A Goal-Based Organizational Perspective on Multi-Agent Architectures

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

[1] Department of Computer Science - University of Toronto, 10 King's College Road
M5S 3G4, Toronto, Canada, tel.: 1-416-978 7569, {mkolp,jm}@cs.toronto.edu
[2] Department of Mathematics - University of Trento, 5 via Inama, I-38100, Trento, Italy,
tel.: 39-0461-88 2114, pgiorgini@science.unitn.it

Abstract. A Multi-Agent System (MAS) is an organization of coordinated autonomous agents that interact in order to achieve particular, possible 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 modeled using the *i** framework which offers the notions of actor, goal and actor dependency and specified as metaconcepts in Telos. Each style is evaluated with respect to a set of software quality attributes, such as predictability or adaptability. A macro level defines these organizational styles and a micro level refers to patterns coming from the agent community. These patterns will define how goals assigned to actors of an organizational architecture will be fulfilled by agents. An e-business example illustrates our purposes. The research is conducted in the context of *Tropos*, a comprehensive software system development methodology.

1 Introduction

Multi-Agent System (MAS) architectures can be considered organizations (see e.g., [Fox81], [Mal88], [Feb98]) composed of *autonomous* and *proactive* agents that interact with one another in order to cooperate and then achieve either a common goal or simply their own goals. Using real world organizations as an a metaphor, distributed and open systems involving many software entities, such as MAS, could benefit from organizational structures (or *styles*) identified in organization theory and strategic alliances literature [Kol01]. In this paper, we adopt (some of) these styles for multi-agent architectural design, using the strategic dependency model of *i** [Yu95], and specifying them in Telos [Myl90]. We then present multi-agent patterns to be used in designing MAS architectures to define how the goals assigned to each actor in an organizational architecture are fulfilled by agents.

This research is being conducted in the context of the Tropos project [Cas01], which is developing a requirements-driven methodology for software systems. 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 [Dar93, Yu95]. In particular, it adopts Eric Yu's *i** model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an

application during early requirements analysis. The key assumption which distinguishes Tropos from other methodologies is that actors and goals are 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 we use to design the overall architecture of a system. It then proposes a set of software quality attributes for these styles in terms of which one can evaluate architectural alternatives. Section 3 introduces the micro level catalogue of goal-based multi-agent patterns for designing the identified organizational architecture of the system in terms of agents. Section 4 presents fragments of an e-business architecture which illustrates the use of styles and patterns proposed in the paper. Finally, Section 5 summarizes the contributions and points to further work.

2 Organizational Styles

Organizational theory [Min92, Sco98] and strategic alliances [Gom96, Seg96, Yos95] 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, using the strategic dependency model of *i**, and specifying them in Telos [Myl90].

A strategic dependency model is a graph, where each node represents an actor (an agent, position, or role within an organization) 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 called 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 [Chu00].

For instance, in Figure 1, the coordination, 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 Coordination and Support respectively through goal dependencies Control and Logistics represented as oval-shaped icons. The Operational Core 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 the typical strategic and logistic components generally 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.

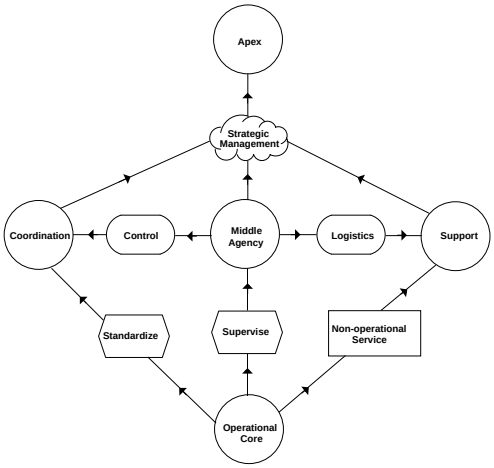

Fig. 1. Structure-in-5

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 [Myl90]. 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, primarily based on the aggregation semantics for Telos presented in [Mot93].

```
TELL CLASS StructureIn5MetaClass
  IN Class WITH /*Class is here used as a MetaMetaClass*/
   attribute name: String
       part, exclusivePart, dependentPart
             ApexMetaClass: Class
             CoordinationMetaClass: Class
             MiddleAgencyMetaClass: Class
             SupportMetaClass: Class
             OperationalCoreMetaClass: Class
END StructureIn5MetaClass
```

Fig. 2. Structure-in-5 in TELOS

The structure-in-5 style is then a metaclass - `StructureIn5MetaClass` - aggregation of five (*part*) metaclasses: `ApexMetaClass`, `CoordinationMeta-Class`, `MiddleAgencyMetaClass`, `SupportMetaClass` and `OperationalCore-MetaClass`, one for each actor composing the structure-in 5 style. Each of these five

components exclusively belongs (`exclusivePart`) to the composite (`Structure-In5MetaClass`) and their existence depend (`dependentPart`) on the existence of the composite. A structure-in-5 specific to an application domain will be defined as a Telos class, instance of `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.

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 [Yu95]. 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 Class WITH /*Class is here used as a MetaMetaClass*/
     attribute name: String
    taskDepended
         s:StandardizeTask
    WITH depender
    OperationalCoreMetaClass: Class
    END
    goalDepended
         c:ControlGoal
    WITH depender
    MiddleAgencyMetaClass: Class
    END
    softgoalDepender
         s:StrategicManagementSoftGoal
    WITH dependee
    ApexMetaClass: Class
    END
END CoordinationMetaclass
```

Fig. 3. Structure-in-5 Coordination actor in TELOS

The **pyramid** (pyr) style is the well-known hierarchical authority structure exercised within organizational boundaries. Actors at the lower levels depend on actors of the higher levels. The crucial mechanism is direct supervision from 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 by their own but only at a local level. This style can be applied when deploying simple distributed systems. Moreover, it encourages dynamicity since coordination and decision mechanisms are direct, not complex and immediately identifiable. Evolvability and modifiability can thus be implemented in terms of this style at low costs. However, it is not suitable for huge distributed systems like multi-agent systems requiring many kinds of agents. Even tough, it can be used by these systems to manage and resolve crisis situations. For instance, a complex multi-agent system faced with a non-authorized intrusion from external and non trustable agents could dynamically, for a short or long time, decide to migrate itself into a pyramid organization to be able to resolve the security problem in a more efficient way.

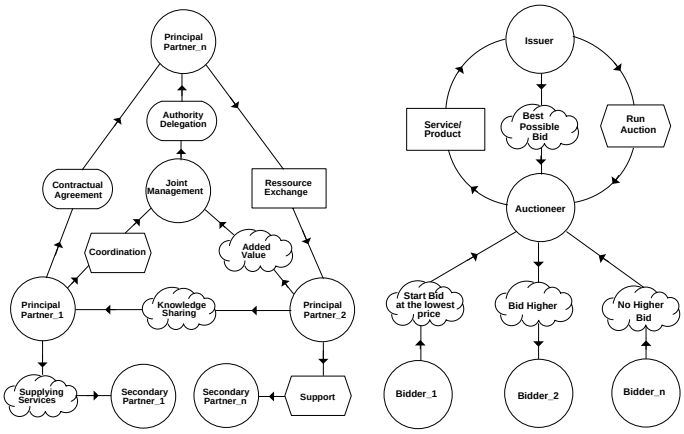

Fig. 4. Joint Venture (a) and Bidding (b)

The **joint venture** (jo-ve) style (Figure 4a) involves agreement between two or more principal partners to obtain the benefits of larger scale, partial investment and lower maintenance costs. Through the delegation of authority to a specific Joint Management actor that coordinates tasks and manages sharing of knowledge and resources, they pursue joint objectives. Each principal partner can manage and control itself on a local dimension and interact directly with other principal partners to exchange, 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 supports tasks for the organization core.

The **bidding** (bidd) style (Figure 4b) involves competitivity mechanisms and actors behave as if they were taking part in an auction. The Auctioneer actor runs the show, 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 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 delegated or lost from a collaborator to another.

The **hierarchical contracting** (hi-co) 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 (top to bottom) restrict autonomy and underlie a cooperative venture between the contracting parties. Such dual and 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 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, and its local contractors, has to come to terms with what is doing 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.

The evaluation of the previous styles can be done with respect to 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 [Kol01] 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 [Chu00].

3 Multi-agent patterns

A further step in the architectural design consists in defining how the goals assigned to each actor are fulfilled by agents. For this end, designers can be guided by a catalogue of multi-agent patterns in which a set of pre-defined solutions are available. A lot of work has been done in software engineering for defining software patterns, and many of them, such as those identified in [Gam95, Pre95], can be incorporated into multi-agent system architectures. Unfortunately, they focus on object-oriented not on the inherent characteristics of agents.

In the area of MAS, some work has been done in designing agent patterns, see for instance [Ari98, Deu99, Ken98]. However, these contributions focus on problems like how agents communicate one another, get information from an information sources, and establish a connection with a specific host. Differently, in Tropos, multi-agent patterns are used for solving a specific goal at organization level.

In the following we present a micro level catalogue involving some multi-agent pattern recurrent in multi-agent literature; in particular, some of the federated patterns introduced in [Hay99, Woo99] we will use in Section 4. For lack of space, we do not use any particular template, but we simple describe the patterns giving a brief description and in some case modeling them with *i**.

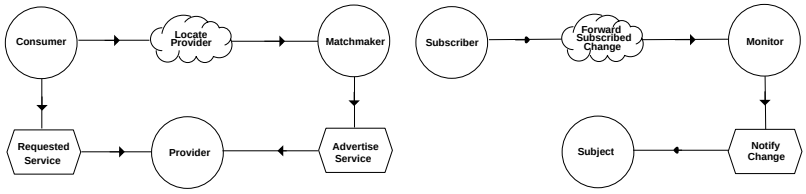

Fig. 5. Matchmacker (a) and Monitor (b)

A **broker** is an arbiter and intermediary accessing services of an agent (provider) to satisfy the request of a consumer. It is especially used in the horizontal integration and joint venture styles.

A **matchmaker** (Figure 5a) locates a provider corresponding to a consumer request for service, and then hands the consumer a handle to the chosen provider directly. Contrary to the broker who directly handles all interactions between the consumer and the provider, the negotiation for service and actual service provision are separated into two distinct phases. It can also be used in horizontal integrations and joint ventures.

A **monitor** (Figure 5b) alerts a subscriber about relevant events. It accepts subscriptions, request notifications from subjects of interest, receive such notifications of events and alerts subscribers to relevant events. The subject provides notifications of state changes as requested. The subscriber registers for notification of state changes to distributed subjects, receive notifications with current state information, and update its local state information. This pattern is used in the horizontal contracting, vertical integration, arm's-length and bidding styles implying observation activities.

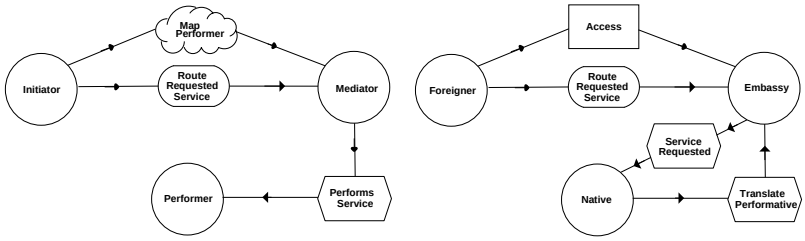

Fig. 6. Mediator (a) and Embassy (b)

A **mediator** (Figure 6a) mediates interactions among different agents. An initiator addresses the mediator in place of asking directly another colleague, the performer. It has acquaintance models of colleagues and coordinates the cooperation between them. Inversely, each colleague has an acquaintance model of the mediator. While a broker simply matches providers with consumers, a mediator encapsulates interactions and maintains models of initiators and performers behaviors over time. It is used in the pyramid, vertical integration and horizontal contracting styles since it underlies direct cooperation and encapsulation features reinforcing authority.

An **embassy** (Figure 6b) routes a service requested by an foreign agent to local one and handle back the response. If the access is granted, the foreign agent can submit messages to the embassy for translation. The content is translated in accordance with

a standard ontology. Translated messages are forwarded to target local agents. The results of the query are passed back out to the foreign agent, translated in reverse. This pattern can be used in the structure-in-5, arm's-length, bidding and co-optation styles to take in charge security aspects between systems component related to the competitivity mechanisms inherent to these styles.

A **wrapper** incorporates a legacy system into a multi-agent system**.** The wrapper interfaces the clients to the legacy by acting as a translator between them. This ensures that communication protocols are respected and the legacy system remains decoupled from the clients. This pattern can be used in the co-optation style when one of the co-optated actor is a representative for a legacy system.

The **Contract-Net** pattern selects an agent to which assign a task. The pattern involves a manager and any number of participants. The manager issues a request for proposal for a particular service to all participants and then accepts "proposals" to meet the service request at a particular "cost". The manager selects one participant who performs the contracted work and informs the manager upon completion. This pattern is especially used in the arm's-length and bidding and co-optation styles due to their inherent competitive features.

A detailed analysis of each multi-agent pattern allows to define a set of capabilities associated with the agents involved in the pattern. Such capabilities are not exhaustive and concern exclusively the agents activities about the pattern's goal. Due to the lack of space, we only present a set of capabilities for the matchmaker pattern (Table 2).

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 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, but simply that the agent should be capable to achieve in some way a given goal. In the Tropos methodology plans are defined in the detail design phase. It is important to notice that we have common capabilities for different agents; for instance, the capability handle with services ontology is common to all the three agents of the Matchmaker pattern. This suggests to build a sort of capability patterns repository usable later for implementing the single agents.

4 An E-business Example

E-business systems are designed to implement "virtual enterprises". By now, software architects have developed catalogues of web architectural style (e.g., [Con00]). Some most common styles are 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 non functional requirements of the application. As a result, the organizational architecture styles are not described nor the conceptual high-level perspective of the e-business application. The following software quality attributes for an e-commerce architecture could be stated according [Kol01]: *Security*, *Availability* and *Adaptability*.

To cope with these software quality attributes and select the architecture of the system, we go through a means-ends analysis using the non functional requirements (NFRs) framework[1]. We refine the identified attributes to sub-attributes that are more precise and evaluates alternative architectural styles against them, as shown in Figure 7. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level attributes. The styles are represented as operationalized attributes (saying, roughly, "make the architecture of the multi agent system *pyramid, co-optation, joint venture, arm's-leng*th-based, …").

The evaluation results in contribution relationships from the architectural styles to the quality attributes, labeled "+", "++", "-", "--". Design rationale is represented by claims drawn as dashed clouds. They make it possible for domain characteristics (such as priorities) to be considered and properly reflected into the decision making process, e.g., to provide reasons for selecting or rejecting possible solutions (+, -). Exclamation marks (! and !!) are used to mark priority attributes while a check-mark "✔" indicates an accepted attribute and a cross "✘" labels a denied attribute.

In Figure 7, *Adaptability* has been AND-decomposed into *Dynamicity* and *Updatability*. For our e-commerce example, *dynamicity* should deal with the way the system can be designed using generic mechanisms to 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 be fashionable for marketing reasons. Frameworks like Active Server Pages (ASP), Server Side Includes (SSI) to create dynamic pages make this attribute easier to achieve. *Updatability* should be strategically important for the viability of the application, the stock management and the business itself since *Media Shop*

[1] In the NFR framework, software quality attributes are called non functional requirements represented as softgoals (cloudy shapes)

employees have to very regularly bring up to date the catalogue by for inventory consistency. Comparable analysis are carried out in turn for newly identified sub-attributes as well as for the other top-level quality attributes *Security* and *Availability*

Eventually, the analysis shown in Figure 6 allows us to choose the joint venture architectural style for our e-commerce example (the operationalized attribute is marked with a "✔"). The analysis uses the correlation catalogue depicted in Table 1 and the top level quality attributes *Adaptability*, *Security* and *Availability* identified. They are respectively marked ++, +, ++ for the selected style. More specific attributes have also been identified during the decomposition process, such as *Integrity* (*Accuracy*, *Completeness*), *Usability*, *Response Time*, *Maintainability*, *Updatability*, *Confidentiality*, *Authorization* (*Identification*, *Authentication*, *Validation*) and need to be considered in the system architecture.

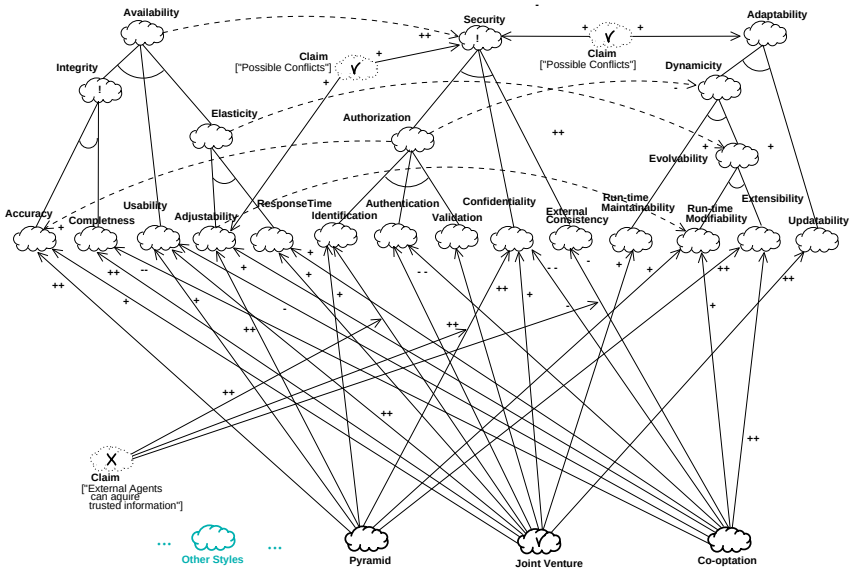

Fig. 7. Partial Architecture Evaluation for Organizational Styles.

Figure 8 suggests a possible assignment of system responsibilities, based on the joint venture architectural style for such an application. The system is decomposed into three principal partners (Store Front, Billing Processor and Back Store) controlling themselves on a local dimension and exchanging, providing and receiving services, data and resources with each other. 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, and other financial data; also of interactions to Bank Cpy. Joint Manager manages all of them controlling *security* gaps, *availability* bottlenecks and *adaptability* issues.

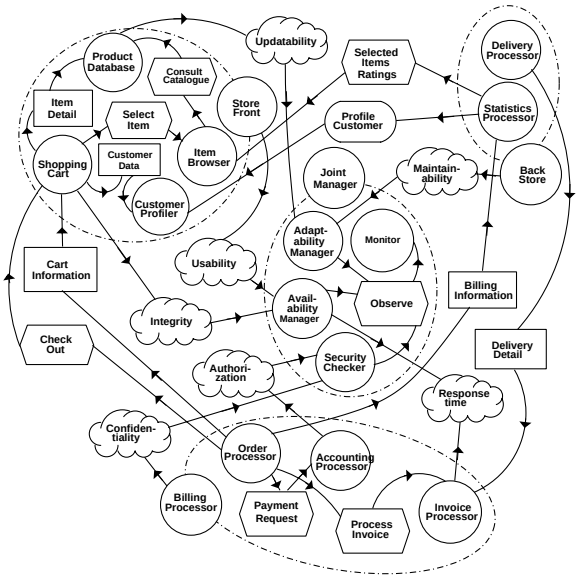

Fig. 8. An e-commerce system in joint venture architecture.

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 system sub-actors Availability Manager, Security Checker and Adaptability Manager each of them assuming one of the main softgoals (and their more specific subgoals) and observed by a Monitor. Further refinements are shown on Figure 8.

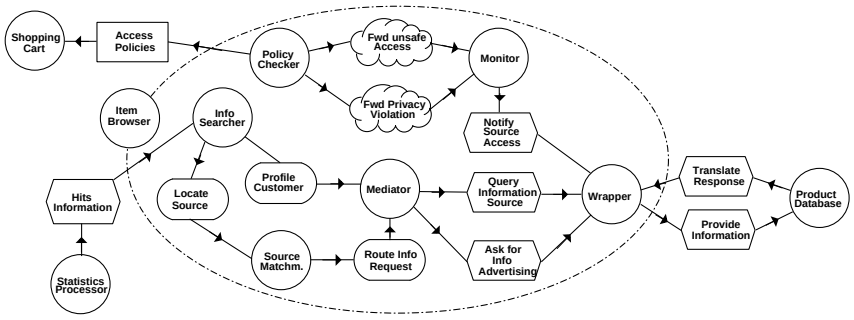

Fig. 9. Multi-agents patterns for Item Browser

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 the Info Searcher, the Source Matchmaker, and the Wrapper, while the wrapper pattern makes the interaction between the Item Browser and the Product Database possible. Of course, other patterns can be applied. For instance, we could use the contract-net pattern to select a wrapper to which delegate the interaction with the Product Database, or the embassy to route the request of a wrapper to the Product Database.

5 Conclusion

Designers rely on styles, patterns, or idioms, to describe the architectures of their systems — i.e., the configurations of components that make up the systems. MAS can be defined as *organizations* of agents that interact to achieve common goals. This paper proposes a catalogue of architectural styles which adopt concepts from organization theory and strategic alliances literature. Indeed, considering real world organizations as a metaphor, systems involving many software entities, such as MAS, could benefit from the same organizational concepts. The contribution also includes the evaluation of software quality attributes identified for these styles.

The organizational styles we have described constitute an architectural macro level. At a micro level we have been focusing on patterns. Many existing patterns can be incorporated into system architecture, such as those identified in [Gam95, Pre95]. For agent inherent characteristics, patterns for distributed, and open architectures like the broker, matchmaker, embassy, mediator, wrapper, mediator are more appropriate [Hay99, Woo99]. Agent capabilities for these patterns have been also introduced.

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 to 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

[Ari98] Y. Aridor and D. B. Lange. *Agent Design Patterns: Elements of Agent Application Design*. Proceeding of Autonomous Agents'98, ACM Press, 1998.

[Cas01] J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology," To appear in *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering* (CAiSE'01), Interlaken, Switzerland, June 2001.

[Chu00] L. K. Chung, B. A. Nixon, E. Yu and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.

[Dar93] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal–directed Requirements Acquisition", *Science of Computer Programming, 20*, 1993, pp. 3-50.

[Deu99] D.Deugo, F. Oppacher, J.Kuester, I. V. Otte. *Patterns as a Means for Intelligent Software Engineering*. Proceedings of the International Conference of Artificial Intelligence (IC-AI'99), Vol II, CSRA Press, 605-611, 1999.

[Feb98] J. Ferber and O. Gutknecht. *A meta-model for the analysis and design of organizations in multi-agent systems*. In Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98). IEEE CS Press, June, 1998.

[Fox81] M.S. Fox. *An organizational view of distributed systems*. IEEE Transactions on Systems, Man, and Cybernetics, 11(1):70-80, January 1981.

[Gam95] E. Gamma., R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995

[Gom96] B. Gomes-Casseres. *The alliance revolution : the new shape of business rivalry*, Cambridge, Mass., Harvard University Press, 1996.

[Hay99] S. Hayden, C. Carrick, and Q. Yang. *Architectural Design Patterns for Multiagent Coordination*. Proc. of the International Conference on Agent Systems , Agents'99 Seattle, WA, May 99.

[Ken98] E. Kendall, P.V. Murali Krishna, C. V. Pathak, and C.B. Suersh. *Patterns of Intelligent and Mobile Agents*. In K. Sycara and M. Wooldridge, eds., Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS-98), pages 92—98, New York, May 9—13, 1998, ACM Press.

[Kol01] M. Kolp, J. Castro and J. Mylopoulos, *A Social Organization Perspective on Software Architectures*. To appear in *Proceedings of the First International Workshop From Software Requirements to Architectures (STRAW'01)*, Toronto, May 2001.

[Min92] H. Mintzberg, Structure in fives : designing effective organizations, Englewood Cliffs, N.J., Prentice-Hall, 1992.

[Myl90] 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.

[Mal88] T.W. Malone. Organizing Information Processing Systems: Paralles Between Human Organizations and Computer Systems. In W. Zachry, S. Robertson and J. Black, eds. *Cognition, Cooperation and Computation*, Norwood, NJ: Ablex, 1988.

[Mot93] R. Motschnig-Pitrik, "The Semantics of PartsVersus Aggregates in Data/Knowledge Modeling", Proc. of the 5th Int. Conference on Advanced Information Systems Engineering (CAiSE'93), Paris, June 1993, pp 352-372.

[Pre95] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.

[Sha96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Upper Saddle River, N.J., Prentice Hall, 1996.

[Sco98] W. Richard Scott. *Organizations : rational, natural, and open systems*, Upper Saddle River, N.J., Prentice Hall, 1998.

[Seg96] L. Segil. *Intelligent business alliances : how to profit using today's most important strategic tool*, New York, Times Business, 1996.

[Zam00] F. Zambonelli, N.R. Jennings, and M. Wooldridge. *Organisational Abstractions for the Analysis and Design of Multi-Agent Systems*. In Proc. of the 1st International Workshop on Agent-Oriented Software Engineering at ICSE 2000, Limerick (IR), June 2000.

[Yos95] M.Y. Yoshino and U. Srinivasa Rangan. *Strategic alliances : an entrepreneurial approach to globalization*, Boston, Mass., Harvard Business School Press, 1995.

[Yu95] E. Yu. Modelling *Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[Woo99] S. G. Woods and M. Barbacci. "Architectural Evaluation of Collaborative Agent-Based Systems." Technical Report, CMU/SEI-99-TR-025, SEI, Carnegie Mellon University, PA, USA, 1999.