From Social Designs to Multi-Agent Architectures

Manuel Kolp, T. Tung Do, Stéphane Faulkner and T. T. Hang Hoang

ISYS - Information Systems Research Unit
Catholic University of Louvain
Place des Doyens, 1, 1348, Louvain-La-Neuve, Belgium
e-mail: {kolp,do,faulkner,hoang}@isys.ucl.ac.be

Abstract

A Multi-Agent System (MAS) is an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real world social organizations as an analogy, this paper proposes architectural styles and design patterns for MAS which adopt concepts from social theories. The styles are intended to represent a macro-level architecture of a MAS in terms of actor, goal and actor dependency and are evaluated with respect to software quality attributes. At a micro-level, social patterns give a finer-grain description of the MAS architecture and define how goals assigned to agents will be fulfilled. They are modeled within a conceptual framework analyzing them from five points of view: social, intentional, structural, communicational and dynamic. An e-business example illustrates our purpose.

Keywords: Multi-Agent Systems, Organizational Styles, Social Patterns, BDI Model, Software Architectural and Detailed Design, E-business Architectures, Tropos.

1 Introduction

The characteristics and expectations of new application domains for the enterprise such as e-business, knowledge management, peer-to-peer computing or web services are deeply modifying software architecture engineering. Most of the system architecture designed for these kinds of application areas are now de facto concurrent and distributed. They tend to be open and dynamic, in that they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

For these reasons - and more - Multi-Agent Systems (MAS) architectures are gaining popularity over traditional systems, including object-oriented ones. MAS architectures do allow dynamic and evolving structures and components which can change at run-time to benefit from the capabilities of new system entities or replace obsolete ones.

Such architectures become rapidly complicated due to the ever-increasing complexity of these new business IT domains and their actors: as the expectations of users and business stakeholders change day after day, as the complexity of systems, information and communication technologies and organizations continually increases in today's dynamic environments, developers are expected to produce architectures that must handle more difficult and intricate requirements that were not taken into account ten years ago, making architectural design a central engineering issue in modern enterprise information system life-cycle.

An important technique that helps to manage this complexity when constructing and documenting such architectures is the reuse of design experience and knowledge. Thus, architectural styles and design patterns have become an attractive approach to reusing design knowledge.

Architectural styles are intellectually manageable abstractions of system structure that describe how system components interact and work together [21].

Design patterns describe a problem commonly found in software designs and prescribe a flexible solution for the problem, so as to ease the reuse of that solution. This solution is repeatedly applied from one design to another, producing design structures that look quite similar across different applications [11].

MAS architectures can be considered social structures composed of autonomous and proactive agents that interact and cooperate with each other to achieve common or private goals. Since the fundamental concepts of multi-agent systems are intentional and social, rather than implementation-oriented, theories which study social structures could provide inspiration and insights to define a catalogue of styles and patterns for designing MAS architectures.

In this paper, we will present socially based catalogues of styles and patterns to construct such architectures and apply them on an e-business case study.

Taking real-world social structures as metaphors, we will propose a set of generic architectural structures [10, 14] for the Tropos methodology [4], whose aim is to construct and validate a software development methodology for agent-based software systems :

- At the architectural design level, organizational styles inspired from organization theory and strategic alliances will be used to design the overall MAS architecture. Styles from organization theory will describe the internal structure and design of the MAS architecture, while styles from strategic alliances will model the cooperation of independent architectural organizational entities that pursue shared goals.

- At the detailed design level, social patterns drawn from research on cooperative and distributed architectures, will offer a more macroscopic view of the social MAS architecture description. They will define the agents and the social dependencies that are necessary for the achievement of agent goals.

The Tropos methodology adopts ideas from MAS technologies and concepts from requirements engineering, where agents and goals have been used heavily

for organizational modeling. The key premise of the project is that agents and goals can be used as fundamental concepts for analysis and design during *all the phases of the software development life cycle*, and not just for requirements analysis.

The paper will be organized as follows. Section 2 will introduce the macro level catalogue of organization-inspired architectural styles, and proposes a set of software quality attributes for evaluating architectural alternatives. Section 3 will introduce the micro level catalogue of social design patterns for finer-grain design of an organizational architecture. An e-business case study will illustrate the use of styles and patterns proposed in the paper. Finally, Section 4 will summarize the contributions of the paper and point to further work.

2 Organizational Architectural Styles

Software architectures describe a software system at a macroscopic level in terms of a manageable number of subsystems, components and modules inter-related through data and control dependencies [2].

System architectural design has been the focus of considerable research during the last fifteen years that has produced well-established architectural styles and frameworks for evaluating their effectiveness with respect to particular software qualities. Examples of styles are pipes-and-filters, event-based, layered, control loops and the like [21]. Examples of software qualities include maintainability, modifiability, portability etc [2]. We are interested in developing a suitable set of architectural styles for multi-agent software systems. Since the fundamental concepts of a Multi-Agent System (MAS) are intentional and social, rather than implementation-oriented, we turn to theories which study social structures for motivation and insights. But, what kind of social theory should we turn to? There are theories that study group psychology, communities (virtual or otherwise) and social networks. Such theories study social structure as an emergent property of a social context. Instead, we are interested in social structures that result from a design process. For this, we turn for guidance to organizational theories, namely *Organization Theory* and *Strategic Alliances*. Organization Theory (e.g., [17, 19, 23]) describes the structure and design of an organization; *Strategic Alliances* (e.g., [8, 18, 20]) models the strategic collaborations of independent organizational stakeholders who pursue a set of agreed upon business goals.

2.1 Organization Theory

"An organization is a consciously coordinated social entity, with a relatively identifiable boundary, that functions on a relatively continuous basis to achieve a common goal or a set of goals" [18]. Organization theory is the discipline that studies both structure and design in such social entities. Structure deals with the descriptive aspects while design refers to the prescriptive aspects of a social entity. Organization theory describes how practical organizations are actually

structured, offers suggestions on how new ones can be constructed, and how old ones can change to improve effectiveness. To this end, schools of organization theory have proposed styles to try to find and formalize recurring organizational structures and behaviors.

In the following, we briefly present organizational styles identified in Organization Theory. The structure-in-5 will be studied in detail in Section 2.3.

The Structure-in-5. An organization can be considered an aggregate of five sub-structures, as proposed by Minztberg [17]. At the base level sits the *Operational Core* which carries out the basic tasks and procedures directly linked to the production of products and services (acquisition of inputs, transformation of inputs into outputs, distribution of outputs). At the top lies the *Strategic Apex* which makes executive decisions ensuring that the organization fulfils its mission in an effective way and defines the overall strategy of the organization in its environment. The *Middle Line* establishes a hierarchy of authority between the Strategic Apex and the Operational Core. It consists of managers responsible for supervising and coordinating the activities of the Operational Core. The *Technostructure* and the *Support* are separated from the main line of authority and influence the operating core only indirectly. The Technostructure serves the organization by making the work of others more effective, typically by standardizing work processes, outputs, and skills. It is also in charge of applying analytical procedures to adapt the organization to its operational environment. The Support provides specialized services, at various levels of the hierarchy, outside the basic operating work flow (e.g., legal counsel, R&D, payroll, cafeteria).

The pyramid style is the well-know hierarchical authority structure. Actors at lower levels depend on those at higher levels. The crucial mechanism is the direct supervision from the Apex. Managers and supervisors at intermediate levels only route strategic decisions and authority from the Apex to the operating (low) level. They can coordinate behaviors or take decisions by their own, but only at a local level.

The chain of values merges, backward or forward, several actors engaged in achieving or realizing related goals or tasks at different stages of a supply or production process. Participants who act as intermediaries, add value at each step of the chain. For instance, for the domain of goods distribution, providers are expected to supply quality products, wholesalers are responsible for ensuring their massive exposure, while retailers take care of the direct delivery to the consumers.

The matrix proposes a multiple command structure: vertical and horizontal channels of information and authority operate simultaneously. The principle of unity of command is set aside, and competing bases of authority are allowed to jointly govern the work flow. The vertical lines are typically those of functional departments that operate as "home bases" for all participants, the horizontal lines represents project groups or geographical arenas where managers com-

bine and coordinate the services of the functional specialists around particular projects or areas.

The bidding style involves competitivity mechanisms, and actors behave as if they were taking part in an auction. An auctioneer actor runs the show, advertises the auction issued by the auction issuer, receives bids from bidder actors and ensures communication and feedback with the auction issuer who is responsible for issuing the bidding.

2.2 Strategic Alliances

A strategic alliance links specific facets of two or more organizations. At its core, this structure is a trading partnership that enhances the effectiveness of the competitive strategies of the participant organizations by providing for the mutually beneficial trade of technologies, skills, or products based upon them. An alliance can take a variety of forms, ranging from arm's-length contracts to joint-ventures, from multinational corporations to university spin-offs, from franchises to equity arrangements. Varied interpretations of the term exist, but a strategic alliance can be defined as possessing simultaneously the following three necessary and sufficient characteristics:

- The two or more organizations that unite to pursue a set of agreed upon goals remain independent subsequent to the formation of the alliance.
- The partner organizations share the benefits of the alliances and control over the performance of assigned tasks.
- The partner organizations contribute on a continuing basis in one or more key strategic areas, e.g., technology, products, and so forth.

In the following, we briefly present organizational styles identified in Strategic Alliances. The joint-venture will be studied in details in Section 2.3.

The joint-venture style involves agreement between two or more intra-industry partners to obtain the benefits of larger scale, partial investment and lower maintenance costs. A specific joint management actor coordinates tasks and manages the sharing of resources between partner actors. Each partner can manage and control itself on a local dimension and interact directly with other partners to exchange resources, such as data and knowledge. However, the strategic operation and coordination of such an organization, and its actors on a global dimension, are only ensured by the joint management actor in which the original actors possess equity participations.

The arm's-length style implies agreements between independent and competitive, but partner actors. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise

common goals. No authority is lost, or delegated from one collaborator to another.

The hierarchical contracting style identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority. Coordination mechanisms developed for arm's-length (independent) characteristics involve a variety of negotiators, mediators and observers at different levels handling conditional clauses to monitor and manage possible contingencies, negotiate and resolve conflicts and finally deliberate and take decisions. Hierarchical relationships, from the executive apex to the arm's-length contractors restrict autonomy and underlie a cooperative venture between the parties.

The co-optation style involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of an initiating organization. By co-opting representatives of external systems, organizations are, in effect, trading confidentiality and authority for resource, knowledge assets and support. The initiating system has to come to terms with the contractors for what is being done on its behalf; and each co-optated actor has to reconcile and adjust its own views with the policy of the system it has to communicate.

2.3 Modeling Organizational Styles

We will define an organizational style as a metaclass of organizational structures offering a set of design parameters to coordinate the assignment of organizational objectives and processes, thereby affecting how the organization itself functions [15]. Design parameters include, among others, goal and task assignments, standardization, supervision and control dependencies and strategy definitions.

This section describes two of the organizational styles presented in Section 2: the structure-in-5 and the joint-venture. For further details see [6, 16]

2.3.1 Structure-in-5

Figure 1 models the structure-in-5 style using the *i** strategic dependency model. *i** is a modeling framework for early requirements analysis [24], which offers goal- and actor-based notions such as *actor*, *agent, role, position, goal, softgoal*, *task*, *resource, belief* and different kinds of social *dependency* between actors. It is a graph, where each node represents an *actor* and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, and the *dependee,* the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely (for instance, the appreciation is subjective or fulfillment is obtained only to a given extent); *task*

dependencies are used in situations where the dependee is required to perform a given activity; and *resource* dependencies require the dependee to provide a resource to the depender.

Actors are represented as circles; dependums – goals, softgoals, tasks and resources – are respectively represented as ovals, clouds, hexagons and rectangles; dependencies have the form *depender* → *dependum* → *dependee*.

For instance in Figure 1, the *Technostructure, Middle Agency* and *Support* actors depend on the *Apex* for strategic management. Since the goal *Strategic Management* does not have a precise description, it is represented as a softgoal (cloudy shape). The *Middle Agency* depends on the *Technostructure* and *Support* respectively through goal dependencies *Control* and *Logistics* represented as oval-shaped icons. The *Operational Core* is related to the *Technostructure* and *Support* actors through the *Standardize* task dependency and the *Non-operational Service* resource dependency, respectively.

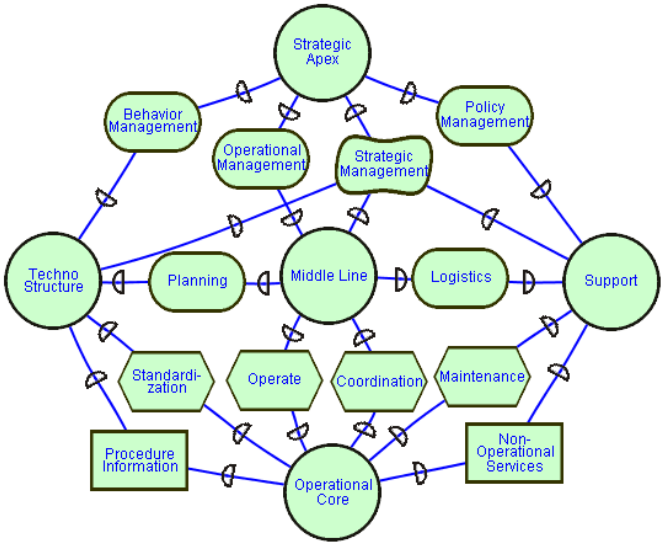

Figure 1: The Structure-in-5 Style

A number of constraints also apply [16]:

- the dependencies between the *Strategic Apex* as depender and the *Technostructure*, *Middle Line* and *Support* as dependees must be of type goal
- a softgoal dependency models the strategic dependence of the *Technostructure*, *Middle Line* and *Support* on the *Strategic Apex*
- the relationships between the *Middle Line* and *Technostructure* and *Support* must be of goal dependencies

- the *Operational Core* relies on the *Technostructure* and *Support* through task and resource dependencies
- only task dependencies are permitted between the *Middle Line* (as depender or dependee) and the *Operational Core* (as dependee or depender).

2.3.2 Joint-venture

Figure 2 models the joint-venture style using i*.

A number of constraints also apply [16]:

- Partners depend on each other for providing and receiving resources.
- Operation coordination is ensured by the joint manager actor which depends on partners for the accomplishment of these assigned tasks.
- The joint manager actor must assume two roles: a private interface role to coordinate partners of the alliance and a public interface role to take strategic decisions, define policy for the private interface and represents the interests of the whole partnership with respect to external stakeholders.

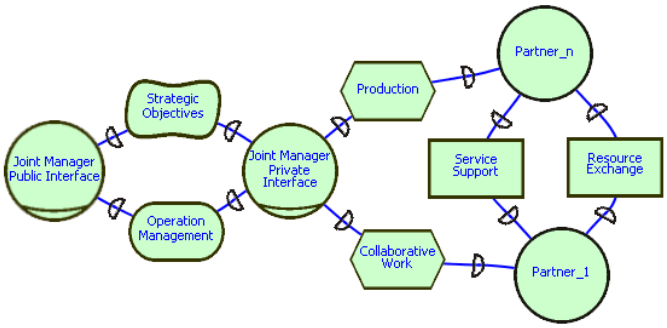

Figure 2: The Joint-venture Style

2.4 Quality Attributes

Software quality attributes (i.e., non functional requirements describing how well the system accomplishes its functions) relevant for MAS have been studied in [13]. Generally the following software qualities are addressed to characterize multi-agent system architectures:

Predictability [22]. Autonomous components like agents have a high degree of autonomy in the way that they undertake action and communication in their domains. It can be then difficult to predict individual characteristics as part of

determining the behavior of a distributed and open system at large.

Security. Agents are often able to identify their own data and knowledge sources and they may undertake additional actions based on these sources [22]. Protocols and strategies for verifying authenticity for these data sources by individual agents are an important concern in the evaluation of overall system quality since, in addition to possibly misleading information acquired by agents, there is the danger of hostile external entities spoofing the system to acquire information accorded to trusted domain agents.

Adaptability. Agents may be required to adapt to modifications in their environment. They may include changes to the component's communication protocol or possibly the dynamic introduction of a new kind of component previously unknown or the manipulations of existing agents.

Coordinability. Agents are not particularly useful unless they are able to coordinate with other agents.

Coordination can be realized in two ways:

- **Cooperativity**. Agents must be able to coordinate with other entities to achieve a common purpose or simply their local goals.
- **Competitivity**. Deliberative negotiating systems are like deliberative systems, except that they have an added dose of competition. The success of one agent implies the failure of others.

Availability. Agents that offer services to other agents/humans must implicitly or explicitly guard against the interruption of offered services. Availability must actually be considered a sub-attribute of security. Nevertheless, we deal with it as a top-level software quality attribute due to its increasing importance in multi-agent system design.

Fallibility-Tolerance. A failure of one agent does not necessarily imply a failure of the whole system. The system then needs to check the completeness and the accuracy of data, information and knowledge transactions and flows. To prevent system failure, different agents can have similar or replicated capabilities and refer to more than one component for a specific behavior.

Modularity increases efficiency of task execution, reduces communication overhead and usually enables high flexibility. On the other hand, it implies constraints on inter-module communication.

Aggregability. Some agent components are parts of other agent components. They surrender to the control of the composite entity. This control results in efficient tasks execution and low communication overhead, however prevents the system to benefit from flexibility.

2.5 Applying Organizational Styles

This section overviews the use of the structure-in-5 and joint-venture styles with the design of an architecture for a business-to-consumer (B2C) application called *E-Media* [6].

E-Media is a business-to-consumer system allowing on-line customers to buy different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes and the like on the Internet. Customers can search the on-line store by either browsing the catalogue or using a search engine to query the database. *E-Media* also allows to process on-line orders, bills and delivery invoices, and keeps track of all web information of strategic importance for statistical analysis. The full case study is described in [4]

Structure-in-5. Figure 3 suggests a possible assignment of system responsibilities for *E-Media* following the structure-in-5 style. It is decomposed into five principal components *Store Front*, *Coordinator*, *Billing Processor*, *Back Store* and *Decision Maker*. *Store Front* serves as the *Operational Core.* It interacts primarily with Customer and provides her with a usable front-end web application for consulting and shopping media items. *Back Store* constitutes the *Support* component. It manages the product database and communicates to the *Store Front* information on products selected by the user. It *stores* and *backs up* all web information from the *Store Front* about customers, products, sales, orders and bills to produce *statistical information* to the *Coordinator*. It provides the *Decision Maker* with *strategic information* (analyses, historical charts and sales reports).

The *Billing Processor* is in charge of handling orders and bills for the *Coordinator* and implementing the corresponding procedures for the *Store Front*. It also ensures the secure management of financial transactions for the *Decision Maker*. As the *Middle Line*, the *Coordinator* assumes the central position of the architecture. It ensures the coordination of *e-shopping* services provided by the *Operational Core* including the management of conflicts between itself, the *Billing Processor*, the *Back Store* and the *Store Front*. To this end, it also handles and implements strategies to manage and prevent *security* gaps and *adaptability* issues. The *Decision Maker* assumes the *Strategic Apex* role. To this end, it defines the *Strategic Behavior* of the architecture ensuring that objectives and responsibilities delegated to the *Billing Processor*, *Coordinator* and *Back Store* are consistent with that global functionality.

Joint-venture. Following the joint-venture style, the *E-Media* architecture in Figure 4 is organized around a joint manager assuming two roles: the *E-store* role defines the *Customer Relationship Management* and the operational strategies of *E-Media*, i.e., *Sales* and *DB Management Strategy*. The *Back Store* deals with coordinativity supervising the other actors: a *Data Mining Processor* handling *Business Knowledge* processes, a *DataBase* storing the on-line catalogue and allowing *Catalogue Browsing*, a *Billing Processor* managing all *Financial Transactions* and a *Shopping Cart* implementing *E-Shopping* activities.

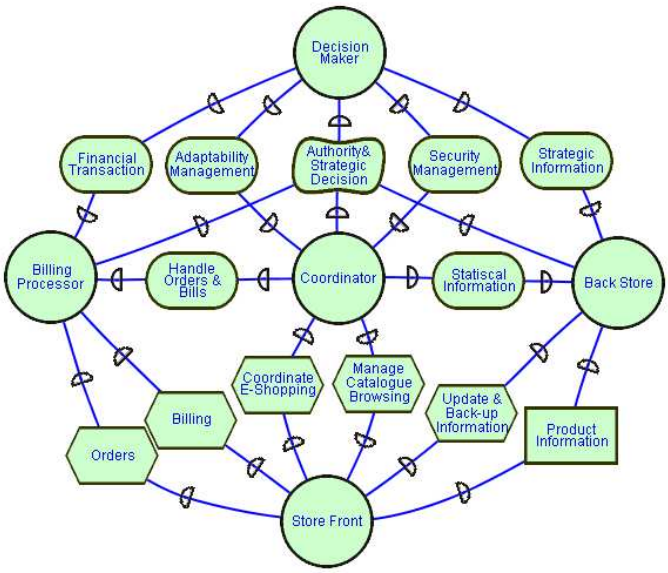

Figure 3: The E-media Architecture in Structure-in-5

Each of these four last actors also interacts directly with each other to exchange data, information and knowledge: the *Shopping Cart* needs data and information about selected products from the *DataBase* and provides *Billing Processor* with financial information about purchased products and on-line customers. The *Data Mining Processor* gets sales data and information from the *Data Base* to produce business knowledge, i.e., historical charts sales reports and business forecasts.

2.6 Evaluation

Three of the software quality attributes presented in Section 2.4 have been identified particularly strategic for e-business systems [13, 6]:

Adaptability deals with the way the system can be designed using generic mechanisms to allow web pages to be dynamically changed. It also concerns the catalogue update for inventory consistency.

The *structure-in-5* separates independently each typical component of the *E-Media* architecture isolating them from each other and allowing dynamic manipulation. In the *joint-venture*, manipulation of partner components can be done easily by registering new components to the joint manager. However, since partners can also communicate directly with each other, existing dependencies should be updated as well.

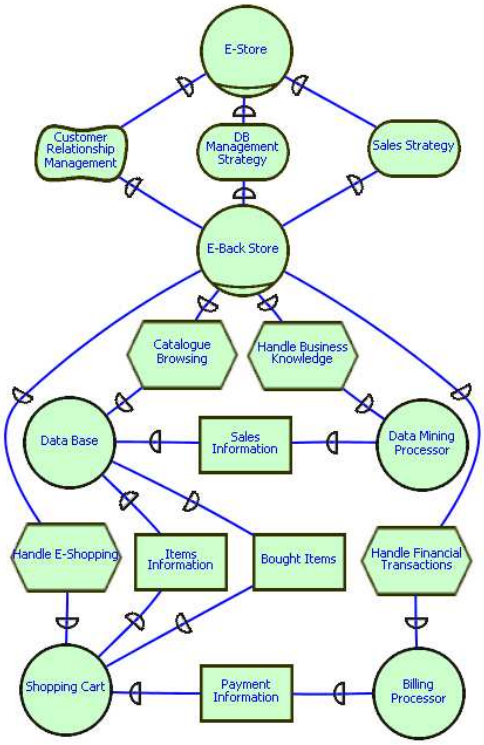

Figure 4: The E-media Architecture in Joint-venture

Security. Clients, exposed to the internet are, like servers, at risk in web applications. It is possible for web browsers and application servers to download or upload content and programs that could open up the client system to crackers and automated agents. JavaScript, Java applets, ActiveX controls, and plug-ins represent a certain risk to the system and the information it manages. Equally important are the procedures checking the consistency of data transactions.

In the *structure-in-5*, checks and control mechanisms can be integrated at different assuming redundancy from different perspectives. Contrary to the classical layered architecture [21], checks and controls are not restricted to adjacent levels. Besides, since the structure-in-5 permits to separate process (*Store Front*, *Billing Processor* and *Back Store*) from control (*Decision Maker* and *Monitor*), security and consistency of these two hierarchies can also be verified independently. The *joint-venture*, through its joint manager, proposes a central message server/controller. Exception mechanism, wiretapping supervising or monitoring can be supported by the joint manager to guarantee non-failability, reliability and completeness.

Availability. Network communication may not be very reliable causing sporadic loss of the server. There are data integrity concerns with the capability of the e-business system to do what needs to be done, as quickly and efficiently as possible in particular with the ability of the system to respond in time to client requests for its services.

The *structure-in-5* architecture prevents availability problems by differentiating process from control. Besides, contrary to the classical layered architecture [21], higher levels are more abstract than lower levels: lower levels only involve resources and task dependencies while higher ones propose intentional (goals and softgoals) relationships. In the *joint-venture*, the central position and role of the joint manager is a means for resolving conflicts between components and prevent availability issues. Through its joint manager, the architecture proposes a central message server/controller. Exception mechanisms, wiretapping supervising or monitoring can be centrally supported by to guarantee non-failability, reliability and completeness.

Table 1 summarizes the strengths and weaknesses of the two architectures with respect to the software quality attributes detailed below.

	Structure-in-5	**Joint-venture**
Security	+	+
Availability	+	+
Adaptability	+-	++

Table 1: Strengths and Weaknesses of E-business Organizational Architectures

3 Social Patterns

A further step in the architectural design of MAS consists of specifying how the goals delegated to each actor are to be fulfilled [10, 13]. For this step, 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., [11]). Unfortunately, little emphasis has been put on social and intentional aspects. Moreover, the proposals of agent patterns that address these aspects (see e.g., [1, 5]) are not intended to be used at a design level, but rather during implementation when low-level issues like agent communication, information gathering, or connection setup are addressed.

This section details the notion of social patterns according to five complementary dimensions [7]: social, intentional, structural, communicational, and dynamic. As an illustration, it defines and studies a social pattern called *Booking*. Social patterns are design patterns focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems. In particular, the structures are inspired by the federated patterns introduced in [12, 13]. We have classified them in two categories. The *Pair* patterns describe direct interactions between negotiating agents. The *Mediation* patterns feature intermediate

agents that help other agents to obtain some agreement about an exchange of services.

Some of the patterns are depicted by figures that reflect their projections on a particular aspect (called modeling dimension). The meaning of these modeling dimensions is detailed later.

3.1 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 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 offer, raise the bid, or cancel the process.

3.2 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 SKwyRL.

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 into a multi-agent system. A wrapper agent interfaces system agents with the legacy system by acting as a translator. This ensures that communication protocols are respected and the legacy system remains decoupled from the rest of the agent system.

3.3 Modeling Social Patterns

This section describes a conceptual framework [7], based on five complementary modeling dimensions, to introspect social patterns. Each dimension reflects a particular aspect of a MAS architecture, as follows.

- The *social dimension* identifies the relevant agents in the system and their intentional interdependencies.
- The *intentional dimension* identifies and formalizes the 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 last three dimensions (structural, communicational, and dynamic) of the architecture are also relevant for traditional (non-agent) systems, but we have adapted and extended them with agent-oriented concepts.

The rest of the section details the dimensions. Each of them will be illustrated through the Booking pattern.

3.3.1 Social Dimension

The social dimension specifies a number of agents and their intentional interdependencies using the i* model [24]. Figure 5 shows a social-dimension diagram for the Broker pattern.

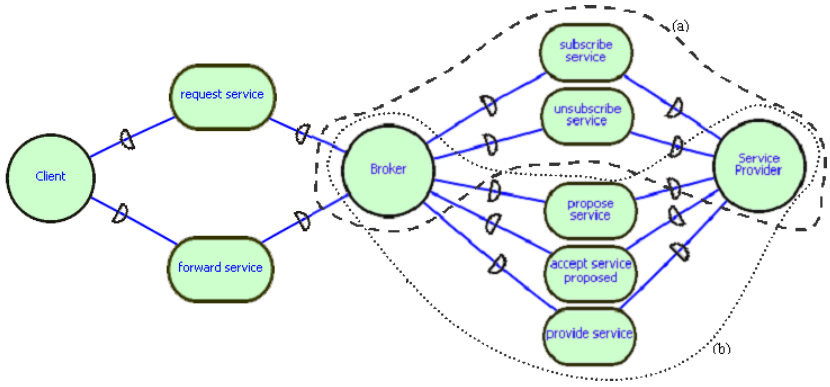

Figure 5: Social diagram for the Broker pattern

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 - Booking, Call-for-Proposals, or Bidding - whereby the Broker agent requests and receives services from service providers (in Figure 5, it is a Call-for-Proposals pattern, shown enclosed within dotted boundary (b)), and (3) interaction between broker and the 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.

3.3.2 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 lists several services of the Broker pattern with an informal definition. With the `FindBroker` service, a client finds a broker that can handle a given service request. The request is then sent to the broker through the

Service Name	Informal Definition	Agent
FindBroker	Find a broker that can provide a service	Client
SendServiceRequest	Send a service request to a broker	Client
QuerySPAvailability	Query the knowledge for information about the availability of the requested service	Broker
SendService RequestDecision	Send an answer to the client	Broker
RecordBRRefusal	Record a negative answer from a broker	Client
RecordBRAcceptance	Record a positive answer from a broker	Client
RecordClient ServiceRequest	Record a service request received from a client	Broker
CallForProposals	Send a call for proposals to service providers	Broker
RecordAndSend SPInformDone	Record a service received from a service provider	Broker

Table 2: Some services of the Broker pattern

`SendServiceRequest` 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 Call-For-Proposals 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).

3.3.3 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 is, 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 is modeled using a UML style class diagram extended for MAS engineering.

The required agent concepts extending the class diagram model are defined below. The structural dimension of the Broker pattern illustrates them.

Structural concepts Figure 6 depicts concepts and their relationships to build the structural dimension. Each concept defines a common template for classes of concrete MAS (for example, `Agent` in Figure 6 is a template for the agent class `Broker` of Figure 7).

A **Belief** describes a piece of the knowledge that an agent has about itself and its environment. Belief are represented as tuples composed of a key and value fields.

Events describe stimuli, emitted by agents or automatically generated, in response to which the agents must take action. As shown in Figure 6, the structure of an event is composed of three parts: declaration of the attributes of the event, declaration of the methods to create the event, declaration of the beliefs and the condition used for an automatic event. The third part only appears for automatic events. Events can be described along three dimensions:

- *External / Internal* event: event that the agent sends to other agents / event that the agent posts to itself. This property is captured by the *scope* attribute.
- *Normal / BDI* event: An agent has some alternative plans in response to a BDI event and only one plan in response to a normal event. Whenever an event occurs, the agent initiates a plan to handle it. If the plan execution fails and if the event is a normal event then the event is said to have failed. If the event is a BDI event, a set of plans can be selected for execution and these are attempted in turn. If all selected plans fail, the event is also said to have failed. The event type is captured by the *type* attribute.
- *Automatic / Nonautomatic* event: an automatic event is automatically created when certain belief states arise. The *create when* statement specifies the logical condition which must arise for the event to be automatically created. The states of the beliefs that are defined by *use belief* are monitored to determine when to automatically create events.

A **Plan** describes a sequence of actions that an agent can take when an event occurs. As shown by Figure 6, plans are structured in three parts: the Event part, the Belief part, and the Method part. The Event part declares events that the plan `handles` (i.e., events that trigger the execution of the plan) and events that the plan produces. The latter can be either `posted` (i.e., sent by an agent only to itself) or `sent` (i.e., sent to other agents). The Belief part declares beliefs that the plan `reads` and those that it `modifies`. The Method part describes the plan itself, that is, the actions performed when the plan is executed.

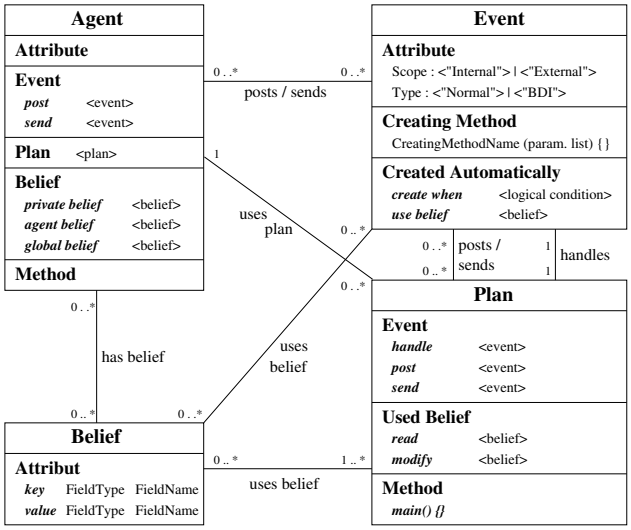

Figure 6: Structural Diagram Template

The **Agent** concept defines the behavior of an agent, as composed of five parts: the declaration of its attributes, of the events that it can post or send explicitly (i.e., without using its plans), of the plans that it uses to respond to events, of the beliefs that make up its knowledge, and of its methods.

The beliefs of an agent can be of type *private*, *agent*, or *global*. A *private* access is restricted to the agent to which the belief belongs. *Agent* access is shared with other agents of the same class, while *global* access is unrestricted.

Broker Pattern Structural Model As an example, Figure 7 depicts the Broker pattern components. For brevity, each construct described earlier is illustrated only through one component. Each component can be considered as an instantiation of the (corresponding) template in Figure 6.

`Broker` is one of the three agents composing the Broker pattern. It has plans such as `QuerySPAvailability`, `SendServiceRequestDecision`, etc. When there is no ambiguity, by convention, the plan name is the same as the as the name of the service that it operationalizes. The private belief `SPProvidedService` is used to store the service type that each service provider can provide. This belief is declared as private since the broker is the only agent that can manipulate it. The `ServiceType` belief stores the information about types of service provided by service providers and is declared as global since its must be known both by the service provider and the broker agent.

The constructor *method* allows to give a name to a broker agent when created. This method may call other methods, for example `loadBR()`, to initialize agent beliefs.

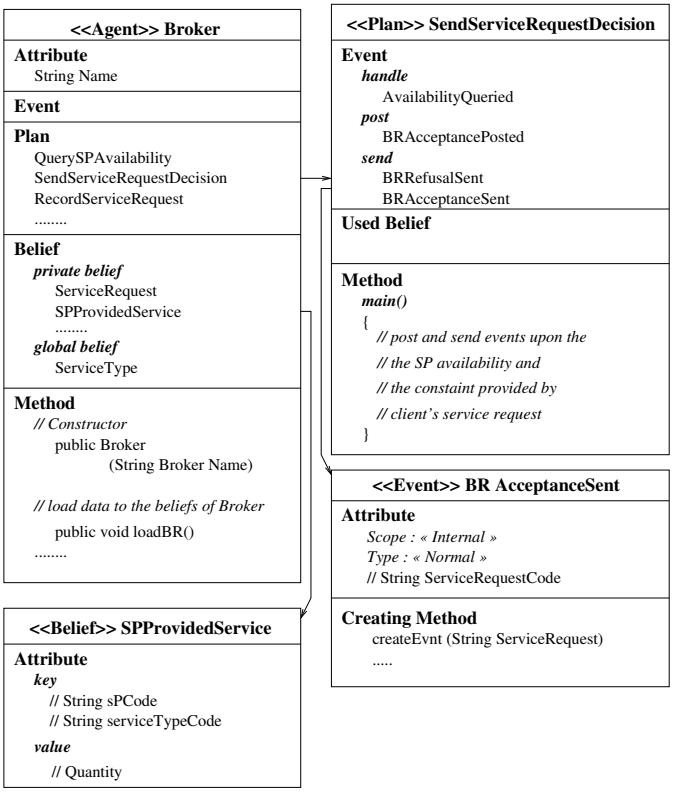

Figure 7: Structural Diagram - Some components of the Broker pattern

`SendServiceRequestDecision` is one of the Broker pattern plans the broker uses to answer the client: the `BRRefusalSent` event is sent when the answer is negative, `BRAcceptanceSent` when the broker has found some service provider(s) that may provide the service requested by the client. In the latter case, the plan also posts the `BRAcceptancePosted` event to invoke the process of recording the client's service request and the process of 'call for proposal' between the broker and the services providers. The SendServiceRequestDecision plan is executed when the `AvailabilityQueried` event (containing the information about the availability of the service provider to realize the client's request) occurs.

`SPProvidedService` is one of the broker's beliefs used to store the services provided by the service providers. The service provider code `sPCode` and the service type code `serviceTypeCode` form the belief key. The corresponding `quantity` attribute is declared as value field.

`BRAcceptanceSent` is an event that is sent to inform the client that its request is accepted.

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 [3] and extend it: *agent_name/Role:pattern_name* expresses the role (*pattern_name*) of the agent (*agent_name*) in the pattern; the arrows are labeled with the name of the exchanged events.

Figure 8 shows a sequence diagram for our Broker pattern. The client (`customer1`) sends a service request (`ServiceRequestSent`) containing the characteristics of the service it wishes to obtain from the broker. The broker may alternatively answer with a denial (`BRRefusalSent`) or a acceptance (`BRAcceptanceSent`).

In the case of an acceptance, the broker sends a call for proposal to the registered service providers (`CallForProposalSent`). The call for proposal (CFP) 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 about this result by sending a `InformFailureServiceRequestSent` or a `ServiceForwarded`, respectively.

The communication dimension of the subscription pattern (SB) is given at the top-right and the communication dimension of the call-for- proposals pattern (CFP) is given at the bottom-right part of the Figure 8. The communication specific for the broker pattern is given at the left part of this figure.

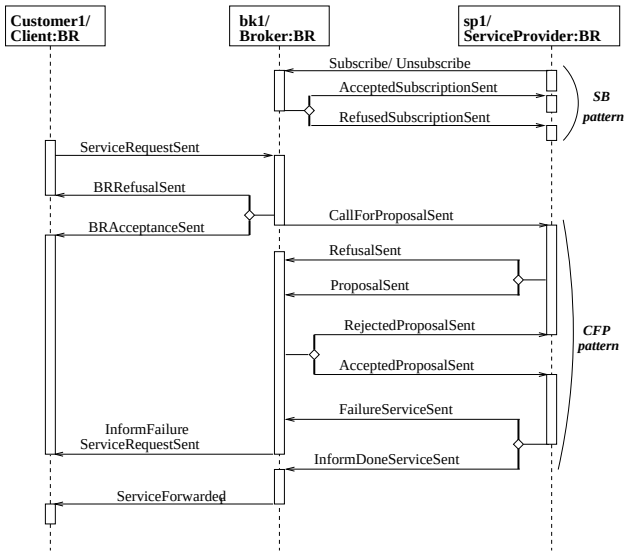

Figure 8: Communication Diagram - Broker

3.5 Dynamic Dimension

As described earlier, 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. 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.

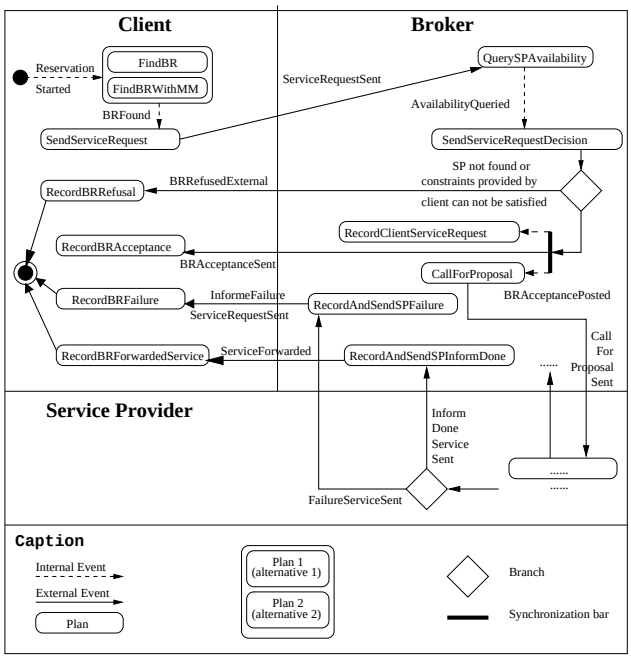

Figure 9: Dynamic Diagram - Broker

An internal event is represented by a dashed arrow and an external event by a solid arrow. As mentioned earlier, a BDI event may be handled by alternative plans. They are enclosed in a round-corner box. A plan is represented by a lozenge shape. Synchronization and branching are represented as usual.

We omit the dynamic dimension of the Subscription and the CFP patterns, and only present in Figure 9 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 the 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.

3.6 Applying the Patterns

Figure 10 shows a possible use of the patterns in the e-business system of Figure 3. In particular, it shows how to realize the dependencies *Manage catalogue browsing*, *Update Information* and *Product Information* from the point of view of the Store Front. The Store Front and the dependencies are decomposed into a combination of social patterns involving agents, pattern agents, subgoals and subtasks.

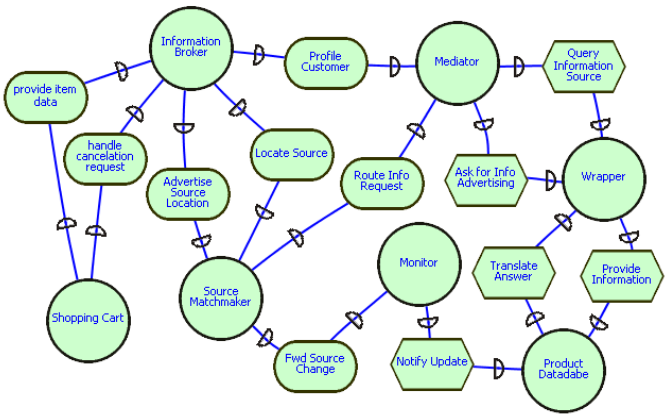

Figure 10: Decomposing the Store Front with Social Patterns

The booking pattern is applied between the *Shopping Cart* and the *Information Broker* to reserve available items. The broker pattern is applied to the *Information Broker*, which satisfies the Shopping Cart 's requests of information by accessing the *Product Database*. The *Source Matchmaker* applies the matchmaker pattern to locate the appropriate source for the *Information Broker*, and the monitor pattern is used to check any possible change in the *Product Database*. Finally, the mediator pattern is applied to dispatch the interactions between the *Information Broker*, the *Source Matchmaker*, and the *Wrapper*, while the wrapper pattern makes the interaction between the *Information Broker* and the *Product Database* possible.

4 Conclusion

Styles and Patterns ease the task developers describing system architectures.

We are working towards the definition of a collection of specific organizational architecture styles and social design patterns for designing multi-agent systems architectures. Since the fundamental concepts of this kind of systems are intentional and organizational, rather than implementation-oriented, they can be viewed as social organizations composed of autonomous and proactive agents that interact to achieve common or private goals.

We propose to use human organizations as a metaphor to suggest a set of generic styles for agent systems, with a preference for organizational design theories over social emergence theories.

To this end, the paper has proposed architectural styles for MAS inspired from organization theory that describes the internal structure and design of organizations and from theories for strategic alliances that model the collaboration of independent organizations that pursue agreed goals.

In particular we have detailed and adapted the structure-in-5, a well understood organizational style used by organization theorists and the joint-venture, used to describe cooperative strategies in the business world.

The contribution also includes the presentation and evaluation of software qualities identified for these styles and a comparison of organizational and conventional styles conducted on an e-business case study.

The organizational styles constitute an architectural macro level. At a micro level we focus on the notion of social design patterns for agents. Many existing patterns can be incorporated into system architecture, such as those identified in [11]. For agent inherent characteristics, patterns for distributed, and open architectures like the broker, matchmaker, embassy, mediator, wrapper, mediator are more appropriate [12, 22]. They detail how goals and dependencies identified in an organizational style can be refined and achieved.

We have introduced a design framework to formalize the *code of ethics* for social patterns – MAS design patterns inspired by social and intentional characteristics –, answering the question: what can one expect from a broker, mediator, embassy, etc.? The framework is used to:

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

We have overviewed some social design patterns. The five dimensions of the framework are illustrated through the definition of a social pattern that we called *Booking*.

Future research directions will extend and formalize precisely [9] the catalogue of organizational styles and social patterns and characterize specifically how a particular model can be seen as an instance of a style or a configuration of

patterns. We will also compare and contrast social patterns with classical design patterns proposed in the literature, and relate them to lower-level architectural components involving (software) components, ports, connectors, interfaces, libraries and configurations.

References

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

[2] Bass, L., Clements, P., and Kazman, R. (1998) *Software Architecture in Practice*, Addison-Wesley.

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

[4] Castro, J., Kolp, M., and Mylopoulos, J. (2002) "Towards requirements-driven information systems engineering: The Tropos Project", in *Information Systems* (27), Elsevier.

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

[6] Do, T. T., Faulkner, S., and Kolp, M. (2003) "Organizational Multi-Agent Architectures for Information Systems", in *Proc. of the 5th Int. Conf. on Enterprise Information Systems* (ICEIS'03), Angers, France.

[7] Do, T. T., Kolp, M., and Pirotte, A. (2003a) "Social Patterns for Designing Multi-Agent Systems", in *Proc. of the 15th Int. Conf. on Software Engineering and Knowledge Engineering* (SEKE'03), San Fransisco, USA.

[8] Dussauge, P. and Garrette, B. (1999) *Cooperative Strategy: Competing Successfully Through Strategic Alliances*, Wiley and Sons.

[9] Faulkner, S. and Kolp, M. (2003) "Towards an Agent Architectural Description Language for Information Systems", in *Proc. of the 5th Int. Conf. on Enterprise Information Systems* (ICEIS'03), Angers, France.

[10] Fuxman, A., Giorgini, P., Kolp, M., and Mylopoulos, J. (2001) "Information Systems as Social Structures", in *Proc. of the 2nd Int. Conf. on Formal Ontologies for Information Systems* (FOIS'01), Ogunquit, USA.

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

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

[13] Kolp, M., Giorgini, P., and Mylopoulos, J. (2001) "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.

[14] Kolp, M., Giorgini, P., and Mylopoulos, J. (2002) "Information Systems Development through Social Structures", in *Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering* (SEKE'02), Ishia, Italy.

[15] Kolp, M., Giorgini, P., and Mylopoulos, J. (2002a) "Organizational Multi-Agent Architectures: A Mobile Robot Example", in *Proc. of the 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems* (AAMAS'02), Bologna, Italy.

[16] Kolp, M., Giorgini, P., and Mylopoulos, J. (2003) "Organizational patterns for early requirements analysis", in *Proc. of the 15th Int. Conf. on Advanced Information Systems* (CAiSE'03), Velden, Austria, 2003.

[17] Mintzberg, H. (1992) *Structure in fives: designing effective organizations*, Prentice-Hall.

[18] Morabito, J., Sack, I., and Bhate, A. (1999) *Organization modeling: innovative architectures for the 21st century*, Prentice Hall.

[19] Scott, W.R. (1998) *Organizations: rational, natural, and open systems*, Prentice Hall.

[20] Segil, L. (1996) *Intelligent business alliances: how to profit using today's most important strategic tool*, Times Business.

[21] Shaw, M. and Garlan, D. (1996) *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall.

[22] 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, Pittsburgh, USA.

[23] Yoshino, M.Y. and Srinivasa Rangan, U. (1995) *Strategic alliances: an entrepreneurial approach to globalization*, Harvard Business School Press.

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