Socio-Intentional Architectures for Multi-Agent Systems: the Mobile Robot Control case

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

[1]Department of Information and Communication Technology - University of Trento, via Sommarive, 14, I-38100, Trento, Italy, tel.: 39-0461-88 2052, pgiorgini@science.unitn.it
[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 Computer Science - University of Toronto, 6 King's College Road M5S 3H5, Toronto, Canada, tel.: 1-416-978 5180, jm@cs.toronto.edu

Abstract. This paper proposes architectural styles for multi-agent systems (MAS) which adopt concepts from organization theory and strategic alliances. These styles are socio-intentional in the sense that they consist of actors who have goals to fulfil and social dependencies describing their obligations. They are represented in *i**, a framework designed to model social and intentional primitives and formalized in terms of *Formal Tropos*. Each proposed style is evaluated with respect to a set of agent software qualities, such as predictability, adaptability and openness. The use of the styles is illustrated and contrasted with a software architecture for mobile robots reported in the literature.

1 Introduction

System architectures describe 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 software qualities. Examples of styles are pipes-and-filters, event-based, layered and the like [Gar93]. Examples of software qualities include maintainability, modifiability, portability, etc. [Bas98]. Multi-Agent System (MAS) architectures can be considered as organizations (see e.g., [Fer98, Fox81, Mal88]) 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 socio-intentional 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 further work on the development of a set of architectural styles for multi-agent systems motivated by these theories. This paper builds on earlier work

reported in [Kol01]. The styles are modeled using the strategic dependency model of *i** [Yu95], and they are further specified in *Formal Tropos* [Fux01a]. To illustrate these styles, we use a case study comparing socio-intentional with conventional software architectural styles for mobile robot control software.

This research is being conducted within the context of the Tropos project [Cas01, Gio01]. Tropos adopts ideas from MAS technologies, mostly to define the detailed design and implementation phases, and ideas from requirements engineering, where agents/actors and goals have been used heavily for early requirements analysis [Dar93, Yu95]. In particular, Tropos is founded on Eric Yu's *i** modeling framework which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modeling an application during early requirements analysis. The key premise of the project is that actors and goals can be used as fundamental concepts for analysis and design during *all phases of software development*, not just requirements analysis.

Section 2 presents samples of socio-intentional structures that have been identified from organizational theory and strategic alliances and offers some specifications in *Formal Tropos*. Section 3 introduces the mobile robot control case study, identifies relevant software qualities for mobile robots and reports on earlier work that use conventional architectures. It then applies the socio-intentional structures proposed here and compares these with some conventional architectural solutions with respect to identified qualities. Finally, Section 4 discusses related work, and Section 5 summarizes the results of the paper and points to further work.

2 Socio-Intentional Structures

Organization theory (e.g., [Min92, Sco98]) and strategic alliances (e.g., [Gom96, Seg96,Yos95]) study alternatives to model (business) organizations. An organizational style represents a possible social way to structure the stakeholders – individuals, physical or social systems – of an organization in order to meet its strategic goals and intentions.

The structure of an organization defines the social roles of the various components (actors), their responsibilities for tasks and goals, the way in which the resources are allocated, and the strategies that must be adopted. Moreover, the structure defines how to coordinate the activities of the various actors and how they depend on each other. Social dependencies can involve both actors of the organization and actors of the environment in which the organization is located (e.g., partners, competitors, clients, etc.).

An organizational style offers also a set of design parameters that can be selected and turned in order to influence the division of labor and the coordinating mechanisms, thereby affecting how the organization functions. Design parameters include, among others, tasks assignment, standardization, supervision and control. The organization designer can use these parameters in order to deal with, so called, *situational* or *contingency factors*, namely organizational states or conditions that are associated with the use of certain design parameters. Contingency factors can involve age and size of the organization, the technical system it uses, and various aspects of the environment, such as stability, complexity, diversity, and hostility.

We propose a catalogue of socio-intentional structures adopting (some of) the styles offered in organization theory and strategic alliances for designing multi-agent architectures. In the following we present briefly some of these styles using the strategic dependency model of *i**.

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].

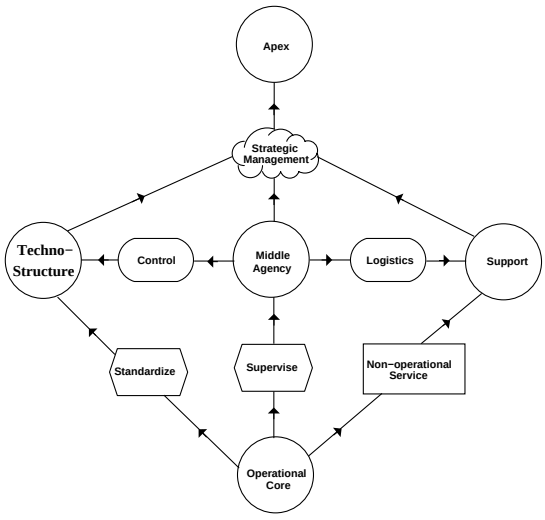

Fig. 1. Structure-in-5

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

The **structure-in-5** (Figure 1) is a typical organizational style. At the base level, the *Operational Core* takes care of the basic tasks — the input, processing, output and direct support procedures — associated with running the organization. At the top

lies the *Apex*, composed of strategic executive actors. Below it, sit the *Technostructure*, *Middle Agency* and *Support* actors, who are in charge of control/standardization, management and logistics procedures, respectively. The *Technostructure* component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the organization 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 flow of operational tasks and procedures.

To specify the structure and formal properties of the style, we use *Formal Tropos* [Fux01a] which offers the primitive concepts of *i** augmented with a rich specification language inspired by KAOS [Dar93]. *Formal Tropos* offers a textual notation for *i** models and allows one to describe dynamic constraints among the different elements of the specification in a first order linear-time temporal logic. Moreover, *Formal Tropos* has a precise semantics which makes Tropos specifications amenable to formal analysis.

For example, in the following, we focus on the Operational Core specification with respect to the performance of basic tasks. The following specification says that each basic task must be performed within a precise time period that depends on the type of the task. For instance, providing raw materials is a task that must be performed before the production process begins.

Entity BasicTask
Attribute constant *taskType: TaskType, resourceNeed: Resource, performed: Boolean, timePeriod: Time, output: OutputType*
Entity Resource
Attribute constant *resourceType:ResourceType*
Actor OperationalCore
Attribute optional resource: Resource
Goal PerformBasicTasks
Mode achieve
Fulfillment definition
$\forall task{:}BasicTask\ (Perform(self,task) \wedge TimePerforming(task) \leq task.time)$

[each basic task in the organization will be performed by the Operational Core within the allotted time period for that type of task]

The **joint venture** style (Figure 2a) is a more decentralized style that involves an agreement between two or more principal partners in order to obtain the benefits derived from operating at a larger scale and reusing the experience and knowledge of the partners. 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 is delegated to a *Joint Management* actor, who coordinates tasks and manages the sharing of knowledge and resources.

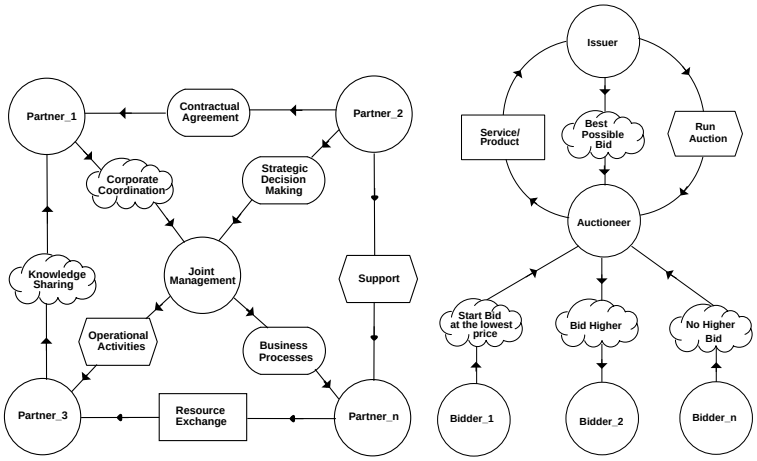

Fig. 2. Joint Venture (a) and Bidding (b)

The **bidding** style (Figure 2b) is founded on competition mechanisms and actors behave as if they were taking part in an auction. The Auctioneer actor runs the whole show. It advertises the auction issued by the auction Issuer, receives bids from bidder actors and ensure communication and feedback with the auction Issuer. The auction Issuer is responsible for issuing the bidding.

The **vertical integration** style (Figure 3a) merges, backward or forward, several actors engaged in achieving or realizing related goals or tasks at *different stages* of a production process. An *Organizer* merges and synchronizes interactions/dependences between participants, who act as intermediaries. Figure 3a 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*.

The **hierarchical contracting** style (Figure 3b) identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority. Coordination here uses mechanisms with arm's-length (i.e., high independence) characteristics involving a variety of negotiators, mediators and observers. These work at different levels and handle conditional clauses, monitor and manage possible contingencies, negotiate and resolve conflicts and finally deliberate and take decisions. Hierarchical relationships, from the executive apex to the arm's-length contractors (top to bottom) restrict autonomy and underlie a cooperative venture between the contracting parties. Such, admittedly complex, contracting arrangements can be used to manage conditions of complexity and uncertainty deployed in high-cost-high-gain (high-risk) applications.

For a more detailed presentation of organizational styles (takeover, hierarchical contracting, bidding, arm's-length, pyramid, flat structure, co-optation, …) we have defined, see [Fux01].

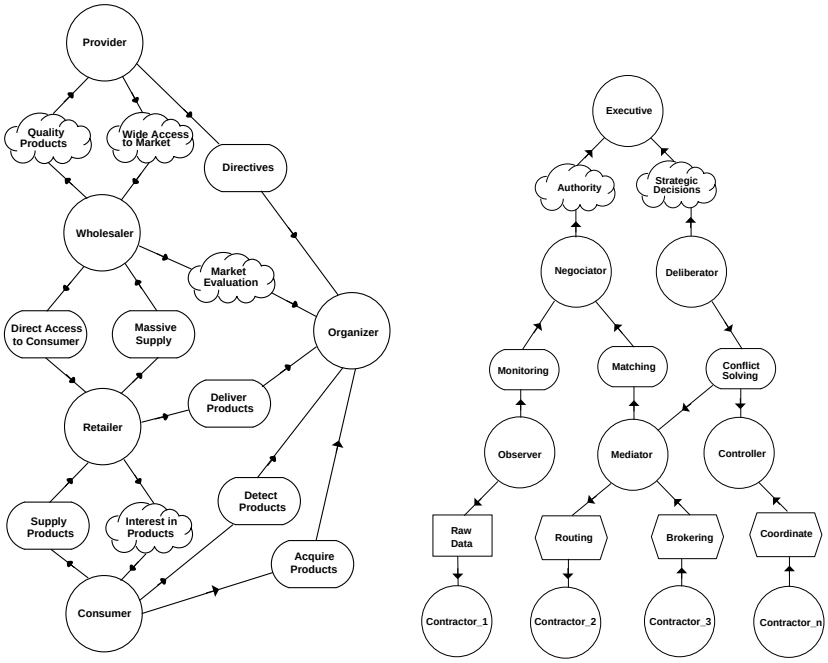

Fig. 3. Vertical Integration (a) and Hierarchical Contracting (b)

3 Architectures for Mobile Robot Control

In this section we apply our socio-intentional architectures to design the architecture of a simple mobile robot control software and compare them to conventional architectural solutions.

The problem focuses on embedded real-time systems. Mobile robot control systems must deal with external sensors and actuators and must respond in time commensurate with the activities of the system in its environment.

Consider the following activities [Sha96] an office delivery mobile robot typically has to accomplish: acquiring the input provided by sensors, controlling the motion of its wheels and other moveable part, planning its future path. In addition, a number of factors complicate the tasks: obstacles may block the robot's path, sensor inputs may be imperfect, the robot may run out of power, mechanical limitations may restrict the accuracy with which the robot moves, the robot may manipulate hazardous materials, unpredictable events may leave little time for responding.

3.1 Agent Software Qualities

With respect to the activities and factors enumerated above, the following agent software qualities can be stated for an office delivery mobile robot's architecture [Sha96].

SQ1 - Coordinativity. Agents must be able to coordinate with other agents to achieve a common purpose or simply their local goals.

A mobile robot has to coordinate the actions it deliberately undertakes to achieve its designated objective (e.g., collect a sample of objects) with the reactions forced on it by the environment (e.g., avoid an obstacle).

SQ2 - Predictability. Agents can have a high degree of autonomy in the way they undertake action and communication in their domains. It can be then difficult to predict individual characteristics as part of determining the behavior of the system at large.

For a mobile robot, never will all the circumstances of the robot's operation be fully predictable. The architecture must provide the framework in which the robot can act even when faced with incomplete or unreliable information (e.g., contradictory sensor readings).

SQ3 – Failability-Tolerance. A failure of one agent does not necessarily imply a failure of the whole system. The system then needs to check the completeness and the accuracy of data, information and transactions. To prevent system failure, different agents can, for instance, implement replicated capabilities.

The architecture must prevent the failure of the robot's operation and its environment. Local problems like reduced power supply, dangerous vapors, or unexpectedly opening doors should not necessarily imply the failure of the mission.

SQ4 - Adaptability. Agents must to adapt to modifications in their environment. They may allow changes to the component's communication protocol, dynamic introduction of a new kind of component previously unknown or manipulations of existing agents.

Application development for mobile robots frequently requires experimentation and reconfiguration. Moreover, changes in robot assignments may require regular modification.

3.2 Conventional Architectures

For sample classical solutions, due to lack of space, we only examine three major conventional architectures - the layered architecture [Sim97], control loops [Loz90] and task trees [Sim92] - that have been implemented on mobile robots.

Layered Architecture. A classical layered architecture is depicted in Figure 4a. 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 level 7 responsibilities. The top level provides the user interface and overall supervisory functions.

Control loop. A controller component initiates the robot actions. Since mobile robots have responsibilities with respect to their operational environment, the controller also monitors the consequences of the robot actions adjusting the future plans based on the return information (Figure 4b).

Task Trees. The architecture is based on hierarchies of tasks. Parent tasks initiate child tasks. For instance the task *Gather Object* initiates the tasks *Go to Position*, *Grab Object*, *Lift Object*, the task *Go to Position* initiates *Move Left* and *Move Forward* and so on. The software designer can define temporal dependencies between pairs of tasks. An example is: "*Grab Object* must complete before *Lift Object* starts." These features permit the specification of selective concurrency.

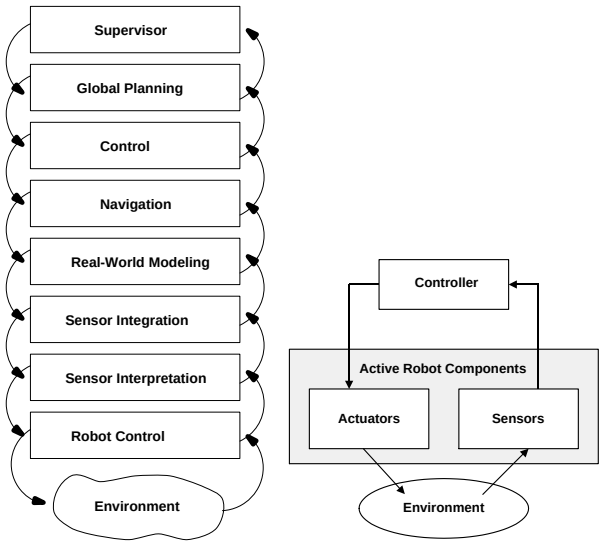

Fig 4. Mobile robot layered [Sim97] (a) and control loop [Loz90] (b) architectures

3.3 Socio-Intentional Architectures

We are currently developing and testing socio-intentional architectures for a miniature office delivery robot (See Figure 5) using the Lego Mindstorms Robotics Invention Systems [Leg02] and the Legolog programming platform based on the Golog Planner [Leg00]. Currently, we are testing two architectures working with abstractions reminiscent of those encountered in the layered architecture: the structure-in-5 and the joint-venture.

Structure-in-5. Figure 6 depicts a structure-in-5 robot architecture in *i**. The *control routines* component is the *operational core* managing the robot motors, joints, etc. *Planning/Scheduling* is the *technostructure* 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.

Fig. 5. Legolog Socio-Intentional Architectures for Lego Mindstorms Robots in Action

Joint Venture. Following the style depicted in Figure 2a, the robot architecture is organized around a central joint manager assuming the overall supervisor/coordinator role for the other agent components: a high level path planner, a module that monitors the environment for landmarks, a low level path planner, a motor controller and a perception subsystem that receives sensors data and interprets it. As said in Section 2, each of these agent components can also interact directly with each other.

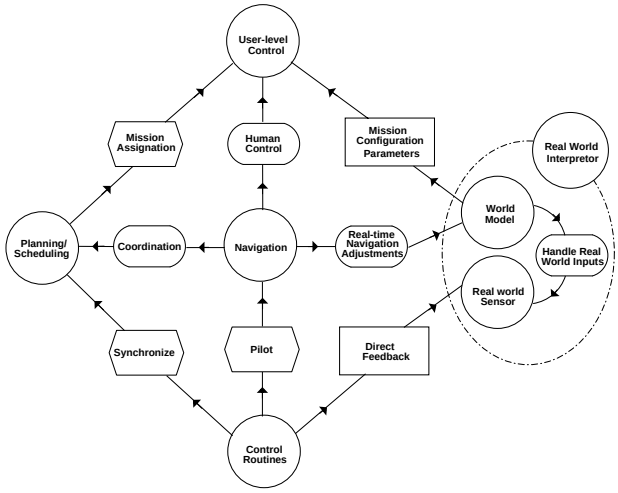

Fig 6. A structure-in-5 mobile robot architecture.

3.4 Evaluation

We evaluate each of the five styles – control loop, layered architecture, task trees, structure-in-5 and joint-venture described in Sections 3.2 and 3.3 with respect to the four agent software quality attributes identified in Section 3.1. Table 1 summarizes the strengths and weaknesses of the five reviewed architectures.

	Loop	Layers	Task Tree	S-in-5	Joint-Vent.
Coordinativity	-	-	+-	++	++
Predictability	+-	+	+-	+	++
Failability-Tol.	+	+-	+	+	+
Adaptability	+-	+-	+	+	+-

Table 1. Strengths and Weaknesses of Robot Architectural Solutions

Coordinativity. The simplicity of the control loop is a drawback when dealing with complex tasks since it gives no leverage for decomposing the software into more precise cooperative agent components.

The layered architecture style suggests that services and requests are passed between adjacent agent layers. However, information exchange is actually not always straight-forward. Commands and transactions may often need to skip intermediate layers to establish direct communication and coordinate behavior.

A task tree permits a clear-cut separation of action and reaction. It also allows incorporation of concurrent agents in its model that can proceed at the same time to. Unfortunately, components have little interaction with each other.

Unlike the previous architectures, the structure-in-5 separates the data (sensor control, interpreted results, world model) from control (motor control, navigation, scheduling, planning and user-level control). The architecture improves coordinativity among components by differentiating both hierarchies – data is implemented by the support component, while control is implemented by the operational core, technostructure, middle agency and strategic apex – as shown in Figure 7.

In the joint venture, each partner component interacts via the joint manager for strategic decisions. Components indicate their interest, and the joint manager returns them such strategic information immediately or mediates the request to some other partner component.

Predictability. The control loop only reduces the unpredictable through iteration. Actions and reactions eliminate possibilities at each turn. Unfortunately, more subtle steps are needed, the architecture offers no framework for delegating them to separate agent components.

In the layered architecture, the existence of abstraction layers addresses the need for managing unpredictability. What is uncertain at the lowest level become clear with the added knowledge in the higher layers.

How task trees address predictability is less clear. If imponderables exist, a tentative task tree can be built, to be adapted by exception handlers when the assumptions it is based on turn out to be erroneous.

Like in the layered architecture, the existence of different abstraction levels in the structure-in-5 addresses the need for managing unpredictability. Besides, contrary to the layered architecture, higher levels are more abstract than lower levels: lower levels only involve resources and task dependencies while higher ones propose intentional (goals and softgoals) relationships.

In the joint-venture, the central position and role of the joint manager is a means for resolving conflicts and prevent unpredictability in the robot's world view and sensor data interpretation.

Failability-Tolerance. In the control loop, it is supported in the sense that its simplicity makes duplication of components and behavior easy and reduces the chance of errors creeping into the system.

In the layered architecture, failability-tolerance could be served, when the robot architect strives *not* do something, by incorporating many checks and balances at different levels into the system. Again the drawback is that control commands and transactions may often need to skip intermediate layers to check the system behavior.

In the task trees, exception, wiretapping and monitoring features can be integrated to take into account the needs for integrity, reliability and completeness of data.

In the structure-in-5, checks and control mechanisms can be integrated at different abstractions levels assuming redundancy from different perspectives. Contrary to the layered architecture, checks and controls are not restricted to adjacent layers. Besides, since the structure-in-5 permits to separate the data and control hierarchies, integrity of these two hierarchies can also be verified independently.

The jointure venture, through its joint manager, proposes a central message server/controller. Like in the task trees, exception mechanism, wiretapping supervising or monitoring can be supported by the joint manager to guarantee non-failability, reliability and completeness.

Adaptability. In the control loop, the robot components are separated from each other and can be replaced or added independently. Unfortunately, precise manipulation has to take place inside the components, at a level detail the architecture does not show.

In the layered architecture, the interdependencies between layers prevent the addition of new components or deletion of existing ones. The fragile relationships between the layers can become more difficult to decipher with change.

Task trees, through the use of implicit invocation, make incremental development and replacement of component straightforward: it is often sufficient to register new components, no existing one feels the impact.

The structure-in-5 separates independently each typical component of the robot architecture isolating them from each other and allowing dynamic manipulation. The structure-in-5 is restricted to no more than 5 major components then, as in the control loop, more refined tuning has to take place inside the components.

In the joint venture, manipulation of partner components can be done easily by registering new components to the joint manager. However, since partners can also communicate directly with each other, existing dependencies should be updated as well. The joint manager cannot be removed due to its central position.

To cope with these quality attributes and select the appropriate structure, more refined analysis and decomposition can be done with frameworks like KAOS [Dar93] or the NFR framework [Chu00]. In the NFR framework, we go through a means-ends refining the identified quality attributes to sub-attributes that are more precise and evaluate social structures against them, as shown partially in Figure 5.

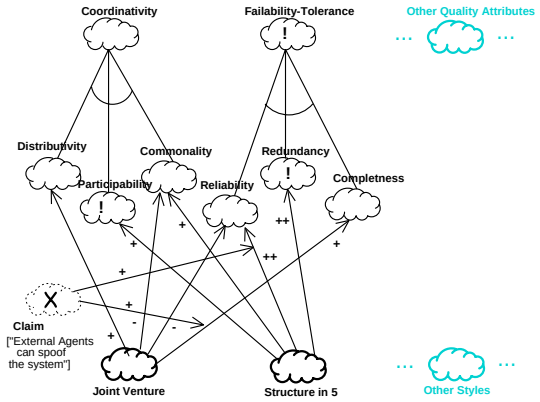

Fig. 7. Partial Evaluation for Organizational Styles

The evaluation results in contribution relationships from the social structures to the quality attributes, labeled "+", "++", "-", "--" that mean respectively *partially satisfied*, *satisfied*, *partially denied* and *denied*. 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. Exclamation marks are used to mark priority attributes while a check-mark "✓" indicates an accepted attribute and a cross "✗" labels a denied attribute.

Relationships types (AND, OR, ++, +, -, and --) between quality attributes are formalized to offer a tractable proof procedure. Attributes can be labeled as Satisfied (S), Partially Satisfied (PS), Denied (D), or Partially Denied (PD), and are not required to be logically exclusive since they may be contradictory. Table 2 shows propagation rules for ++, +, -, and -- relationships with respect to satisfiability (S) and partial satisfiability (PS). A dual table can be given for the deniability and the partial deniability.

	++	+	-	--
S	S	PS	PD	D
PS	PS	PS	PD	PD

Table 2. Propagation rules for S and PS

Under the assumption that D < PD < PS < S, we use min-value and max-value functions respectively for AND and OR relationships.

We are currently working on two different approaches. The first is a logic approach in which S, PS, PD, and D are four truth values and each node can assume the values S (or PS) and D (or PD) (conflicts are allowed; e.g., a node can be satisfied and partially denied). For each type of relationship the propagation rules are defined by a

set of axioms. The second approach uses a numerical interval to define the degree of satisfiability and deniability of a node. Here, we are working in two different directions: one is based on the probability theory and the other on the Dempster-Shafer theory (see, e.g, [Par94]).

The layered architecture gives precise indications as to the components expected in a robot. The other two classical architectures (control loop and task trees) define no functional components and concentrate on the dynamics. The organizational styles (Structure-in-5 and Joint Venture) focus on how to organize components expected in a robot but also on the intentional and social dependencies governing these components. Exhaustive evaluations are difficult to be established at that point. But, considering preliminary results we can deduce in Table 1, from the discussion in the present section, we can argue that the Structure-in-5 and the Joint-Venture, since they are patterns governed by organizational characteristics, fit better systems and applications that need open and cooperative components like the mobile robot example.

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 [Fer98] 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 [Bas98]).

6 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 socio-intentional 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 architectures.

Future research directions include formalizing precisely the socio-intentional 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 involving (software) components, ports, connectors, interfaces, libraries and configurations [Fux01, Kol01]. We are still working on contrasting our structures to conventional styles [Sha96] and patterns [Gam95] 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

[Bas98] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*, Reading, Addison-Wesley, 1998.

[Cas01] 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.

[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.

[Fer98] J. Ferber and O. Gutknecht."A meta-model for the analysis and design of organizations in multi-agent systems". In *Proc. of the 3rd Int. Conf. on Multi-Agent Systems* (ICMAS'98), June, 1998.

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

[Fux01] 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.

[Fux01a] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. "Model Checking Early Requirements Specification in Tropos". In *Proc. of the 5th Int. Symposium on Requirements Engineering* (RE'01), Toronto, Canada, Aug. 2001.

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

[Gar93] D. Garlan and M. Shaw. "An Introduction to Software Architectures", in *Advances in Software Engineering and Knowledge Engineering*, volume I, World Scientific, 1993.

[Gio01] 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.

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

[Kol01] M. Kolp, P. Giorgini, and J. Mylopoulos. "An Organizational Perspective on Multi-agent Architectures". In *Proc. of the Eighth International Workshop on Agent Theories, architectures, and languages* (ATAL'01), Seattle, USA, August 2001.

[Leg00] Legolog. At http:// www.cs.toronto.edu/cogrobo/Legolog, 2000.

[Leg02] Lego Mindstorms Robotics Invention System. At http://mindstorms.lego.com, 2002.

[Loz90] T. Lozano-Perez. Preface to *Autonomous Robot Vehicles*. Cox, L.J. and Wilfong G.T., eds, Springer Verlag, 1990.

[Mal88] 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.

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

[Par94]S. Parsons, "Some qualitative approaches to applying the Dempster-Shafer theory". In *Information and Decision technologies*, 19 (1994), pp 321- 337.

[Sco98] W. Richard Scott. *Organizations: rational, natural, and open systems*, Prentice Hall, 1998.

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

[Sh96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Sim92] R. Simmons. "Concurrent Planning and Execution for Autonomous Robots". In *IEEE Control Systems*, n• 1, 1992. pp. 49-56.

[Sim98] 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.

[Yos95] M.Y. Yoshino and U. Srinivasa Rangan. *Strategic alliances : an entrepreneurial approach to globalization*, 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.