Describing Agent-Oriented Design Patterns in Tropos

Carla Silva, Jaelson Castro, Patrícia Tedesco and Ismênia Silva

Centro de Informática – Universidade Federal de Pernambuco (UFPE) Av. Prof. Luiz Freire S/N, 50732-970, Recife PE, Brazil

{ctlls, jbc, pcart, igls}@cin.ufpe.br

Abstract. *The increasing interest in software agents and multi-agent systems has recently led to the development of new methodologies based on agent concepts. The Tropos framework offers an approach to guide the development of agent-oriented systems by using concepts based on requirements engineering. In this paper, we concentrate on the detailed design and implementation phases of the Tropos approach. In particular, we outline a method for choosing and applying agent-oriented design patterns. Moreover, we discuss how agent-oriented design patterns could be properly described, and provide some means of implementing them in a particular agent environment - JADE. The proposed pattern description includes a template as well as three UML extended diagrams to capture the behaviour, structure and collaboration of each pattern. By doing so, we hope to improve the understanding and usage of agent-oriented design patterns. We apply the proposal to an e-News example.*

1. Introduction

Agent-oriented software engineering is concerned with the use of agents in the development of complex distributed systems, especially in open and dynamic environments. In order to reduce development costs and promote reuse, efforts are underway to investigate the concept of patterns [Tahara, Ohsuga and Honiden 1999].

A software pattern describes a recurring problem and solution; it may address conceptual, architectural or design problems [Kendall, Krishna, Pathak and Suresh 1998]. This concept has proved highly useful within the object-oriented field, and can make the applications more flexible, understandable, and reusable [Aridor and Lange 1998]. Indeed, the efficient utilization of design patterns has the potential for bringing significant benefits to the architectural process and to the successful implementation of multi-agent systems [Hayden, Carrick and Yang 1999].

We are working on the improvement of Tropos [Castro, Kolp and Mylopoulos 2002] - a requirements-driven framework that proposes to use the same concepts at various stages during the software development lifecycle. Tropos supports four phases of software development:

- Early requirements: It is concerned with the understanding of a problem by studying an organisational setting.
- Late requirements: The system-to-be is described within its operational environment, along with relevant functions and qualities (e.g. performance, security, availability).
- Architectural design: The system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies.

– Detailed design: The behaviour of each architectural component is defined in further detail.

Tropos has proposed a set of design patterns focusing on social and intentional aspects [Kolp, Giorgini and Mylopoulos 2002]. The so called social patterns offer a microscopic view of the multi-agent system (MAS) at the detailed design phase to express in richer details each component present in the architectural design of the system [Kolp, Do, Faulkner and Hoang 2005]. However, Tropos does not define a detailed description of the social patterns as well as a systematic way to choose and apply them in order to refine the architectural components of the software under development. Hence, in this paper we introduce a process for addressing this issue and a template for describing the social patterns. This template is complemented by three extended UML diagrams [Rumbaugh, Jacobson and Booch 1999] aimed at JADE [Bellifemine, Caire, Trucco and Rimassa 2005] oriented implementation.

This paper is organised as follows: Section 2 reviews the Tropos framework. Section 3 introduces a process for choosing and applying the social patterns. This section also depicts the application of the proposed process to an e-News example. Section 4 presents an overview of the JADE environment. Section 5 introduces our proposal for describing social patterns and implementing them with JADE. This section also describes in detail the e-News agents resulting from application of the patterns. Section 6 discusses related work. Finally, section 7 summarises our work and points out urgent and still open issues.

2. The Tropos Approach

Tropos [Castro, Kolp and Mylopoulos 2002] is a requirements-driven framework aimed at building software that operates within a dynamic environment. It adopts the concepts and models offered by *i** [Yu 1995] framework which includes concepts such as *actor* (actors can be *agents*, *positions* or *roles*)[1], as well as social dependencies among actors including *goal*, *softgoal*, *task* and *resource* dependencies. This means that both the system's environment and the system itself are seen as organizations of actors, each having goals to be fulfilled and each relying on other actors to help them with goal fulfilment. The i* framework also includes the Strategic Dependency (SD) model (Figure 1) for describing the network of relationships among actors, as well as the Strategic Rationale (SR) model for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors by using a means-ends analysis.

Tropos has defined organizational architectural styles [Kolp, Giorgini and Mylopoulos 2002] for agent, cooperative, dynamic and distributed applications to guide the design of the system architecture. These architectural styles *(pyramid, joint venture, structure in 5, takeover, arm's length, vertical integration, co-optation and bidding)* are based on concepts and design alternatives coming from research on organization management. From this perspective, a software system is like a social organization of coordinated autonomous components that interact in order to achieve specific and possibly common goals. Figure 1 shows a MAS architecture in i* for an e-News system that applies the joint-venture style [Kolp, Giorgini and Mylopoulos 2002] at the

[1] A role is an abstract actor. Concrete, physical agents such as human beings or software agents play roles. A position is a collection of roles that are typically played by a single agent [Yu 1995].

architectural design level. An e-News allows users to obtain news from a newspaper web site. News related to a specific subject are extracted from different and distributed news' agencies on the Internet, translated and merged in order to provide good quality news. In a *joint venture* organisational style, a specific joint management actor (e.g. Chief Editor) coordinates tasks and manages the sharing of resources between partner actors (e.g. Editor, Webmaster, Reporter and Photographer). Each partner can manage and control itself on a local dimension and may interact directly with other partners to exchange resources, such as data and knowledge. However, the strategic operation and coordination of such system, and its actors on a global dimension, are the only responsibility of the joint management actor (i.e. Chief Editor).

When an user wishes to read news, she accesses the newspaper website maintained by a Webmaster agent which is responsible for updating the published information. The information to be published is provided and authorised by the Chief Editor agent. The Chief Editor depends on the Editor agents to have the news composed according to the newspaper guideline. Thus, it is necessary for the Chief Editor to share the newspaper guideline among the Editor agents. Each of them is responsible for bringing news about a specific subject. For example, an Editor may be responsible for political news while another one may be responsible for sports news. Each Editor selects one or many reporters/photographers which can find the news of specific subjects (e.g., basketball) to fulfill his sub-guideline (e.g., about sports news). The Chief Editor then edits the news from the Editor agents and forwards the edited news (if authorized) to the Webmaster to publish them.

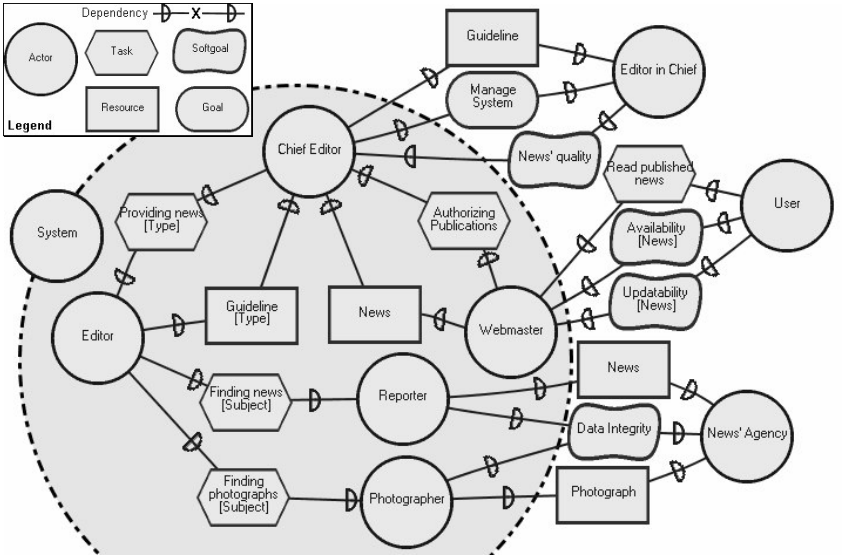

Figure 1. An e-News Joint Venture architecture

As shown in Figure 1, actors are represented as circles; dependums -- goals, softgoals, tasks and resources -- are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form depender⇒dependum⇒dependee. To

support modelling and design during the Detailed Design phase, which is the focus of this paper, Tropos also adopts extensions to UML [Rumbaugh, Jacobson and Booch 1999], like AUML, the Agent Unified Modelling Language [Odell, Parunak and Bauer 2000].

In the sequel we provide an overview of the Detailed Design phase of Tropos.

2.1. Detailed Design Phase

The detailed design phase is intended to introduce additional detail for each architectural component of a system. It consists of defining how the goals assigned to each actor present in the architectural model, are fulfilled by agents performing roles according to social patterns.

Designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Considerable work has been done in software engineering for defining software patterns (see e.g., [Gamma, Helm, Johnson and Vlissides 1995]). Tropos has defined a set of design patterns, named social patterns [Kolp, Giorgini and Mylopoulos 2002], focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems. In particular, the social patterns are inspired by the federated patterns introduced in [Hayden, Carrick and Yang 1999] [Woods and Barbacci 1999]. The framework presented in [Kolp, Do, Faulkner and Hoang 2005] has classified them into two categories:

– *Pair* patterns -- such as booking, call-for-proposal, subscription, or bidding – which describe direct interactions between negotiating agents.

The **Booking Pattern** involves a client and a number of service providers[2]. The client issues a request to book some resource from a service provider. The provider can accept the request, deny it, or propose to place the client on a waiting list, until the requested resource becomes available when some other client cancels a reservation.

The **Subscription Pattern** involves a yellow-page agent and a number of service providers. The providers advertise their services by subscribing to the yellow pages. A provider that no longer wishes to be advertised can request to be unsubscribed.

The **Call-For-Proposals Pattern** involves an initiator and a number of participants. The initiator issues a call for proposals for a service to all participants and then accepts proposals that offer the service for a specified cost. The initiator selects one participant to supply the service.

The **Bidding Pattern** involves an initiator and a number of participants. The initiator organizes and leads the bidding process, and receives proposals. At every iteration, the initiator publishes the current bid; it can accept an order, raise the bid, or cancel the process.

– *Mediation* patterns -- such as monitor, broker, matchmaker, mediator, embassy, or wrapper – which feature intermediary agents that help other agents to reach an agreement on an exchange of services.

In the **Monitor Pattern**, subscribers register for receiving, from a monitor agent, notifications of changes of state in some subjects of their interest. The monitor accepts

[2] Service providers are those agents which can perform some services.

subscriptions, request notifications from subjects of interest, receives notifications of events and alerts subscribers to relevant events. The subject provides notifications of state changes as requested.

In the **Broker Pattern**, the broker agent is an arbiter and intermediates the access to services of an agent (provider) to satisfy the request of a client. Clients access the service of providers by sending requests via the broker. A broker tasks include locating the appropriate provider, forwarding the request to the provider and transmitting results and exceptions back to the client.

In the **Matchmaker Pattern**, a matchmaker agent locates a provider corresponding to a consumer request for service, and then hands the consumer a direct handle to the chosen provider. Contrary to the broker who directly handles all interactions between the consumer and the provider, the negotiation for service and actual service provision are two distinct phases.

In the **Mediator Pattern**, a mediator agent mediates interactions among agents. An initiator addresses the mediator in place of asking directly another colleague, the performer. The mediator has acquaintance models of colleagues and coordinates the cooperation between them. Conversely, each performer has an acquaintance model of the mediator. While a broker only intermediates providers with consumers, a mediator encapsulates interactions and maintains models of initiators and performers behaviours over time.

In the **Embassy Pattern**, an embassy routes a service requested by a foreign agent to a local one and handle the response back. If the access to the local agent is granted, the foreign agent can submit messages to the embassy for translation. The content is translated in accordance to a standard ontology. Translated messages are forwarded to target local agents. The results of the query are passed back out to the foreign agent, translated in reverse.

The **Wrapper Pattern** incorporates a legacy system into a multi-agent system**.** The wrapper agent interfaces the clients to the legacy by acting as a translator between them. This ensures that communication protocols are respected and the legacy system remains decoupled from the rest of the agent system.

Having described the social patterns characteristics, the next step is to have a process for choosing and applying those patterns that will be used for detailing the software architectural design. In the next section, we outline our process proposal and the process usage in the e-News example.

3. Detailed Design Process

In this paper, we propose a process for choosing and applying the social patterns which consists of the following activities:

– **Activity 1.** For each role present in the architectural design of the system, identify some constraints in performing its responsibilities (e.g. some agents in MAS society may request services from strange agents or vice-versa. In this case, an embassy agent may translate the exchanged messages between them. This entails using the Embassy Pattern [Kolp, Giorgini and Mylopoulos 2002]).

Applying this activity to the e-News architecture (Figure 1), we have enumerated the following constraints of e-News system:

i. Necessity to interact with possible non-agent based systems (e.g. News' Agency)

ii. The localisation of all service provider agents is not known at design time.

iii. Presence of a goal (e.g. newspaper guideline) that can be subdivided into sub-goals (a sub-guidelines for each type of news) and whose fulfilment can be achieved through a cooperation of many service providers

iv. Presence of a goal (sub-guideline fulfilment) that may be achieved through activity delegation.

– **Activity 2.** In order to select the proper patterns to be applied, evaluate the suitability of the social patterns in solving the constraints of each role present in the architectural design of the system.

Applying this activity to the e-News architecture (Figure 1), we have enumerated the following considerations:

i. The Wrapper pattern enables the Reporters and Photographers to interact with News' Agency and to handle the information exchanged by them.

ii. The Matchmaker pattern offers a Yellow Pages service for locating service provider agents (e.g. used by both the Chief Editor agent to locate the Editor agents and the Webmaster agent to locate the Chief Editor agent).

iii. The Mediator pattern allows the Chief Editor to coordinate the cooperation of Editor agents to fulfill the newspaper guideline.

iv. The Broker pattern enables the fulfillment of a sub-guideline (about a specific type of news) by requesting to a reporter/photographer to find news related to this sub-guideline.

– **Activity 3.** Matching the actors present in MAS architecture with the roles composing the chosen social patterns. When applying a pattern we may need to add new roles in the system, according to that pattern.

Applying this activity to the e-News architecture (Figure 1), we have matched the following roles:

i. Each reporter/photographer can play the role of a Wrapper agent interacting with news' agencies and translating the query for news into the format of the News' Agency and translating the News' Agency response into the data model used by the Editor agents.

ii. A Matchmaker agent must be added to the system in order to maintain the location of service provider agents (e.g. Chief Editor and Editor) and answer the clients' requests (e.g. Chief Editor and Webmaster) letting the clients to interact directly with the providers.

iii. The Chief Editor can play the role of a Mediator agent decomposing the newspaper guideline into sub-guidelines for each type of news (e.g., sports, politics, etc.) to be fulfilled by each Editor agent.

iv. Each Editor can play the role of a Broker agent selecting one or many reporters/photographers which can provide the news of specific subjects (e.g., basketball) to fulfil his sub-guideline (e.g., about sport news).

– **Activity 4**. According to the social patterns that have been applied, establish the relationships among the roles.

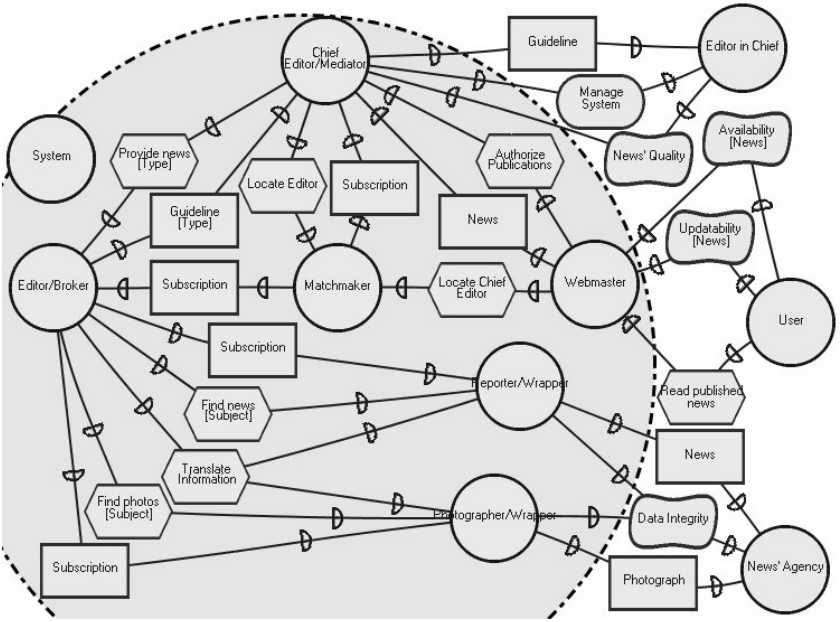

Figure 2. Detailed design of the e-News architecture

Applying this activity to the e-News architecture (Figure 1), we can define various relationships among roles in the system. For example, the Editor and Chief Editor agents, which are service providers, depend on the Matchmaker agent to subscribe themselves in the Matchmaker's yellow page service (illustrated as Subscription resource dependency in Figure 2). The Webmaster agent, which is a service client[3], depends on the Matchmaker agent to get the identification of a Chief Editor (illustrated as Locate Chief editor task dependency in Figure 2). The Chief Editor, which is also a service client, depends on the Matchmaker agent to get the identification of one or more Editor agents (illustrated as Locate Editor task dependency in Figure 2). Each Editor agent has its own catalogue of Reporters and Photographers which can be contacted to fulfill its sub-guideline. This catalogue works as a yellow pages service and the Reporters/Photographers depend on the Editor agent to subscribe themselves to the Editor's catalogue (illustrated as Subscription resource dependency in Figure 2). The Editor, after locating the suitable Reporters/Photographers, depends on them to find the news about a specific subject (illustrated as Find News [Subject] and Find Photos [Subject] tasks dependencies in Figure 2) in order to fulfill its sub-guideline. The Reporters/Photographers, in their turn, interacts directly with the News' Agency, which is an external non-agent based system. The Reporters and Photographers agents depend on News' Agency system to get the news and photos related to a specific

[3] Service clients are those agents which needs the services performed by other agents.

subject. However, this information may be in a format different from the one used by the agents composing the e-News system. This kind of translation is provided by the Reporters/Photographers (illustrated as Translate Information task dependency in Figure 2).

The performance of these activities will produce the global MAS architectural detailed design depicted in Figure 2. The application of the patterns will promote decoupling among the system's agents and therefore, flexibility in the system architectural design. Hence, if a Reporter agent contacted by some Editor agent stop running, for example, the Editor may replace that Reporter by locating another one in the yellow pages.

The Detailed Design phase also includes a careful description of each agent present in the MAS detailed architecture (Figure 2). UML based diagrams are used to represent the collaboration, structure and behaviour of the agents (explained later in section 5). We also investigate how these diagrams could be implemented in JADE [Bellifemine, Caire, Trucco and Rimassa 2005], a popular agent oriented environment.

4. JADE Overview

Several agent oriented development environments have been proposed in the literature. Some are compliant with the FIPA specifications [FIPA 2005], such as JACK [JACK 2005], FIPA-OS [FIPA-OS 2005] and JADE [Bellifemine, Caire, Trucco and Rimassa 2005]. In this paper, we have chosen JADE as a suitable agent platform to support the implementation of MAS.

In JADE, a behaviour represents a task that an agent can carry out. Behaviours are logical execution threads that can be composed in various ways to achieve complex execution patterns and can be initialised, suspended and spawned at any given time [Moraitis, Petraki and Spanoudakis 2002]. One of the most important features that JADE agents provide is the ability to communicate. Messages exchanged by JADE agents have a format specified by the ACL language defined by FIPA [FIPA 2005]. This format comprises a number of fields and in particular the performative field which was very important in the development of our proposal. The performative is the communicative intention indicating what the sender intends to achieve by sending the message. It will be further explained in the Section 5.

The yellow pages service in JADE (according to the FIPA specification) is provided by an agent called DF (Directory Facilitator). JADE also provides a support for content languages and ontologies which, among other things, verifies if the message to be exchanged complies with the rules of the defined ontology [Bellifemine, Caire, Trucco and Rimassa 2005].

In the next section, we discuss how agent-oriented design patterns could be properly described, and provide some means of implementing them in JADE. Examples of patterns specializations in the e-News system are also considered.

5. Describing Social Patterns

Tropos proposes social patterns but does not provide a standard template to describe them in order to facilitate the solution deployment. To address this issue, we introduce a

template complemented by three UML extended diagrams to capture the behaviour, structure and collaboration of each pattern.

5.1. Social Pattern Template

In this work we present the social patterns following a subset of the template proposed by GOF [Gamma, Helm, Johnson and Vlissides 1995]. For example, in Table 1 we define the Matchmaker Pattern that has been applied to our e-News system.

Table 1. The Template for a Pattern Description

Template Element	Description
Name	Matchmaker Pattern
Intent	To locate a provider for a given service requested by a client and letting the client directly interact with the provider.
Applicability	Use when an agent (client) needs to directly interact with another agent (provider) to use its services but does not know what agent offers the desired service.
Motivation Example	An agent (client) may need a specific service provided by another unknown agent (provider). An intermediary agent (matchmaker) can find the provider agent which offers the requested service and give the provider's identification to the client, which can then directly interact with it. For example, a reporter agent needs to interact with a news' agency to obtain news about a specific subject. The news' agency to be contacted is going to be known at run time by a matchmaker agent which works as a yellow page informing clients (i.e., the reporter) the identification of the agent (i.e., the news' agency) that provides the requested service (i.e., the news).
Participants	The Client requests the identification of an agent that provides a specific service. The Matchmaker finds the Provider of the requested service and gives its identification to the Client. The Provider must subscribe the Matchmaker yellow page service in order to be found by the clients requesting its services.

It is expected that the template is accompanied by some detailed examples given in a certain implementation platform. In this paper, we have chosen JADE as the target platform. Hence, we can now specify more clearly the interaction protocols (collaboration) among the agents involved in each pattern. The agent's internal behaviour description is also facilitated when we have a specific implementation platform in mind. Moreover, we could promote a direct code generation from the pattern structural diagram.

5.2. Collaboration

The collaboration among agents involved in a pattern can be described in term of UML's sequence diagrams. For example, in Figure 3 we have Client, Matchmaker and Provider agents. These agents are exchanging messages, characterised by FIPA performatives. Notice that the information between "[]" suggests the message content. These messages define the Matchmaker pattern protocol.

A Provider can send a message characterised by an INFORM to the Matchmaker agent indicating a subscription/unsubscription of its services into the yellow page

maintained by the Matchmaker. The Matchmaker can reply to it with a message characterised by either:

i. an ACCEPT_PROPOSAL performative, indicating that the Matchmaker has accepted the subscription/unsubscription to the yellow page;

ii. a REFUSE performative, indicating that the Matchmaker has refused the subscription/unsubscription to the yellow page.

A Client agent can send a message characterized by a REQUEST performative indicating a request to locate one or more providers of a specific service. The Matchmaker agent can reply with a message characterized by either:

i. a REFUSE performative, indicating that the service providers have not been located;

ii. an INFORM performative, indicating not only that the service providers have been located but also the providers' identification.

If the Client agent receives the identification of the service provider(s), it can send a message characterized by a REQUEST performative, indicating that the Client requests the Provider to perform a specific service. The provider can reply with:

i. a REFUSE performative, indicating that the service has not been performed; or

ii. an INFORM performative, indicating that the service has been performed.

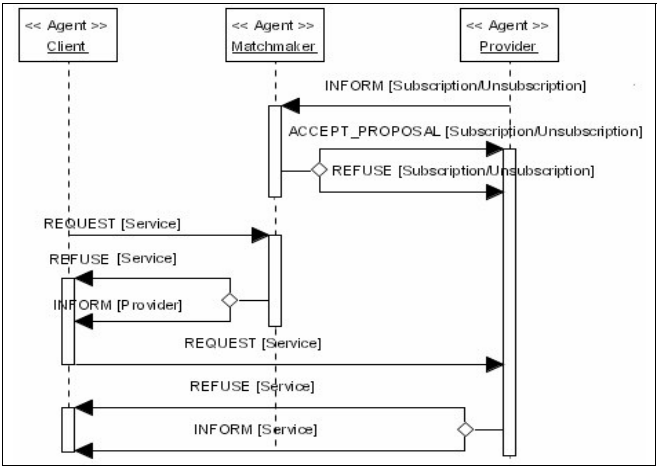

Figure 3. Collaborations

A specific scenario reflecting the usage of the Matchmaker pattern in the e-News architectural detailed design (Figure 2) could involve the Chief Editor and the Editor agents. In Figure 3, the Client agent would be specialized as a Chief Editor, while the Provider agent would be an Editor. Notice that the scenario we are going to describe will use a subset of the protocol's messages depicted in Figure 3. The Chief Editor agent requests the identification of one or more Editor agents (by specializing the REQUEST [Service] message between Client and Matchmaker agents) to the Matchmaker agent. If there is some Editor subscribed in the Matchmaker's yellow

pages, the Chief Editor's request will be answered with the Editor's identification (by specializing the INFORM [Provider] message between Matchmaker and Client agents). Thus, the Chief Editor agent will be able to request directly to the specific Editor agent the achievement of a sub-guideline (by specializing the REQUEST [Service] message between Client and Provider agents). Assuming that the Editor agent has fulfilled its sub-guideline, the Chief Editor's request will be answered with the required news (by specializing the INFORM [Service] message between Provider and Client agents).

5.3. Structure

The diagram described in Figure 4 shows the structure of the Matchmaker pattern. This diagram is an extension of the UML's class diagram in order to provide a sufficient description of the agents involved in the pattern aiming at a JADE implementation. The agent is featured as a class stereotyped with <<Agent>>. Every agent performs one or more behaviours which are also featured as a class stereotyped with <<Behaviour>>. To represent that an agent can perform a behaviour we use an association between that agent and its behaviour. The navigability is placed from agent to behaviour (Figure 4).

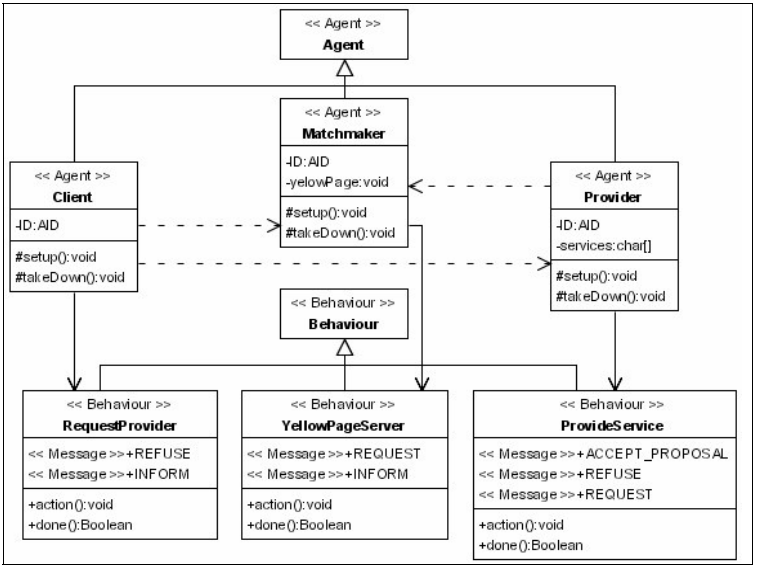

Figure 4. Structure

The agents' ability to establish a communication with other agents is represented by a dependency between the communicating agents. Notice that the <<Agent>> and the <<Behaviour>> classes extend the JADE's Agent and Behaviour classes, respectively. The <<message>> stereotype in the <<Behaviour>> class indicates the FIPA performatives that an agent performing that behaviour may handle, i.e. the agent communication interface.

The class diagram depicted in Figure 4 can be specialized to reflect some agents involved in the Matchmaker pattern. For example, if we apply it to the e-News architectural detailed design. The Client agent would be specialized as the Webmaster,

while the Provider agent would be a Chief Editor. The Behaviour classes (RequestProvider, YellowPageServer and ProvideService) illustrated in the Figure 4 are maintained in the specialization of the Matchmaker pattern. Observe that several other behaviours can be added to each agent in the specialization of the pattern. For instance, the Webmaster agent could have a behaviour to maintain the website while the Chief Editor agent could have a behaviour for handling the guideline decomposition and the newspaper composition.

5.4. Behaviour

The behavioural diagram is an extension of the UML's activity diagram. It shows activities which trigger message sending, activities performed after the reception of certain messages and the evaluation of conditions deciding which activity is going to be performed. In particular, the diagram depicted in Figure 5 describes the whole behaviour of the Matchmaker pattern, capturing all the possible activities to be performed by agents involved in the pattern.

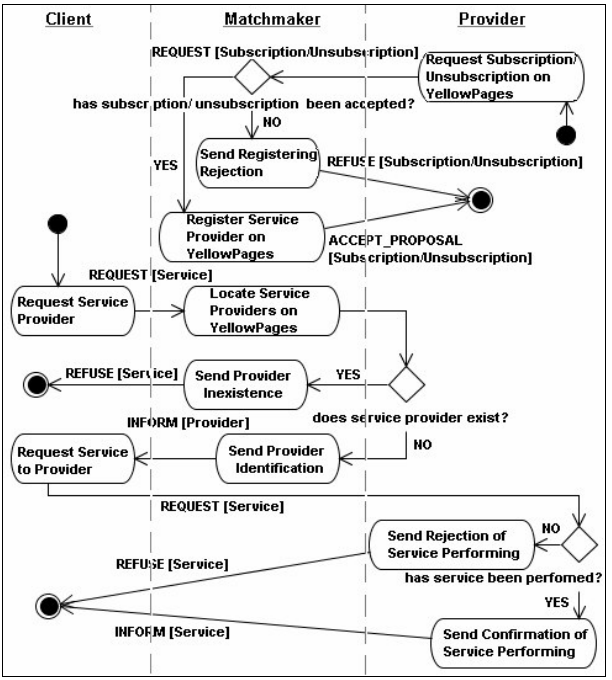

Figure 5. Behaviour

Initially the Client agent sends a REQUEST message to the Matchmaker agent aiming to get a provider of a specific service. The Matchmaker agent verifies if there is some agent providing the required service registered in the yellow pages. If no Provider is available for the required service the Matchmaker agent sends a REFUSE message warning the Client agent about the inexistence of the Provider. Otherwise the Matchmaker sends an INFORM message along with the Provider(s) identification(s). When the Client agent receives the Provider identification, it can establish a direct

interaction with the Provider and request the execution of the required service. If the Provider has performed the service, it sends an INFORM message to the Client agent, otherwise he sends a REFUSE message. To register/unregister a Provider of a service in the Yellow Pages, the Matchmaker must receive a REQUEST message from this Provider agent. The Matchmaker can accept or reject registering/unregistering the service Provider. In the former case, the Matchmaker sends an ACCEPT_PROPOSAL message to the service Provider agent. In the latter case, the Matchmaker sends a REFUSE message.

In our e-News case study we could have specialized this behavioural diagram to reflect a specific scenario involving the agents of the Matchmaker pattern. However, for the sake of space, this description is not shown in this paper.

5.5. JADE implementation

We can also use JADE to define the code skeleton of a social pattern. In fact our purpose is to promote, in the future, an automatic code generation from the detailed design diagrams. Observe the association between <<Agent>> class and <<Behaviour>> class in Figure 4. It indicates that the <<Behaviour>> class is an inner class of the <<Agent>> class in the JADE code (Figure 6). In this case, the behaviour can only be performed by the agent which possesses its declaration. Notice that when a behaviour can be performed by an agent, this behaviour is added to the agent's setup() method.

```
public class Client extends Agent {
    protected void setup() {// Put agent initializations here
        this.addBehaviour(              new RequestProvider()); // Add the behaviour }
    protected void takeDown() {         // Put agent clean-up operations here }
    private class RequestProvider extends Behaviour {   /**  Inner class  */
        public void action() {
              ACLMessage reply = myAgent.receive(mt);
              if  (reply != null) {
                   if (reply.getPerformative() == ACLMessage.INFORM )  {   }
                   else if   (reply.getPerformative()==ACLMessage.REFUSE)  {   } }
               else {     block();   } }//end of action method
         public boolean done() {    return true;   }
     }  // End of inner class RequestProvider      } // End of agent class
```

Figure 6. Sample Code for Client class

In Figure 4, the dependency between the <<Agent>> classes indicates that a Client agent can establish a communication with a Provider agent. Also a Client agent and a Provider agent can establish communication with the Matchmaker agent. It is important to note that the Matchmaker agent is already implemented by *jade.domain.DFService* library (the Matchmaker is called Directory Facilitator in JADE). Thus, any agent implemented in JADE may use the yellow pages service to locate service providers. The information stereotyped with <<Message>> in the second compartment of the <<Behaviour>> class (Figure 4) indicates the FIPA performatives that can be handled by the agent performing that behaviour. Thus, each performative present in the Behaviour class will generate an IF inside its action() method (Figure 6). Methods, attributes, generalization, GUI classes and both the Agent and Behaviour classes are implemented similarly to object-oriented in Java [Java 2005]. For the sake of

space, in the sequel, we only show the code generated from the <<Agent>> Client class presented in Figure 4.

Although the patterns' descriptions are driven to an implementation using JADE, they are general descriptions which can be specialised according to the domain of the application under development. Our intention is to produce a pattern description which is detailed enough to facilitate the specialisation of the pattern when it is to be applied in architectural detailed design, as well as the latter JADE implementation.

For instance, let's consider the class diagram specialization where the Webmaster agent is the Client interacting with the Matchmaker agent. The resulting JADE code generation of the Webmaster agent class would be similar to those illustrated in Figure 6. The only difference is that the class name would be changed to Webmaster.

6. Related Works

This section discusses the main design patterns proposed in agent-oriented community. Many of them deal with patterns for designing mobile agent-based applications, such as [Aridor and Lange 1998] [Tahara, Ohsuga and Honiden 1999] [Deugo, Oppacher, Kuester and Von Otte 1999]. Others propose a catalogue of coordination patterns for multi-agent systems [Hayden, Carrick and Yang 1999], or a social perspective on agent-oriented design patterns [Kolp, Do, Faulkner and Hoang 2005].

The proposal presented in [Kolp, Giorgini and Mylopoulos 2002] shows a preliminary catalogue of social patterns and the approach proposed in [Kolp, Do, Faulkner and Hoang 2005] conceptualises a framework to explore these patterns. In particular, five different complementary dimensions are required to describe the detailed design models reflecting particular aspects of MAS architectures, such as social, intentional, structural, communicational and dynamic dimensions. The first dimension is described using the i* framework [Yu 1995], while the last three dimensions are described using extensions of UML [Rumbaugh, Jacobson and Booch 1999]. The agent architecture focused by this framework is a deliberative architecture called BDI (belief-desire-intention) [Rao and Georgeff 1995]. The chosen target environment for agent-oriented development is JACK [JACK 2005], since it supports implementation for BDI architectures. This work also introduces the generation of code from a given agent specifications to JACK platform. However, this approach does not present an explicit method for applying the patterns during the development of multi-agent systems. The patterns are applied in an ad hoc way and their choice is not clearly justified.

On the other hand, in our work we outline a process which facilitates the choice and application of the social patterns to MAS architectural detailed design. Moreover, although JADE enables a direct codification of reactive agent architecture, an extension for supporting implementation of BDI agent architecture has been proposed and is called JADEX [Braubach, Pokahr and Lamersdorf 2004]. Our approach also includes a template for specifying the social patterns in order to promote a better understanding of them. To support MAS development using JADE, we have proposed three UML extended diagrams to capture the behaviour, structure and collaboration of each social pattern. We claim that these three diagrams along with a template are sufficient to capture the information required to describe an agent-oriented design pattern. Notice that these diagrams focus on JADE implementation aiming at an automatic code generation.

7. Conclusions and Future Work

This paper focuses on the detailed design phase of Tropos aiming to provide a template and some extended UML diagrams for describing the social patterns as well as provide some means of implementing them in JADE. Our purpose is to promote an efficient utilization of social patterns in order to achieve a successful detailed architectural design of multi-agent systems. To this end, we have also outlined a method for choosing and applying the social patterns to a specific application in order to detail the MAS architecture in terms of more specific software agents.

Although our approach offers a pattern description which facilitates its application and subsequent JADE implementation, it presents some limitations. For example, if we want to implement a MAS using FIPA-OS [FIPA-OS 2005], the diagrams proposed in this work will not help the designer in the detailed design of the system. Since each MAS implementation platform differs considerably in the concepts used during implementation, it would be necessary to have a description/specification of the patterns for each different target platform. On the other hand, the process outlined in this work is independent of agent implementation environment.

We are also working on the social patterns description by using UML extended diagrams which are platform independent. Our purpose is to enable easy implementation of the social patterns using other FIPA compliant platforms, besides JADE. Some future work includes investigating how agents developed with Tropos can be implemented using JADEX - a software framework for the creation of goal-oriented agents following the BDI model. Moreover, applying the social patterns to other real case studies is also required in order to further detail the process guidelines. To validate the usefulness of our proposal, we intend to compare it with other approaches for describing, choosing and applying design patterns, both object-oriented and agent-oriented ones.

Acknowledgements

This work was partially supported by CNPq and CAPES grants.

8. References

Aridor, Y. and Lange, D. (1998) “Agent design patterns: Elements of agent application design”. In: Proceedings of the 2nd International Conference on Autonomous Agents, Agents’98, St. Paul, USA, p. 108–115.

Bellifemine, F., Caire, G., Trucco, T. and Rimassa, G. (2005) Jade Programmer’s Guide - JADE 3.3. http://sharon.cselt.it/projects/jade/, Last access in March.

Braubach, L., Pokahr, A. and Lamersdorf, W. (2004) “Jadex: A Short Overview”, In: Main Conference Net.ObjectDays 2004, AgentExpo.

Castro, J., Kolp, M. and Mylopoulos, J. (2002) “Towards Requirements-Driven Information Systems Engineering: The Tropos Project”, Information Systems News, Elsevier, vol 27, p. 365-89.

Deugo, D., Oppacher, F., Kuester, J. and Von Otte, I. (1999) “Patterns as a Means for Intelligent Software Engineering”, In: Proceedings of the International Conference on Artificial Intelligence, School of Computer Science, Carleton University, Ottawa, Ontario, Canada, p. 605-611.

FIPA. The Foundation for intelligent agents (2005), http://www.fipa.org, Last access in March.

FIPA-OS (2005) "A component-based toolkit enabling rapid development of FIPA compliant agents", http://fipa-os.sourceforge.net/, Last access in March.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

Hayden, S., Carrick, C. and Yang, Q. (1999) "Architectural design patterns for multiagent coordination", In: Proceedings of the 3rd International Conference on Autonomous Agents, Agents'99, Seattle, USA.

JACK Intelligent Agents (2005) http://www.agent-software.com/, Last access in March.

Java Technology (2005) http://www.java.sun.com/, Last access in March.

Kendall, E. A., Krishna, P. V. M., Pathak, C. V. and Suresh, C. B. (1998) "Patterns of Intelligent and Mobile Agents", In: Proceedings of the 2nd International Conference on Autonomous Agents, Agents'98, St. Paul, USA, p. 92 – 99.

Kolp, M., Giorgini, P. and Mylopoulos, J. (2002) "Information Systems Development through Social Structures", In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ishia, Italy.

Kolp, M., Do, T. T., Faulkner, S. and Hoang, H. T. T. (2005) "Introspecting Agent Oriented Design Patterns", In: S. K. Chang (Eds), Advances in Software Engineering and Knowledge Engineering, vol. III, World Publishing.

Moraitis, P., Petraki, E. and Spanoudakis, N. (2002) "Engineering JADE Agents with the Gaia Methodology", In: Agent Technologies, Infrastructures, Tools and Applications for E-Services, R. Kowalczyk, J. Muller, H. Tianfield, R. Unland (editors), Best (revised) papers of NODe 2002 Agent-Related Workshops, (LNAI2592), p. 77-92.

Odell, J., Parunak, H. V. D. and Bauer, B. (2000) "Extending UML for Agents", In: Proceedings of the Agent-Oriented Information Systems at the 17th National Conference on Artificial Intelligence. iCue Publishing, p. 3–17.

Rao, A.S. and Georgeff, M.P. (1995) "BDI agents: from theory to practice", Technical Note 56, Australian Artificial Intelligence Institute.

Rumbaugh, J., Jacobson, I. and Booch, G. (1999) The Unified Modeling Language – Reference Manual. Addison Wesley.

Tahara, Y., Ohsuga, A. and Honiden, S. (1999) "Agent system development method based on agent patterns", In: Proceedings of the 21st International Conference on Software Engineering, Los Angeles, California, USA, p. 356 – 367.

Woods, S. G. and Barbacci, M. (1999) "Architectural Evaluation of Collaborative Agent-Based Systems", Technical Report, CMU/SEI-99-TR-025, SEI, Carnegie Mellon University, USA.

Yu, E. (1995) "Modelling Strategic Relationships for Process Reengineering". Ph.D. thesis. Department of Computer Science. University of Toronto. Canada.