A Framework for Patterns in Gaia: A case-study with Organisations

Jorge Gonzalez-Palacios and Michael Luck

School of Electronics and Computer Science
University of Southampton, Southampton, SO17 1BJ, UK
{jlgp02r, mml}@ecs.soton.ac.uk

Abstract. The agent-oriented approach has been successfully applied to the solution of complex problems in dynamic open environments. However, to extend its use to mainstream computing and industrial environments, appropriate software tools are needed. Arguably, software methodologies form the most important type of these software tools. Although several agent-oriented methodologies have been proposed to date, none of them is mature enough to be used in industrial environments. In particular, they typically don't include catalogues of patterns that are necessary for addressing issues of reuse and speed of development. Two possible approaches to overcome such weaknesses in current agent-oriented methodologies are: to propose new methodologies, or to enhance existing ones. In this paper, the latter approach is taken, offering an enhancement of the Gaia methodology to include a catalogue, specifically concerned with a set of organisational patterns. Each of these patterns contains the description of a structure that can be used to model the organisation of agents in specific applications. The use of these patterns helps to reduce development time and promotes reusability of design models.

1 Introduction

The agent-oriented paradigm views a software system as composed of autonomous, pro-active entities that interact using a high level of discourse to achieve overall goals. Research and experimental work in the agent community have shown that this approach is suitable for modelling complex dynamic problems. However, to encourage its use in mainstream computing, some means of engineering agent-based applications is needed. Software methodologies provide one way to engineer such applications in an efficient, repeatable, robust and controllable fashion.

As is indicated in the previous paragraph, interaction plays a relevant role in any multi-agent system. For that reason, the particular focus of our work is on the use of *organisations*, which are important because they provide suitable abstractions to describe, analyse and design the interactions between agents operating in a multi-agent system; thus, organisations may serve as a first-class abstraction to model applications. In the organisational approach, each agent plays one or more roles that interact according to predefined protocols. However, an organisation is more than just a set of interactive roles, so that when modelling an organisation, some means of capturing its essential properties is needed (*organisational abstractions*).

Although many agent-oriented methodologies have been proposed, none is mature enough to be used in industrial and commercial environments [17]. To overcome this, new methodologies are being proposed, but there are two disadvantages. First, increasing the number of available methodologies may cause undesirable effects, such as difficulty in reaching standards, and confusion in selecting a methodology for a specific problem [19]. These effects are particularly undesirable in industrial environments; for instance, adopting a technology in which standards have not been reached is a highly risky decision. Second, there is a tendency to bypass the reuse of work contained in current methodologies. Although they are not mature enough, recent evaluations ([4, 19]) show that some contain valuable contributions. Thus, we focus our research on enhancing current work rather than creating yet another methodology.

Among all the current methodologies, those that encourage the use of organisational abstractions are of special interest, since some evidence suggests that they are potentially suitable for building open systems in complex dynamic environments [22, 15]. One of the first methodologies to incorporate organisational abstractions was that presented in [23] (hereafter called Gaia Extended with Organisational Abstractions, or simply GaiaExOA), which is an evolving extension of the Gaia methodology [20]. In particular, GaiaExOA offers the following valuable features.

– It is easy to understand even by non-specialists, since the process is straightforward and the modelling language simple.
– It is also architecture-independent so that no commitment is made to a specific agent architecture, allowing different architectures to be used in the development process.
– Equally important, GaiaExOA is very well known, being one of the most cited (and consequently used) methodologies, and is suitable for extensions and enhancements. This is already indicated by the various different extensions that have been built around Gaia itself [21, 11].

However, GaiaExOA also requires further work. For example, it lacks a catalogue of patterns to support the development of applications and, in particular, it lacks *organisational patterns*, the importance of which is highlighted in [23], but no such set exists. In this paper, we fill this gap by presenting a framework in which organisational patterns may be developed. We also present an example of such a pattern (but lack of space prohibits other cases). Since the advantages of using patterns are multiple (they reduce development time, promote reusability, act as communication facilitators, and serve as reference and documentation), a valuable enhancement has been achieved.

The rest of the document is organised as follows: Section 2 presents a review of the GaiaExOA methodology with the aim of providing the reader with the context in which the patterns will be used. In Section 3 the exact purpose and scope of the patterns is established. The way patterns are described is important for a fast and appropriate selection; thus in Section 4 we justify the layout used. Due to space limitations, we present only one pattern, and this is done in Section 5. Section 6 contains a review of related work. Finally, Section 7 presents our conclusions.

2 Methodological Context

With the aim of providing the reader with the context in which the patterns will be used, a summary of the GaiaExOA methodology is presented in this section (for a more detailed description see [23] and [20]).

2.1 Organisations in GaiaExOA

As suggested earlier, GaiaExOA is based on the organisational metaphor, which implies that a multi-agent system is seen as a set of agents playing *roles* and *interacting* to achieve individual or societal goals. In turn, these interactions give rise to the composition of *organisations*. In this section, we first describe how the components of roles and interactions are characterised in GaiaExOA, and then outline the mechanism used to describe what constitutes an organisation.

name	
description	
protocols	
activities	
responsibilities	
	liveness properties
	safety properties
permissions	

Table 1. Roles in GaiaExOA

First, we consider the various components. Roles are characterised by a set of features defining their nature and activity as shown in Table 1. The *name* identifies the role and reflects its main intent; the *description* provides a brief textual description of the role; the *protocols* describe the interactions with other roles; the *activities* detail those computations that the role performs without interacting with other roles; the *responsibilities* express the functionality of the role (divided into two parts: *liveness properties* and *safety properties*, which relate to states of affairs that a role must bring about, and the conditions whose compliance the role must ensure, respectively); and the *permissions* identify both the resources that the role needs in order to fulfil its responsibilities, and its rights of access to use them. The characterisation of a role is depicted graphically by means of a *role schema*, an example of which is shown in Figure 4.

Interactions are characterised by means of *protocol definitions*, which comprise the following features: a *purpose* that provides a brief textual description of the interaction; a list of *initiators* that enumerates the roles that start the interaction (usually a single element); a list of *responders* that enumerates the roles involved in the interaction, apart from the initiators; a list of *inputs* and *outputs* that provides the information required or produced during the interaction; and a brief textual *description* that outlines the processing performed by the initiators during the interaction. This characterisation is represented graphically using a diagram like that shown in Figure 1.

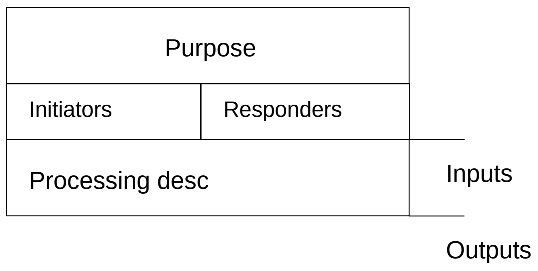


Fig. 1. Protocols in GaiaExOA

Organisations are characterised using two concepts: *organisational rules* and *organisational structures*. The former are constraints imposed on the components of the organisation; that is, roles and protocols. In other words, organisational rules express relationships and constraints between roles, between protocols and between protocols and roles. For example, in an electronic commerce application, an organisational rule might state that an agent cannot play the roles of seller and buyer in the same transaction. To specify organisational rules, first-order temporal logic is used in GaiaExOA [23], together with the temporal connectives. The following formula makes use of this logic to express the rule cited above.

$$\neg\Diamond\,[plays(a, seller) \wedge plays(a, buyer)]$$

Organisational structures encompass two aspects: *topology* and *control regime*. The topology of an organisation is formed of all the communication paths between the member roles, and it is commonly depicted using a diagram in which roles are represented by nodes, and communication paths are represented by arcs between nodes (see Figure 2). The control regime refers to the power relationship between the member roles. Common control regimes are peer-to-peer (when no role is subordinated to another) and master-slave (when the existence of one role is justified only in terms of supporting another role).

2.2 Phases of Development

Although GaiaExOA is one of the best known and most used methodologies, it is limited in terms of its applicability to the full cycle of development, primarily addressing the analysis and upper-design phases, leaving the rest (e.g. detailed design and implementation) largely unconsidered. A brief description of each of its phases follows.

The analysis phase deals with collecting the features needed to model the system, and consists of the following three steps. First, the overall goals of the organisation and its expected global behaviour are identified. Second, based on this information, the basic skills the organisation must accomplish and the interactions needed to achieve them are recognised. Then, a preliminary set of roles is derived from the basic skills. The set of the corresponding role schemata is called the *preliminary roles model*. In addition, the interactions are used to define a preliminary set of protocols. The corresponding set of

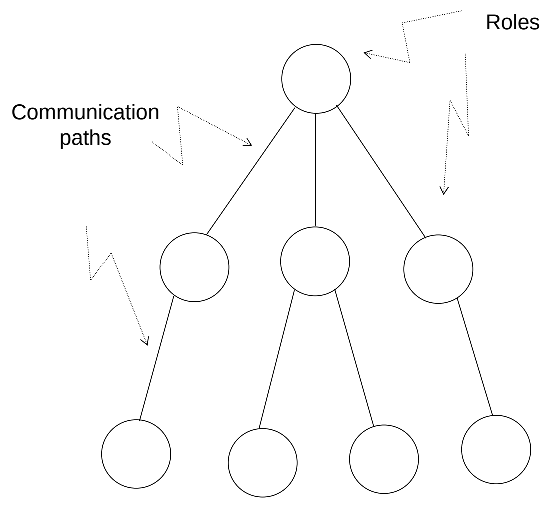


Fig. 2. Topology representation

protocol definitions is called the *preliminary interactions model*. During this step it is important to keep the roles and interactions independent of any specific organisational structure. Finally, the organisational rules of the system are collected.

The design phase deals with building a specification of the system and consists of five steps. To begin with, the organisational structure of the system is determined. This structure might not resemble the physical structure in which the system is immersed, since additional considerations must be taken into account, like efficiency. Secondly, the roles and interactions models are completed. This activity includes the incorporation of new roles and interactions that may have resulted from the application of the previous step, and it is suggested that structure-dependent aspects be separated from those independent of the structure. Thirdly, *organisational patterns* are exploited with the aim of designing the organisational structure and the final interactions model (see Section 3 for details). Next, a decision is made about which roles will be played by which agents at runtime. This decision must consider factors such like coupling, coherence and efficiency. Finally, a list with the *services* (or coherent blocks of activity in which an agent is engaged) of all the roles in the system is produced. A service is a coherent block of activity in which an agent is engaged. Services are derived from the functionality of the roles played by the agents, and their description is shown in Table 2.

name	name of the service
inputs	information needed
outputs	information produced
pre-conditions	constraints
post-conditions	effects

Table 2. Services in GaiaExOA

3 Organisational Patterns: Purpose

The use of the patterns is highlighted in Section 2.2 of the design phase of the methodology. It is observed that before selecting a pattern, the developer has already completed the roles and interactions models and has also identified the organisational rules and defined the organisational structure (topology and control regime). The developer chooses the pattern that best matches the structure that he or she has defined. Since the organisational structure has already been decided, we believe that the organisational pattern must provide at least the following additional value to the developer, all of them oriented towards facilitating the development process.

- A more formal description.
  For example, the developer may sketch the structure using an informal diagram, while the pattern must describe the structure in a more formal way. A more formal description is helpful to reduce ambiguity.
- Description of each role in the structure.
  Each pattern must include the schemata of the participating roles, including the characteristics related to the structure, so that the developer can focus his or her attention on application-specific features.
- Suggestions about their use.
  These suggestions would provide general advice on matters related to implementation. An appropriate level of detail is needed here for avoiding technology dependence.
- Description of the situations in which the pattern's use is appropriate.
  This should include a list of known situations in which the pattern or a closely related one has been used.
- A detailed description of the structure.
  This would be more detailed than a developer normally would achieve at this stage of the design, such as a list of organisational rules corresponding to the management of the organisation. Another example is the inclusion of extensions or variations of the pattern in which greater efficiency is achieved but perhaps the structure is less intuitive. This would let the developer focus on domain specific details rather than general design matters.

After selecting the right pattern, the developer must be ready to complete the final roles and interactions models. GaiaExOA goes as far as this in modelling the system, but a complete methodology should continue towards implementation.

It should be noted that the main question when selecting one of these patterns is what organisational structure best models the characteristics of the system-to-be. As pointed out in [23], such a structure must not only appropriately describe the characteristics of the system but must also take into account issues like efficiency and flexibility. According to Fox [6], when designing a distributed system, one must consider two issues: task decomposition and selection of a control regime. In GaiaExOA, a preliminary task decomposition is achieved during the analysis phase, but the decision of the definitive *topology* is postponed until design. For that reason the selected pattern must provide the topology and the control regime for the organisation.

A first attempt to create a set of patterns may be to take all possible combinations of known topologies and control regimes. This, of course, would lead to an unmanageable number of patterns. Another approach is to consider only those combinations that are potentially useful, either based on experience, or by analogy to other areas in which organisational structures have been applied. We assume that a small number of organisational structures would suit a broad range of applications.

4 The description of the patterns

The importance of a comprehensive structured description for the patterns is twofold. First, it must facilitate the selection of the most appropriate pattern for a specific application. Second, it must be meaningful and helpful when patterns are used as part of a methodology. Some layouts have previously been proposed to describe software patterns (e.g., [8]), and in particular agent patterns (e.g., [5, 14]). In [5] the following sections are suggested as mandatory in any layout: *name*, *context*, *problem*, *forces* and *solution*. Apart from these, *rationale* and *known uses* are also included, specifically for the description of patterns for agent coordination. The layout employed in [14] to describe a catalogue of agent patterns is also divided into two parts: one common to all patterns, and one specific to each of the categories presented in it. The common part includes: name, *alias*, problem, forces, *entities*, *dynamics*, *dependencies*, *example*, *implementation*, *known uses*, *consequences*, and *see also* (a description of the meaning of these components is presented below).

It should be noted, however, that there is no common agreement about what constitutes a good pattern description. For instance, there are different opinions about whether a unique description is appropriate to encompass a variety of patterns. On the one hand, doing this could result in a superficial description. On the other hand, different descriptions make it difficult to compare patterns when selecting one for a specific application. In the agent-oriented approach, it is even less clear what a good pattern description is, mainly because of the immaturity and diversity of agent-oriented methodologies. Rather than engage in that debate, we adopt a very pragmatic approach and opt for a simple description that complies with the Context-Problem-Solution notion (CPS)[2]. According to CPS, the essence of a pattern relies on the relationship between the problem, the situations in which it commonly occurs, and its solution. Thus, every pattern description must include these three elements.

The layout proposed in this document is divided into two parts. It includes a general part, similar to those found in other pattern descriptions; and a particular part, which is specific to organisational patterns. Since the purpose of the latter part is to describe an organisation, we consider the following sections to be necessary: *roles*, *structure*, *dynamics* and *rules*. The sections of the pattern layout, together with a brief explanation, are presented next.

– Name: short descriptive name for the pattern.
– Alias: other names for the pattern.
– Context: a description of the situation in which the pattern applies. Note that the context is a general description and thus is not sufficient to determine if the pattern is applicable. To this end, the context is complemented with the *forces* (see below).

- Problem: the problem solved by the pattern.
- Forces: description of factors that influence the decision as to when to apply the pattern in a *context*. Forces push or pull the problem towards different solutions or indicate possible trade-offs [5]. We have identified the following forces in organisational patterns.
  - Coordination efficiency: organisation structure strongly influences efficiency of coordination in terms of information shared and number of messages interchanged.
  - Coupling: the degree of interdependence between the *roles*. Although coupling is inherent in all the structures, it varies in degree. A structure with high coupling imposes strong constraints on joining the system.
  - Subordination relationships: some structures impose specific control regimes on their roles, which may not be appropriate for some applications.
  - Topology complexity: simple topologies exhibit low coordination overhead but require powerful *roles* in terms of resources and task processing.
- Solution: a description of the solution.
- Restrictions: scope of the pattern.
- Consequences: side-effects of the use of the pattern.
- Implementation: short advice on how to implement the pattern.
- Based on: the references that served as a basis for the pattern.
- Roles: the participating *roles* and their characteristics. When appropriate, the roles in the pattern are described using role schemata. Since these patterns are intended for the development of open systems, a certain characterisation of roles is needed. For example, in a hierarchy, the manager role is more critical than those of subordinates in terms of integrity of the organisation. Thus, a simple characterisation would be: highly critical and less critical. Regarding their complexity, roles may be qualified as basic or potentially decomposable.
- Structure: the topology and the control regime between roles. As stated above, an organisational structure is defined by the topology and control regime of the organisation. Although the structure of an organisation is easily understood by means of a diagram, a formalism is needed to express it for purposes of manipulation, validation and comparison, particularly if tools are developed to support the design process (although this is out of the scope of this paper). In [23] the importance of such a formalism is recognised but the choice of it is left for future work. Here, we propose a simple formalism inspired by the ontology in [7], based on first-order predicate logic, and using the following predicates:
  - $hasInteraction(r, s)$: There exists an interaction protocol that involves role r and role s.
  - $subordinated(r, s)$: The role r is subordinated to the role s.
- Dynamics: the way the roles interact to solve the problem. The interactions between the roles are described using protocol definitions (see Section 2).
- Rules: Constraints to be respected in the organisation independent of the application domain. There are two types: those that spread over all the protocols and roles, and those that express relations between roles, protocols, or between roles and protocols. The formalism used to express the rules is the one proposed in [23] (see Subsection 2.1), together with the following predicates:

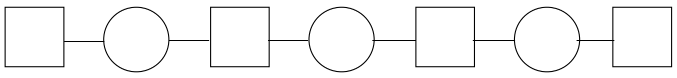


Fig. 3. Pipeline topology

- $plays(a, r)$: agent a performs role r.
- $initiates(r, p)$: role r begins protocol p.
- $participates(r, p)$: role r participates in the execution of protocol p.

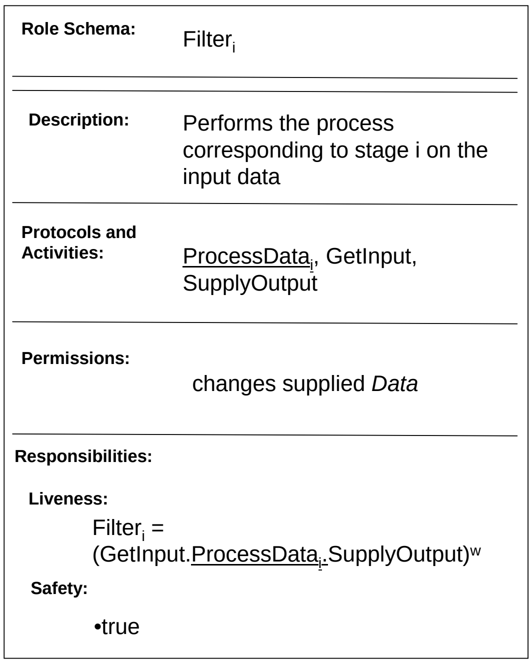
5 The Pipeline pattern

After the trivial case in which a structure contains only one role, the group is the simplest type of organisation. One characteristic of this type of structure is that the control regime is peer-to-peer; that is, no role is subordinated to another. According to the complexity of the topology, two patterns can be easily devised: pipeline and network. The former has a simple linear topology while the latter has a topology in which every role is connected to every other role. Below, the pipeline organisational pattern is described (see Figure 3).

– Name: Pipeline.
– Alias: Flat.
– Context: according to the GaiaExOA process, before selecting a pattern the developer has already completed the roles and the interactions models, and has also compiled the organisational rules and defined the organisational structure (topology and control regime). After selecting the appropriate pattern, the developer must be ready to complete the final roles and interactions models.
– Problem: to find the organisational structure that best describes the system under development. In GaiaExOA, the processes of the organisation are provided by the roles model, so what is missing is to define the topology and the control regime of the organisation. On the other hand, some characteristics of the problem have been identified. First, the overall goals are achieved by a strong collaboration among the participating roles. Second, such a collaboration can be seen as a processing line in which each role performs a transformation on a given information and delivers it to the next member of the line.
– Forces:
  - Coordination efficiency: low.
  - Coupling: low.
  - Subordination relationships: none.
  - Topology complexity: very simple.
– Solution: this structure has been extensively used in mainstream software engineering to design applications in which the overall processing can be decomposed into independent sequential tasks. The tasks are performed by *filters*, which are the processing components. Each filter is connected to the next by means of a *pipe*, which

transfers data from the filter to its successor. Usually, the data are uniform and the tasks apply some sort of transformation on them, such as addition, modification or reduction of information. Although several descriptions exist for this style [2, 16, 10], the pattern presented here is suitable for the agent paradigm and has been adapted to be useful within the methodological context of GaiaExOA. In particular, the components have been modelled as roles and agents, and their interactions as organisations.

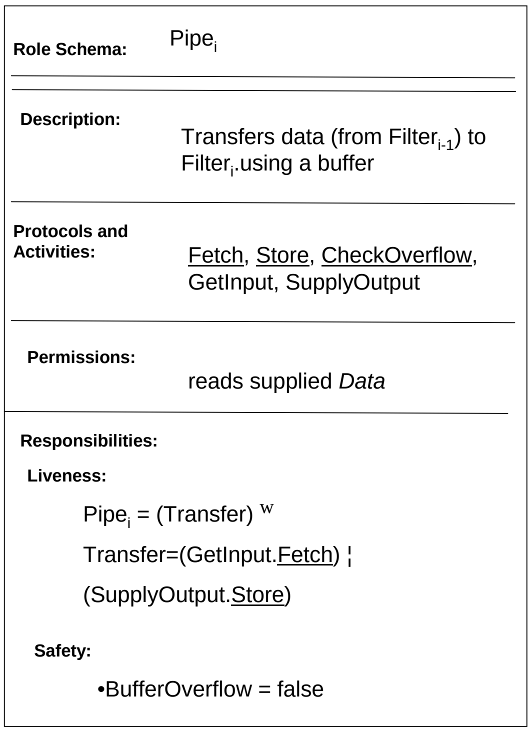
– Restrictions: first, the overall task must be decomposable into independent sequential tasks. Second, the flow of information is restricted to be linear, sequential and only in one direction (no loops or feedback). Third, the processing speed is determined by the slowest filter, although the use of buffers in pipelines can alleviate this restriction to some extent. Finally, to avoid bandwidth and storage problems, the data transferred from stage to stage must be small.
– Consequences: the mechanism of coordination provided is rather simple and is not suitable for operations such as error management. This structure is flexible, since filters can be replaced or bypassed and new filters can be added easily.
– Implementation: the overall task of the system has to be decomposed into independent sequential tasks, with each assigned to one filter. The pipelines may be immersed in the communication layer.



Role Schema:	Filter_i
Description:	Performs the process corresponding to stage i on the input data
Protocols and Activities:	$\underline{\text{ProcessData}_i}$, GetInput, SupplyOutput
Permissions:	changes supplied *Data*
Responsibilities:	
Liveness:	$\text{Filter}_i = (\text{GetInput}.\underline{\text{ProcessData}_i}.\text{SupplyOutput})^{w}$
Safety:	•true

Fig. 4. The Filter role

– Roles: filters are obvious candidates to become roles. In addition, we decided to model pipes also as roles since this highlights their existence within the structure. (The decision of joining a filter and a pipe in a single agent can be postponed to the detailed design phase. Alternatively, pipes could have been modelled as resources, but GaiaExOA does not provide an explicit environment model.) However, it should be noted that filters are *active* entities while pipes are *passive* ones. Filters are allowed to be organisations themselves, but pipes are assumed to be primitive entities. For simplicity of the pattern, we decided not to include the roles of data source (the component which supplies data to the first pipe) and the data sink (the component to which the data to the last pipe is supplied). Figures 4 and 5 show templates of role schemata for the filter and pipe roles respectively.



Role Schema: Pipe$_i$

Description: Transfers data (from Filter$_{i-1}$) to Filter$_i$.using a buffer

Protocols and Activities: Fetch, Store, CheckOverflow, GetInput, SupplyOutput

Permissions: reads supplied *Data*

Responsibilities:

Liveness:

Pipe$_i$ = (Transfer) ^{w}

Transfer=(GetInput.Fetch) ¦ (SupplyOutput.Store)

Safety:

•BufferOverflow = false

Fig. 5. The Pipe role

– Structure: let us denote with N the number of filters in the structure and with $Filter_i$ and $Pipe_j$ the filters and pipes ($1 \leq i \leq N$ and $1 \leq j \leq N+1$) respectively (note that the number of pipes is $N+1$). Figure 3 depicts a pipeline for the case $N = 3$. The structure is described by the following constraints:
Each role has interaction only with its neighbours:

$$\forall i : hasInteraction(Filter_i, p) \Rightarrow$$

$$(p = Pipe_i \vee p = Pipe_{i+1})$$

$$\forall i : hasInteraction(Pipe_i, f) \Rightarrow \\ (f = Filter_{i-1} \vee f = Filter_i)$$

There are no subordinate relations between the roles:

$$\forall s, t : \neg subordinated(s, t)$$

– Dynamics: as shown in Figures 4 and 5, the protocols involved in the coordination of the organisation are $GetInput$ and $SupplyOutput$. Their descriptions are shown in Figure 6 a) and b) respectively. The typical operation of the structure at stage i is the following. First, the filter requests the pipe to its left for the next data using the $GetInput$ protocol (the filter confirms the correct reception of the data). Next, the filter processes the data. Then, the filter requests the pipe to its right to store the processed data using the $SupplyOutput$ protocol.

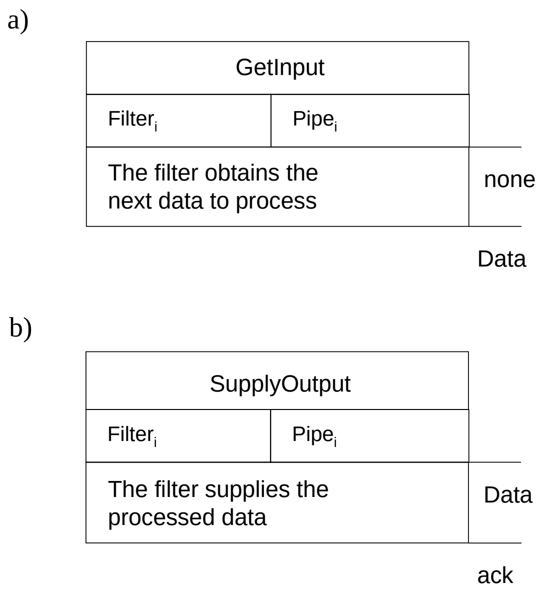

a)
GetInput
Filter$_i$
Pipe$_i$
The filter obtains the next data to process
none
Data
b)
SupplyOutput
Filter$_i$
Pipe$_i$
The filter supplies the processed data
Data
ack

Fig. 6. Pipeline Protocols

– Organisational rules:
All the roles are played by at least one agent:

$$\forall i : \exists a \mid plays(a, Filter_i)$$

$$\forall j : \exists a \mid plays(a, Pipe_j)$$

All the roles are played by at most one agent:

$$\forall i : plays(a, Filter_i) \wedge$$
$$plays(b, Filter_i) \Rightarrow (a = b)$$

$$\forall j : plays(a, Pipe_j) \wedge$$
$$plays(b, Pipe_j) \Rightarrow (a = b)$$

6 Related work

The patterns presented here are intended to be used during the methodological process outlined by Zambonelli et al. [23], in which the importance of a set of organisational patterns is stated but no such set is presented.

Patterns are extensively used to facilitate the development of software systems; in the agent-oriented approach they have been employed to design multiple aspects of an application. Some examples of agent-based methodologies that include the use of patterns in their processes are Tropos, Kendall's methodology and PASSI, considered below. Kolp et al. present a set of patterns in [13] as part of the Tropos methodology, which uses patterns (called *styles*) to describe the general architecture of a system under construction. Although there are similarities with our work, we include organisational rules and classify structures based on topology and control regime (or task decomposition).

Kendall [12] also includes a catalogue of patterns as a part of a technique to analyse and design agent-based systems. The patterns in that catalogue are more general than those presented here, since they include not only interactions but also the roles themselves (it should be noted that the concept of *role* there comes from role theory and is not identical to the concept used here). Since there is no reference to organisational abstractions, however, that work cannot be directly used in the GaiaExOA methodology, but perhaps the structure of those patterns may be used as a base to populate the set of patterns proposed here.

Cossentino et al. present in [3] the design of a particular type of agent pattern immersed in the PASSI methodology. They define a pattern as consisting of a model and an implementation. The model includes two parts: structure and behaviour. Structural patterns are classified into: action patterns, which represent the functionality of the system; behaviour patterns, which can be viewed as a collection of actions; component patterns, which encompass the structure of an agent and its tasks; and service patterns, which describe the collaboration between two or more agents. Implementations are available for two agent platforms, namely, JADE and FIPA-OS. As can be noted from this brief description, the concept of organisation is not explicitly addressed in their work.

Other patterns in the agent literature do not use a specific methodology. For instance, Aridor and Lange [1] present a catalogue that covers different aspects of an application: *travelling*, *task* and *interaction* but these are appropriate only for mobile-agent systems, and are *object-based* rather than role-based. Lind [14] proposed a structure of a pattern catalogue in which the work presented here may fit in their *Society* section, but it is not always clear how to apply the general-purpose patterns within a specific methodology. This is also true for [5], in which Deugo et al. present a set of coordination patterns that are not embedded in a methodology process, so usage is not clear. In fact, there is no separation of coordination patterns, task delegation patterns and matching patterns. A similar set of patterns is presented by Hayden et al. [9] but this focuses on defining how a goal assigned to a particular agent is fulfilled by interacting with other agents.

Finally, Silva and Delgado [18] present an agent pattern that provides distribution, security and persistence transparency. This does not suit our purposes because it focuses on access to a single agent rather than considering a society of them.

7 Conclusions

Although several agent-oriented methodologies have been proposed recently, none of them is mature enough to develop commercial and industrial applications. One step towards achieving mature methodologies is to enhance current methodologies with the inclusion of software engineering best practices, and one such best practice is the use of patterns in key parts of the design process. In this paper, we have presented a framework in which organisational patterns may be developed to model the organisational structure of software applications. Also included is an example of such a pattern (but lack of space prohibits other cases). No framework or set of patterns like these have been proposed before.

We argue that the development presented here is useful for the following reasons. First, it completes Gaia, which is one of the most used methodologies, and the exploitation of organisational patterns is an integral part of its process. Second, it increases the accessibility of the methodology, since the inclusion of patterns makes the methodology easier to use, especially by non-expert users. Third, it helps to reduce development time since developers may reuse the models to avoid building their applications from scratch. Finally, it provides a basis on which further patterns can be developed and improvements can be discussed.

It should be noted that although some patterns are very simple in concept, like the one presented in this document, their usefulness is twofold: they explicitly state the structure a system must conform to; and they serve as a basis for designing complex applications, since most real applications can be described by a composition of several simpler structures. Though this is a first step, it is an important one if the move to industrial-strength design and development is to be successful.

References

1. Y. Aridor and D. Lange. Agent design patterns: Elements of agent application design. In *Autonomous Agents (Agents'98)*. ACM Press, 1998.
2. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture*. Wiley, 1996.
3. M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems, 2002.

4. K. Hoa Dam and M. Winikoff. Comparing agent-oriented methodologies. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS03)*, 2003.
5. D. Deugo, M. Weiss, and E. Kendall. *Coordination of Internet Agents: Models, Technologies and Applications*, chapter Reusable Patterns for Agent Coordination. Springer, 2001.
6. M. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, 1981.
7. M. Fox, M. Barbuceanu, M. Gruninger, and J. Lin. *Simulating Organizations*, chapter An Organizational Ontology for Enterprise Modeling. AAAI Press/The MIT Press, 1998.
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
9. S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multiagent coordination. In *International Conference on Agent Systems '99 (Agents'99)*, 1999.
10. G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2003.
11. T. Juan, A. Pearce, and L. Sterling. Roadmap: Extending the gaia methodology for complex open systems. In *AAMAS '02*. ACM, 2002.
12. E. Kendall. Role models: Patterns of agent system analysis and design. *BT Technology Journal*, 17(4):46–57, 1999.
13. M. Kolp, J. Castro, and J. Mylopoulos. A social organization perspective on software architectures. In *First Int. Workshop From Software Requirements to Architectures*, 2001.
14. J. Lind. Patterns in agent-oriented software engineering. In Fausto Giunchiglia, James Odell, and Gerhard Weiss, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*. Springer, 2003.
15. X. Mao and E. Yu. Organizational and social concepts in agent-oriented software engineering. In *this volume*, 2004.
16. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
17. O. Shehory and A. Sturm. Evaluation of modelling techniques for agent-based systems. In J.P. Muller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 624–631. ACM Press, 2001.
18. A. Silva and J. Delgado. The agent pattern for mobile agent systems. In *3rd European Conference on Pattern Languages of Programming and Computing, EuroPLoP'98*, 1998.
19. A. Sturm and O. Shehory. A framework for evaluating agent-oriented methodologies. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.
20. M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
21. F. Zambonelli, N. Jennings, A. Omicini, and M. Wooldridge. *Coordination of Internet Agents: Models, Technologies and Applications*, chapter Agent-Oriented Software Engineering for Internet Applications. Springer, 2001.
22. F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In *First International Workshop on Agent-Oriented Software Engineering*, pages 127–141, 2000.
23. F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.