A Goal-Based Organizational Perspective on Multi-Agent Architectures

Manuel Kolp[1] Paolo Giorgini[2] John Mylopoulos[3]

[1]IAG - Information Systems Research Unit - University of Louvain, 1 Place des Doyens, B-1348 Louvain-La-Neuve, Belgium, tel.: 32-10 47 83 95, kolp@isys.ucl.ac.be
[2] Department of Mathematics - University of Trento, 4 via Sommarive, I-38100, Trento, Italy, tel.: 39-0461-88 2052, pgiorgini@science.unitn.it
[3] Department of Computer Science - University of Toronto, 6 King's College Road M5S 3H5, Toronto, Canada, tel.: 1-416-978 5180, jm@cs.toronto.edu

Abstract. A Multi-Agent System (MAS) is an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real world organizations as an analogy, this paper proposes architectural styles for MAS which adopt concepts from organization theory and strategic alliances literature. The styles are intended to represent a macro-level architecture of a MAS, and they are modeled using the *i** framework which offers the notions of actor, goal and actor dependency for modeling multi-agent settings. The styles are also specified as metaconcepts in the Telos modeling language. Moreover, each style is evaluated with respect to a set of software quality attributes, such as predictability and adaptability. The paper also explores the adoption of micro-level patterns proposed elsewhere in order to give a finer-grain description of a MAS architecture. These patterns define how goals assigned to actors participating in an organizational architecture will be fulfilled by agents. An e-business example illustrates both the styles and patterns proposed in this work. The research is being conducted within the context of *Tropos*, a comprehensive software development methodology for agent-oriented software.

1 Introduction

Multi-Agent System (MAS) architectures can be considered as organizations (see e.g., [6, 7, 15]) composed of *autonomous* and *proactive* agents that interact and cooperate with one another in order to achieve common or private goals. In this paper, we propose to use real world organizations as a metaphor in order to offer a set of generic architectural patterns (or, *styles*) for distributed and open systems, such as MAS. These styles have been adopted from the organization theory and strategic alliances literature [12]. The styles are modeled using the strategic dependency model of *i** [22], and they are further specified in the Telos modeling language [14]. We also present multi-agent patterns to design MAS architectures at a finer-grain.

To illustrate the use of these styles and patterns, we use as example a (fictitious) *Media Shop.* This is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. *Media Shop* customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order. *Media Shop* is supplied latest releases and in-catalogue items by *Media Supplier*. To increase market share, *Media Shop* has decided to open up a B2C retail sales front on the internet. With the new setup, a customer can order *Media Shop* items in person, by phone, or through the internet. The system has been named *Medi@* and is available on the world-wide-web using communication facilities provided by *Telecom Co*. It also uses financial services supplied by *Bank Co.*, which specializes on on-line transactions.

This research is being conducted within the context of the Tropos project [2]. Tropos adopts ideas from MAS technologies, mostly to define the detailed design and implementation phase, and ideas from requirements engineering, where agents/actors and goals have been used heavily for early requirements analysis [4, 22]. In particular, Tropos is founded on Eric Yu's *i** modeling framework which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an application during early requirements analysis. The key premise of the project is that actors and goals can be used as fundamental concepts for analysis and design during *all phases of software development*, not just requirements analysis.

Section 2 introduces the macro level catalogue of organization-inspired architectural styles, and proposes a set of software quality attributes for evaluating architectural alternatives. Section 3 introduces the micro level catalogue of goal-based multi-agent patterns for finer-grain design of an organizational architecture. Section 4 presents fragments of an e-business case study to illustrate the use of styles and patterns proposed in the paper. Finally, Section 5 summarizes the contributions of the paper and points to further work.

2 Organizational Styles

Organizational theory [13, 19] and strategic alliances [9, 20, 21] study alternative styles for (business) organizations. These styles are used to model the coordination of business stakeholders -- individuals, physical or social systems -- to achieve common goals. We propose a macro level catalogue adopting (some of) these styles for designing MAS architectures.

A strategic dependency model is a graph, where each node represents an actor (an agent, position, or role) and each link between two actors indicates that one actor depends on another for a goal to be fulfilled, a task to be carried out, or a resource to be made available. We call the depending actor of a dependency the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers (goal, task or resource) is the *dependum*. The model distinguishes among four types of dependencies -- goal-, task-, resource-, and softgoal-dependency -- based on the type of freedom that is allowed in the relationship between depender and dependee. Softgoals are distinguished from goals because they do not have a formal definition, and are amenable to a different (more qualitative) kind of analysis [3].

For instance, in Figure 1, the 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 Coordination and Support respectively through goal dependencies Control and Logistics represented as oval-shaped icons. Likewise, the Operational Core actor is related to the Coordination and Support actors through the Standardize task dependency and the Non-operational Service resource dependency, respectively.

The **structure-in-5** (s-i-5) style (Figure 1) consists of five typical strategic and logistic components found in many organizations. At the base level one finds the Operational Core where the basic tasks and operations -- the input, processing, output and direct support procedures associated with running the system -- are carried out. At the top lies the Apex composed of strategic executive actors. Below it, sit the control/standardization, management components and logistics: Coordination, Middle Agency and Support, respectively. The Coordination component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the system adapt to its environment. Actors joining the Apex to the Operational Core make up the Middle Agency.

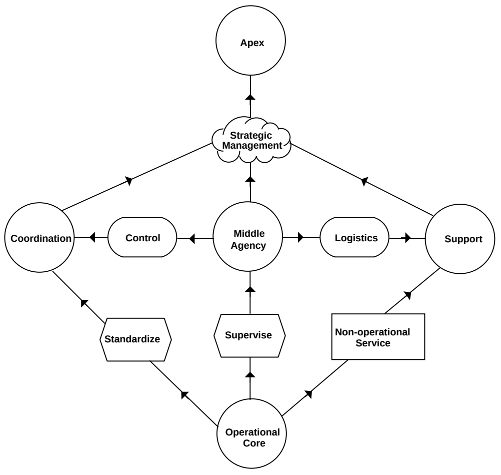

Fig. 1. Structure-in-5

The Support component assists the Operational Core for non-operational services that are outside the basic flow of operational tasks and procedures.

Figure 2 specifies the structure-in-5 style in Telos. Telos is a language intended for modeling requirements, design and implementation for software systems. It provides features to describe metaconcepts used to represent the knowledge relevant to a variety of worlds – subject, usage, system, development worlds - related to a software system. Our organizational styles are formulated as Telos metaconcepts, using heavily aggregation semantics, as proposed in [16].

`MetaClass` is a metametaclass with all metaclasses as instances. `ApexClass`, `CoordinationClass`, `MiddleAgencyClass`, `SupportClass`, and `OperationalCoreClass` are also metaclasses whose instances are actor classes of five different kinds, as indicated by their names. The structure-in-5 style is then a metaclass defined as an aggregation of five (*part*) metaclasses. Each of these five components exclusively belongs (`exclusivePart`) to the aggregate and their existence depends (`dependentPart`) on the existence of the composite. A structure-in-5 architecture specific to an application domain can now be defined as a Telos class, which instantiates `StructureIn5MetaClass`. Similarly, each structure-in-5 component specific to a particular application domain will be defined as a class, instance of one of the five `StructureIn5Metaclass` components.

```
    TELL CLASS StructureIn5Class IN MetaClass WITH
     attribute name: String
         part, exclusivePart, dependentPart
                apex: ApexActorClass
                coordination: CoordinationActorClass
                middleAgency: MiddleAgencyActorClass
                support: SupportActorClass
                operationalCore: OperationalCoreClass
  END StructureIn5MetaClass
```

Fig. 2. Structure-in-5 in Telos

Figure 3 formulates in Telos one of these five structure-in-5 components: the Coordination actor. Dependencies are described following Telos specifications for *i** models. The Coordination actor is a metaclass, `CoordinationMetaclass`. According to Figure 1, the Coordination actor is the dependee of a task dependency `StandardizeTask` and a goal dependency `ControlGoal`, and the depender of a softgoal dependency `StrategicManagementSoftGoal`.

```
TELL CLASS CoordinationMetaclass IN MetaClass WITH
  attribute name: String
   taskDepended
        s:StandardizeTask
        WITH depender
           opCore: OperationalCoreClass
        END
   goalDepended
        c:ControlGoal
        WITH depender
           midAgency: MiddleAgencyClass
        END
   softgoalDepender
        s:StrategicManagementSoftGoal
        WITH dependee
           apex: ApexClass
        END
END CoordinationMetaclass
```

Fig. 3. Structure-in-5 coordination actor.

The **pyramid** (pyr) style is the well-known hierarchical authority structure exercised within organizational boundaries. Actors at the lower levels depend on and report to actors of higher levels. The crucial mechanism here is direct or indirect supervision by the apex. Managers and supervisors are then only intermediate actors routing strategic decisions and authority from the apex to the operating level. They can coordinate behaviors or take decisions on their own, but only at a local level. This style can be applied when designing simple distributed systems. Moreover, this style encourages dynamicity since coordination and decision mechanisms are direct and immediately identifiable. For applications which require a high degree of evolvability and modifiability, this is a good style to use. However, this style is not suitable for complex distributed systems requiring many kinds of agents and supervision relationships. On the other hand, it can be used by such systems to manage and resolve crisis situations. For instance, a complex multi-agent system faced with a non-authorized intrusion from external agents could dynamically reconfigure itself into a pyramid structure in order to resolve the security problem in an effective way.

The **joint venture** (jo-ve) style (Figure 4a) involves agreement between two or more principal partners to obtain the benefits of larger scale operation, with only partial investment and lower maintenance costs. This is accomplished by delegating authority to a specific Joint Management actor who coordinates tasks and manages sharing of knowledge and resources. Each principal partner can manage and control its own operations on a local dimension, but also interact directly with other principal partners to provide and receive services, data and knowledge. However, the strategic operation and coordination of such a system and its partner actors on a global dimension are only ensured by the Joint Management actor. Outside the joint venture, secondary partners supply services or support tasks for the organization core.

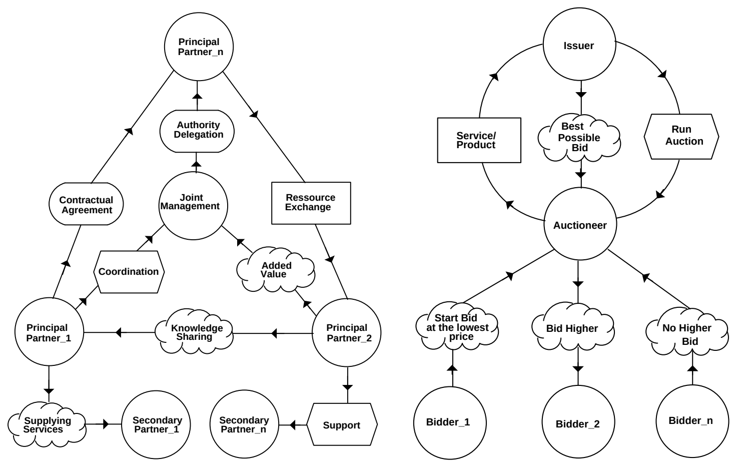

Fig. 4. Joint Venture (a) and bidding (b)

The **bidding** (bidd) style (Figure 4b) is founded on competition mechanisms and actors behave as if they were taking part in an auction. The Auctioneer actor runs the whole show. It advertises the auction issued by the auction Issuer, receives bids from bidder actors and ensure communication and feedback with the auction Issuer. The auction Issuer is responsible for issuing the bidding.

The **arm's-length** (ar-le) style implies agreement between independent and competitive actors who are willing to join a partnership. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise common goals. Authority is not delegated by any partner.

The **hierarchical contracting** (hi-co) style identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority. Coordination here uses mechanisms with arm's-length (i.e., high independence) characteristics involving a variety of negotiators, mediators and observers. These work at different levels and handle conditional clauses, 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 (top to bottom) restrict autonomy and underlie a cooperative venture between the contracting parties. Such, admittedly complex, contracting arrangements can be used to manage conditions of complexity and uncertainty deployed in high-cost-high-gain (high-risk) applications.

	s-i-5	pyr	jo-ve	bidd	ar-le	hi-co	co-op
Predictability	+	++	+	--	-		-
Security	+	++	+	--	--		-
Adaptability		+	++	++	+	+	++
Cooperativity	+	++	+	-	-	+	++
Competitivity	-	-	-	++	++	+	+
Availability	+	+	++	-	--	+	--
Failability-Tolerance		--		--	++		-
Modularity	++	-	+	++	+	+	--
Aggregability	++		++			+	

Table 1. : Correlation catalogue.

The **co-optation** (co-op) style involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of a newly-created organization. By co-opting representatives of external systems, an organization is, in effect, trading confidentiality and authority for resource, knowledge assets and support.

The evaluation of these styles can be done with respect to desirable software quality attributes identified as relevant for distributed and open architectures such as multi-agent ones. For lack of space, we do not detail them here and refer to [12] where a full description of such attributes is presented. Table 1 summarizes correlations for our styles and the quality attributes: +, ++, -, -- respectively model partial/positive, sufficient/positive, partial/negative and sufficient/negative contributions [3].

3 Multi-agent patterns

A further step in the architectural design of MAS consists of specifying how the goals delegated to each actor are to be fulfilled. For this step, designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Design patterns have received considerable attention in Software Engineering [8, 17] and some of these could be adopted for MAS architectures. Unfortunately, they focus on object-oriented rather than agent-oriented systems.

In the area of MAS, some work has been done in designing agent patterns, e.g.,[1, 5, 11]. However, these contributions focus on agent communication, while we are interested in specifying how a goal is to be achieved at an organization level.

In the following we present a micro level catalogue of often-encountered multi-agent patterns in the MAS literature. In particular, some of the federated patterns introduced in [10, 23] will be used in Section 4.

A **broker** is an arbiter and intermediary accessing services from a provider to satisfy the request of a consumer. It is used in vertical integration and joint venture architectures.

A **matchmaker** (Figure 5a) locates a provider for a given consumer service request, and then lets the consumer interact directly with the provider. In this respect, matchmakers are different from brokers who directly handle all interactions between the consumer and the provider. This pattern is also useful for horizontal integration and joint venture architectures.

A **monitor** (Figure 5b) alerts a subscriber when certain events occur. This type of agent accepts subscriptions, requests notifications from subjects of interest, receives such notifications of events and alerts subscribers accordingly. The subject provides notifications of state changes as requested. The subscriber registers for notification of state changes to subjects, and receives notifications of changes. This pattern is used in horizontal contracting, vertical integration, arm's-length and bidding styles all of which require monitoring activities.

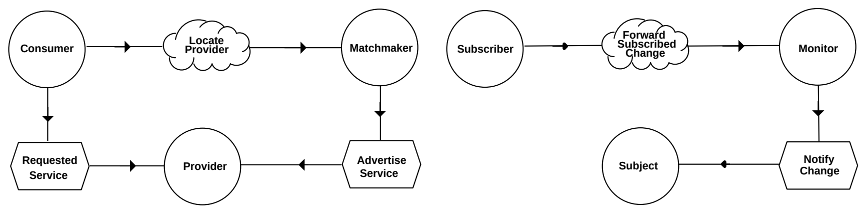

Fig. 5. Matchmacker (a) and monitor (b)

A **mediator** (Figure 6a) mediates interactions among different agents. An initiator addresses the mediator instead of asking directly another agent, the performer. A mediator has acquaintance models of other actors and coordinates cooperation among them. Conversely, other agents have an acquaintance model for the mediator. While a matchmaker simply matches providers with consumers, a mediator encapsulates interactions and maintains models of initiators and performers behaviors over time. It is used in pyramid, vertical integration and horizontal contracting styles because it underlies direct cooperation and encapsulation features reinforcing authority.

An **embassy** (Figure 6b) routes a service requested by an external agent to a local agent and hands back to the foreigner the response. If the request is granted, the external agent can submit messages to the embassy for translation. The content of each such message is translated in accordance with a standard ontology. Translated messages are forwarded to requested local agents. The results of the query are passed back out to the foreign agent, after translation. This pattern is useful for the structure-in-5, arm's-length, bidding and co-optation styles because it copes well with security issues that arise because of the competition mechanisms inherent to these styles.

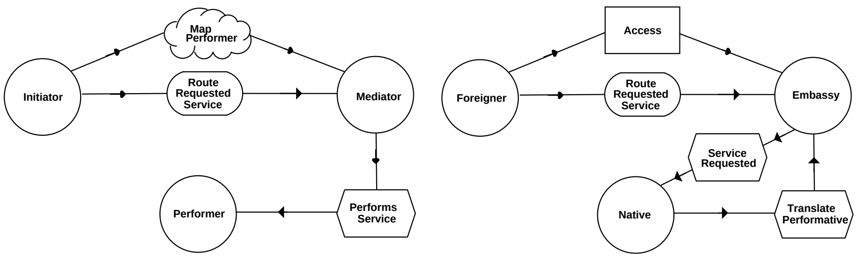

Fig. 6. Mediator (a) and embassy (b)

A **wrapper** incorporates a legacy system into a multi-agent system**.** The wrapper interfaces system agents with the legacy system 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 system. This pattern can be used in the co-optation style when one of the co-optated actor is a representing a legacy system.

The **Contract-Net** pattern selects an agent to which it assigns a task. This pattern includes a manager and any number of participants. The manager issues a request for proposals for a service to all participants and then accepts "proposals" that offer the service for a particular "cost". The manager selects one participant who performs the contracted work and informs the manager upon completion. This pattern is useful in the arm's-length, bidding and co-optation styles due to their competition mechanisms.

MATCHMAKER	
Agent	**Capabilities**
Customer	– Build a request to query the matchmaker – Handle with a services ontology – Query the matchmaker for a service – Find alternative matchmakers – Request a service to a provider – Manage possible provider failures – Monitor the provider's ongoing processes – Ask the provider to stop the requested service
Provider	– Handle with a services ontology – Advertise a service to the matchmaker – Withdraw the advertisement – Use an agenda for managing the requests – Inform the customer of the acceptance of the request service – Inform the customer of a service failure – Inform the customer r of success of a service
Matchmaker	– Update the local database – Handle with a services ontology – Use an agenda for managing the customer requests – Search the name of an agent for a service – Inform the customer of the unavailability of agents for a service

Table 2. : Agents' capabilities for the matchmaker pattern.

A detailed analysis of each pattern allows us to define a set of capabilities for agents playing a role in the pattern. Such capabilities are not exhaustive and concern exclusively agents activities related to each pattern's goal. For lack of space, we only present a set of capabilities for the matchmaker pattern (Table 2).

A capability states that an agent is able to act in order to achieve a given goal. In particular, for each capability the agent has (knows) a set of plans that may apply in different situations. A plan describes the sequence of actions to perform and the conditions under which the plan is applicable. At this stage we do not need to define the plans in detail. Instead, we simply specify that the agent needs to be capable of achieving in one or more ways a given goal. In the Tropos methodology plans are defined in the detail design phase. Sometimes several agents participating in a pattern need to have common capabilities. For instance, the capability handle with services ontology is common to all three agents of the Matchmaker pattern. This suggests a need for a capability pattern repository to be used during the implemention phase.

4 An E-business Example

E-business systems are essential components of "virtual enterprises". By now, software architects have developed catalogues of web architectural styles. Common styles include the *Thin Web Client*, *Thick Web Client* and *Web Delivery*. These architectural styles focus on web concepts, protocols and underlying technologies but not on business processes nor on non-functional requirements for a given application. As a result, the organizational architecture styles are not described well within such a framework. Three software quality attributes are often important for an e-commerce architecture, according to [12]: *Security*, *Availability* and *Adaptability*.

To cope with these software quality attributes and select a suitable architecture for a system under design, we go through a means-ends analysis using the non functional requirements (NFRs) framework. This framework analyses desirable qualities by going through an iterative goal refinement and decomposition as shown in Figure 7. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level qualities. The styles are represented as design decisions (saying, roughly, "make the architecture of the system respectively *pyramid, co-optation, joint venture, arm's-length*-based, …").

The evaluation results in contribution relationships from architectural styles to quality attributes, labeled "+", "++", "-", "--". Design rationale is represented by claims drawn as dashed clouds. These can represent priorities and other meta-information about the decision making process. Exclamation marks (! and !!) are used to mark priorities while a check-mark indicates an achieved quality, while and a cross indicates an un-achievable one.

In Figure 7, *Adaptability* has been AND-decomposed into *Dynamicity* and *Updatability*. For our e-commerce example, *dynamicity* is concerned with the use of generic mechanisms that allow web pages and user interfaces to be dynamically and easily changed. Indeed, information content and layout need to be frequently refreshed to give correct information to customers or simply follow fashion trends for marketing reasons. Using frameworks such as Active Server Pages (ASP) and Server

Side Includes (SSI) to create dynamic pages goes some distance towards addressing this quality. *Updatability* is strategically important for the viability of an application. For our example, *Media Shop* employees have to update regularly the catalogue for inventory consistency. This type of analysis is to be carried out in turn for newly identified sub-qualities as well as for other top-level qualities such as *Security* and *Availability*.

Eventually, the analysis shown in Figure 6 allows us to choose the joint venture architectural style for our e-commerce example (qualities that have been achieved are marked with a check mark). The analysis uses the correlation catalogue shown in Table 1 and the top level qualities *Adaptability*, *Security* and *Availability*. These are respectively marked ++, +, ++ for the selected style. More fine-grain qualities have been identified during the decomposition process, such as *Integrity* (*Accuracy*, *Completeness*), *Usability*, *Response Time*, *Maintainability*, and more.

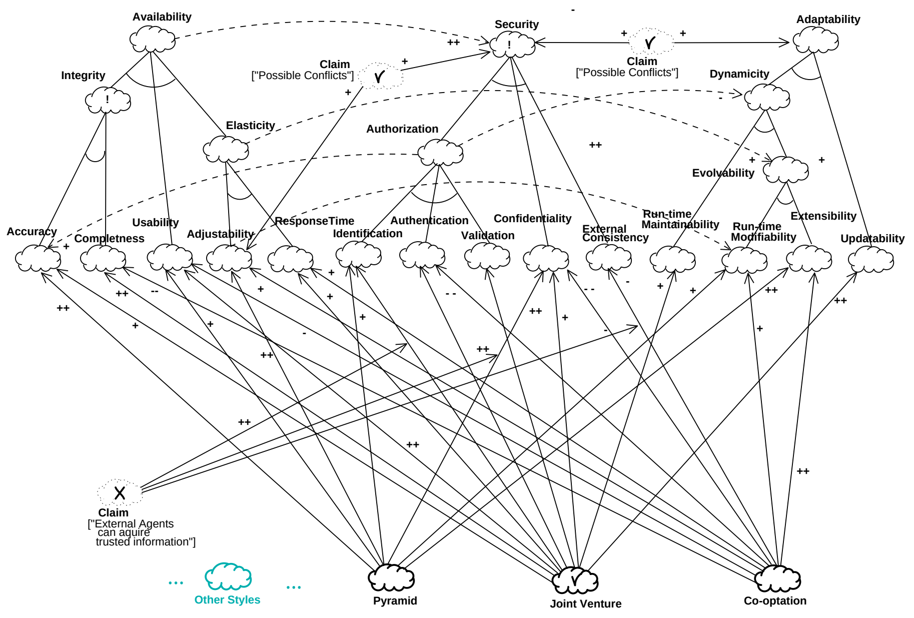

Fig. 7. Partial architecture evaluation for organizational styles.

Figure 8 offers a possible assignment of system responsibilities for our example, based on the joint venture style. The system consists of three principal partners, Store Front, Billing Processor and Back Store. Each of them delegates authority to, and is controlled and coordinated by, the joint management actor (Joint Manager) managing the system on a global dimension. Store Front interacts primarily with Customer and provides her with a usable front-end web application. Back Store keeps track of all web information about customers, products, sales, bills and other data of strategic importance to Media Shop. Billing Processor is in charge of the (secure) management of orders and bills, as well as other financial data. It is also in charge of interactions with Bank Cpy. Joint Manager manages all of the above, controlling *security*, *availability* and *adaptability* concerns.

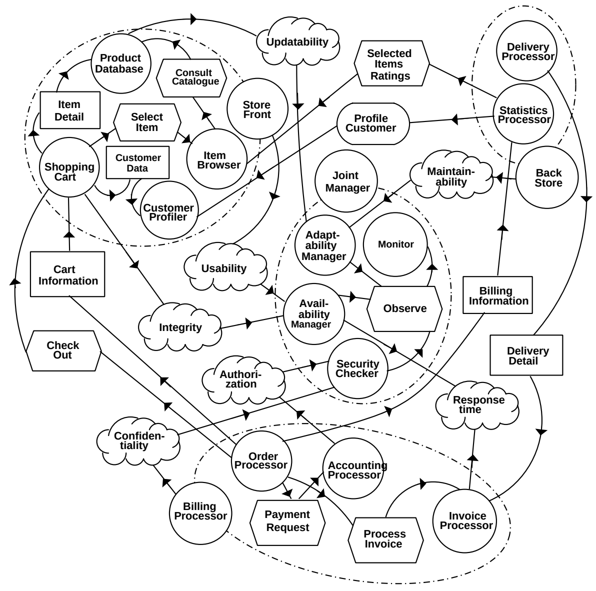

Fig. 8. An e-commerce system in joint venture architecture.

To accommodate the responsibilities of Store Front, we introduce Item Browser to manage catalogue navigation, Shopping Cart to select and custom items, Customer Profiler to track customer data and produce client profiles, and Product Database to manage media items information.

To cope with the identified software qualities (*security*, *availability* and *adaptability*), Joint Manager is further refined into four new sub-actors: Availability Manager, Security Checker and Adaptability Manager assume one of the main softgoals (and their subgoals). They are all monitored by Monitor. Further refinements are shown on Figure 8.

Figure 9 shows a possible use of some of the multi-agent patterns for designing in terms of agents the architecture of the e-business system shown in Figure 8. In particular, the broker pattern is applied to the Info Searcher, which satisfies requests of searching information by accessing Product Database. The Source Matchmaker applies the matchmaker pattern locating the appropriate source for the Info Searcher, and the monitor pattern is used to check both the correct use of the user data and the security for the sources accesses. Finally, the mediator pattern is applied to mediate the interaction among Info Searcher, Source Matchmaker and the Wrapper, while the wrapper pattern makes the interaction between Item Browser and Product Database possible. Other patterns can be applied as well. For instance, we could use the contract-net pattern to delegate to a wrapper the interaction with the Product Database, or the embassy to route the request of a wrapper to the Product Database.

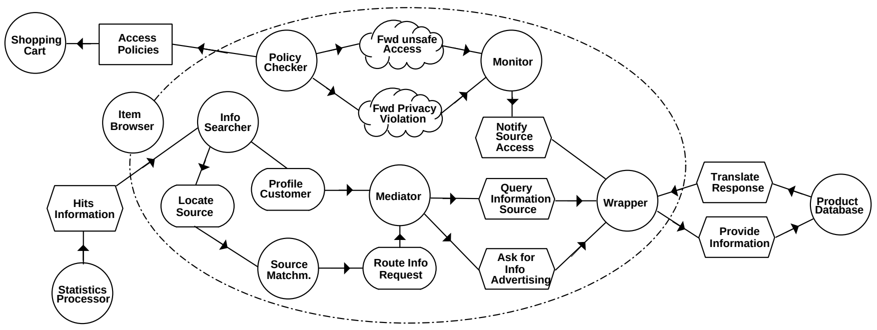

Fig. 9. Multi-agents patterns for item browser

5 Conclusions

Designers rely on styles, patterns, or idioms, to describe the architectures of their choice. We propose that MAS can be conceived as *organizations* of agents that interact to achieve common goals. This paper proposes a catalogue of architectural styles and agent patterns for designing MAS architectures at a macro- and micro-level. The proposed styles adopt concepts from organization theory and strategic alliances literature. The proposed patterns are based on earlier research within the agents community. The paper also includes an evaluation of software qualities that are relevant to these styles.

Future research directions include formalizing precisely the organizational styles and agent patterns that have been identified, as well as the sense in which a particular model is an instance of such a style and pattern. We also propose to compare and contrast them with classical software architectural styles and patterns proposed in the literature, and relate them to lower-level architectural components involving (software) components, ports, connectors, interfaces, libraries and configurations.

References

[1] 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), New York, USA, May 1998.

[2] J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology", In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering* (CAiSE'01), Interlaken, Switzerland, June 2001.

[3] L. K. Chung, B. A. Nixon, E. Yu and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal–directed Requirements Acquisition", *Science of Computer Programming, 20*, 1993, pp. 3-50.

[5] D.Deugo, F. Oppacher, J.Kuester, I. V. Otte. “Patterns as a Means for Intelligent Software Engineering”. In *Proc. of the Int. Conf. of Artificial Intelligence* (IC-AI’99), Vol II, CSRA Press, 605-611, 1999.

[6] J. Ferber and O. Gutknecht.“A meta-model for the analysis and design of organizations in multi-agent systems”. In *Proc. of the 3rd Int. Conf. on Multi-Agent Systems* (ICMAS’98), June, 1998.

[7] M.S. Fox. “An organizational view of distributed systems”. In *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70-80, January 1981.

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

[9] B. Gomes-Casseres. *The alliance revolution : the new shape of business rivalry*, Harvard University Press, 1996.

[10] S. Hayden, C. Carrick, and Q. Yang. “Architectural Design Patterns for Multiagent Coordination”. In *Proc. of the 3rd Int. Conf. on Autonomous Agents* (Agents’99), Seattle, USA, May 1999.

[11] E. Kendall, P.V. Murali Krishna, C. V. Pathak, and C.B. Suersh. “Patterns of Intelligent and Mobile Agents”. In *Proc. of the 2nd Int. Conf. on Autonomous Agents* (Agents’98), pages 92—98, New York, May 1998.

[12] M. Kolp, J. Castro and J. Mylopoulos, “A Social Organization Perspective on Software Architectures”. In *Proc. of the First Int. Workshop From Software Requirements to Architectures* (STRAW'01), Toronto, May 2001.

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

[14] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: “Telos: Representing Knowledge About Information Systems”. In *ACM Trans. Info. Sys.*, 8 (4), Oct. 1990, pp. 325 – 362.

[15] T.W. Malone. “Organizing Information Processing Systems: Parallels Between Human Organizations and Computer Systems”. In W. Zachry, S. Robertson and J. Black, eds. *Cognition, Cooperation and Computation*, Norwood, NJ: Ablex, 1988.

[16] R. Motschnig-Pitrik, “The Semantics of Parts Versus Aggregates in Data/Knowledge Modeling”, *Proc. of the 5th Int. Conf. on Advanced Information Systems Engineering* (CAiSE’93), Paris, June 1993, pp 352-372.

[17] W. Pree. *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.

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

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

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

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

[22] E. Yu. Modelling *Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.