Agent-Oriented Software Development

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

[1] 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
[2]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
[3] Department of Mathematics - University of Trento, 4 via Sommarive, I-38100, Trento, Italy, tel.: 39-0461-88 2052, pgiorgini@science.unitn.it

Abstract. The Tropos project is developing concepts, tools and techniques for building agent-oriented software. This paper presents a quick overview of the project and then focuses on a specific problem: the identification of architectural styles for multi-agent systems (MAS). The proposed styles have been adopted from the literature on organization theory and strategic alliances. The styles are represented in *i**, a framework designed to model social and intentional concepts. Each proposed style is evaluated with respect to a set of agent software qualities, such as predictability, adaptability and availability. The use of the styles is illustrated and contrasted with a software architecture for mobile robot reported in the literature.

1 Introduction

The explosive growth of application areas such as electronic commerce, enterprise resource planning and mobile computing has profoundly and irreversibly changed our views on software and Software Engineering. Software must now be based on open architectures that continuously change and evolve to accommodate new components and meet new requirements. Software must also operate on different platforms, without recompilation, and with minimal assumptions about its operating environment and its users. As well, software must be robust and autonomous, capable of serving a naïve user with a minimum of overhead and interference. These new requirements, in turn, call for new concepts, tools and techniques for engineering and managing software.

For these reasons -- and more -- agent-oriented software development is gaining popularity over traditional software development techniques, including structured and object-oriented ones. After all, agent-based architectures (known as multi-agent systems in the Agent research community) *do* provide for an open, evolving architecture which can change at run-time to exploit the services of new agents, or replace under-performing ones. In addition, software agents can, in principle, cope with unforeseen circumstances because they include in their architecture goals, along with a planning capability for meeting them. Finally, agent technologies have matured to the point where protocols for communication and negotiation have been standardized [7].

We are developing a software development methodology for agent-based software systems. The methodology adopts ideas from multi-agent system technologies, mostly to define the implementation phase of our methodology [4]. We are also adopting ideas from Requirements Engineering, where agents and goals have been used heavily for early requirements analysis [5, 26]. In particular, we adopt 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 our work from others in Requirements Engineering is that actors and goals are used as fundamental concepts for modelling and analysis during *all phases of software development*, not just early requirements[1].

Our methodology, named Tropos, is intended to support five phases of software development:

Early requirements, concerned with the understanding of a problem by studying an existing organizational setting; the output of this phase is an organizational model which includes relevant actors and their respective dependencies;

Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities; this description models the system as a (small) number of actors which have a number of dependencies with actors in their environment; these dependencies define the system's functional and non-functional requirements;

Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data and control flows; within our framework, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies.

Detailed design, where each architectural component is defined in further detail in terms of inputs, outputs, control, and other relevant information; our framework adopts elements of AUML [19] to complement the features of *i**;

Implementation, where the actual implementation of the system is carried out, consistently with the detailed design; we use a commercial agent programming platform, based on the BDI (Beliefs-Desires-Intentions) agent architecture for this phase.

The motivations behind the Tropos project are presented in [2] and [12], including an early glimpse of how the methodology would work for particular case studies.

In this paper, we focus on a specific problem related to the Tropos methodology: the identification of architectural styles for Tropos models. System architectures describe a software system at a macroscopic level in terms of a manageable number of subsystems/components/modules inter-related through data and control dependencies. The design of software architectures has been the focus of considerable research for the past decade which has resulted in a collection of well-understood architectural styles and a methodology for evaluating their effectiveness with respect to particular

[1] Analogously to the use of concepts such as `object`, `class`, `inheritance` and `method` in object-oriented software development.

software qualities. Examples of styles are pipes-and-filters, event-based, layered and the like [11]. Examples of software qualities include maintainability, modifiability, portability, etc. [1]. Multi-Agent System (MAS) architectures can be considered as organizations (see e.g., [6, 8, 16]) composed of *autonomous* and *proactive* agents that interact and cooperate with one another in order to achieve common or private goals. Since the fundamental concepts of multi-agent systems are intentional and social, rather than implementation-oriented, we turn to theories which study social and intentional structures for motivation and insights. But, what kind of social theory should we turn to? There are theories that study group psychology, communities and social networks. Such theories study social and intentional structure as an *emergent property* of a social context. Instead, we are interested in organizational structures that emerge from a *design* process. For this, we turn to organizational theory and strategic alliances for guidance. The purpose of this paper is to present a set of architectural styles for multi-agent systems motivated by these theories. The styles are modeled using the strategic dependency model of *i** [26]. To illustrate these styles, we use a case study comparing organizational with conventional software architectural styles for mobile robot control software.

Section 2 presents our organization-inspired architectural styles described in terms of the strategic dependency model from *i** and specified in Telos. Section 3 introduces a set of desirable software quality attributes for comparing them. Section 4 overviews a mobile robot example while Section 5 discusses related work. Finally, Section 6 summarizes the contributions of the paper and points to further research.

2 Organizational Structures

Organizational theory (such as [14, 18]) and strategic alliances (e.g., [13, 25]) study alternatives for (business) organizations. These alternatives are used to model the coordination of business stakeholders -- individuals, physical or social systems -- to achieve common goals. Using them, we view a software system as a social organization of coordinated autonomous components (or agents) that interact in order to achieve specific, possibly common goals. We adopt (some of) the styles defined in organizational theory and strategic alliances to design the architecture of the system, model them with *i**, and specify them in Telos [17].

In *i**, a strategic dependency model is a graph, in which 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. By depending on another actor for a dependum, an actor is able to achieve goals that it is otherwise unable to achieve, or not as easily or as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals.

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 the structure-in-5 style (Figure 1), the coordination, middle agency and support actors depend on the apex for strategic management purposes. Since the goal *Strategic Management* is not well-defined, it is represented as a softgoal (cloudy shape). The middle agency actor depends on both the coordination and support actors respectively through goal dependencies *Control* and *Logistics* represented as oval-shaped icons. The operational core actor is related to the coordination and support actors respectively through the *Standardize* task dependency and the *Non-operational service* resource dependency.

In the sequel we briefly discuss ten common organizational styles.

The **structure-in-5** (Figure 1) 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.

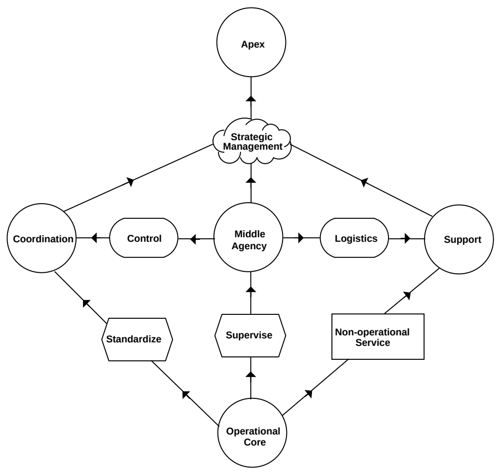

Fig. 1. Structure-in-5

Below it sit the control/standardization, management components and logistics, respectively coordination, middle agency and support. 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. 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, implementation and design decisions for software systems [17]. It provides features to describe metaconcepts that can be used to represent the knowledge relevant to a variety of worlds – subject, usage, system, development worlds - related to a software system. Our styles are formulated as Telos metaclasses, primarily based on the aggregation semantics for Telos presented in [15].

The structure-in-5 style is then a metaclass - *StructureIn5MetaClass* - aggregation of five (*part*) metaclasses: *ApexMetaClass*, *CoordinationMetaClass*, *MiddleAgencyMetaClass*, *SupportMetaClass* and *OperationalCoreMetaClass*, one for each actor composing the structure-in 5 style depicted in Figure 1. Each of these five components exclusively belongs (*exclusivePart*) to the composite (*StructureIn5MetaClass*) 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* (See Section 4). 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 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

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 [26].

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

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

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*.

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.

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 distributed systems.

Moreover, this style 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.

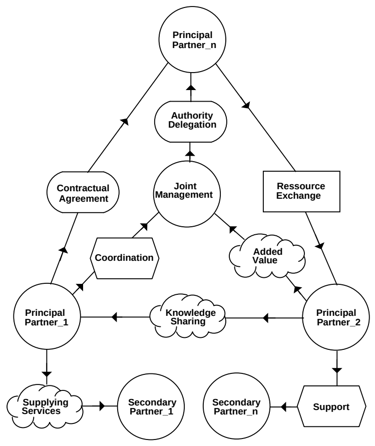

Fig. 4. Joint Venture

The **joint venture** style (Figure 4) 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 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.

The **takeover** style involves the total delegation of authority and management from two or more partners to a single collective *takeover* actor. It 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 **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.

The **bidding** style (Figure 5) 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 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.

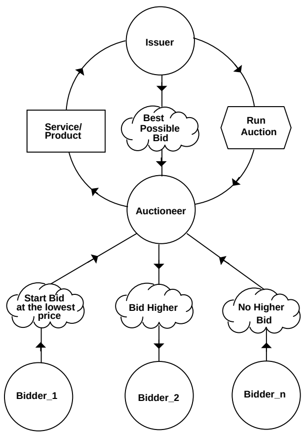

Fig. 5. Bidding

The **hierarchical contracting** style (Figure 6) 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.

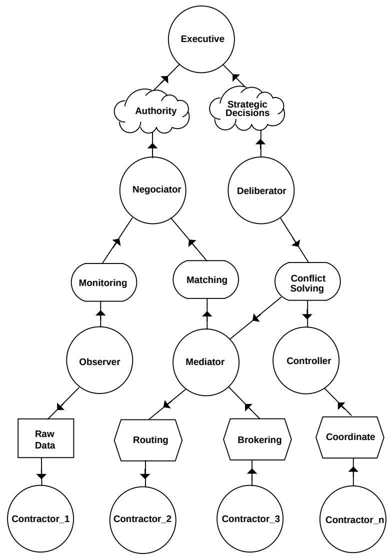

Fig. 6. Hierarchical Contracting

The **co-optation** style (Figure 7) involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of an initiating organization.

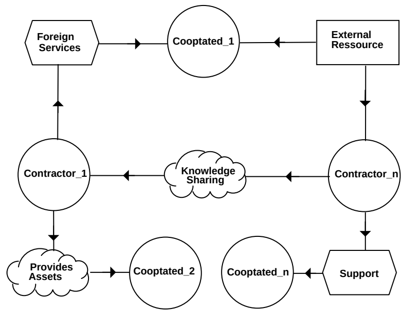

Fig. 7. Cooptation

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 **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 intermediate workshops. Vertical integrations take place between exchange partners, actors symbiotically related. Figure 8 presents a vertical integration style for the domain of goods distribution. *Provider* is expected to supply quality products, *Wholesaler* is responsible for ensuring their massive exposure, while *Retailer* takes care of the direct delivery to the *Consumers*.

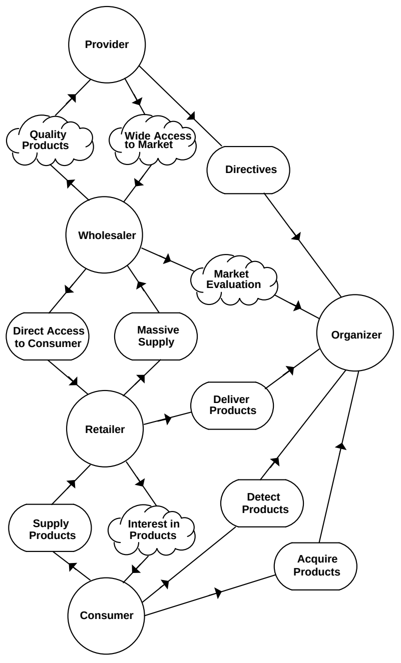

Fig. 8. Vertical Integration

3 Evaluating Architecture

The organizational styles defined in Section 2 can be evaluated and compared using the following software quality attributes identified for multi-agent architectures:

1 - Predictability [24]**.** Agents have a high degree of autonomy in the way that they undertake action and communication in their domains. It can be then difficult to predict individual characteristics as part of determining the behavior of a distributed and open system at large.

2 - Security. Agents are often able to identify their own data sources and they may undertake additional actions based on these sources [24]. Protocols and strategies for verifying authenticity for these data sources by individual agents are an important concern in the evaluation of overall system quality since, in addition to possibly misleading information acquired by agents, there is the danger of hostile external entities spoofing the system to acquire information accorded to trusted domain agents.

3 - Adaptability. Agents may be required to adapt to modifications in their environment. They may include changes to the component's communication protocol or possibly the dynamic introduction of a new kind of 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 ways:

4 - Cooperativity. They must be able to coordinate with other entities to achieve a common purpose.

5 - Competitivity. The success of one agent implies the failure of others.

6 - Availability. Components 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 [3]. Nevertheless, we deal with it as a top-level quality attribute due to its increasing importance in multi-agent system design.

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

	1	2	3	4	5	6	7	8	9
Flat	--	--	-			+	+	++	-
Struct-5	+	+		+	-	+	++	++	++
Pyramid	++	++	+	++	-	+	--	-	
Joint-Vent	+	+	++	+	-	++		+	++
Bid	--	--	++	-	++	-	--	++	
Takeover	++	++	-	++	--	+		+	+
Arm's-Lgth	-	--	+	-	++	--	++	+	
Hierch Ctr			+	+	+	+		+	+
Vert Integr	+	+	-	+	-	+	--	--	--
Coopt	-	-	++	++	+	--	-	--	

Table 1. Correlation Catalogue

8 - Modularity [23] increases efficiency of task execution, reduces communication overhead and usually enables high flexibility. On the other hand, it implies constraints on inter-module communication.

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

Table 1 summarizes the correlation catalogue for the organizational patterns and top-level quality attributes we have considered. Following notations used by the NFR (non functional requirements) framework [3], +, ++, -, --, respectively model partial/positive, sufficient/positive, partial/negative and sufficient/negative contributions.

4 Example

To motivate our organizational styles, we consider an application domain where distributed and open architectures are increasingly important: mobile robots.

The mobile robot example presented in [22] studies notably the layered architecture (Figure 9) implemented in the Terregator and Neptune office delivery robots [20].

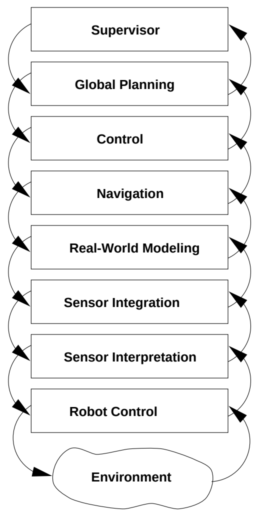

Fig. 9. Classical Mobile Robot Layered Architecture

According to [22] at the lowest level, reside the robot control routines (motors, joints,...). Levels 2 and 3 deal with the input from the real world. They perform sensor interpretation (the analysis of the data from one sensor) and sensor integration (the combined analysis of different sensor inputs). Level 4 is concerned with maintaining the robot's model of the world. Level 5 manages the navigation of the robot. The next two levels, 6 and 7, schedule and plan the robot's actions. Dealing with problems and replanning is also part of the level-7 responsibilities. The top level provides the user interface and overall supervisory functions.

The following software quality attributes are relevant for the robot's architecture [22]: *Cooperativity*, *Predictability*, *Adaptability*, *Integrity*. Let consider, for instance, *Cooperativity* and *Predictability*.

Cooperativity: the robot has to coordinate the actions it undertakes to achieve its designated objective with the reactions forced on it by the environment (e.g., avoid an obstacle). The idealized layered architecture (Figure 9) implemented on some mobile robots does not really fit the actual data and control-flow patterns. The layered architecture style suggests that services and requests are passed between adjacent layers. However, data and information exchange is actually not always straight-forward. Commands and transactions may often need to skip intermediate layers to establish direct communication. A structure-in-5 proposes a more distributed architecture allowing more direct interactions between component.

Another recognized problem is that the layers do not separate the data hierarchy (sensor control, interpreted results, world model) from the control hierarchy (motor control, navigation, scheduling, planning and user-level control). Again the structure-in-5 could better differentiate the data hierarchy - implemented by the operational core, and support components - from the control structure – implemented by the operational core, middle agency and strategic apex as will be described in Figure 10.

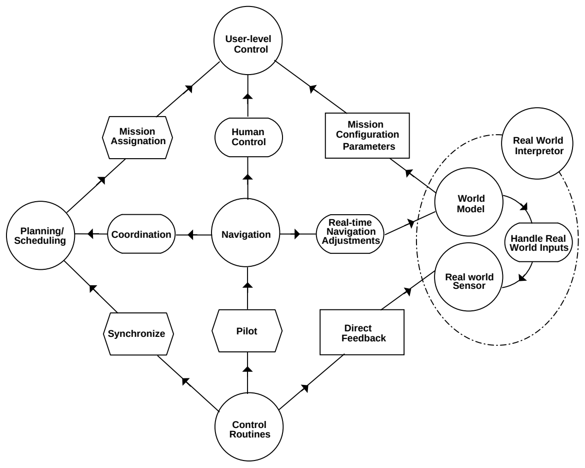

Fig. 10. A Structure-in-5 Mobile Robot Architecture

Adaptability: application development for mobile robots frequently requires customization, experimentation and dynamic reconfiguration. Moreover, changes in tasks may require regular modification. In the layered architecture, the interdependencies between layers prevent the addition of new components or deletion of existing ones. The structure-in-5 style separates independently each typical component of an organizational structure but a joint venture isolating components and allowing autonomous and dynamic manipulation should be a better candidate. Partner components, except the joint manager, can be added or deleted in a more flexible way.

Figure 10 depicts a mobile robot architecture following the structure-in-5 style from Figure 1. The *control routines* component is the *operational core* managing the robot motors, joints, etc. *Planning/Scheduling* is the *coordination* component scheduling and planning the robot's actions. The *real world interpreter* is the *support* component composed of two sub-components: *Real world sensor* accepts the raw input from multiple sensors and integrates it into a coherent interpretation while *World Model* is concerned with maintaining the robot's model of the world and monitoring the environment for landmarks. *Navigation* is the *middle agency* component, the central intermediate module managing the navigation of the robot. Finally, the *user-level control* is the human-oriented *strategic apex* providing the user interface and overall supervisory functions.

Figure 11 formulates the media robot structure-in-5 in Telos. *MobileRobotClass* is a Telos class, instance of the *StructureIn5Metaclass* specified in Figure 2. This aggregation is composed of five exclusive and dependent parts *ControlRoutinesClass, RealWorldInterpreterClass*, *NavigationClass*, *PlanningClass* and *UserLevelControl-Class*, each of them is instance of one metaclass, component of *StructureIn-5MetaClass*.

```
TELL CLASS MobileRobotClass
   IN StructureIn5MetaClass WITH
         attribute
                  name: String
         part, exclusivePart, dependentPart
                  ControlRoutinesClass: OperationalCoreMetaClass
                  RealWorldInterpreter: SupportMetaClass
                  NavigationClass: MiddleAgencyMetaClass
                  PlanningClass: CoordinationMetaClass
                  UserLevelControl: ApexMetaClass
END MobileRobotClass
```

Fig. 11. Mobile Robot Structure-in-5 Architecture in Telos

5 Related Work

Other research work on multi-agent systems offers contributions on using organization concepts such as agent (or agency), group, role, goals, tasks, relationships (or dependencies) to model and design system architectures.

For instance, Aalaadin [6] presents a model based on two level of abstraction. The concrete level includes concepts such as *agent*, *group* and *role* which are used to describe the actual multi-agent system. The methodological level defines all possible roles, valid interactions, and structures of groups and organizations. The model describes an organization in terms of its structure, and independently of the way its agents actually behave. Different types of organizational behavioral requirement patterns have been defined and formalized using concepts such as groups and roles within groups and (inter-group and intra-group) role interactions.

In our work the concepts Aalaadin uses in the concrete level are contained in the concept of actor. An actor can be a single or a composite agent, a position covered by an agent, and a role covered by one or more agents. Unlike ours, Aalaadin's proposal does not include goals in the description of an organization. Moreover, in Aalaadin's work these descriptions include details (e.g., interaction languages and protocols) which we deal with at a later stage of design, typically called *detailed design*.

On a different point of comparison, Aalaadin uses *rules*, *structures* and *patterns* to capture respectively how the organization is expected to work, which kind of structure fits given requirements, and whether reuse of patterns is possible. In our framework, some rules are captured by social dependencies in terms of which one defines the obligations of actors towards other actors. Moreover, other rules can be captured during detailed design instead of earlier phases, i.e., early and late requirements, or architectural design (see [1]).

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 designing MAS architectures as organizational architectures, i.e, at a macro- and micro-level. The proposed styles adopt concepts from organization theory and strategic alliances literature. The paper also includes an evaluation of software qualities that are relevant to these styles. A standard case study (the mobile robot case control) illustrates and compares them with respect to conventional architecture.

Future research directions include formalizing precisely the organizational structures 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 relate them to social or agent patterns (e.g, the broker, matchmaker, embassy, facilitator, …) and lower-level architectural components [21] involving (software) components, ports, connectors, interfaces, libraries and configurations [9]. We are still working on contrasting our structures to conventional styles [22] and patterns [10] proposed in the software engineering literature. To this end, as mentioned in the paper, we are defining algorithms to propagate evidences of satisfaction and denial of each conventional or social structure with respect to a set of non-functional requirements.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*, Reading, Addison-Wesley, 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, pp. 108-123.

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

[4] Coburn, M., *Jack Intelligent Agents: User Guide version 2.0*, AOS Pty Ltd, 2000.

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

[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*, June, 1998.

[7] *The Foundation for Intelligent Physical Agents*, http://www.fipa.org, 2001.

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

[9] A. Fuxman, P. Giorgini, M. Kolp, and J. Mylopoulos. "Information systems as social structures". In *Proc. of the 2nd Int. Conf. on Formal Ontologies for Information Systems* (FOIS'01), Ogunquit, USA, October 2001.

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

[11] D. Garlan and M. Shaw. "An Introduction to Software Architectures", in *Advances in Software Engineering and Knowledge Engineering*, volume I, World Scientific, 1993.

[12] P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia and P. Bresciani. "Agent-Oriented Software Development: A Case Study". In *Proc. of the 13th Int. Conference on Software Engineering & Knowledge Engineering* (SEKE01), Buenos Aires, Argentina, June 2001.

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

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

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

[16] 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*, Ablex, 1988.

[17] 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.

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

[19] Odell, J., Van Dyke Parunak, H. and Bauer, B., "Extending UML for Agents", *Proceedings of the Agent-Oriented Information System Workshop at the 17th National Conference on Artificial Intelligence*, pp. 3-17, Austin, USA, July 2000.

[20] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan. "A modular architecture for office delivery robots". In *Proc. Of the 1st Int. Conf. on Autonomous Agents* (Agents '97), Marina del Rey. CA, Feb 1997, pp.245 - 252.

[21] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik. "Abstractions for software architecture and tools to support them." In *IEEE Transactions on Software Engineering*, 21(4), pp. 314 - 335, 1995.

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

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

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

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

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