AGENT-ORIENTED DESIGN PATTERNS

T. Tung Do, Manuel Kolp, Stéphane Faulkner, Alain Pirotte
Information System Unit, University of Louvain
1, Place des Doyens, 1348 Louvain-La-Neuve, Belgium
Email: {do, kolp, faulkner, pirotte}@isys.ucl.ac.be

Keywords: agent oriented software engineering, design patterns, multi-agent systems

Abstract: Multi-Agent Systems (MAS) architectures are gaining popularity over traditional ones for building open, distributed, and evolving software required by today's corporate IT applications such as eBusiness systems, web services or enterprise knowledge bases. Since the fundamental concepts of multi-agent systems are social and intentional rather than object, functional, or implementation-oriented, the design of MAS architectures can be eased by using social patterns. They are detailed agent-oriented design idioms to describe MAS architectures as composed of autonomous agents that interact and coordinate to achieve their intentions, like actors in human organizations. This paper presents social patterns and focuses on a framework aimed to gain insight into these patterns. The framework can be integrated into agent-oriented software engineering methodologies used to build MAS. We consider the Broker social pattern as a combination of patterns and use it to illustrate the framework. The automatation of patterns design is also overviewed.

1 INTRODUCTION

The explosive growth of application areas such as electronic commerce, knowledge management, peer-to-peer and mobile computing has profoundly changed our views on information systems engineering. Systems must now be based on open architectures that continuously evolve to accommodate new components and meet new requirements. These new requirements call, in turn, for new concepts and techniques for engineering and managing information systems. Therefore, Multi-Agent System (MAS) architectures are gaining popularity over traditional systems, including object-oriented ones.

Developing enterprise information systems with a MAS architecture permits a better match between system architectures and their operational environment. A MAS can be seen as a *social organization* of autonomous software entities (agents) that can flexibly achieve agreed-upon *intentions* through their interactions. MAS support dynamic and evolving structures which can change at run-time to benefit from the capabilities of new system entities or replace obsolete ones.

MAS architectures become rapidly complicated due to the ever-increasing complexity of these new business domains and their human or organizational actors. An important technique that helps to manage this complexity when constructing and documenting such architectures is the reuse of development experience and knowhow. Over the past few years, *design patterns* have significantly contributed to the reuse of design expertise, improvement application documentation and more flexible and adaptable designs. The idea behind a pattern is to record the essence of a solution to a design problem so as to facilitate its reuse when similar problems are encountered.

Considerable work has been done in software engineering on defining design patterns (see e.g., (Gamma, 1995)). Unfortunately, they focus on object-oriented rather than agent-oriented systems. In the area of MAS, little emphasis has been put on social and intentional aspects. Moreover, the proposals of agent patterns that address those aspects (see e.g., (Aridor, 1998; Deugo, 1999; Hayden, 1999)) are not aimed at the design level, but rather at the implementation of lower-level issues like agent communication, information gathering, or connection setup. For instance, the FIPA (FIPA) identified and defined a set of agent's interaction protocols that are only restricted to the communication.

Since there is a fundamental mismatch between the concepts used by the object-oriented paradigm (and other traditional mainstream software engineering approaches) and the agent-oriented view, there is a need to develop patterns that are specifically tailored to the development of multi-agent systems using agent-oriented primitives.

The catalogue of social patterns proposed in (Kolp, 2001) constitutes a contribution to the definition of agent-oriented design patterns for MAS architectures. This paper proposes and conceptualizes the SKwyRL patterns framework to explore social patterns and facilitate the building of MAS during detailed design as well as the generation of code for agent implementation. It models and introspects the patterns along different complementary dimensions.

The paper is organized as follows. In Section 2, we overview the patterns. Section 3 proposes the framework and illustrates its different modeling dimensions through the Broker pattern. The automation of social patterns is overviewed in Section 4. Section 5 points to some conclusions.

2 SOCIAL PATTERNS

In the following, we present patterns based on social and intentional dimensions that are recurrent in multi-agent and cooperative systems. *Pair* patterns describe direct interactions between negotiating agents, while *Mediation* patterns feature intermediate agents that help other agents reach agreement about an exchange of services.

2.1 Pair Patterns

The **Booking** pattern involves a client and a number of service providers. The client issues a request to book some resource from a service provider. The provider can accept the request, deny it, or propose to enter the client in a waiting list, until the requested resource becomes available.

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 a client and a number of service providers. The client issues a call for proposals for a service to all service providers and then accepts proposals that offer the service for a specified cost. The client selects one service provider to supply the service.

The **Bidding** pattern involves a client and a number of service providers. The client organizes and leads the bidding process, and receives proposals. At every iteration, the client publishes the current bid; it can accept an offer, raise the bid, or cancel the process.

2.2 Mediation Patterns

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 subscriptions, requests information from the subjects of interest, and alerts subscribers accordingly.

In the **Broker** pattern, the broker agent is an arbiter and intermediary that requests services from providers to satisfy the request of clients. The rest of the paper details latter the pattern to illustrate the SKwyRL social patterns framework.

In the **Matchmaker** pattern, a matchmaker agent locates a provider for a given service requested by a client, and then lets the client interact directly with the provider, unlike brokers, who handle all interactions between clients and providers.

In the **Mediator** pattern, a mediator agent coordinates the cooperation of performer agents to satisfy the request of an initiator agent. While a matchmaker simply matches providers with clients, a mediator encapsulates interactions and maintains models of the capabilities of initiators and performers over time.

In the **Embassy** pattern, an embassy agent routes a service requested by an external agent to a local agent. If the request is granted, the external agent can submit messages to the embassy for translation in accordance with a standard ontology. Translated messages are forwarded to the requested local agent and the result of the query is passed back out through the embassy to the external agent.

The **Wrapper** pattern incorporates a legacy system or database into a multi-agent system. A wrapper agent is an embassy that interfaces system agents with the legacy system/database by acting as a translator. This ensures that communication protocols are respected and that the legacy system/database remains decoupled from the rest of the agent system.

3 THE SKwyRL SOCIAL PATTERNS FRAMEWORK

This section describes SKwyRL[1], a conceptual framework based on five complementary modeling di-

[1] Socio-Intentional ArChitecture for Knowledge Systems and Requirements ELicitation (http://www.isys.ucl.ac.be/skwyrl/)

mensions to investigate social patterns. Each dimension reflects a particular aspect of a MAS architecture.

Other modeling frameworks like AUML (Bauer, 2001) represent the detailed design of a MAS by adapting and extending object-oriented concepts. We believe that a conceptual framework primarily based on MAS concepts (such as beliefs, intentions, goals, ...) is more appropriate than extended object-oriented frameworks that treat agents as objects of a particular type.

- The *social dimension* identifies the relevant agents in the system and their intentional interdependencies.
- The *intentional dimension* identifies and formalizes services provided by agents to realize the intentions identified by the social dimension, independently of the plans that implement those services. This dimension answers the question: "What does each service do?"
- The *structural dimension* operationalizes the services identified by the intentional dimension in terms of agent-oriented concepts like beliefs, events, plans, and their relationships. This dimension answers the question: "How is each service operationalized?"
- The *communicational dimension* models the temporal exchange of events between agents.
- The *dynamic dimension* models the synchronization mechanisms between events and plans.

The social and the intentional dimensions are specific to MAS. The structural, communicational, and dynamic dimensions of the architecture are also relevant for traditional (non-agent) systems, but we have adapted and extended them with agent-oriented concepts. This section details the dimensions and illustrates them with the Broker pattern.

3.1 Social Dimension

The social dimension specifies a number of agents and their intentional interdependencies using the i* model (Yu, 1995). Figure 1 shows a social-dimension diagram for the Broker pattern. Agents are drawn as circles and their intentional dependencies as ovals. An agent (the *depender*) depends upon another agent (the *dependee*) for an intention to be fulfilled (the *dependum*). Dependencies have the form *depender* → *dependum* → *dependee*.

The Broker pattern can be considered as a combination of (1) a Subscription pattern (shown enclosed within dashed boundary (a)), that allows service providers to subscribe their services to the Broker agent and where the Broker agent plays the role of yellow-page agent, (2) one of the other pair patterns

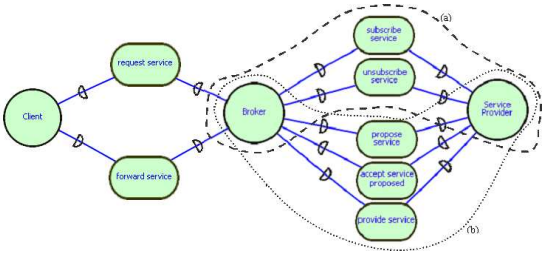

Figure 1: Social diagram for the Broker pattern

(Booking, Call-for-Proposals, or Bidding) whereby the Broker agent requests and receives services from service providers (in Figure 1, it is a Bidding pattern, shown enclosed within dotted boundary (b)), and (3) interaction between Broker and client: the Broker agent depends on the client for sending a service request and the client depends on the Broker agent to forward the service.

To formalize intentional interdependencies, we use Formal Tropos (Fuxman, 2001), a first-order temporal-logic language that provides a textual notation for i* models and allows to describe dynamic constraints. A *forward service* dependency can be defined in Formal Tropos as follows.

Dependum Forward Service
Mode: Achieve
Depender: Client cl
Dependee: Broker br
Fulfillment:

$$(\forall sr : ServiceRequest, st : ServiceType)$$
$$request(cl, br, sr) \wedge provide(br, st)$$
$$\wedge\, ofType(sr, st) \rightarrow \Diamond\, received(cl, br, st)$$

[Broker *br* successfully provides its service to client *cl* if all requests *sr* from *cl* to *br*, that are of a type *st* that *br* can handle, are eventually satisfied]

3.2 Intentional Dimension

While the social dimension focuses on interdependencies between agents, the intentional view aims at modeling agent rationale. It is concerned with the identification of *services* provided by agents and made available to achieve the intentions identified in the social dimension. Each service belongs to one agent. Service definitions can be formalized as intentions that describe the fulfillment condition of the service. The collection of services of an agent defines its behavior.

Table 1 gives an informal definition for several services of the Broker pattern. The `FindBroker` service allows a client to find a broker that can handle a given service request. The client then sends the request to the broker through the `SendServiceRequest`

Table 1: Some services of the Broker pattern

Service Name	Informal Definition	Agent
FindBroker	Find a broker that can provide a service	Client
SendService Request	Send a service request to a broker	Client
QuerySP Availability	Query the knowledge for information about the availability of the requested service	Broker
SendService RequestDecision	Send an answer to the client	Broker
RecordBR Refusal	Record a negative answer from a broker	Client
RecordBR Acceptance	Record a positive answer from a broker	Client
RecordClient ServiceRequest	Record a service request received from a client	Broker
CallFor Proposals	Send a call for proposals to service providers	Broker
RecordAndSend SPInformDone	Record a service received from a service provider	Broker

service. The broker can query its belief knowledge with the `QuerySPAvailability` service and answer the client through the `SendServiceRequestDecision` service. If the answer is negative, the client records it with its `RecordBRRefusal` service. If the answer is positive, the broker records the request (`RecordClientServiceRequest` service) and then broadcasts a call (`CallForProposals` service) to potential service providers. The client records acceptance by the broker with the `RecordBRAcceptance` service.

The Bidding pattern could be used here, but this presentation omits it for brevity.

The broker then selects one of the service providers among those that offer the requested service. If the selected provider successfully returns the requested service, it informs the broker, that records the information and forwards it to the client (`RecordAndSendSPInformDone` service).

Services can also be formalized in Formal Tropos (Fuxman, 2001).

3.3 Structural Dimension

While the intentional dimension answers the question "What does each service do?", the structural dimension answers the question "How is each service operationalized?".

Services are operationalized as *plans*, that are sequences of actions. The knowledge that an agent has (about itself or its environment) is stored in its *beliefs*. An agent can act in response to the *events* that it handles through its plans. A plan, in turn, is used by the agent to read or modify its beliefs, and send events to other agents or post them to itself.

The structural dimension models these agent-oriented concepts (*plan, belief, event, agent*) with an UML style class diagram extended for MAS. Due to lack of space, in this paper, we do not study them. A more detail definition of these concepts could be found in (Do, 2003).

3.4 Communication Dimension

Agents interact with each other by exchanging events. The communicational dimension models, in a temporal manner, events exchanged in the system. We adopt the sequence diagram model proposed in AUML (Bauer, 2001) and extend it: *agent_name/role:pattern_name* expresses the role (*role*) of the agent (*agent_name*) in the pattern (*pattern_name*); the arrows are labeled with the name of the exchanged events.

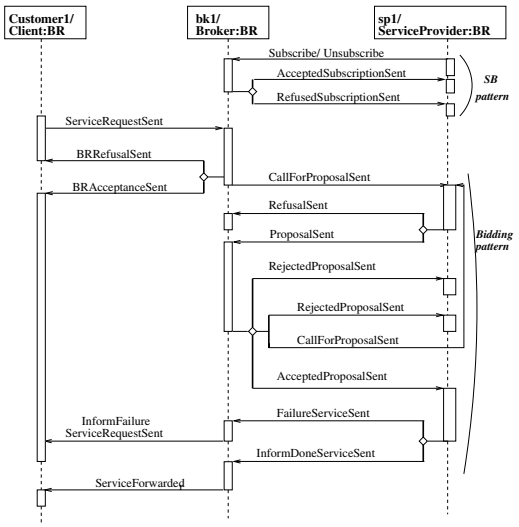

Figure 2: Communication Diagram - Broker

Figure 2 shows the sequence diagram of our Broker pattern. It also depicts the subscription (SB) and bidding patterns the broker pattern embeds. The client (`customer1`) sends a service request (`ServiceRequestSent`) containing the characteristics of the service that it wishes to obtain from the broker. The broker may alternatively answer with a denial (`BRRefusalSent`) or a acceptance (`BRAcceptanceSent`).

In case of acceptance, the broker sends a call for proposals to the registered service providers (`CallForProposalSent`). The bidding pattern is then applied to model the interaction between

the broker and the service providers. The service provider either fails or achieves the requested service. The broker then informs the client by sending a `InformFailureServiceRequestSent` or a `ServiceForwarded`, respectively.

3.5 Dynamic Dimension

In MAS, a plan can be invoked by an event that it handles and it can create new events. Relationships between plans and events can rapidly become complex. To cope with this problem, we propose to model the synchronization and the relationships between plans and events with activity diagrams extended for agent-oriented systems (See Figure 3). These diagrams specify the events that are created in parallel, the conditions under which events are created, which plans handle which events, and so on.

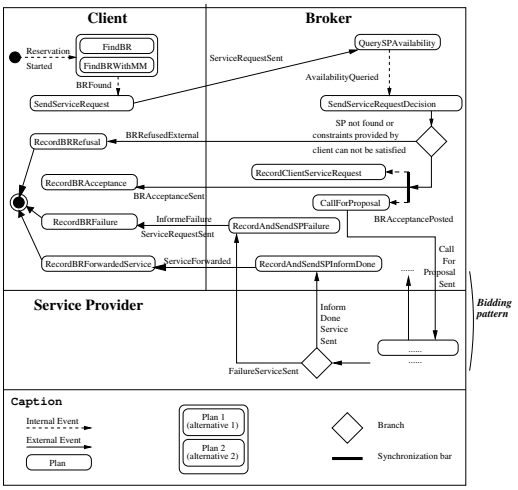

Figure 3: Dynamic Diagram - Broker

An event is represented by a arrow, a plan is represented by an oval. If an event is handled by alternative plans, they are enclosed in a round-corner box. Synchronization and branching are represented as usual.

We omit the dynamic dimension of the Subscription and the Bidding patterns, and only present in Figure 3 the activity diagram specific to the Broker pattern. It models the flow of control from the emission of a service request sent by the client to the reception by the same client of the realized service result sent by the broker. Three swimlanes, one for each agent of the Broker pattern, compose the diagram. In this pattern, the `FindBroker` service described in Section 3.2, is either operationalized by the `FindBR` or the `FindBRWithMM` plans (the client finds a broker based on its own knowledge or via a matchmaker).

At a lower level, each plan could also be modeled by an activity diagram for further detail if necessary.

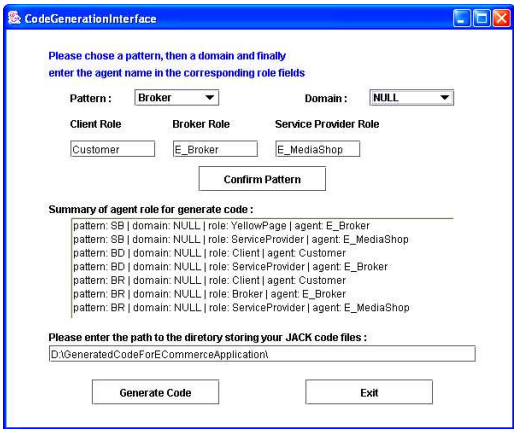

Figure 4: JACK Code Generation

4 CODE GENERATION

The main motivation behind design patterns is the possibility of reusing them during detailed design and implementation. Numerous CASE tools such as Rational Rose (Rose) and Together (Together) include code generators for object-oriented design patterns. Programmers identify and parameterize, during detailed design, the patterns that they use in their applications. A code skeleton for patterns is then automatically generated and programming is thus made easier.

Such tools are not yet available for agent design patterns. A recent development in the SKWYRL patterns framework is the proposal a code generator for the social patterns introduced in Section 2. Figure 4 shows the main window of the tool. It is developed with Java and produces code for JACK (JACK), an agent-oriented development environment built on top of Java. JACK extends Java with specific capabilities to implement agent behaviors. On a conceptual point of view, the relationship of JACK to Java is analogous to that between C++ and C. On a technical point of view, JACK source code is first compiled into regular Java code before being executed.

In the SKWYRL code generator, the programmer first chooses which social pattern to use, then the roles for each agent in the selected pattern (e.g., the `E_Broker` agent plays the *broker* role for the Broker pattern but can also play the *client* role for the Bidding pattern and the *yellow-page* role for the Subscription pattern in the same application). The process is repeated until all the patterns in the application are identified. The code generator then produces the generic code for the patterns (the .agent, .event, .plan, .bel JACK files).

Figure 5 shows an e-business broker developed with JACK. The code skeleton was generated with

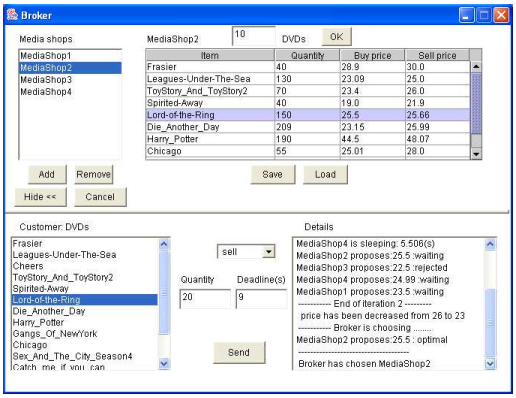

Figure 5: An E-Business Broker

our code generator using the Broker pattern. The customer sends a request to the broker to buy or sell DVDs, specifying titles, quantity, and deadline for answering the request. When receiving the customer request, the broker interacts with the E_MediaShops to obtain the DVDs (bottom-right corner of the figure). At every iteration, the broker can accept an offer, raise the bid, or cancel the process.

5 CONCLUSION

Patterns ease the task of developers describing system architectures. This paper has introduced the SKWYRL social patterns framework designed for agent methodologies like TROPOS to formalize a *code of ethics* for social patterns (MAS design patterns inspired by social and intentional characteristics), answering questions like: "what can one expect from a broker, mediator, or embassy?". The framework is used to:

- define social patterns and answer the above question according to the five modeling dimensions of TROPOS: social, intentional, structural, communicational, and dynamic.
- drive the design of the details of a MAS architecture in terms of those social patterns during the detailed-design phase.

The paper has overviewed some social design patterns on which we are working. The five dimensions of the framework have been illustrated through the Broker pattern.

Future research directions include the precise formalization of a catalog of social patterns, including the characterization of the sense in which a particular MAS architecture is an instance of a configuration of patterns. We will also compare and contrast social patterns with classical design patterns proposed in the literature.

REFERENCES

Y. Aridor and D. B. Lange. "Agent Design Patterns: Elements of Agent Application Design", in *Proc. of the 2nd Int. Conf. on Autonomous Agents* (Agents'98), St Paul, Minneapolis, USA, 1998.

B. Bauer, J. P. Muller and J. Odell "Agent UML: A Formalism for Specifying Multiagent Interaction", in *Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering* (AOSE'00), Limerick, Ireland, 2001. he Netherlands, 2002.

D. Deugo, F. Oppacher, J. Kuester and I. V. Otte. "Patterns as a Means for Intelligent Software Engineering", in *Proc. of the Int. Conf. on Artificial Intelligence* (IC-AI'99), Vol. II, CSRA, 1999.

T. Tung Do, M. Kolp, T. T. Hang Hoang and A. Pirotte. "A Framework for Design Patterns for Tropos", in *Proc. of the 17th Brazilian Symposium on Software Engineering* (SBES 2003), Maunas, Brazil, 2003.

Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org

A. Fuxman, M. Pistore, J. Mylopoulos and P. Traverso. "Model Checking Early Requirements Specifications in Tropos", in *Proc. of the 5th IEEE Int. Symposium on Requirements Engineering* (RE'01), Toronto, Canada, 2001.

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

S. Hayden, C. Carrick and Q. Yang. "Architectural Design Patterns for Multiagent Coordination", in *Proc. of the 3rd Int. Conf. on Agent Systems* (Agents'99), Seattle, USA, 1999.

JACK Intelligent Agents. http://www.agent-software.com/.

M. Kolp, P. Giorgini and J. Mylopoulos. "A Goal-Based Organizational Perspective on Multi-Agents Architectures", in *Proc. of the 8th Int. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages* (ATAL'01), Seattle, USA, 2001.

Rational Rose. http://www.rational.com/rose/.

Together. http://www.togethersoft.com/.

E. Yu. *Modeling Strategic Relationships for Process Reengineering*, PhD thesis, University of Toronto, Department of Computer Science, Canada, 1995.