(No) More design patterns for multi-agent systems

Mario Henrique Cruz Torres  Tony Van Beers  Tom Holvoet
Department of Computer Science
Katholieke Universiteit Leuven
Leuven, Belgium
{MarioHenrique.CruzTorres, Tony.VanBeers, Tom.Holvoet}@cs.kuleuven.be

Abstract

A multi-agent systems (MAS) can be used to solve several problems that permeate current complex software systems design, specially distributed systems. The MAS research community has extensively studied protocols, algorithms, methodologies, and architectures to create autonomous, adaptable, robust, and scalable systems. Despite the MAS research achievements, mainstream software developers do not leverage from this knowledge.

Design patterns provide a simple, although effective, mechanism to disseminate knowledge about well proven solutions for recurrent software design problems. The widespread acceptance, by the software development industry, of design patterns shows their effectiveness.

Some researchers attempted to identify MAS design patterns, what is a first step towards broader acceptance of MAS solutions. However, we believe that the MAS research community needs to embrace the identification of MAS design patterns in order to spread MAS solutions to the software industry.

Categories and Subject Descriptors: C.2.11 [Software Engineering]: Software Architectures

General Terms: Algorithms, Design, Standardization

Keywords: coordination mechanisms, multi-agent system, service allocation, distributed systems, design patterns

1. Introduction

It is common to identify well known MAS architectures, coordination protocols, etc, being re-created from scratch, either by peer researchers from other domains, or by mainstream software developers. With the advent of new technologies, such as the internet of things, smart grids, and cloud computing, MAS solutions will be even more demanded and useful. How can the implementors of these new technologies leverage from the enormous MAS knowledge base?

The MAS research community have created solutions to several problems that permeate current complex software systems, specially distributed systems. The MAS research community has extensively studied protocols, algorithms, methodologies, and architectures for many years. MAS solutions can be used to create autonomous, adaptable, robust, and scalable systems. Despite the MAS research community achievements, mainstream software developers do not leverage from the available MAS solutions. We believe that the MAS research community should struggle to identify design patterns, which basically are good design solutions for recurring problems that occur in software design [10], for MAS. This way, using design patterns, the MAS solutions will be more accessible to the software industry, or other peer-researchers.

In the late 1980’s and beginning 1990’s, the object oriented research community saw the rise of software design patterns. The widespread acceptance of design patterns culminated with two seminal works: the book Design Patterns: elements of reusable object oriented software by Gamma et. al. [10], and the Pattern-Oriented Software Architecture series of books, by Buschmann et. al. [2]. This acceptance translated into benefits to both researchers, which received feedback from practitioners, and practitioners whom benefited from the techniques explained in the pattern catalogs.

To the best of our knowledge, design patterns had limited acceptance by the MAS research community. There are works identifying patterns that occur in diverse MAS domains, such as [17–20, 28]. Our team also did work in the direction of defining a pattern language for MAS [29]. However it is not possible to find papers including design patterns in the major MAS events.

To ground our position that the MAS research should identify and document design patterns, we provide a case study of a MAS mechanism for coordination and control, called Composite Delegate MAS, as a design pattern. We also analyze the benefits and drawbacks of using a design pattern to explain such coordination mechanism. We conclude the paper presenting the challenges that need to be faced to identify and document design patterns in MAS.

The rest of the paper is organized as follows. Section 2 depicts works related to MAS design patterns. We present a decentralized coordination mechanism solution as a design pattern in Section 3. In Section 4 we discuss design patterns in the MAS community. Finally, we present our conclusions in Section 5.

2. MAS design patterns

It is possible to works related to using design patterns to document MAS solutions. There are mainly two types of works about patterns in the agent community. The first type is about identifying patterns on agent development methodologies. The second type focus on identifying design patterns in MAS design.

We can cite the works of [17–21, 28], which identified patterns in agent oriented software development methodologies. [28] describes an agent development methodology based on the usage of patterns. This work shows how software developers can apply the patterns iteratively to solve the problem at hand, until they finish designing the agent system. It is a software development methodology based on patterns and MASs.
The work done at [21] describes several MAS patterns. This work describes agent oriented development methodologies as well as agent design patterns. It also describes common templates found in the MAS design patterns. Besides identifying MAS patterns it also proposes a methodology to help software developers to choose the appropriated MAS design pattern to use to solve the problem at hand.

In [19], Lind proposes a template for writing patterns. He also identifies several patterns that occur in MAS. He does not explain, though, in detail any of the cited patterns. The work also arguments in favor of identifying the patterns that occur in the MAS research.

The second type of works, focus more more on the design of MAS. Works as [7–9, 14] describe design patterns that occur in coordination and control systems. [14] describes the delegate MAS coordination mechanism as an architectural pattern. We believe it is a good starting point for introducing the coordination mechanism, but does not give enough details about how can a software developer design a solution using the pattern.

[8] describes a number of coordination patterns, at a higher level of abstraction. It also discusses the causes for MAS solutions not being used by software developers from industry. It also arguments in favor of using patterns to document MAS solutions.

The Gradient coordination mechanism is described as a pattern in [9]. This work shows how this mechanism was inspired on nature and discuss alternate variations of it. We believe, though, that it is still hard for the reader to fully understand how to design a software using such a pattern. Nonetheless the work describes a MAS solution, the Gradient coordination mechanism, as a pattern, clearly stating the context in which the pattern is applicable and its benefits.

In the next section we describe a coordination and control mechanism as a pattern. We describe the pattern in such a way that, we believe, provides clear guidelines on how to design a software using the pattern.

3. Case Study: the Composite DelegateMAS design pattern

The Composite DelegateMAS pattern allows resource users to coordinate and control the usage of resources available in a distributed environment. It allows the coordination of the collective behaviour of independent resource users in a decentralized way. The pattern can be used when the underlying environment is highly dynamic and large scale, both in terms of physical distribution and in the number of entities.

3.1 Example

Everyone is familiar with traffic and - unfortunately - with traffic jams. Traffic jams are not only a source of huge economic loss. They also with influence the environment deterioration, and are extremely stressful to road users. Often, however, traffic jams could diminished, by having a better vehicle routing coordination.

Traffic jams are often a consequence of poor usage of available resources. In this setting roads are the resources used by cars, which are the resource users. The many different road users are clearly independent and have no way of directly communicating with each other. This is a large scale application and is also a very dynamic one. Some dynamic events are that roads can be blocked, renovation works can be done, and cars enter and leave the system.

3.2 Context

A distributed environment in which a heterogenous set of resource users need to execute tasks. Tasks require the consumption of one or more resources, but these resources are scarce in the environment.

3.3 Problem

The main problem is how to decentralized coordinate the usage of scarce resources in a dynamic environment. There are several ways to implement a solution for this problem. One way is to take a centralistic approach, where resources register themselves with a single resource provider and resource users need an initial reference to this provider. The coordination of the resource usage is then the responsibility of the resource provider. However, in highly dynamic, large scale environments, such approaches tend to break [12, 22].

Another way might be to have resource providers broadcasting messages about their availability, or, the other way around, have resource users broadcasting resource requests. The problem with such approach is that it can be quite inefficient, since all decisions are based on the ‘current’ state of the environment. However in highly dynamic settings, the state can have already been changed at the decision making point. This can result in all resource users competing for the same resources, not taking into account the quality of the overall solution or the intentions of other resource users.

To address the problem of coordinating resource usage amongst heterogeneous resource users in a non-centralized manner, requires the resolution of following forces:

- **Heterogeneity**: resource users are independent entities, which have no way of directly communicating with each other. An infrastructure needs to be provided, which allows information exchange between such entities, enabling a global coordination.
- **Availability**: resource users need to be able to locate the resources they require in the environment.
- **Traversability**: resource users need to be able to traverse the environment, in order to consume resources located at remote nodes.
- **Foreseeability**: The coordination of the resource usage should be based on the future state of the resources available in the environment. This increases the efficiency of this coordination compared to approaches based only on the current state of the resources, which gets outdated fast in a highly dynamic setting.

3.4 Solution

The following functionality is provided to solve the coordination of resource usage described above:

- **Feasibility**: Feasibility information describing which resources are available at which location. This information needs to be made available in the environment. Resource providers are responsible for providing this information. The feasibility information can be retrieved from the environment and used for routing tasks through the environment.
- **Exploration**: Resource users need to explore relevant paths in the environment. A relevant path is a path which covers all resources needed by a task. A task agent needs to consider all possible paths (i.e. sequences of operations) which can bring the current task towards its goal. Exploring means evaluating several paths in a what-if mode in order to judge timing and quality of a path if it would be followed by the task agent. Resource agents are responsible for answering "what-if". These queries ask for an estimate of the quality and/or cost of resource usage at a certain time.
- **Intention**: Based on the options a resource user has to complete its task, it chooses one path as its intention. Adopting an intention obviously has implications on future state of the resources that will be visited as part of the intention. The resource user should make its intention known in the environment. This in-
tention information describes which resources the user will be consuming at what time. Resource providers can then use this information to provide a more accurate estimate for "what-if" queries.

Several applications of the delegate ant MAS pattern, as described in [15], are combined into a whole, that solves the coordination problem at hand. A delegate MAS is a behavioural module, which manages a set of light-weight agents that are able to traverse, query and/or update the environment in which they are deployed, and optionally report back to their delegate MAS. Delegate ant MAS is a refinement of Delegate MAS, in which the managed agents are ant agents. The ant agents behaviour and state is directly related to ant colony optimization techniques. These ant agents query and update the environment through the use of a pheromone infrastructure offered by the environment. The details for these two patterns can be found in [15].

For this pattern, two delegate ant MAS’s are introduced. The first one is the resource agent. A resource agent is an agent which manages all resources at a certain location. It also spreads feasibility information, information about its resources availability, in the environment. To spread this information it uses feasibility ants, which drop information in the form of pheromones in a pheromone infrastructure.

A second delegate ant MAS introduced in this pattern, is the task agent. A task agent is an agent that autonomously executes a task. This agent manages two types of ant agents to realize its goals. The first ant agent type is the exploration ant. Exploration ants explore feasible paths in the environment, based on the feasibility information contained in the pheromones dropped in the environment. Exploration ants address different resource agents, asking them 'what if' questions (i.e. what would be the cost of executing task t, requiring resource r, if it arrived at time t). At the end of their itinerary, exploration ants report back to the task agent informing it about the results they have obtained.

The second type of ant agents managed by the task agent is the intention ant. Task agents use intention ants to spread the task agent’s intentions in the environment. The task agent sends out intention ants along the selected path. The intention ants drop reservation information both along the path and at the resource nodes that are part of the selected path. The affected resource agents use this intention information to provide more accurate cost estimates, based on the future state of its resources.

3.5 Structure

The structure of the Composite DelegateMAS pattern is shown in Figure 3.5. The pattern is described in terms of roles and responsibilities, together with the relationships between those roles. The different roles are discussed below:

- **Resource**: a resource represents an entity which can be consumed. What a resource actually represents depends on the application domain. A few examples are: roads in a traffic coordination problem, a software service in a services composition problem, etc.

- **ResourceAgent**: a resource agent manages resources. It offers an interface to these resources and is able to provide cost estimates for the execution of tasks. It answers "what if" questions based on the schedule of the resources it is responsible for. Resource agents represent a delegate ant MAS as described in [15]. In this context, the resource agent manages a set of ant agents, which are the feasibility ants. These ants collectively disperse feasibility information about the resources managed by the resource agent.

- **TaskAgent**: a task agent is an entity whose sole purpose is to execute a task. Just as the resource agent this role represents a delegate ant MAS. It manages both exploration ants and intention ants.

- **Task**: a task is a set of activities that need to be performed.

- **AntAgent**: an ant agent is a constrained agent, with very limited behaviour. Its goal is to traverse the environment in search for a solution on behalf of its delegate ant MAS. The traversal of the environment is based on pheromone values, which are dropped in the environment by other ant agents. As the ant agent traverses the environment, it collects information about it.

- **FeasibilityAnt**: an ant agent is managed by a resource agent. Feasibility ants are responsible for dropping feasibility information in the environment.

- **ExplorationAnt**: an ant agent is managed by a task agent and has the responsibility of exploring all feasible paths that might lead the task agent towards its goals. Exploration ants collect necessary information on their way and report their results back to their task agent.

- **IntentionAnt**: an ant agent is managed by a task agent and has the responsibility of dropping intention information in the environment. It notifies the different resource agents that will be used by its task agent.

- **Environment**: the environment offers an execution context to ant agents. It allows ant agents to traverse it, offering a pheromone infrastructure in which ant agents can drop and smell pheromones.

- **PheromoneInfrastructure**: the pheromone infrastructure offers a way to drop and smell pheromones in the environment. It supports the ACO-like behaviour of ant agents. When a pheromone is dropped in the pheromone infrastructure, it will increase the level of that pheromone. The pheromone level can be retrieved later on by other ant agents. Furthermore, the pheromone infrastructure ensures that pheromones evaporate over time, to avoid stale information.

- **Pheromone**: a pheromone represents quality information about a path, which can be shared amongst ant agents.

![Figure 1. Static structure of the Composite DelegateMAS design pattern.](image-url)
3.6 Dynamics

This section describes the behaviour exhibited by the different roles introduced in Section 3.5. Note that, due to size limitations, certain behaviour has been omitted here. An ant agent possesses two main behaviors: (i) mobility behaviour, which determines how the ant traverses the environment; and (ii) computational behaviour, which represents the task an ant agent has to perform at a single node.

Making feasibility information available

Once a resource agent is started, it keeps track of the feasibility ants that pass through it. If there are no feasibility ants passing by, the resource agent creates and deploys new feasibility ants itself. This guarantees that information about its resources are sent out out periodically. Figure 3.6 shows the computational behaviour of a feasibility ant. The feasibility ant keeps track of the resources available at the nodes it visits and drops pheromones at these resources. It indicates that the resource can be reached through.

Exploring feasible paths

The behaviour of a task agent is illustrated in Figure 3.6. Once the task agent starts executing a task, it sends out exploration ants to explore feasible paths. The computational behaviour of these ants is illustrated in Figure 3.6. Exploration ants obtain cost estimates through the what-if questions. The exploration ants calculate the costs associated to the traveled path and, when a possible path is found, report back it to their task agent. Exploration ants also drop quality information along the path.

Intention propagation

After a task agent has received information about possible paths to follow, it chooses a path based on the desired qualities and starts following the path. The task agent send out intention ants to indicate its intention to use the resources along the path. This intention information is used by resource agents to provide more accurate resource usage estimates.

3.7 Implementation

This section gives some guidelines on implementing the pattern. Note that, because of size restrictions, this is not a full coverage of all implementation issues, but it highlights the most important concern. Below is a list of the different attention points:

Software environment

A choice needs to be made about the software environment and how this maps to the actual underlying distributed environment. We give a few examples of how the software environment can be implemented:

- code migration:
  It is possible to let the software environment mimic the underlying distributed environment, in which the different resource and task agents are deployed at different locations and the ant agents actually migrate to different nodes.

- virtual environment:
  Another possibility is to create a virtual environment representing the real underlying environment, but which is centralized on a single node or a small set of nodes. In this case, the resource and task agents are deployed on the same nodes.

Pheromone infrastructure

There are several ways to implement the pheromone infrastructure. Some possibilities are listed below:
in the form of pheromones. Pheromones are sent out by resource agents, planting roadsigns in the infrastructure elements, which are located next to the roads. Feasibility ants are task agents. Resource agents are deployed in "road infrastructure MAS" pattern can be used. In this setting, roads are resources that can be seen as an optimization of the node-based pheromone infrastructure. Indeed, in this case an exploration ant can determine in a certain node to which node to go next, since it can smell in which direction the appropriate pheromone level is the highest.

Pheromones

It is important to decide which information to put in the pheromones. For example, pheromones dropped by feasibility ants may not only represent the reachable resources, but can also include certain quality information about those resources. Another decision is to decide if the system will support different pheromone types. Different pheromone types can indicate different qualities of the resources.

3.8 Example resolved

In the traffic setting, described in section 3.1, the Composite DelegateMAS pattern can be used. In this setting, roads are resources and vehicles are task agents. Resource agents are deployed in "road infrastructure elements" which are located next to the roads. Feasibility ants are sent out by resource agents, planting roadsigns in the environment in the form of pheromones.

Vehicles have a task to complete, i.e. reach a certain destination. They sent out exploration ants, which explore all reachable paths to the vehicle’s destination. On their way they collect information about traffic jams, road works, etc. from the road infrastructure elements through "what if" questions. Exploration ants ask what would be the time to pass a road if the vehicle would arrive at a certain time.

Once the exploration ants have returned and their results collected, the vehicle selects a single path to follow. Simultaneously, the vehicle starts following the path and starts sending out feasibility ants, informing the road infrastructure elements about when they will pass the associated road. This allows the road infrastructure elements to better estimate the future road load.

3.9 Known uses

The Composite DelegateMAS pattern has been successfully applied in different domains. Below is an overview of these domains with references to analysis of the use of the pattern.

- **Manufacturing control**: Verstraete et al. have applied the pattern in the domain of manufacturing control. In [27] they explain how they combine the pattern with the PROSA reference architecture [26].
- **Traffic control**: Claes et al. applied the pattern in the domain of traffic control. A description of how the pattern is used can be found in [4].
- **Pickup-and-Delivery**: Claes et al. applied the pattern as a solution to the pickup and delivery problem, as described in [3].
- **Service composition**: M.H.C. Torres uses Composite DelegateMAS to select services that participate in a supply chain process. DelegateMAS is used to improve the robustness of the created compositions [25].

3.10 Consequences

The use of the Composite DelegateMAS pattern has following advantages:

- **Independence**: The way the pattern is designed, allows for the different resource and task agents in a system to be developed completely independently, possibly deployed by different independent organisations. The shared pheromone infrastructure offers a way for these independent entities to collaborate within a predefined context. Furthermore, organisations can easily join/leave the system without this having an impact on the overall solution.

- **Robustness**: No central management point has a global overview of the system and its current state. Therefore, the usage of this pattern increases the robustness of solution when compared to centralized approaches, which have a single point of failure.

- **Configurability**: The pattern as presented here leaves quite some parameters open. For instance, a choice needs to be made on when to send out feasibility ants, which information to put in the pheromones, how fast the pheromones evaporate, etc. Instead of fixing this information, these parameters are intentionally left open. Indeed, the optimal configuration of a Composite DelegateMAS pattern depends heavily on the specific domain in which it is applied. Therefore, the pattern indicates what needs to be configured where.

The Composite DelegateMAS pattern has following drawbacks:

- **Shared infrastructure**: It is not simple to agree on a shared infrastructure interfaces. Problem can emerge especially when multiple, independent organisations are involved.

- **Configurability**: The configurability of the Composite DelegateMAS pattern, stated above as an advantage, also adds complexity. It is needed to wisely choose configuration parameters and modeling the pheromone information. As this is left open by the pattern, it is also open for bad configurations which might easily flood the environment or introduce incomplete/irrelevant information in the system. Thus, the configuration of the pattern requires a substantial effort from the software designer.

- **Trust**: As no central management point is introduced, the pattern relies on all agents in the system willing to globally optimizing the system, instead of just looking at their own goals. Hence, a high level of trust is required from the agents.

3.11 Remarks

As already mentioned, the Composite DelegateMAS pattern is a composite pattern, composing several applications of the delegate ant MAS pattern, as described in [15]. Some patterns solving a similar problem can be found in literature. Most of these patterns promote a centralized solution to the resource allocation problem. A few examples are: lookup, lazy acquisition, eager acquisition, and partial acquisition [16].

4. Discussion

We argue in favor of identifying and describing MAS solutions as design patterns. However there are clear challenges that should be faced as well, in order to have useful MAS design patterns.

The first challenge is that it is difficult to have a concrete pattern, but not missing generality. We believe the pattern should be concrete enough to provide clear guidelines to how the software
developer should design a software using the pattern. However, the design pattern should be generic enough not to be tied to a single design.

Another challenge is the lack of proper tool support for helping software developers to design software using patterns. There is research about tool support for using patterns, but these tools are still not available for developers in general.

In the next section we discuss the patterns in the MAS community.

4.1 Design patterns in the MAS community

As mentioned before, several attempts have been made to propose patterns and pattern-based design methodologies for MAS [1, 6, 15, 24, 30], but they lack the success of other design patterns like observer, state [11]. One of the main reasons why mainstream software developers do not benefit from these MAS patterns, is that they simply do not know them. There is no single point of reference or body of knowledge as [11] was for the other patterns movement in object oriented programming. Similar problems can be seen in other software branches, like the secure software community [13].

What [11] did for patterns like the observer pattern, was to define a pattern language which became used by the entire software community. Every developer has encountered these patterns somewhere in his/her educational program. This resulted in a universal vocabulary and syntax, i.e. a universal pattern language, supported both by the educational system and the many resources available online.

As noted by Cline in [5], to maximize the benefits of design patterns, they should be applied uniformly throughout an organization. If the word organization is replaced with software engineering community, this is clearly what has happened with the [11] patterns. Like Schmidt pointed out in [23]: ultimately, patterns are successful because people take time to read them, learn them, use them and write them.

This pattern language should be of sufficient quality and should be backed up with example applications and implementations from real world scenarios. The patterns described in the language should capture the minimal solution to a recurring problem, but care needs to be taken to avoid too abstract or vague patterns. A pattern should clearly indicate how it should be used, what are possible extension points are and how it can be extended.

Therefore, it is important that the MAS community, first identifies more patterns, second, creates and agrees upon a base pattern language. This language could introduce a common vocabulary and syntax for designing MAS applications, allowing more efficient communication within and outside the MAS community.

5. Conclusion

The body of knowledge created by the multi-agent system research is extensive and valuable. However, the impact on mainstream software developers could be higher if the MAS solutions were easily available and shared.

We believe that the best way to spread the knowledge about MAS solutions is identifying the design patterns that arise from these solutions. The design patterns should describe the forces acting upon the problem, the consequences of using the pattern, and the solution. Even more, the design pattern should be concrete enough to provide guidance to the software developer about how she can implement the solution.

We show, in this paper, a coordination and control pattern, called Composite Delegate MAS, that describes a decentralized solution and goes into enough depth to really help a possible developer. We did this by the use of simple UML diagrams, describing the static and dynamic nature of the pattern.

We also believe that the MAS research community can better serve software developers by identifying the design patterns found in MAS solutions and creating a reference pattern catalog. This MAS pattern catalog can, on the one hand, spread MAS solutions, and, on the other hand, be used to test the MAS solutions giving valuable feedback to the MAS research community.

Acknowledgments

This research is partially funded by the Inter-university Attraction Poles Program Belgian State, Belgian Science Policy, and by the K.U.Leuven Research Fund.

References


