Architectural Styles for Information Systems: An Organizational Perspective

Manuel Kolp **John Mylopoulos**

Department of Computer Science
University of Toronto
6 King's College Road
Toronto M5S 3H5, Canada
{mkolp, jm}@cs.toronto.edu

ABSTRACT

Enterprise information systems such as ERP, Knowledge Management or e-business systems need to deploy information system architectures which match the organization of the enterprise within which they operate. This calls for system architectures which adopt models from research in organization theory. With this idea in mind, we offer a set of organization–oriented architectural styles for information systems. Our approach complements well proposals for multi-agent architectures which are becoming increasingly important for business information systems. In this paper, we adopt an e-business example to illustrate how to design an organizational architecture for a business-to-consumer application. The research is conducted in the context of a comprehensive information system development methodology called *Tropos*.

Keywords
Architectural design, organization theory, multi-agent systems, software architectures, organization models.

1 INTRODUCTION

Enterprise information systems have traditionally suffered from an impedance mismatch. While the information system itself is conceived as a collection of (software) modules -- including objects, data structures and interfaces -- the enterprise operational environment for/in which the system is designed is understood in terms of agents, positions, roles, responsibilities, objectives, tasks, resources and organizational structures. This mismatch is one of the factors for the poor quality of enterprise information systems, also the frequent failure of their development processes.

Development methodologies for ERP, Knowledge Management and e-business systems need to integrate organizational and software system models to avoid this semantic gap. Indeed, ERP systems are designed to implement a process view of the enterprise to meet organizational goals, tightly integrating all functions from the enterprise organization. Knowledge management systems are designed to help the enterprise gain insight and understanding from its own knowledge and expertise. Much of this knowledge is tacitly hidden in the enterprise organization itself. Finally e-business systems are designed to implement "virtual enterprises" based on organizational patterns that drive their business processes.

This realization calls for information system architectures which adopt models from research in organizational theory and strategic alliances. In this paper, we offer a set of information system architectural styles which are motivated by organizational theory. Our perspective complements well, but also subsumes, proposals for

multi-agent architectures. Multi-agent systems are organizations composed of agents - autonomous entities who can act and interact with their environment. Coordination is achieved through inter-dependencies which define potential interactions and cooperations in order to achieve common goals.

Section 2 sketches the context of the Tropos project, which offers a comprehensive methodology for agent-oriented information system development. Section 3 describes an e-commerce example and introduces the primitive concepts offered by *i** [Yu95], the organizational modelling framework we have adopted for Tropos. In section 4 we present our organization-inspired architectural styles modeled with *i**, while in section 5 we present a set of software quality attributes in terms of which one can evaluate architectural alternatives, using the non-functional requirements framework [Chu00]. Based on such a comparison, we propose a system architecture for our example. Finally, section 6 summarizes the contributions of the paper and points to further work.

2 A METHODOLOGICAL CONTEXT : TROPOS[1]

Tropos [Cas00a] is an information system development methodology which is founded on the concepts of *actor* and *goal*. Tropos is intended as a seamless methodology which describes in terms of the same concepts the organizational environment within which an information system will eventually operate, as well as the system itself. The proposed methodology supersedes traditional development techniques, such as structured and object-oriented ones in the sense that it is tailored to information systems that will operate within an organizational context and is founded on concepts used during early requirements analysis. To this end, we adopt the concepts offered by *i** [Yu95], a modeling framework offering concepts such as *actor*, *agent*, *position* and *role*, as well as social *dependencies* among actors, including *goal*, *softgoal*, *task* and *resource* ones. Previous versions of Tropos appeared in [Myl00, Cas00].

The proposed methodology spans four phases of software development:

• Early requirements, concerned with the understanding of a problem by studying an organizational setting; the output is an organizational model which includes relevant actors, their goals and tinter-dependencies.

• Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities.

• Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies.

• Detailed design, where behaviour of each architectural component is defined in further detail.

3 MODELING A B2C SYSTEM WITH *I**[2]

Media Shop is a store selling and shipping different kinds of media items such as books, newspapers, audio CDs, videotapes, and the like. *Media Shop* is supplied with the latest releases by *Media Supplier* and customers can use a catalogue describing available media items to specify their order. To increase market share, *Media Shop* has decided to open up a 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, *Medi@*, is available on the web using communication facilities provided by *Telecom Cpy*. It also uses financial services supplied by *Bank Cpy*, which specializes on on-line transactions.

The basic objective for the new system is to allow an on-line customer to examine the items in the *Medi@* internet catalogue, and place orders. Customers can search the on-line store by either browsing the catalogue or querying the item database. An on-line search engine allows customers with particular items in mind to search title, author/artist and description fields through keywords or full-text search.

---

[1] For a detailed description of Tropos, see [Cas00a].

[2] For a detailed description of the *Medi@* case study, see [Cas00a].

During early requirements analysis, the analyst captures the intentions of stakeholders. These are modeled as goals which, through some form of a goal-oriented analysis, eventually lead to the functional and non-functional requirements of the system-to-be [Dar93]. In *i**, early requirements are assumed to involve social actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The *i** framework includes the *strategic dependency model* for describing the relationships among actors, as well as the *strategic rationale model* for supporting the reasoning that each actor goes through concerning its relationships with other actors.

A strategic dependency model is a graph, where each node represents an *actor*, and each link between two actors indicates that one actor depends on another for something in order that the former may attain some goal. We call the depending actor the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers is called the *dependum*. Figure 1 shows the beginning of an *i** model.

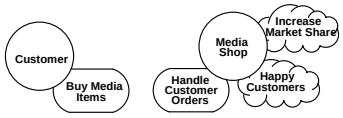

Figure 1: Requirements and needs for *Customers* and *Media Shop*

The two main stakeholders for our e-commerce application are *Customer* and *Media Shop*. The customer has one relevant goal *Buy Media Items* (represented as an oval-shaped icon), while the media store has goals *Handle Customer Orders*, *Happy Customers*, and *Increase Market Share*. Since the last two goals are not well-defined, they are represented as softgoals (shown as cloudy shapes).

A strategic rationale model determines through a means-ends analysis how these goals (including softgoals) can actually be fulfilled through the contributions of other actors. A strategic rationale model is a graph with four types of nodes -- *goal*, *task*, *resource*, and *softgoal* -- and two types of links -- means-ends links and process decomposition links. It captures the relationship between the goals of each actor and the dependencies through which the actor expects these dependencies to be fulfilled.

Figure 2 focuses on one of the (soft)goal *Increase Market Share*. The analysis postulates a task *Run Shop* (represented in terms of a hexagonal icon) through which it can be fulfilled. Tasks are partially ordered sequences of steps intended to accomplish some (soft)goal. Tasks can be decomposed into goals and/or subtasks, whose collective fulfillment completes the task. In the figure, *Run Shop* is decomposed into goals *Handle Billing* and *Handle Customer Orders*, tasks *Manage Staff* and *Manage Inventory*, and softgoal *Improve Service* which together accomplish the top-level task. As shown in the figure, subgoals and subtasks can be specified more precisely through refinement (see [Cas00a]).

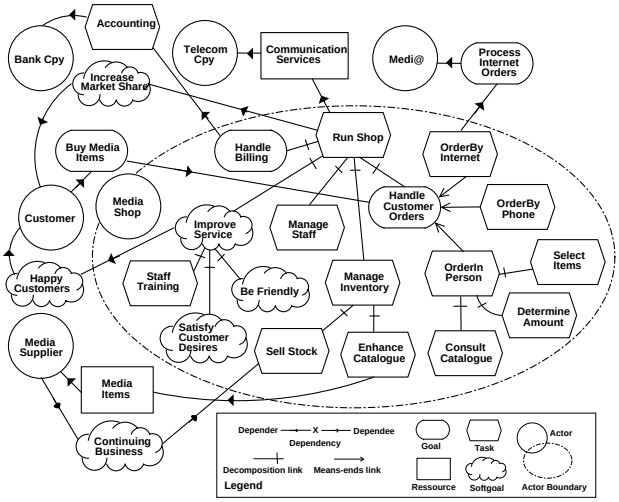

Figure 2: Means-ends analysis for the softgoal *Increase Market Share*

Late requirements analysis results in a requirements specification which describes all functional and non-functional requirements for the system-to-be. In Tropos, the system is represented as one or more actors

contributing to the fulfillment of stakeholder goals, along with other actors from the system's operational environment. For our example, the *Medi@* system is introduced as an actor in the strategic dependency model depicted in Figure 3.

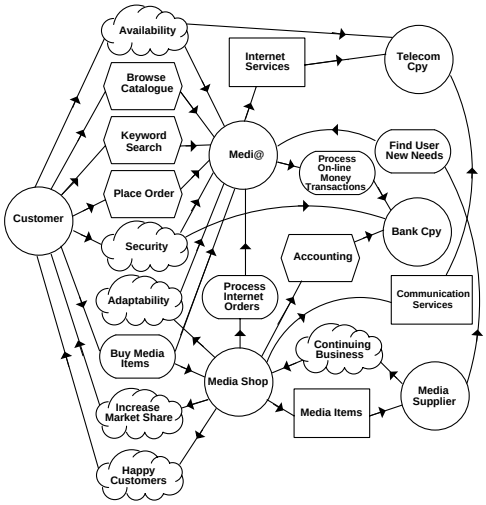

Figure 3: Strategic dependency model for a media shop

With respect to the actors identified in Figure 2, *Customer* depends on *Media Shop* to buy media items while *Media Shop* depends on *Customer* to increase market share and remain happy (with *Media Shop* service). *Media Shop* depends on *Medi@* for processing internet orders and on *Bank Cpy* to process business transactions. *Customer*, in turn, depends on *Medi@* to place orders through the internet, to search the database for keywords, or simply to browse the on-line catalogue. With respect to relevant qualities, *Customer* requires that transaction services be secure and usable, while *Media Shop* expects *Medi@* to be easily adaptable. Further dependencies are shown on Figure 3 and explained in [Cas00a].

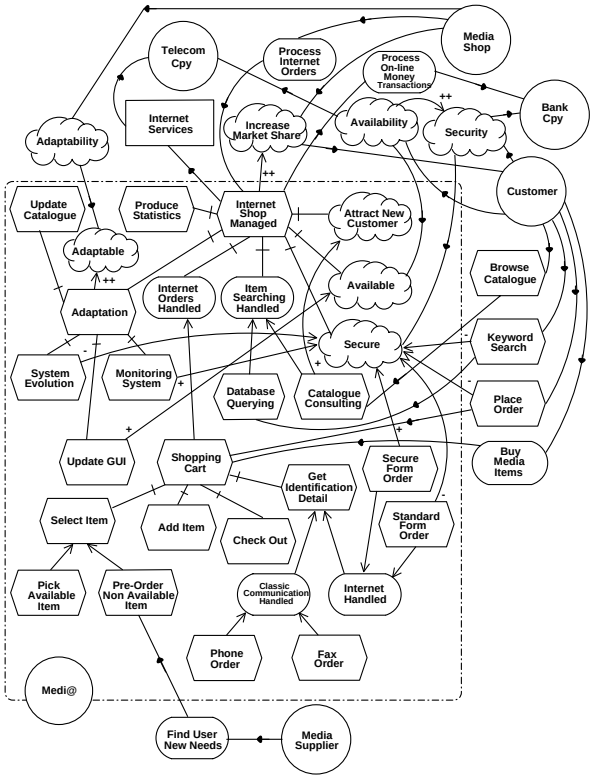

Figure 4: Strategic rationale model for *Medi@*

As late requirements analysis proceeds, *Medi@* is given additional responsibilities, and ends up as the depender of several dependencies. This responsibility assignment is realized using the same kind of means-ends analysis illustrated in Figure 2. Hence, the analysis in Figure 4 focuses on the system itself, instead of a external stakeholder starting from a root task *Internet Shop Managed* providing sufficient support (++) [Chu00] to the softgoal *Increase Market Share*.

Softgoal contributions are introduced to model sufficient or partially positive (++ and +) or negative (- - and -) support to softgoals *Secure*, *Available*, *Adaptable*, *Attract New Customers* and *Increase Market Share*. The result of this means-ends analysis is a set of (system and human) actors who are dependees for some of the dependencies that have been postulated.

4 ORGANIZATIONAL ARCHITECTURE STYLES

Architectural design has emerged as a crucial phase of the design process of an information system. A system architecture constitutes a small intellectually manageable model of system structure, which describes how system components work together. Several works have identified architectural styles (see e.g., [Sha96]) to guide high-level system design and have discussed how these drive the composition of a system from particular types of components. However, such work does not focus on multi-agent architectures. Moreover, previously developed styles are not applicable to agent-oriented systems. For instance, the client-server architecture that executes a program on the server triggered by a client is no longer relevant with mobility and nomadic features, such as those that are available for agent software.

Organizations exist primarily to coordinate the actions of many individuals for some purpose. That purpose could be to develop and manage structures as business units, profitable enterprises, multi-national alliances, governmental institutions, public administrations, charitable associations, theatre companies or sport leagues. Furthermore, real world organizations are not constructed with a population of identical individuals doing the same thing; instead, they diversify, delegate, negotiate, manage, cooperate, compete, and so on. Using real world organizations as an analogy, systems involving many software entities, such as mutli-agent systems (MAS), could benefit from the same organizational models and architectural designs understood in terms of organizational concepts. Moreover, when designing enterprise MAS, this approach is particularly relevant to match the enterprise operational environment.

We represent such styles inspired by organizational theory (such as [Min93, Sco98]) as well as strategic alliances (e.g., [Gom96, Seg96, Yos95]) in terms of the *i** modeling framework.

The **flat structure** has no fixed structure and no control of one actor over another is assumed. The main advantage of this architecture is that it supports autonomy, distribution and continuous evolution of an actor architecture. However, the key drawback is that it requires an increased amount of reasoning and communication by each participating actor.

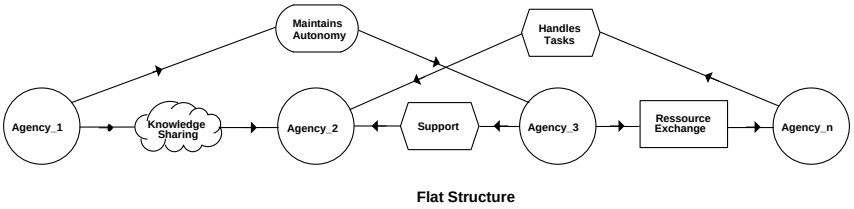

Figure 5 : Flat Structure

The **structure-in-5** style 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 of the organization lies the apex composed of strategic executive actors. Below it sit the logistics, control/standardization and management components respectively support, coordination and middle agency. The support component assists the operation core for non-operational services that is outside the basic flow of operational tasks and procedures. 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 who join the strategic apex to the operational core make up the middle agency.

The **pyramid** 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 multi agent 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. On the contrary, it is not suitable for huge MAS requiring many kinds of agents. However, it can be used by these MAS to manage and resolve crisis situations. For instance, a complex 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. When considering this style for applications in which the computation can appropriately be defined via a hierarchy of procedure definitions, it can also be related to the classical *main program and subroutines* architectural style [Sha96].

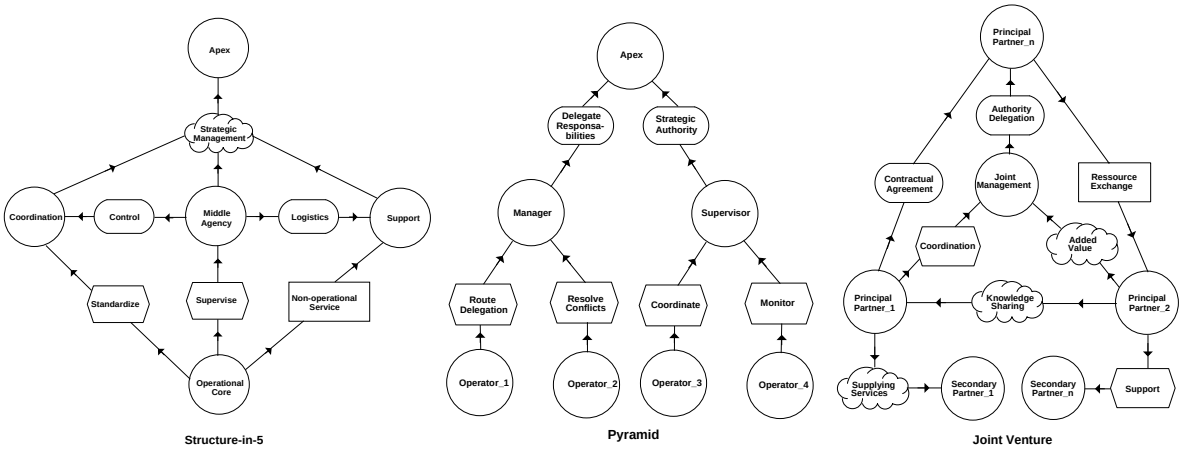

Figure 6: Structure-in-5, Pyramid and Joint Venture

The **joint venture** style 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 operations and manages sharing of knowledge and resources, they pursue joint objectives and common purpose. 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 system 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 **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 delegated or lost from a collaborator to another. Since this style is suitable for applications that involve a collection of distinct, largely independent computations whose execution should proceed competitively, it can be considered a derivation of the classical *communicating processes* architectural style [Sha96].

The **bidding** style involves competitivity mechanisms and actors needed to run 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 auctioneer might be a system actor that merely organizes and operates the auction and its mechanisms. It can also be one of the bidders (for example selling an item which all other bidders are interested in buying). The auction issuer is responsible for issuing the bidding. This style implies fast response time and adjustability for the system.

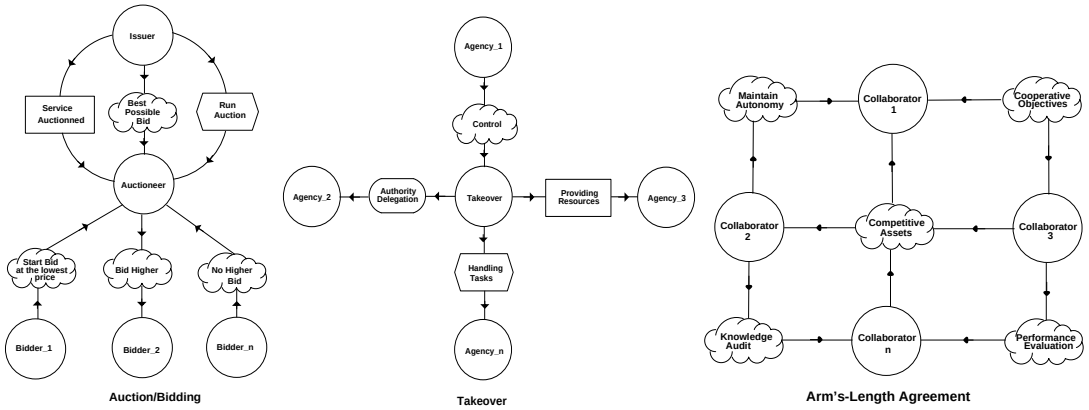

Figure 7 : Bidding, Takeover and Arm's-Length Agreement

The **takeover** style involves the total delegation of authority and management from two or more partners to a single collective *takeover* actor. The takeover style is similar in many ways to the joint venture style. The major and crucial difference is that while in a joint venture identities and autonomies of the separate units are preserved, the takeover absorbs these critical units in the sense that no direct relationships, dependencies or communications are tolerated except those involving the takeover.

The **hierarchical contracting** style identifies coordinating mechanisms that combine arm's-length agreement features with aspects associated with pyramidal authority. Coordination mechanisms developed to manage 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. Since this style is suitable for applications that involve distinct classes of layered services that can be arranged hierarchically, it can be considered a specialization of the classical *layered* architectural style [Sha96].

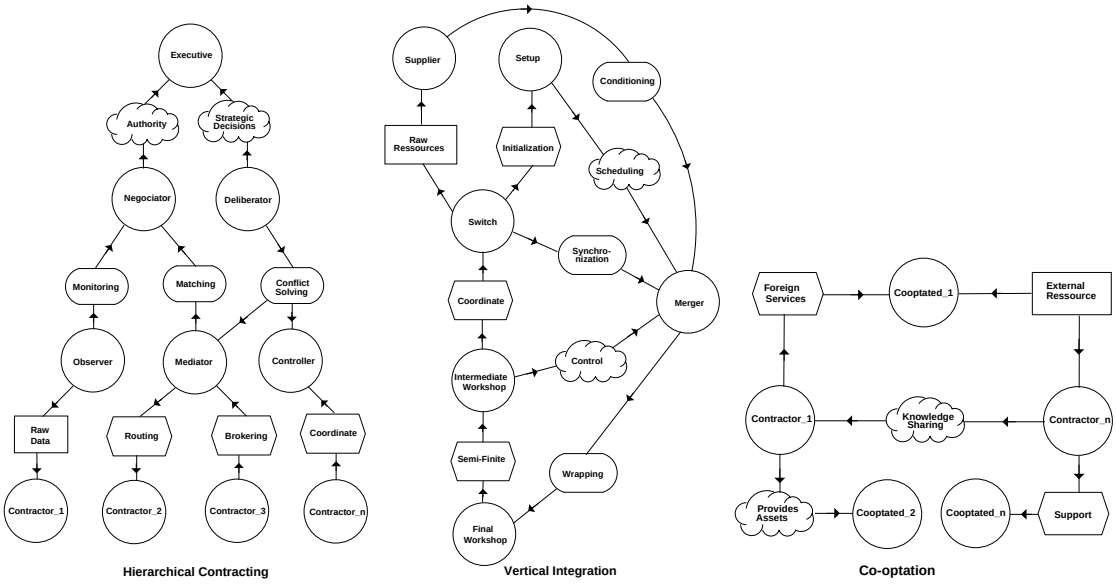

Figure 8 : Hierarchical Contracting, Vertical Integration and Co-optation

The **vertical integration** style merges, backward or forward, one or more system actors engaged in related tasks but at different stages of a production process. A merger synchronizes and controls interactions between each of the participants that can be considered as workshops. Since this style is suitable for applications that require a defined series of independent computations to be performed on ordered data, it can be viewed as a

specialization within organization boundaries of the classical *pipe and filter* architectural style [Sha96].

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, 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 his own views with the policy of the system he has to communicate. The receiving system's boundary is also crossed and the local contractors have to come to terms with the intrusion from the external environment and with the temporary or permanent addition to their number.

5 EVALUATING ARCHITECTURES WITH NFRS

During architectural design we concentrate on the key system actors, defined during requirements analysis, and their responsibilities. These include the desired functionality of the system-to-be, as well as a number of non functional requirements (also called software quality attributes) related to usability, security, availability, reusability, evolvability, extensibility, reusability, …

Due to the organizational nature of multi-agent systems, we have found the following non functional requirements relevant for architectural evaluation when deploying a MAS.

- **Predictability** [Woo99]**.** 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 such as timeliness and latency as part of determining the behavior of a multi-agent system at large.
- **Security.** Agents are often able to identify their own data sources and they may undertake additional actions based on these sources [Woo99]. 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 agents 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 agent's communication protocol or possibly the dynamic introduction of a new kind of agent previously unknown or the manipulations of existing agents.
- **Coordinability.** Agents are not particularly useful unless they are able to coordinate with other agents. This can be realized in two different antagonist ways:

  Cooperativity. Agents must be able to coordinate with other agents to achieve a common purpose.

  Competitivity. Agent must be able to coordinate with other agents except that the success of one agent implies the failure of others.
- **Availability.** Agents that offer services to other agents must implicitly or explicitly guard against the interruption of offered services. Availability must actually be considered a sub-attribute of security [Chu00]. Nevertheless, we prefer to deal with it as a top-level non functional requirements due to its increasing importance in multi-agent system design.
- **Failability-Tolerance.** A MAS is composed by several agents and a failure of one agent does not necessarily imply a failure of the whole system. To prevent system failure, different agents can have similar or replicated capabilities and refer to more than one agent for a specific behavior. However, this replication of capabilities induces redundancy in the system.
- **Modularity** [She98] 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 agents are components of other agents. They surrender to the control of the composite agent. This control results in efficient tasks execution and low communication overhead, however prevents the system to benefit from flexibility.

Figure 9 resumes the correlation catalogue for the ten architectural styles and nine top-level quality sofgoals

we have identified. HELP, MAKE, HURT, BREAK, respectively model partial/positive, sufficient/positive, partial/negative and sufficient/negative contributions.

Correlation Catalog	Predict.	Secur.	Adapt.	Cooperat.	Compet.	Availab.	Failabil.	Modul.	Aggreg.
Flat Structure	BREAK	BREAK	MAKE			HELP	HELP	MAKE	HURT
Structure-in-5	HELP	HELP		HELP	HURT	HELP		MAKE	MAKE
Pyramid	MAKE	MAKE	HELP	MAKE	BREAK	HELP	BREAK	HURT	
Join Venture	HELP	HELP	MAKE	HELP	HURT	MAKE		HELP	MAKE
Bidding	BREAK	BREAK	MAKE	HURT	MAKE	HURT	BREAK	MAKE	
Takeover	MAKE	MAKE	HURT	MAKE	BREAK	HELP		HELP	HELP
Arm's-Length	HURT	BREAK	HELP	HURT	MAKE	BREAK	MAKE	HELP	
Hierarch. Cont.			HELP	HELP	HELP	HELP		HELP	HELP
Vert. Integ.	HELP	HELP	HURT	HELP	HURT	HELP	BREAK	BREAK	BREAK
Co-optation	HURT	HURT	MAKE	MAKE	HELP	HURT	HELP		

Figure 9 : Correlation Catalogue for Organizational Architectures and Top-Level Quality Softgoals.

To cope with these quality softgoals, the software architect goes through a means-ends analysis refining them to sub-goals that are more precise and evaluates alternative architectural styles against them, as shown in Figure 10. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level quality softgoals. The styles are represented as operationalized softgoals (saying, roughly, "make the architecture of the multi agent system *pyramid*, *takeover*, *co-optation*, *joint venture*, *arm's-length*-based, …").

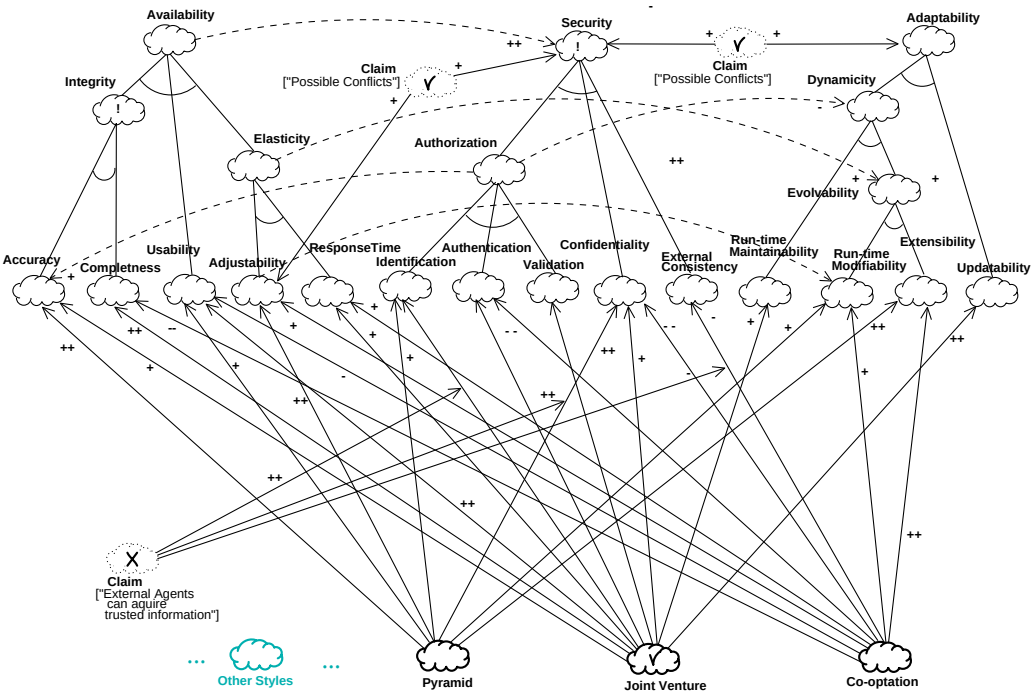

Figure 10 : Partial Architecture Evaluation for Organizational Styles.

The evaluation results in contribution relationships from the architectural styles to the quality softgoals, labeled "+" (HELP), "++" (MAKE), "-" (HURT), "--" (BREAK). Design rationale is represented by claim softgoals 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 softgoals while a check-mark "✔" indicates an accepted softgoal and a cross "✘" labels a denied softgoal.

In Figure 10, the *Adaptability* softgoal has been AND-decomposed into subgoals *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 softgoal easier to achieve. *Updatability* should be strategically important for the viability of the application, the stock management and the business itself since *Media Shop* 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 quality subgoals as well as for the other top-level quality softgoals *Security* and *Availability*.

Eventually, the analysis shown in Figure 10 allows us to choose the joint venture architectural style for our e-commerce example (the operationalized softgoal is marked with a "✔"). The analysis uses the correlation catalogue depicted in Figure 9 and the top level softgoals *Adaptability*, *Security* and *Availability* identified during late requirements analysis (Figures 3 and 4). They are respectively marked MAKE, HELP, MAKE for the selected style. More specific softgoals have also been identified during the NFR 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.

Figure 11 suggests one possible assignment of system responsibilities, based on the joint venture architectural style. The system has the structure described for this style in Figure 6. It 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* manage all of them controlling *security* gaps, *availability* bottlenecks and *adaptability* issues.

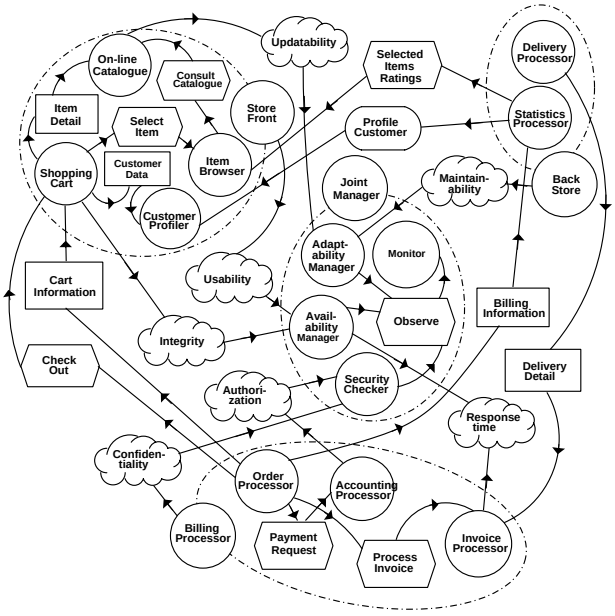

Figure 11 : The e-commerce system architecture in *joint venture*

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 *On-line Catalogue* to deal with digital library obligations. To cope with the non-functional requirement decomposition proposed in Figure 10, *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 11.

6 CONCLUSIONS

Software designers rely on informal styles, patterns, or idioms, to describe the architectures of their systems — i.e., the configurations of components that make up the systems. We argue that the architecture of enterprise information systems should be organized the same way enterprises are. In other words the development process should consider the organizational models of the enterprise for/in which they are designed. The research is conducted in the context of Tropos, a software development methodology driven by early requirements notions such as those of *actor* and *goal*.

This paper has focused on architectural styles taking inspiration from organizational models defined in organizational theory and the strategic alliances literature. This organizational perspective complements well proposals for multi-agent architectures, increasingly used for e-business and enterprise knowledge systems. Indeed, considering real world organizations as a metaphor, systems involving many software actors, such as MAS could benefit from the same organizational models. After all, human organizations diversify, delegate, negotiate, manage, cooperate, compete, and the like. Software architectures could greatly benefit from such qualities. The contributions of this paper include an application of the NFR framework to evaluate relevant software qualities for different architectural styles.

The organizational styles we have described should eventually constitute an architectural macrolevel. At a micro level we will be focusing on the notion of patterns. Many existing patterns can be incorporated into system architecture, such as those identified in [Gam95, Pre95, Bus96]. For agent inherent characteristics, patterns like the broker, matchmaker, embassy, mediator, wrapper, mediator are more appropriate [Hay99, Woo99]. Another direction for further work is to relate the architectural styles proposed in this work to lower-level architectural components involving (software) components, ports, connectors, interfaces, libraries and configurations.

ACKNOWLEDGEMENTS

The Tropos project has been partially funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada, and Communications and Information Technology Ontario (CITO), a Centre of Excellence funded by the province of Ontario.

We are grateful to Jaelson Castro (Federal University of Perambuco), Paolo Giorgini (University of Trento), Eric Yu (University of Toronto), and Axel van Lamsweerde (University of Louvain) for helpful suggestions and feedback to this research.

REFERENCES

[Bus96] F. Buschmann, R. Meunier, H. Rohnet, P. Sommerland and M. Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.

[Cas00] J. Castro, M. Kolp and J. Mylopoulos. Developing Agent-Oriented Information Systems for the Enterprise, *Proceedings of the Second International Conference On Enterprise Information Systems* (ICEIS00), Stafford, UK, July 2000.

[Cas00a] J. Castro, M. Kolp and J. Mylopoulos. “A Requirements-Driven Software Development Methodology,” submitted to the Conference on Advanced Information Systems Engineering (CAiSE’01.

[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.

[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,” In *Proceedings of the International Conference on Agent Systems '99* Agents'99, Seattle, WA, May 1999.

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

[Myl00] J. Mylopoulos and J. Castro. “Tropos: A Framework for Requirements-Driven Software Development”, Brinkkemper, J. and Solvberg, A. (eds.), *Information Systems Engineering: State of the Art and Research Themes*, Springer-Verlag, June 2000, pp. 261-273.

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

[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.

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

[She98] O. Shehory. “Architectural Properties of Multi-Agent Systems,” Technical report CMU-RI-TR-98-28, Carnegie Mellon University, 1998.

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

[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.