Multi-Agent and Software Architectures: A Comparative Case Study

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

[1] Department of Information and Communication Technology - University of Trento, Via Sommarie 14, I-38100, Trento, Italy, tel.: 39-0461-882052, pgiorgini@science.unitn.it
[2] IAG School of Management- Information Systems 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 M5H 3S5, Toronto, Canada, tel.: 1-416-978 5180, jm@cs.toronto.edu

Abstract. We propose a collection of architectural styles for multi-agent systems motivated by organizational theory and enterprise organization structures. One of the styles is discussed in detail and part of it is formalized using the Formal Tropos specification language. In addition, we conduct a comparative study of organizational and conventional software architectures using a mobile robot control example from the Software Engineering literature.

1 Introduction

We are interested in developing a suitable set of architectural styles for multi-agent systems. Since the fundamental concepts of multi-agent systems are intentional and organizational, rather than implementation-oriented, we turn to organizational theories which study structures as *societies* that emerge from a *design* process.

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 and strategic alliances. This paper builds on earlier work reported in [5] by offering some formalization of one of the proposed styles, also a case study comparing organizational with conventional software architectural styles for mobile robot control software.

This research is conducted within the context of *Tropos* [1,8], an agent-oriented software development methodology which is founded on the concepts of *actor* and *goal,* adopted from the *i** [12] modeling framework. Tropos describes in terms of these concepts the organizational environment within which a system will eventually operate, as well as the system itself.

The rest of the paper is organized as follows. Section 2 presents samples of organizational styles that have been identified from organizational theory literature. Section 3 focuses on one of these styles, the structure-in-5, and offers a formalization using the Formal Tropos language. Section 4 presents the mobile robot control case study, identifies relevant software qualities for mobile robots and reports on earlier work that uses conventional architectures. It then applies the organizational styles proposed here and compares these with some conventional architectures with respect to identified qualities. Finally Section 5 summarizes the results of the paper.

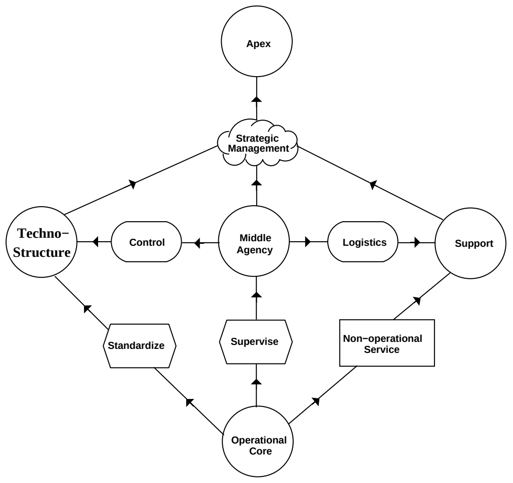
2 Organizational Styles

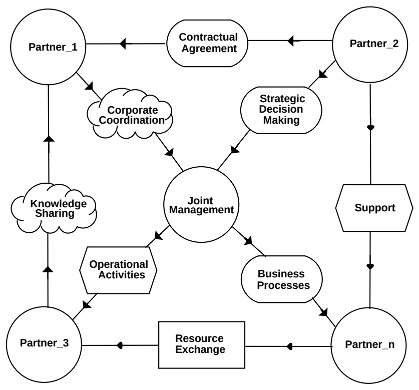
Organizational theory and strategic alliances (e.g., [9]) study alternative styles for (business) organizations. These styles are used to model the coordination of business stakeholders -- individuals, physical or social systems -- to achieve common goals. Each organizational style represents a possible way to structure an organization in order to meet its strategic objectives.

The structure of an organization defines the roles of various intentional components (actors), their responsibilities defined in terms of tasks and goals, and their resources. Moreover, an organizational structure defines how to coordinate the activities of various actors and how they depend on each other.

We propose a macro level catalogue of styles adopting (some of) the abstractions offered by organizational theory for designing multi-agent architectures. In the following we present briefly two of them using *i**. For other styles, see [5].

An *i** strategic dependency model [12] is a graph, where each node represents an actor (an agent, position, or role) and each link between two actors represents a social dependency. Such a dependency can represent the fact that one actor depends on another for a goal to be fulfilled, a task to be performed, or a resource to be made available. The depending actor is called 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 between goals, which are well defined, and softgoals, which do not have a formal definition and are amenable to a different (more qualitative) kind of analysis [2].





Figures 1 and 2. Structure-in-5 and Joint Venture

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 basic tasks — the input, processing, output and direct support procedures — associated with running the organization. At the top lies the *Apex*, composed of executive actors. Below it, sit the *Technostructure*, *Middle Agency* and *Support* actors, who are in charge of control/standardization, management and logistics, 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 basic flow of operational tasks and procedures.

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

3 Structure-in-5

In this section we describe in more detail the structure-in-5 style. To specify the structure and formal properties of the style, we use *Formal Tropos* [4] which offers the primitive concepts of *i** augmented with a rich specification language inspired by KAOS [3]. 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.

Minztberg proposes a basic structure for organizations (for us, organizational styles) based on fives subunits [7], hence its name *structure-in-5* (Figure 1). This decomposition allows one to apply alternative coordination mechanisms (such as mutual adjustment, direct supervision, standardization of skills, outputs and work processes) and design parameters (such as job specialization, behavior formalization, decentralization, unit size, and unit grouping) in order to analyze the different behaviors of the organization.

Basically, this structure defines a hierarchy of roles inside the organization, the responsibilities associated with each subunits, and inter-dependencies among them. Figure 3 shows a more detailed *i** strategic dependency model for this style.

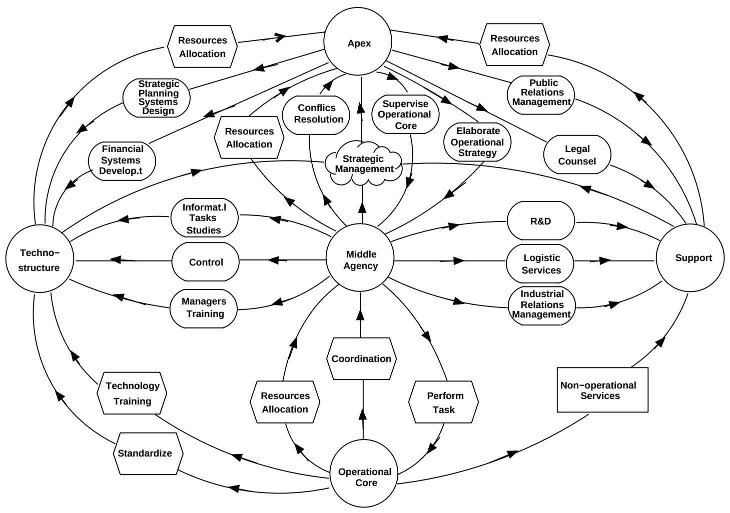


Figure 3. Structure-in-5 in detail

At the base level one finds the Operational Core where the basic tasks and operations are carried out. Basic tasks include securing inputs for production, also transforming these into outputs. For example, in a manufacturing firm, the purchasing department buys raw materials, while the production department transforms these to products. In addition, outputs need to be distributed and support functions need to be performed (e.g., production machine maintenance, inventory control and the like.) In the following, we focus on the Operational Core specification with respect to the performance of basic tasks. The specification below states 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]

At the middle level we have three main actors: the Middle Agency, Technostructure, and Support.

The Middle Agency is composed of a chain of middle-line managers with formal authority that join the Strategic Apex to the Operational Core. The managers in the chain are responsible for supervision and coordination of the Operational Core activities, the allocation of the resources to lower levels, and the formulation of tactics consistent with the strategies of the overall organization. For instance, when the strategic apex of the Postal Service decides to realize a project for e-Postal Services, each regional manager, and in turn, each district manager must elaborate the plan as it applies to its geographical area.

In general, the Middle Agency performs all the managerial tasks of the chief executive, but in the context of managing a particular unit. A Middle Agency actor must lead the members of its unit, develop a network of liaison contacts, monitor the environment and its unit's activities, allocate tasks and resources within, negotiate with outsiders, initiate strategic change, and handle exceptions and conflicts. In the following we present a part of the specification for the *PerformTask* dependency between the Middle Agency and Operational Core that concerns the assignment of a task to the Operational Core.

Dependency PerformTask
 Type task
 Mode achieve
 Depender MiddleAgency
 Depndee OperationalCore
 Attribute constant task: BasicTask
 Creation
  condition *¬task.performed*
  trigger *JustCreated(task)*
 Fulfillment
  condition for depender *task.performed*

[a PerformTask dependency is created when there is a task that has not been performed, and the dependency is fulfilled when the task is performed]

The Technostructure comprises analysts outside the operating work flow, who affect the work of others. They define certain forms of standardization that reduce direct supervision: work process, output, and skills standardization. They are also responsible for training managers of the Middle Agency and operators of the Operational Core. At middle levels, analysts carry out operations research studies of informational tasks, and they design on behalf of the Strategic Apex strategic planning systems and financial systems to control and monitor strategic goals. In the following, we present the specification of the Technostructure with respect to an output standardization goal. In particular, we specify that for each basic task that the Operational Core has to perform, the Technostructure provides a specific output standard to which the task output must conformed to. The output standard depends on the task type and required output properties, such as length, weight, and strength for a machined part, or text length, fonts and document structure for a document.

Actor Technostructure
 Goal StandardizeBasicTasks
  Mode achieve
  Fulfillment definition *∀task:BasicTask (Standardize(task.output))*

[for each basic task in the organization, the Technostructure will standardize the output]

Entity Standard
  Attribute constant *output:OutputType, parameterers : Parameters*

Dependency Standardize
 Type task

```
Mode achieve
Depender OperationalCore
Dependee Technostructure
Attribute constant  task :BasicTask
Creation
    condition ¬∃standard: Standard (standard.output=task.output)
    trigger      JustCreated(task)
Fulfillment
      condition for  depender
                    ∃ standard: Standard (standard.output=task.output)
```

[the Standardize dependency is created when there is no standard for a newly created task, and it is fulfilled when the standard has been created]

The Support is composed of units which specialize in supporting the organization with different services outside its operating work flow. This improves control within the organization and reduces the uncertainty of having to buy services in the open market. The units are self-contained mini-organizations and can support various levels of the structure-in-5 hierarchy: public relations management and legal counsel for the Apex, industrial relations management, logistics, and R&D for the Middle Agency; no-operational services (e.g., cafeteria and mail-room) for the Operational Core.

At the top lies the Apex composed of strategic executive actors responsible for ensuring that the organization serves its mission in an effective way. Their major goals include direct supervision (e.g., allocate resources to the middle level, resolve conflicts within the Middle Agency, and monitor performances), and management of the relations with the environment (e.g., inform influential external actors of organizational activities, develop high-level contact, and negotiating major agreements). They also develop organizational strategies consistent with the interpretation of the environment.

Figure 4 details the Technostructure actor in terms of sub-actors. These include Financial Analysts who develop financial systems for the Apex, also Management and Technology Instructors who train the Middle Agency and Operational Core actors respectively. In addition, there are Technology Analysts that standardize the technology used by the operators and support them in their activities. Work-Study analysts control work process standardization for the Operational Core, while Planning/Control analysts design strategic planning systems for the Apex, control the outputs standardization, and perform quality control for the Middle Agency. Finally, Personnel analysts control skills standardization, and Operations Research analysts carry out operations research studies of informational tasks for the Middle Agency.

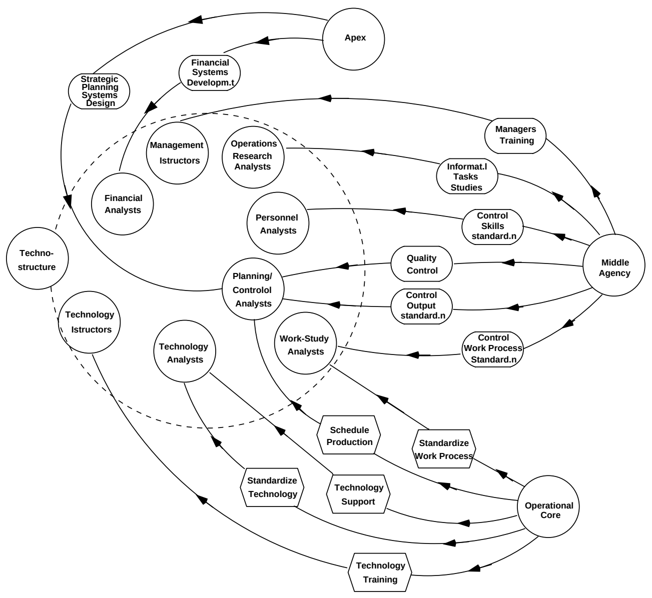


Figure 4. The Technostructure actor

4 A Case Study: Architectures for mobile robots

Mobile robot control systems must deal with external sensors and actuators. They must respond in time commensurate with the activities of the system in its environment.

Consider the following activities [10] 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.

4.1 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 [10].

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 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.

4.2 Classical Styles

For sample classical solutions, due to lack of space, we only examine three major conventional architectures - the layered architecture [10], control loops [11] and task trees [6] - that have been implemented on mobile robots.

Layered Architecture. According to [10], a classical layered architecture can be viewed as a structure composed of 7 layers. 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.

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.

4.3 Organizational styles

We are developing organizational architectures for a miniature office delivery robot using the Lego Mindstorms Robotics Invention Systems. 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.

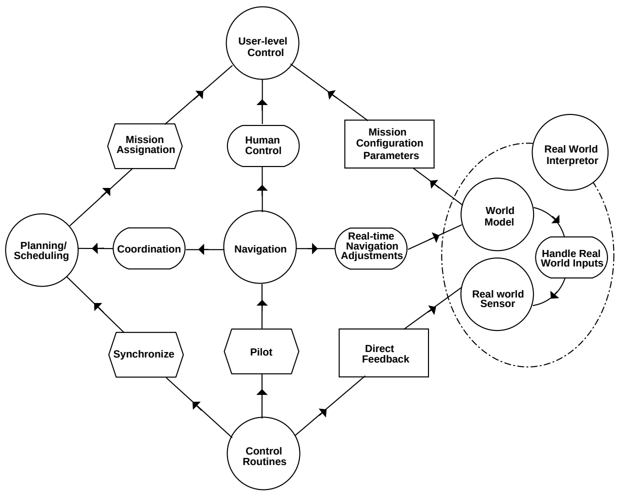


Figure 5. A structure-in-5 mobile robot architecture.

Structure-in-5. Figure 5 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.

Joint Venture. 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.

4.4 Evaluation

In this section, we evaluate each of the five styles – control loop, layered architecture, task trees, structure-in-5 and joint-venture described previously.

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. 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 5.

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 reduces the unpredictable only through iteration. Actions and reactions eliminate possibilities at each turn. Unfortunately, if 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.

The existence of different abstraction levels in the structure-in-5 addresses the need for managing unpredictability. 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 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 detail level 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.

Table 1 summarizes the strengths and weaknesses of the five reviewed architectures.

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.

	Loop	Layers	Task Tree	S-in-5	Joint-Venture
Coordinativity	-	-	+-	++	++
Predictability	+-	+	+-	+	++
Failability-Tolerance	+	+-	+	+	+
Adaptability	+-	+-	+	+	+-

Table 1: Strengths and Weaknesses of Robot Architectures

5 Conclusion

We are working towards a collection of architectural styles for multi-agent systems. In this paper we presented in detail one of the organizational styles, the structure-in-5, and conducted a comparative study of some organizational styles and conventional software architectures on a standard case study (the mobile robot control) selected from the Software Engineering literature.

Considering preliminary results established in the paper we can argue that organizational patterns fit better software and applications that need dynamic manipulation and coordination of components since they are driven by organizational characteristics.

We are currently working on formalizing other organizational styles, also applying them to more examples from the literature, for software as well as organizational structures.

6 References

1. Castro, J., Kolp, M., and Mylopoulos, J. "A Requirements-Driven Development Methodology". In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering, CAiSE'01*, Interlaken, Switzerland, June 2001.
2. Chung, L. K., Nixon, B. A., Yu, E. and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.
3. Dardenne, A., van Lamsweerde, A. and Fickas, S. "Goal–directed Requirements Acquisition", *Science of Computer Programming*, 20, 1993, pp. 3-50.
4. Fuxman, A., Pistore M., Mylopoulos, J., and Traverso, P. "Model Checking Early Requirements Specification in Tropos". In *Proc. of the 5th Int. Symposium on Requirements Engineering, RE'01*, Toronto, Canada, Aug. 2001.
5. Kolp, M., Giorgini P., and Mylopoulos J. "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 1-3, 2001.
6. Lozano-Perez, T., Preface to Autonomous Robot Vehicles. *Cox, L.J. and Wilfong G.T., eds, Springer Verlag*, 1990.
7. Mintzberg, H. Structure in Fives: Designing Effective Organizations, *Prentice-Hall*, 1992.
8. Perini. A, Bresciani, P., Giunchiglia, F., Giorgini, P., Mylopoulos, J., A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proc. Of the 5th International Conference on Autonomous Agents*, Montreal CA, May 2001, ACM.
9. Scott, W. R. Organizations: Rational, Natural, and Open Systems, *Prentice Hall*, 1998.
10. Shaw, M., and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, Upper Saddle River, N.J., *Prentice Hall*, 1996.
11. Simmons, R., Goodwin, R., Haigh, K., Koenig, S., and O'Sullivan, J. "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.
12. Yu E. Modelling Strategic Relationships for Process Reengineering, *Ph.D. thesis, Department of Computer Science, University of Toronto*, Canada, 1995.