes et al., 2010; Tuts, 2010).

This chapter is organized as follows. We first present the related research on solving the PDPs and on patterns for MAS in Section 5.2. Then, in Section 5.3 we present a set of patterns, called "Delegate MAS patterns." We present two PDP case studies in Section 5.4, and discuss solutions for both case studies in Section 5.5 – these solutions make use of the presented patterns, and as such can reuse the know-how and the quality expectations that the patterns aim to accomplish. We conclude our work in Section 5.6.

5.2 Related Work

In the last decade, various taxonomical papers have appeared (Parragh et al., 2008; Berbeglia et al., 2010) which classify the diverse approaches proposed for solving different variants of PDP. Here, we differentiate two major streams of approaches: (1) combinatorial optimization-based approaches and (2) MAS-based approaches. Finally, we present related work with respect to the pattern-based MAS design.

5.2.1 Combinatorial Optimization-Based Approaches for Solving PDP

Traditional approaches mainly treat PDPs as static problems and solve different variants using the combinatorial optimization techniques (Burke and Kendall, 2005). These approaches are not able to effectively handle the dynamism in the problem (Parragh et al., 2008). In order to deal with the dynamic problems, online algorithms (Berbeglia et al., 2010) are designed which add an innovative heuristic on top of the static solutions. When new data becomes available, instead of recomputing the whole solution, they reoptimize the existing one using heuristics and metaheuristics (Gendreau et al., 2006; Gutenschwager et al., 2004).

Moreover, in both the combinatorial optimization and the reoptimization based approaches, the role of a centralized dispatcher is evident. They assume that global information is maintained at the dispatcher, and calculate optimal routes for serving the set of requests available in a certain amount of time. As centralized approaches, reoptimization-based approaches face the challenge of rapidly responding to new information as it becomes available.

In order to meet the challenge of scalability, the idea of breaking the problem into component parts appeared. Ghiani et al. (2003) review parallel computing solutions for solving dynamic

PDP. Such solutions use different control and communication structures (e.g., master-slave, etc.) to efficiently search the solution space on the basis of parallel computing. However, they overlook the possibilities of modeling the system in a decentralized way. For example, PDP could be reframed on the basis of different entities within the problem, that is, the transportation requests and the vehicles. The following section describes the decentralized, MAS-based approaches for PDP.

5.2.2 MAS-Based Approaches for Solving PDP

MAS present a modeling abstraction for building decentralized solutions (Jennings, 2000), with the ability to perform well in uncertain domains (Fischer et al., 1995). Fischer et al. argue that MAS fit the transportation domain particularly well (Fischer et al., 1995). The main motivation of this statement is the following. First, the transportation domain is inherently decentralized (vehicles, transportation requests, companies, etc.). Second, decentralized MAS can handle the dynamism more appropriately. Third, commercial companies may be reluctant to provide proprietary data needed for global optimization whereas agents can use local information.

To support their claims, Fischer et al. present an MAS-based solution in which tasks are allocated to trucks and company agents using an extension of the Contract Net Protocol (CNP). In their approach, the CNP is used to calculate a basic solution, which is later improved by using auction protocols (Wooldridge, 2002). In essence, the authors recognize the limitation of a fully decentralized system, that is, that agents only have access to local information; and hence they make agents coordinate using a two-level approach. The company agent at the top of the hierarchy has more information about the system and makes decisions about the individual truck agents that have limited information. Mes et al. (2007) and Dorer and Calisti (2005) also try to find a balance between the omniscience of a centralized approach and the swiftness of decentralized MAS. Mes et al. (2007) use two high-level agents – the planner and the customer agent – to collect information from and distribute information to the agents under their responsibility. Dorer and Calisti (2005) use a centralized dispatcher agent to allocate incoming requests to regional dispatcher agents. In both approaches, the role of the higher-level agents is to centralize the information essential for the lower-level agents to make effective and more optimized decisions.

Beside such basically market-inspired techniques (Clearwater, 1996), other decentralized coordination mechanisms are proposed, including digital pheromones (Brueckner, 2000), gradient fields (Mamei et al., 2004), and tokens (Xu et al., 2005).

In essence, these solutions try to design smart engineering techniques to provide an optimized solution for a particular PDP variant. However, due to the differences in constraints and business requirements, the tricky balance achieved in one solution for a PDP variant normally could not work in the other variants. This makes the solutions discussed here very hard to be reused. For a new PDP variant, having different business requirements, a new solution is required to be designed. Lack of reusable design knowledge makes the MAS-based PDP solutions very hard to engineer.

This problem results from lack of reusable assets as well as guidance on how to systematically choose and use the most suitable coordination mechanism for MAS. In order to deal with this limitation, some researchers proposed using reusable patterns for designing decentralized MAS.

5.2.3 Patterns for MAS

Aridor and Lange (1998) are among the pioneers who capture MAS experiences as patterns. They identify the importance of structuring coordination mechanisms and present mobile agent patterns under the three classes of Traveling, Task, and Interaction patterns. Their Message pattern (in the class of Interaction patterns) provides the foundation for our Smart Message pattern discussed in Section 5.3.1.

More recently, Oluyomi et al. (2007) presented a framework to facilitate the classification, analysis, and description of agent-oriented patterns. Their primary concern is to present patterns for covering all levels of agent-oriented software engineering but they do not demonstrate the applicability of patterns with fully fledged case studies.

In our previous work, Wolf and Holvoet (2007) distil patterns from the recurrent solutions for the decentralized coordination between agents. They presented the Gradient Field pattern and Market-Based Control pattern. Later, the Delegate MAS pattern (Holvoet et al. 2009) was proposed to build complex interactions behaviors for a set of coordination-and-control applications. It provides an extensible and tunable solution to balance global information and local decisions discussed in Section 5.2.2.

Although dynamic PDP variants exhibit significant differences in their constraints and the business requirements, they still have many common requirements. For instance, there are always vehicles and customer requests in a dynamic and large-scale environment. Having said that, patterns should be able to play an important role in the solution design. However, there is little effort to demonstrate the (re)use of patterns in designing different variants of PDP.

5.3 Delegate MAS Patterns

A literature study on coordination mechanisms for decentralized applications revealed many recurring techniques for typical technical challenges. One line of techniques has been captured as a set of patterns in (Holvoet et al., 2009). In this section, we describe two of these patterns, called Delegate MAS patterns. The patterns are described using a typical template for patterns, including a description of the context, problem, force, solution, and consequences.

This section first introduces the smart message pattern, which is used for communication between agents in changing environments. Then, the Delegate MAS pattern is described for managing such smart messages (Section 5.3.2). For a clear description of concepts, in this section, we will use the vehicle agent and the package agent that are explained in Section 5.5.1.

5.3.1 Smart Message

Context

In large-scale dynamic systems, due to many dynamic factors, the environment of a distributed system normally exhibits dynamic features. In such systems, the environment is represented as a graph in which the nodes have connections to the neighboring nodes with varying quality of service and changing connections. Application-level software entities need to interact with other entities on top of this changing topology for information distribution, information gathering, coordination, and synchronization.

Problem

In general, a problem occurs when multiple and complex interactions are required between one entity and other entities. In MAS, these entities are normally implemented as agents. Under various circumstances, it is not possible or not desirable to have simple direct interactions with an agent. This situation happens when there is no accurate global information available, for instance, if the exact identification or the location of the other agent is not known, or a route to the other agent is not known/broken. Using the traditional mechanism to send messages back and forth between the two remote nodes can be troublesome.

Forces

The dynamic nature of the decentralized systems makes the communication between different agents very hard to engineer. A solution needs to cope with the dynamic topological and quality of service characteristics while communicating with other agents. At the same time, heavy communication between distant nodes needs to be limited to avoid the excessive overhead.

In order to provide all its interactions in an uncertain environment, an agent needs to be developed covering both its functional logic and the nonfunctional communication requirements, while overcoming the previous mentioned problems. This dual responsibility makes the design of such an agent very complex and costly. A solution is needed to alleviate or manage this complexity.

Solution

The pattern identifies the smart message as the core building block to solve the mentioned problem – see Figure 5.1. A smart message is a self-contained entity that comprises both state

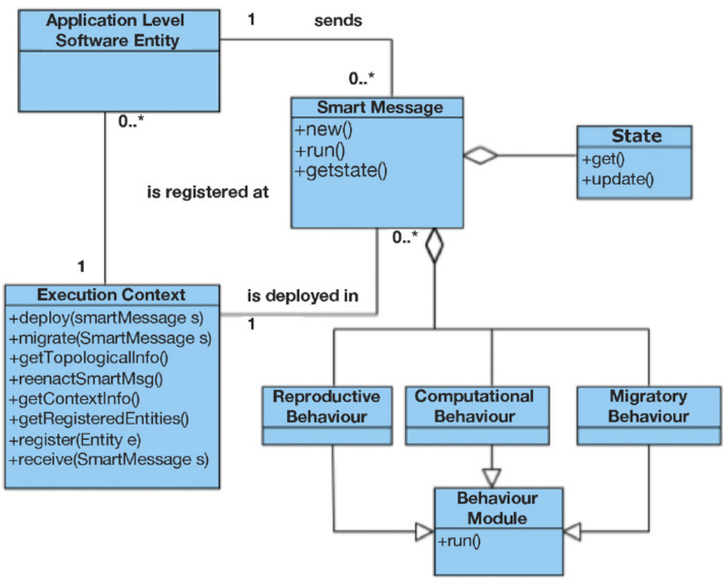

Figure 5.1: A smart message

and behavior, and retains information about the agent that is responsible for the message (e.g., the sender). The smart message autonomously moves in the environment and interacts with nodes on behalf of its owner agent. Behavior of a smart message typically includes:

- querying context information, and possibly interacting with locally active agents;
- local computation that complies with the messages' objective;
- migratory behavior that decides which node to move to next;
- reproductive behavior that allows the 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, that is, an environment in which the smart message can perform its behavior and update its state. This execution context normally links to a node with computing capability. The execution context offers services for:

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

In the PDP, as will be discussed further below, the vehicle agent can send out a smart message to explore nearby packages and evaluate the quality of various paths that the vehicle could make. It can use a disciplined flood (Holvoet et al., 2009) to limit communication overhead in its search, combined with a property itinerary 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 also interrogate the nodes it passes to find out about their current/forecast road conditions.

Consequences

A smart message is a valuable instrument for agents for communicating with each other in a large-scale and unknown environment, which is typical in dynamic PDPs. Care should be taken that their reproductive behavior does not cause flooding or eternal traveling behavior. Due to the limited communication capacity, the state and the mobile code must be limited in size. Due to the limited computational capacities of nodes, the behavior must be of limited complexity.

5.3.2 Delegate MAS

A smart message is a single, mobile unit. The use of a smart message can alleviate the complexity in implementing communication on top of a changing environment. However, the smart messages can be effectively used in a conglomerate way, collectively executing a particular task or role. A Delegate MAS pattern is designed to provide such management.

Context

Similarly, as for the smart messages, the problem that is addressed by this pattern arises in the context of applications in large scale, dynamic decentralized systems. The underlying, distributed communication environment is a (dynamic) graph topology, where nodes have connections to the neighboring nodes, possibly with varying quality of service. Because of dynamism, agents – such as the package agents and vehicle agents – require updated information. They need to interact repeatedly over a period of time with other agents, for coordination, synchronization, and information gathering.

Problem

The repeated interactions with various agents need to be managed appropriately, which includes the specification of the individual interactions, the frequency of interactions, the aggregation, and the processing of the results from the interactions. A software design is needed to make efficient and effective use of the smart messages, without making the implementation of the agent too complex.

Forces

A set of related interactions, collectively pursuing an objective of an agent, needs to be managed in order to ensure coherence of their behavior while avoiding unnecessary overheads. These related interactions (made using smart messages) should be performed

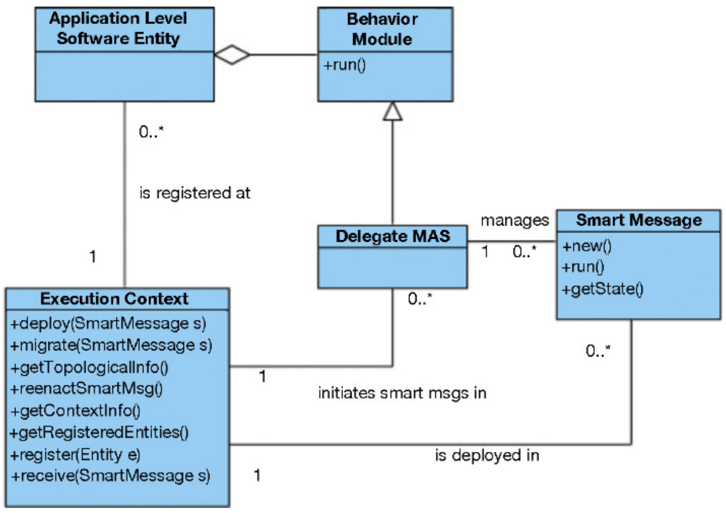

Figure 5.2: Delegate MAS

coherently, according to a shared policy. For example, one agent that needs to interact with five distant nodes in order to find out which of the nodes can be reached the fastest. It 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 the distant nodes should be limited as the overhead of such communication can be substantial. For example, interactions to continuously monitor a path between two nodes should not flood the network.

Solution

A Delegate MAS (see Figure 5.2) is a behavior module (Maes, 1990), that is, a well-defined behavior that an agent can perform to reach a particular objective or task. The behavior module is monitored and controlled by an agent, and fulfils a well-defined objective or task on behalf of the agent. 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 the behavior modules, as well as the coordination between behavior modules.

A Delegate MAS 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 (reconfigurable) behavior 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 preprocessed according the specification of the behavior module. Later these results are forwarded to the agent for further interpretation.

Consequences

This solution allows agents to delegate part of their communication and interaction behavior to a separate module. 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 agent itself.

By using the Delegate MAS pattern, an agent can delegate the task of interactions to the separate behavior module so that the complexity of the agent can be simplified.

5.4 Two Case Studies

In this section, two different PDPs are introduced: the "personal rapid transportation" (Tuts, 2010) and the "hierarchical pickup and delivery" (Claes et al., 2010). These two variants have different business requirements and system objectives. However, they face similar coordination-and-control problems that make agents' implementation complex and error prone.

5.4.1 Personal Rapid Transportation

A Personal Rapid Transportation (PRT) system is a public transportation model featuring small vehicles to provide personalized transportation services to the customers. It consists of a number of pick-up/drop-off points (called stations) scattered in the service area. Customers at the PRT stations wait to be picked up by the vehicles and delivered to his/her destination station with certain time constraints on both waiting and delivery time. A number of PRT vehicles drive around the service area and stop at certain stations to pick up, transport, and deliver customers.

Compared to other public transportation systems, PRT has some following distinct features:

- PRT vehicles do not stop at every station. They are designed to provide a nonstop journey for the customers.
- The routes of PRT vehicles are not fixed. They can autonomously change their routes according to the current conditions.
- PRT vehicles normally have smaller capacity compared to the normal vehicles.

Figure 5.3 shows a simulated PRT for Leuven City (retrieved from MAS-DisCoSim (Gompel et al., 2010)). In this figure, there are 21 PRT stations marked with dots and 7 vehicles to pick up customers. The customers are shown in the picture with package icons.

The general goal is to deliver all customers while maximizing a function of customer satisfaction and minimizing unnecessary travel of the vehicles.

5.4.2 Hierarchical Pickup and Delivery

Large logistics providers organize their transportation network in a hierarchical "hub and spoke" overlay network (Rodrigue et al., 2006). In order to reach their destination, the

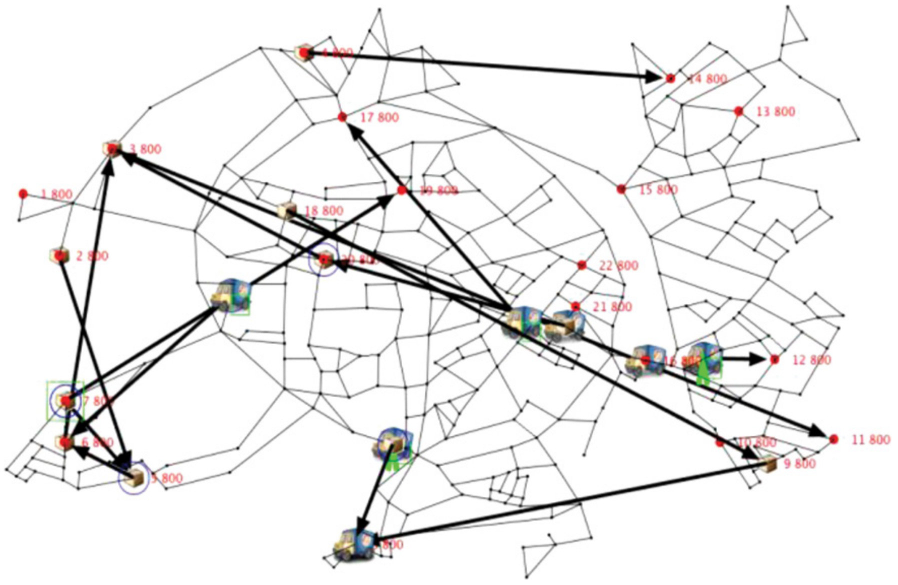

Figure 5.3: Personal Rapid Transport (Leuven Map)

packages are routed from depot to depot in this overlay network. If the destination of a package falls outside the region of a particular depot, it is forwarded to a depot on a higher level. This process is repeated until the package reaches a depot covering its destination or the highest level of the hierarchy is reached. At the highest level, the top-level hubs are connected in a mesh-like fashion. An example of the hierarchical hub-and-spoke overlay network is depicted in Figure 5.4.

The packages need to be routed through this hierarchical network in order to reach their destination. Between each level in the hierarchy, the transportation resources (e.g., trucks, trains, ships, aeroplanes) need to be allocated in order to actually transport the packages up and down the hierarchy. Scheduling these resources is a difficult process due to a number of challenges, including the dynamic nature of the transportation demand, the scale of the hierarchical overlay network, and unforeseen events – for example, the eruption of the Eyjafjallajökull volcano in 2010 – that disrupted global transportation, severely influencing the schedules.

The general goal is to deliver all the packages, while minimizing the resource usage. As the packages move up in the hierarchy, they need to be grouped together to save the resource usage.

5.5 Solutions

This section presents solutions for both PDP variants discussed above. This section, however, does not claim nor aim to present a detailed quantitative evaluation of these solutions. Our main objective is to illustrate the application of the Delegate MAS patterns for engineering valuable solutions.

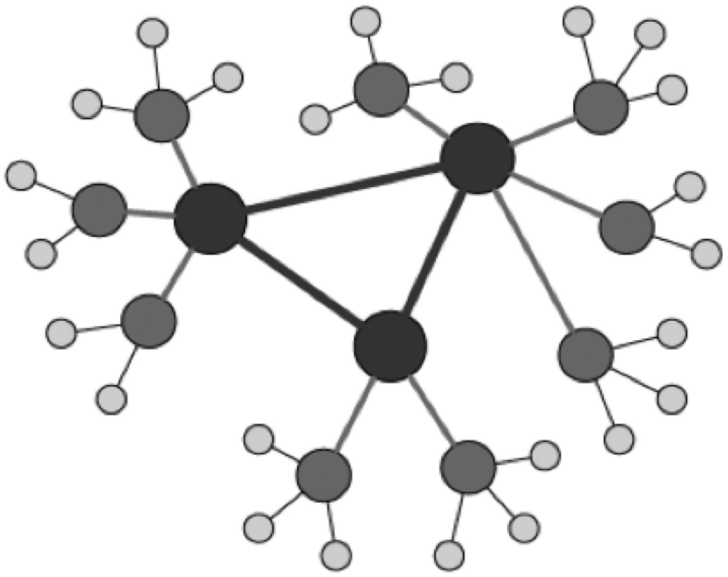

Figure 5.4: Hierarchical PDP, largest circles represent top level of hierarchy

Both solutions start from a core architectural breakdown that is common in MAS-based solutions for PDPs (see Section 5.2), and with a description of common technical challenges. Thereafter, we refine the solutions using the Delegate MAS patterns.

5.5.1 Core Architectural Breakdown

PDPs consist of the transportation requests, which are handled by the transport devices (vehicles). As indicated by Fischer et al., 1995, PDP is an inherently distributed problem, and to model the problem entities, such as vehicles and transportation requests, as agents is a natural metaphor (Fischer et al., 1995). Therefore, in our core architectural breakdown, the transportation requests are modeled as package agents and the transport devices are modeled as vehicle agents, both being deployed in a common environment.

The environment of the targeted PDP is modeled as a directed graph. This graph represents a (simplified) road structure, where the nodes represent physical or virtual road connections. The packages (nonmoving static entities) are deployed in the environment on the nodes. Nodes have storage capability and agents can store/acquire information from nodes. In order to deliver the packages, the mobile entities – vehicles – move from one location to the other location in the environment. To enable communication through the environment, both the packages and the vehicles can communicate with the environment. When a vehicle reaches a particular package, it can perform an action (pickup or enquiry) on it.

Agents

As mentioned earlier, the problem entities that represent the autonomous units of decision-making, are modeled as agents. An agent is deployed on a physical entity, which in turn is situated in the environment. It contains not only entity specific information but also the processing logic related to it. The physical entity can be abstractly described as a set of capabilities, where a capability specifies the operations that the entity is able to perform. For instance, a vehicle has the capability to move, pickup, and deliver packages; both package agents and vehicle agents have the capability to communicate with other agents.

In accordance with the business requirements, an agent is responsible for performing certain tasks correctly (e.g., a vehicle should deliver at the right location) and timely (e.g., a vehicle should pickup packages at the specified time). It needs to explore and plan to achieve its individual goals, which is preferably coherent with the system objectives. For instance, the vehicle agent is responsible for performing its task by guiding its vehicle through the environment, and by communicating and coordinating with the other agents.

Modeling and implementing the internal behavior of the vehicle or package agents is another crucial architectural decision. Various agent architectures can be used, for example, a BDI (Belief–Desire–Intention) architecture (Rao and Georgeff, 1995), or a Reactive Agent Model (Wooldridge, 2002). Again, this chapter does not pretend to "per se" provide the best solution for both PDP variants, but illustrates the application of patterns. For our solutions, we have opted to use a BDI-based architecture for the agents. Its explicit notion of beliefs, desires, and intentions suits the way we modeled the problem.

Agent interactions

The agents are deployed on the physical entities in the environment, and obviously need to interact to achieve the system objectives. A typical interaction amongst these agents could be as follows:

When a new transportation request – that is, a package – enters the system, a package agent is created. The agent is aware of the package details, that is, the pickup and delivery location, the time to pickup, its weight, and so forth. It can respond to the other agents querying this information. Vehicle agents are aware of the current position of their vehicle, and can steer their vehicle to move in the environment and pickup/deliver the packages. It observes the packages in its vicinity and inquires the package details from these package agents. On the basis of its decision logic, it considers the possible and feasible packages to pickup, then selects a particular package and communicates its intention (to pickup) with the package agent by sending a proposal to the package agent.

A package agent in turn can choose to commit to the most suitable proposal offered by various vehicle agents. This allows agents to coordinate their behavior by accommodating their intentions. Coordination is necessary as actions of one vehicle agent may influence the beliefs (and in turn plans) of the other vehicle agents. If the situation in the environment is such that picking up another package becomes substantially more favorable (depending on business requirements), the vehicle agent can reconsider its reservation and adopt a new intention. This process is called intention reconsideration (Wooldridge, 2002).

In this core architectural breakdown, these agent interactions are decentralized and localized, that is, they do not require accessing any global information for coordinating their actions. This feature enables the approach to be scalable – in terms of number of vehicles, number of transportation requests, or geographical expansion. Moreover, agents are able to respond quickly to the local changes in the dynamic environment.

5.5.2 Challenges in Designing MAS-Based Solutions for PDP

There are several distinguishing characteristics of PDP that make the designing of a solution – and, in particular, the interactions between agents – challenging. These challenges are faced in the context of a dynamic and scalable environment. The environment is dynamic since traffic jams, road constructions, vehicle failures, dynamically arriving transportation requests, request cancellations are the causes of uncertainty in the environment. The problem is large in scale – in terms of number of vehicles, types of vehicles (single load carriers, trolley trucks, trains, airplanes, ships), number of transportation requests or widely distributed locations of pickup or delivery.

We describe the most important challenges to address:

1. The resources are limited. Any agent, such as a truck agent, cannot maintain accurate global information due to limited capacity and communication capability. At the same time, the computational resources are limited as well. Similarly, there are other, physical constraints, such as a vehicle's speed range, road capacity, and so forth.
2. The decisions are highly correlated. The decision of one agent influences the behavior of the other agents. When a vehicle finds a more suitable package to pick from the one it reserved, it may cancel the previous reservation and go for a new reservation. This cancellation may cause other vehicle agents to propose the package for the pickup. This implies that an agent needs to notify other agents about its intention and requires the other agents to keep track of resources it has reserved.
3. Purely local decisions are not optimal. If the vehicle and the package agents make decisions purely on the basis of local information, the resulting solution will be far from optimal. On the other hand, as indicated in Section 5.2, centralized approaches tend to breakdown when the underlying system is large scale and agents are unable to give an agile response to the local changes. It is challenging to maintain a decentralized design yet enable agents to: (1) disseminate necessary local information to the remote agents; (2) collect the disseminated information that is necessary for local decisions.

Most of these challenges are related to the coordination between agents, making the design of collaboration crucial for MAS-based solutions, yet highly complex in general.

5.5.3 Typical Applications of Delegate MAS Patterns

We first point out the variety of possible tasks that can be accomplished by using Delegate MAS patterns by referring to the core architectural breakdown, that is, identifying the vehicle agents and the package agents. Later, we refine the solutions for both case studies.

Local-to-(sub)global information dissemination

In a large-scale, decentralized system, local decisions are not and cannot be optimal. Referring back to Delegate MAS patterns discussion in Section 5.3, an agent can only effectively make decisions when it receives the relevant information about its execution context. This means that agents need to attract relevant information toward them, and propagate their local information to the remote nodes. Through this information dissemination process, agents can maintain an updated model of (the relevant part of) the topology. In order to provide this support, agents can use feasibility delegate MAS.

Feasibility delegate MAS is an instance of a delegate MAS pattern for information dissemination and topology discovery by sending lightweight smart messages, so-called feasibility ants. The feasibility ants are issued by their master agent – for instance, a package agent – to propagate local information or changes to the remote nodes within a certain range. A feasibility ant carries out information designated by its master agent (here the package agent) and roams in the environment. At each node it passes, it drops information about the feasible path to its master agent starting from this node. Then, it clones itself for every node directly connected to its current node, except for its incoming node, until it reaches the same node already or exceeds the range limitation. During the roaming process, feasibility ants make local information (sub)globally available and maintain updated information of feasible paths. Vehicles use these feasible paths to reach the package.

Exploring feasible paths

The information spread using the feasibility ants is scattered in the environment. In order to make a decision on the basis of this updated and accurate data, an agent – such as a vehicle agent – needs to actively retrieve information in which it is interested. This challenge can be solved by using the exploration delegate MAS – an instance of Delegate MAS pattern. This instance sends exploration ants. An exploration ant is a smart message that is designed to explore a feasible path for its master agent by collecting and evaluating the information distributed by the feasibility ants. It selects a feasible path and explores it by moving node to node. It evaluates the explored path quality in terms of time, cost, and other quality criteria from its master agent's perspective. In this way, the master agent can be alleviated from the burden of doing such a decentralized and concurrent exploration task.

Negotiation between agents

In order to deal with the challenge of multiinfluential decisions, agents need better coordination by negotiating. Whenever an intention is made that might influence other agents, it is preferable to notify and negotiate with other agents about it. If negotiation succeeds, the other agent might be required to reserve certain resources. To serve this purpose, the intention delegate MAS can be used. This instance of the Delegate MAS pattern manages smart messages, the intension ants, to propagate the intention of its master agent through the environment. For instance, the vehicle agent can send intention ants to propose to a package agent its intention to pick up at a particular time and with a certain delivery cost. The package agent may select one of the best offers proposed by different vehicle agents.

The challenge of dealing with limited resources is also addressed by a Delegate MAS pattern, as the master agent only needs to deal with its core functionality and delegates most of the interaction functionality to the ant agents. Compared to the monolithic solutions where one agent implements all those functionalities, by using the Delegate MAS pattern, a master agent consumes far less resources. Moreover, no entity needs to maintain global information, rather agents disseminate and collect only their relevant information.

Dynamism

Due to the dynamism of the system, these three different kinds of ants need to be sent out periodically. After changes in the environment, old information about feasible paths, path evaluations, and reservations might become outdated. In order to achieve consistency with the changing environments, ants will terminate their execution according to "time to live" stamps assigned to those ants. Detailed design and sample implementation can be found in (Holvoet and Valckenaers, 2007).

These three different instances of a Delegate MAS pattern – feasibility ants, exploration ants, and intention ants – cover the three important requirements typical in the decentralized coordination-and-control applications. Moreover, according to application-specific requirements, other instances can be developed on the basis of the Delegate MAS patterns. In the subsection below, we will introduce how to use a Delegate MAS pattern in designing the solutions for the selected case studies.

5.5.4 Personal Rapid Transportation

Agent-based modeling

As described in the previous section, PRT is intrinsically distributed. Three problem entities can be identified: (1) customers, (2) PRT vehicles, and (3) PRT stations. In order to use an MAS-based approach to model this variant of PDP, we refine the core architectural breakdown for PDP (described in Section 5.5.1) and identify that three types of agents – (1) Customer agent, (2) Vehicle agent, and (3) Station agent – are needed. However, as a customer can only be taken up/dropped off at stations, a Customer agent is represented by the Station agent to reduce modeling and implementation complexity.

Station agent

The Station agents are basically containers for the customers. As a customer can enter the station at an arbitrary time, a Station agent needs to collect this information and propagate it to the Vehicle agents. They also help customers in vehicle selection when multiple options are available.

Vehicle agent

The major functions of a Vehicle agent are to pick up waiting customers and to deliver them to their destinations. In order to achieve this task, a Vehicle agent must collect information about the waiting customers: pickup and delivery locations and the time constraints of a specific customer, and so forth. When multiple customers are discovered, it needs to decide on which customer to take so as to maximize users' satisfaction and reduce unnecessary travel. After a decision is made, this agent should guide the vehicle to the pickup location within changing road conditions. Its functions also include finding the best route to deliver the customer.

Coordination requirements

In PRT, we identify several features that meet the context of the Delegate MAS pattern. It is intrinsically decentralized, with many agents each having their specific goals. It also involves complex interactions between agents to achieve decentralized coordination. In order to deliver the customers in a cost-effective way, these agents must effectively communicate with each other in the changing environment.

Applying Delegate MAS patterns

In a dynamic PRT system, the major function of a Vehicle agent is to provide functions for customer searching, picking up, and delivery. However, due to the dynamic and decentralized nature of the system, an agent needs far more complex implementation.

Key functions of a Vehicle agent are listed as follows:

1. To search for the customer information in a localized way;
2. To find a possible/optimal route to reach the customer for each detected waiting customer;
3. To choose one customer by using heuristics;
4. To make a reservation (negotiates) with a Station agent;
5. To guide the vehicle to the reserved customer (station) in a changing topology.

In all these requirements, from a Vehicle agent's point of view, only the tasks 3 and 5 are its core functionalities. However, the exploration and the coordination requirements though

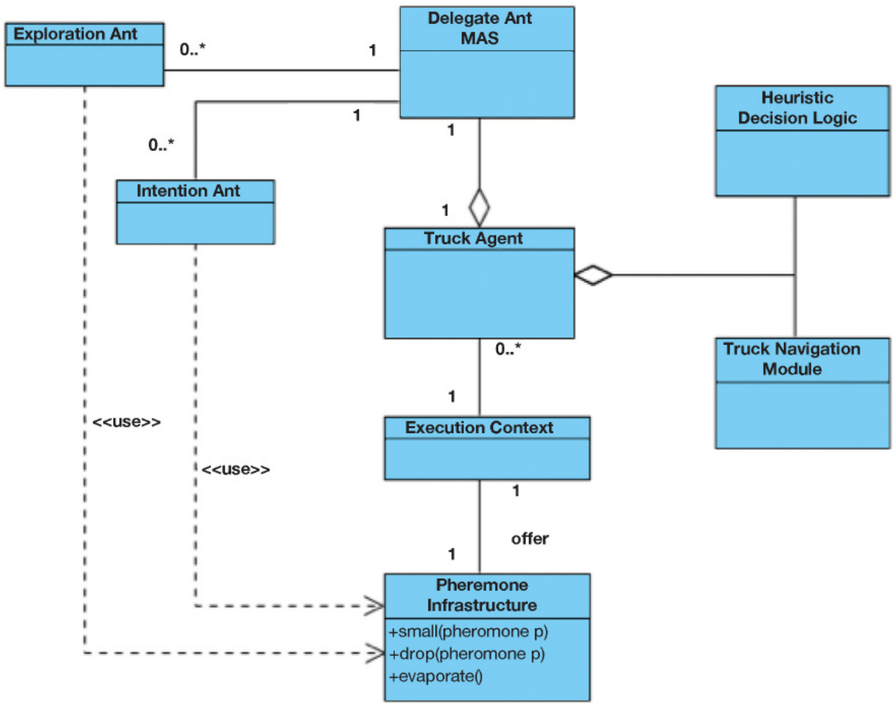

Figure 5.5: Class Diagram for the Vehicle Agent

not core functionality, constitute an important section in the agent implementation and add considerable implementation complexity.

The Delegate MAS pattern provides a systematic approach for implementing the complex interactions between the agents. Thus, this pattern can be used to simplify agent implementation. Here, we only focus on illustrating the design for the Vehicle agent. The class diagram of the Vehicle agent is shown in Figure 5.5.

Vehicle agent

A Vehicle agent uses the exploration ants to explore feasible paths. The exploration ants detect the available customers from the information propagated by the feasibility ants (task 1). It calculates the best route to the target customer on the basis of information collected during the exploration (task 2). Vehicle agents use the intention ants to notify a Station agent that they are likely to visit this station at the estimated time and with certain cost to pay for the delivery (task 4). In this way, the intention ants make a (evaporable) booking on the resources and the Station agent adjusts the status of its customers. Therefore, the Station agent is able to predict the time and the cost for the delivery more accurately. Thus, it is possible to make more refined/timely scheduling for its customers.

Table 5.1: Implementation complexity for different modules.

Functional Modules	Submodules	Line of Code	Percent (%)
Delegate MAS	Feasibility Ant	259	64.5
	Intention Ant	125	
	Exploration Ant	167	
	Delegate MAS Management	218	
Decision logics	Target selection, Route decision, negotiation	423	35.5

By using the Delegate MAS pattern, the functionality of the Vehicle agent is implemented in a simplified way. To quantify this statement, we have analyzed the code for implementing this functionality.

Table 5.1 shows that: (1) A considerable part of the agent's implementation (almost 2/3rd) can be separated from the decision logic of the agent. This implies that the Delegate MAS is able to segregate concerns, and supports the agent's design to be based on the design principle of Separation of Concerns 2). A substantial part of a Vehicle agent's implementation (about 200% of the agent's core functionality) can be explicitly supported by Delegate MAS patterns.

The Station agent was also designed using a Delegate MAS pattern by delegating the information dissemination task to the feasibility ants. It also uses the heuristic to decide which reservation for a customer should be chosen when multiple proposals from the Vehicle agents are received. Detailed discussion of this agent can be found in the reference Tuts (2010).

5.5.5 Hierarchical Pickup and Delivery

Agent-based modeling

Also for the hierarchical PDP, we describe a solution on the basis of a set of coordinating agents. Two types of agents are identified.

Package agent

Every Package agent is responsible for one package. It represents its package's information, such as the source, destination, time constraints, expense constraints, and so forth. Its major function is to find a suitable route through the hierarchical overlay network. It travels alongside the package until it reaches its destination.

Depot agent

For each depot, a Depot agent is created. The major function of this agent is to make a best schedule to allocate optimized resources to the packages among the hierarchical overlay network. This schedule has two constraints: (1) satisfy packages' delivery requirements; (2) reduce the resource usage by combining the delivery of packages as much as possible.

Due to the dynamism in the system, a Depot agent has to continually optimize its local schedule according to the historical data and the reservations proposed by the Package agents.

Compared to the core architectural breakdown, this model does not have Vehicle agents. Since, in this PDP variant, vehicles can only travel between depots and are fully regulated by Depot agents, the Depot agents can take up the functionality of the Vehicle agents.

By communicating with each other, these agents collectively decide upon the route that a package will traverse through the hierarchical overlay network. They also calculate the allocation and the schedule of the transportation resources. Like the previous solution, in this solution also no global knowledge is assumed, every node in the network only knows about its immediate neighbors up and down.

In contrast with the solution technique to the previous case study, this one is package centric, that is, it is the Package agent that needs to find suitable routes for its package. As Package agents can influence the schedule of selected Depot agents and have to compete with the other Package agents, finding a route through the overlay network and the resources necessary to transport the package become challenging.

Coordination requirements

As can be seen from the problem description, Package agents have two different interactions with the Depot agents: (1) Search in the hierarchical overlay network to find a feasible path by retrieving local schedule information from Depot agents; (2) If a suitable route is selected from all the possible explored routes, the Package agent needs to reserve the resources with the Depot agents along the selected route. These two types of interactions require the Package agents to interact with the multiple remote Depot agents in a concurrent way. These interactions meet the context of the Delegate MAS pattern as described in the next section.

Applying Delegate MAS patterns

In order to simplify agent interactions, the Delegate MAS pattern is used to design both the Package agent and the Depot agent. The Package agents can delegate the path exploration task to the exploration ants. Figure 5.6 shows an example of how the exploration ants explore the hierarchical overlay network. The second requirement for the resource reservation can be typically implemented by using the intention ants. The Depot agents are implemented in a passive way. They respond to requests rather than actively initiate interactions.

In order to deliver a package, agents need to interact with others while making local decisions. This process constitutes the following steps:

1. The exploration ants roam the hierarchical overlay network in search of feasible paths;
2. The Depot agents inform passing exploration ants about the schedules and limitations of their resources;
3. The exploration ants finish their exploration phase and gather at the package destination;

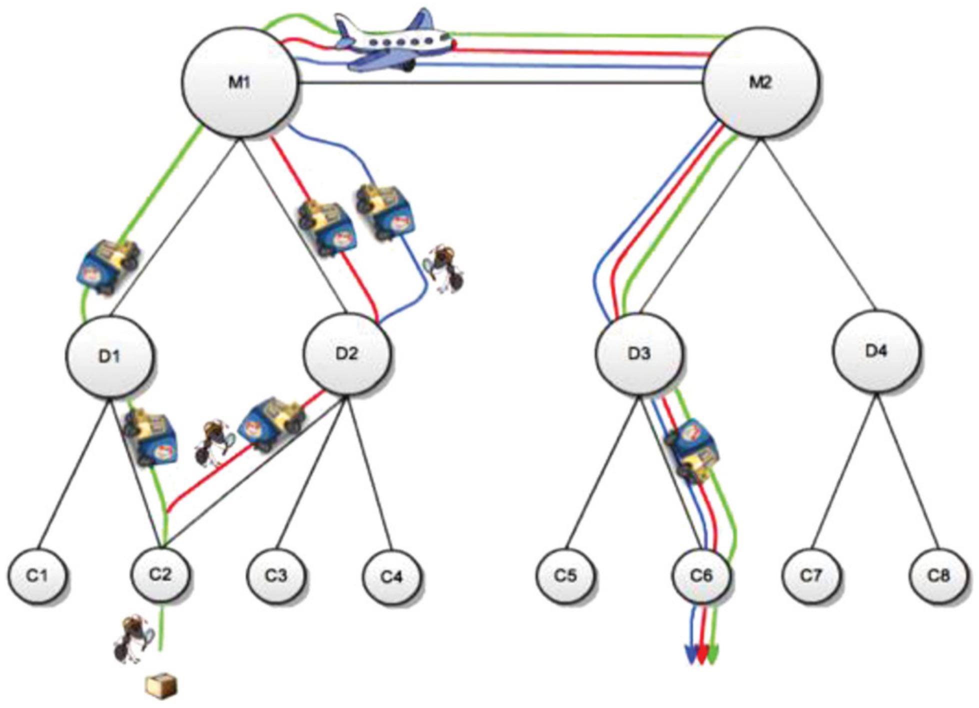

Figure 5.6: Exploration ants explore the hierarchical network.
On arrival the exploration ants have a schedule including arrival and departure times of the transportation resources. In this figure, exploration ants search for feasible paths between nodes C2 and C6.

4. One possible path is selected using the heuristic at destination;
   The reservation for the package schedule is carried out with the Depot agents across the path by an intention ant through the selected path;
5. Depot agents use the reservation information to inform the future exploration ants and optimize their local schedule.

In these steps, we can see the interactions play an important part in the implementation. By using the Delegate MAS pattern, developers of Package agents can focus on the core functionality – the heuristic for path selection. The same applies to the development of the Depot agent, which only needs to implement the local schedule optimization. Due to this design based on separation of concerns, implementation of this solution has similar code features as the solution introduced in Section 5.5.4.

5.5.6 Discussion

In both case studies, we demonstrated the use of the Delegate MAS pattern in designing decentralized solutions for dynamic PDP. Our experience shows that the Delegate MAS pattern gives support in the following way.

Separation of concerns

By applying Delegate MAS patterns, agents can delegate the complex interaction behavior to the lightweight ants. This delegation separates the coordination concerns from the agent's core functionality. Thus programmers can focus on the agent's core programming and most of the skeleton code for interactions can be generated due to usage of the Delegate MAS pattern. Table 5.1, to some extent, gives the evidence.

This separation also makes agent implementation modular and easy to be (re)configured when required. For instance, if the problem details changes and another information dissemination mechanism becomes more appropriate, we can easily locate the reproductive module of the feasibility ant for revision. The Delegate MAS pattern also makes ant behaviors easily tunable – for example, increasing the frequency of sending out ants when there is more dynamism.

Extensible implementation

Compared to the other solutions that implement smart engineering techniques to devise complex MAS only applicable for a particular application, our solution uses reusable patterns to implement MAS based coordination for PDP. The usage of the pattern means it is abstracted from concrete implementation details and it can be used for different requirements.

In these two case studies, we used three types of instances that implemented a Delegate MAS pattern. However, the pattern itself does not restrict what exact functionalities a Delegate MAS module should perform. Different implementations can be created to accomplish application specific tasks in a decentralized and concurrent way. This means our solution can be extended to support a variety of interaction behaviors between agents.

5.6 Conclusion

Large-scale and dynamic, decentralized applications are particularly hard to engineer. This is especially true for designing MAS-based solutions for dynamic PDP. Many solutions have been described in the literature, yet they have not been consolidated into reusable assets and are often optimized for a particular application. This limitation makes them very hard to be reused for designing different PDP variants.

In this chapter, we did not focused on evaluating specific solutions. Rather our focus has been on using reusable patterns to design agent-based solutions for coordination-and-control applications considering PDP as a case study. The Delegate MAS patterns, proposed in our previous work to provide a well-designed solution for implementing decentralized agent interactions, are applied to design the solutions for two PDP variants. Delegate MAS leads to decentralized approaches, that is, solutions that do not require global information to be managed at a centralized entity for optimized decision making. Instead, agents make decisions locally by collaboratively disseminating and collecting information, and by coordinating using lightweight ants in a fully decentralized way. Our experience shows that by using these reusable patterns, an agent's internal design and development can be largely simplified. The use of the Delegate MAS pattern proves to be tailorable and extensible.

Many challenges for further work remain. On the agenda is extending the application of the Delegate MAS pattern to other variants of PDP. In this way, instances other than the three typical instances of delegate MAS can be identified. Another direction is to investigate and abstract the additional recurrent patterns from the existing solutions for PDP. A pattern repository for the PDP can be built to provide a systematic design for the PDP. It is also interesting to investigate how to effectively combine multiple patterns for a solution design. Such a pattern repository can then be studied for its applicability in other application domains that require decentralized coordination in large-scale and dynamic settings.

References

Aridor, Y., Lange, D.B., 1998. Agent design patterns: elements of agent application design presented at the Proceedings of the Second International Conference on Autonomous agents. Minneapolis, Minnesota, United States.

Berbeglia, G., Cordeau, J.-F., Laporte, G., 2010. Dynamic pickup and delivery problems. European Journal of Operational Research 202, 8–15.

Brueckner, S., 2000. Return from the ant – synthetic ecosystems for manufacturing control. PhD Thesis, Humboldt-Universität, Berlin.

Burke, E., Kendall, G., 2005. Search methodologies: introductory tutorials in optimization and decision support techniques. Springer, New York.

Claes, R., Holvoet, T., Van, G.J., 2010. Coordination in hierarchical pickup and delivery problems using Delegate multi-agent systems. In the fourth Workshop on Artificial Transportation Systems and Simulation. 1–7.

Clearwater, S.H., 1996. Market-based control: a paradigm for distributed resource allocation. River Edge, N.J.: World Scientific, Singapore.

Dijkstra, E.W., 1982. On the role of scientific thought. Selected Writings on Computing: A Personal Perspective. Springer-Verlag, New York, pp. 60-66.

Dorer, K., Calisti, K., 2005. An adaptive solution to dynamic transport optimization. Presented at the Proceedings of the Fourth International Joint Conference on Autonomous agents and multi-agent systems, The Netherlands.

Fischer, K., Müller, J.P., Pischel, M., 1995. Cooperative Transportation Scheduling: an Application Domain for DAI. Applied Artificial Intelligence 10, 1–33.

Gendreau, M., Guertin, F., Potvin, J.-Y., SÈguin, R., 2006. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Transportation Research Part C: Emerging Technologies 14, 157–174.

Ghiani, G., Guerriero, F., Laporte, G., Musmanno, R., 2003. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. European Journal of Operational Research 151, 1–11.

Gutenschwager, K., Niklaus, C., Vo, S., 2004. Dispatching of an Electric Monorail System: Applying Metaheuristics to an Online Pickup and Delivery Problem. Transportation science 38, 434–446.

Gompel, J.V., Tuts, B., Claes, R., Torres, M.C., Holvoet, T., 2010. MAS-DiscoSim for PDP:a Testbed for Multi-Agent Solutions to PDPs. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1 (pp. 1639–1640). International Foundation for Autonomous Agents and Multiagent Systems.

Holvoet, T., Valckenaers, P., 2007. Exploiting the environment for coordinating agent intentions. Presented at the Proceedings of the Third International Conference on Environments for multi-agent systems III, Hakodate, Japan.

Holvoet, T., Weyns, D., Valckenaers, P., 2009. Patterns of Delegate MAS. Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems. California, USA, San Francisco, pp. 1–9.

Jennings, N.R., 2000. On agent-based software engineering. Artificial intelligence 117, 277–296.

Maes, P., 1990. Situated agents can have goals. Robotics and Autonomous Systems 6.1, 49–70.

Mamei, M., Zambonelli, F., Leonardi, L., 2004. Co-fields: a physically inspired approach to motion coordination. Pervasive Computing, IEEE 3, 52–61.

Mes, M., Heijden, M. van der, Harten, A. van, 2007. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. European Journal of Operational Research 181, 59–75.

Oluyomi, A., Karunasekera, S., Sterling, L., 2007. A comprehensive view of agent-oriented patterns. Autonomous Agents and Multi-Agent Systems 15, 337–377.

Parragh, S., Doerner, K., Hartl, R., 2008. A survey on pickup and delivery problems. Journal of Betriebswirtschaft 58, 81–117.

Personal rapid transportation system. http://en.wikipe dia .org/wiki/Personal_rapid_transit. (Accessed 25.11.2014)

Rao, A.S., Georgeff, M.P., 1995. BDI-agents: from theory to practice, in Proceedings of the First International Conference on Multi-Agent Systems.

Rodrigue, J.-P., Comtois, C., Slack, B., 2006. The geography of transport systems. New York, London, Routledge.

Shaw, M., Clements, P., 2006. The golden age of software architecture. IEEE Software 23, 31–39.

Tuts, B., 2010. Applying self-adaptation techniques to Delegate MAS. Master Thesis, Computer Science, K.U. Leuven, Leuven.

Wolf, T.D., Holvoet, T., 2007. Design patterns for decentralized coordination in self-organizing emergent systems in Proceedings of the Fourth International Conference on Engineering self-organizing systems. Hakodate, Japan, 28–49.

Wooldridge, M.J., 2002. An introduction to multi-agent systems, xviii, 348.

Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M., Sycara, K., 2005. An integrated token-based algorithm for scalable coordination presented at the Proceedings of the Fourth International Joint Conference on Autonomous agents and multi-agent systems, The Netherlands.