Auton Agent Multi-Agent Sys (2006) 13:3–25
DOI 10.1007/s10458-006-5717-6

Multi-agent architectures as organizational structures

Manuel Kolp · Paolo Giorgini · John Mylopoulos

Published online: 25 February 2006

Abstract A Multi-Agent System (hereafter MAS) is an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real world organizations as an metaphor, this paper proposes architectural styles for MAS which adopt concepts from organizational theories. The styles are modeled in i*/Tropos, using the notions of actor, goal and actor dependency and are intended to capture needs/wants, delegations and obligations. The proposed architectural styles are evaluated with respect to a set of software quality attributes, such as predictability and adaptability. In addition, we report on a comparative study of organizational and conventional software architectures using a mobile robot control example from the Software Engineering literature. The research reported here was conducted within the scope of the Tropos project, whose objective is to develop a comprehensive agent-oriented software development methodology.

Keywords Multi-agent architectures · Organizational styles · Architectural design

1. Introduction

Software architectures describe a software system at a macroscopic level in terms of a manageable number of subsystems, components and modules inter-related through data and control dependencies.

M. Kolp (✉)
Information Systems Research Unit, School of Management, University of Louvain-1, Place des doyens, B-1348, Louvain-La-Neuve, Belgium
e-mail: kolp@isys.ucl.ac.be

P. Giorgini
Department of Information and Communication Technology, University of Trento Via Sommarive, 14, I-38050 Povo, Trento, Italy
e-mail:paolo.giorgini@dit.unitn.it

J. Mylopoulos
Department of Computer Science, University of Toronto-6, King's College Road, M5S 3H5 Toronto, Canada
e-mail:jm@cs.toronto.edu

System architectural design has been the focus of considerable research during the last fifteen years and has produced well-established architectural styles and frameworks for evaluating their effectiveness with respect to particular software qualities. Examples of styles are pipes-and-filters, event-based, layered, control loops and the like [52]. Examples of software qualities include maintainability, modifiability, portability and the like [1]. We are interested in developing a suitable set of architectural styles for multi-agent software systems.

In this paper, we consider that the fundamental concepts of a Multi-Agent System (MAS) are intentional and social, rather than implementation-oriented. We turn to theories which study social structures for motivation and insights since after all, software agents by virtue of their intentional capabilities (e.g., ability to plan and negotiate) are much more than mere software components. Moreover, MAS attempt to emulate or are inspired by our limited understanding of how humans plan and negotiate. In addition, it is precisely these capabilities that social structures take advantage of to enhance flexibility, extensibility, robustness and reliability of the overall system. Finally, social structures have been used in practice for much longer than software architectures. Adopting those among them that have proven most useful makes sense in any attempt to revise our conceptualization of software into one where the basic components are endowed with intentional and social traits.

But, what kind of social theory should we turn to? There are theories that study group psychology, communities (virtual or otherwise) and social networks. Such theories study social structure as an emergent property of a social context. Instead, we are interested in social structures that result from a design process. For this, we turn for guidance to organizational theories, namely *Organization Theory* that describe the internal structure and design of an organization and *Strategic Alliances* that model the strategic cooperation of independent organizational actors that pursue shared goals.

In this paper, we propose a set of architectural styles for MAS, and give an in-depth account of two of them – the structure-in-5 and the joint venture – for multi-agent using the strategic dependency model of *i** [63]. We then analyze these styles with respect to a number of qualities, such as predictability and adaptability. In order to compare our proposal with conventional architectural styles, we also report on the results of a comparative study of organizational and conventional software architectures using a mobile robot control example from the Software Engineering literature.

This research is being conducted within the context of *Tropos* [4,48], a project developing a requirements-driven methodology for software systems. The *Tropos* methodology adopts and integrates ideas from MAS and Requirements Engineering, where agents/actors and goals have been used heavily for early requirements analysis [9,63]. In particular, the *Tropos* project adopts Eric Yu's *i** model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modeling an application during early requirements analysis.

The key assumption which distinguishes *Tropos* from other software development methodologies is that actors and goals can be used as fundamental concepts during *all phases of software development*, not just requirements analysis. Using the same concepts and notations to match requirements analysis with system architecture reduces the semantic gap between the organizational environment and the architecture of the future system. It also leads to a streamlined development process founded on a small set of concepts. More details about Tropos can be found in [4].

The present work integrates and extends research in progress about social abstractions for the Tropos methodology. In [35], we have detailed a social ontology for Tropos that treats information systems as social structures throughout the development life cycle. In [36], we

describe how to use this social ontology for early requirements analysis, where the concern is to model the organizational setting (e.g., an enterprise, a corporate alliance, . . .) for a system-to-be. In the present paper, which is an extended and revised version of [23,34], we emphasize the use of organizational styles based on organization theory and strategic alliances to design architectures.

The interested reader can find detailed information on how the organizational styles have actually been applied to build real e-business applications in [8,16,17,47]. To develop such systems, the proposed architectural styles need to be decomposed during detailed design. Moreover, the behavior of each identified architectural component needs to be defined in further detail, in terms of smaller components using social design patterns [13]. These are generic design structures that define how (a small number) of agents are interacting together in order to fulfill their obligations. This decomposition process – from architectural design (organizational styles) to detailed design (social patterns)– is defined in [33,47], and can guide the modularization of a complex macro-system into a set of interacting micro-structures.

The rest of the paper is structured as follows. Section 2 introduces the styles identified in organization and strategic alliance theories. Section 3 details two of them – the structure-in-5 and the joint venture - and illustrates them using real-world examples of organizations. These two styles are modeled in terms of social and intentional concepts using the *i** framework. Section 4 proposes a set of software quality attributes for these styles in terms of which one can evaluate architectural alternatives. Section 5 presents fragments[1] adopted from the Software Engineering literature to illustrates the use of our styles and compares them to conventional ones. Finally Section 6 discusses related research, while Section 7 summarizes the contributions and outlines future work.

2. Structuring organizations

Organizational structures are primarily studied by *Organization Theory* (e.g., [44,45,50]), where the aim is to understand the structure and design of an organization; also by theories on *Strategic Alliances* (e.g.,[15,25,51,62]), that model the strategic collaborations of independent organizational stakeholders who have agreed to pursue a set of shared business objectives. Both disciplines aim to identify and study organizational patterns that describe a system at a macroscopic level in terms of a manageable number of subsystems, components and modules inter-related through dependencies.

In this paper, we are interested in identifying, formalizing and using for MAS design well-understood and precisely defined patterns from organizational theories. Our purpose is not to categorize them exhaustively, nor to study them from a managerial perspective.

2.1. Organization theory

"An organization is a consciously coordinated social entity, with a relatively identifiable boundary, that functions on a relatively continuous basis to achieve a common goal or a set of goals" [45]. Organization theory is the discipline that studies both structure and design for such social entities. Structure deals with the descriptive aspects while design refers to the prescriptive aspects of a social entity. Organization theory studies how real-world organizations are actually structured, offers suggestions on how new ones can be planned, and how

[1] Although we have worked out other e-business case studies using the styles described in this paper (see, for example, [4,12,16,48]), we decided to use a simpler and more pedagogical example here for understandability purposes.

old ones can change to improve effectiveness. To this end, since Adam Smith, schools of organization theory have proposed patterns to capture and formalize recurring organizational structures and behaviors.

In the following, we briefly present some of the main organizational styles identified in Organization Theory. A more exhaustive list is proposed in [18]. For lack of space, only the structure-in-5 is presented in detail in Section 3.

The Structure-in-5. According to this style, an organization consists of five sub-structures, as proposed by Mintzberg [44]. At the base level sits the *Operational Core* which carries out the basic tasks and procedures directly linked to the production of products and services: acquisition of inputs, transformation of inputs into outputs, distribution of outputs. At the top lies the *Strategic Apex* which makes executive decisions ensuring that the organization fulfils its mission in an effective way and defines the overall strategy whereby the organization operates within its environment. The *Middle Line* establishes a hierarchy of authority between the Strategic Apex and the Operational Core. It consists of managers responsible for supervising and coordinating the activities of the Operational Core. The *Technostructure* and the *Support* are separated from the main line of authority and influence the operating core only indirectly. The Technostructure serves the organization by making the work of others more effective, typically by standardizing work processes, outputs, and skills. It is also in charge of applying analytical procedures to adapt the organization to its operational environment. The Support provides specialized services, at various levels of the hierarchy, outside the basic operating work flow (e.g., legal counsel, R&D, payroll, cafeteria). We describe and model examples of structures-in-5 in Section 3.

The pyramid style is a well-known hierarchical authority structure [44]. Actors at lower levels depend on those at higher ones. A critical mechanism for this style is the direct supervision from the Apex. Managers and supervisors at intermediate levels only route strategic decisions and authority from the Apex to the operating (lowest) level. They can coordinate behaviors or make tactical decisions on their own, but only at a local level.

The chain of values merges, backward or forward, several actors engaged in achieving or realizing related goals or tasks at different stages of a supply or production process [50]. Participants who act as intermediaries, add value at each step of the chain. For instance, for the domain of goods distribution, providers are expected to supply quality products, wholesalers are responsible for ensuring their massive exposure, while retailers take care of the direct delivery to the consumers.

The matrix proposes a multiple command structure: vertical and horizontal channels of information and authority operate simultaneously [50]. The principle of unity of command is set aside, and competing bases of authority are allowed to jointly govern the work flow. The vertical lines are typically those of functional departments that operate as "home bases" for all participants. On the other hand, horizontal lines represents project groups or geographical arenas where managers combine and coordinate the services of the functional specialists around particular projects or areas.

The bidding style involves competition mechanisms where actors behave as if they were taking part in an auction [45]. An auctioneer actor runs the show, advertises the auction issued by the auction issuer, receives bids from bidder actors and ensures communication and feedback with the auction issuer.

2.2. Strategic alliances

A strategic alliance links specific facets of two or more organizations. At its core, this structure is a trading partnership that enhances the effectiveness of the competitive strategies of participant organizations. This is accomplished by providing for the mutually beneficial trade of technologies, skills, or products based upon them. An alliance can take a variety of forms, ranging from arm's-length contracts to joint ventures, from multinational corporations to university spin-offs, from franchises to equity arrangements. Varied interpretations of the term exist, but a strategic alliance can be defined as possessing simultaneously the following three defining characteristics:

- Two or more organizations that unite to pursue a set of agreed upon goals, but remain independent subsequent to the formation of the alliance.
- Partner organizations share the benefits of the alliances and have control over the performance of assigned tasks.
- Partner organizations contribute on a continuing basis in one or more key strategic areas, e.g., technology, products, and/or services.

In the following, we briefly present some of the main organizational styles identified in Strategic Alliances. A more exhaustive list is proposed in [18]. For lack of space, only the joint venture style is studied in detail in Section 3.

The joint venture style involves agreement between two or more intra-industry partners to obtain the benefits of scale, partial investment and lower maintenance costs [15]. A specific joint management actor coordinates tasks and manages the sharing of resources between partner actors. Each partner can manage and control its own on a local dimension and interacts directly with other partners to exchange resources, such as data and knowledge. However, the strategic operation and coordination of such an organization, and its actors on a global dimension, are only ensured by the joint management actor where the original actors possess equity participation. We describe and model examples of joint ventures in Section 3.

The arm's-length style implies agreements between independent and competing actors [51]. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise common goals. No authority is lost, or delegated from one collaborator to another.

The hierarchical contracting style identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority [62]. Coordination mechanisms developed for arm's-length (independent) relationships involve a variety of negotiators, mediators and observers. These operate at different levels and handle conditional clauses, monitor and manage possible contingencies, negotiate and resolve conflicts, and finally deliberate and make decisions. Hierarchical relationships – from the executive apex to arm's-length contractors – restrict autonomy and serve as a basis for a cooperative venture between the partners.

The co-optation style incorporates partner representatives into the decision-making or advisory structure and behavior of a newly-founded organization [25]. By co-opting their representatives, organizations are trading confidentiality and authority for resources, knowledge assets and support. The newly-founded organization has to come to terms with the contractors

for what is being done on its behalf; and each co-opted actor has to reconcile and adjust its own views with the policies of the newly-founded organization.

3. Modeling organizational styles

We define an organizational style as a metaclass of organizational structures offering a set of design parameters to coordinate the assignment of organizational objectives and processes, thereby affecting how an organization functions. Design parameters include, among others, goal and task assignments, standardization, supervision and control dependencies, also strategy definitions. This section offers more detail about two of the organizational styles presented in Section 2: the structure-in-5 and the joint-venture. Specifically, for each style we give examples and then propose a meta-model.

Both the examples and the meta-model use the *i** modeling framework, originally proposed for early requirements analysis [63]. *i** offers goal- and actor-based notions such as *actor, agent, role, position, goal, softgoal, task, resource, belief* and different types of social *dependencies* between actors. A strategic dependency model captures the network of social dependencies among actors in terms of a graph, where each circle represents an *actor* and each link between two actors indicates that one actor depends on the other for something. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely (in the sense that it is subjective, and fulfillment may be partial); *task* dependencies are used in situations where the dependee is required to perform a given activity; and *resource* dependencies require the dependee to provide a resource to the depender.

3.1. Structure-in-5

To detail and specify the structure-in-5 as an organizational style, this section presents two case studies: Agate Ltd [2] and GMT [24]. We then propose a meta-model for the style.

Agate. Agate Ltd is an advertising agency located in Birmingham, UK, employing about fifty staff, as detailed in Table 1 [2].

The *Direction* – four directors responsible for the main aspects of Agate's *Global Strategy* (advertising campaigns, creative activities, administration, and finances) – forms the *Strategic Apex*. The *Middle Line,* composed of the *Campaigns Management* staff, is in charge of *finding* and *coordinating* advertising campaigns (marketing, sales, edition, graphics, budget, . . .). It is supported in these tasks by the *Administration and Accounts* and *IT and Documentation* departments. The *Administration and Accounts* constitutes the *Technostructure* handling administrative tasks and policy, paperwork, purchases and budgets. The *Support* component includes the *IT and Documentation* departments. It defines *IT policies*, provides *technical means* required for the management of campaigns, and ensures services for *system support* as well as information retrieval for (*documentation* resources). The *Operational Core* includes the *Graphics and Editorial* staff in charge of the creative and artistic aspects of *realizing* advertising *campaigns* (texts, photographs, drawings, layout, design, logos).

Figure 1 models Agate as a structure-in-5 using *i**. As shown in Fig. 1, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are represented as

Table 1 Organization of Agate Ltd

Direction
1 Campaigns Director
1 Creative Director
1 Administrative Director
1 Finance Director
Campaigns Management
2 Campaign managers
3 Campaign marketers
1 Editor in Chief
1 Creative Manager
Graphics
6 Graphic designers
2 Photographers
Edition
2 Editors
4 Copy writers
Documentation
1 Media librarian
1 Resource librarian
1 Knowledge worker
Administration
3 Direction assistants
4 Manager Secretaries
2 Receptionists
2 Clerks/typists
1 Filing clerk
IT
1 IT manager
1 Network administrator
1 System administrator
1 Analyst
1 Computer technician
Accounts
1 Accountant manager
1 Credit controller
2 Accounts clerks
2 Purchasing assistants

ovals, clouds, hexagons and rectangles respectively; dependencies have the form *depender* → *dependum* → *dependee*.

GMT is a company specialized in telecom services in Belgium. Its lines of products and services range from phones & fax, conferencing, line solutions, internet & e-business, mobile solutions, and voice & data management. As shown in Fig. 2, the structure of the commercial organization follows the structure-in-5. An *Executive Committee* constitutes the *Strategic Apex*. It is responsible for defining the *general strategy* of the organization. Five chief managers (*finances*, *operations*, *divisions management, marketing,* and *R&D*) apply the specific aspects of the *general strategy* in the area of their competence: *Finances & Operations* is in charge of *Budget* and Sales *Planning* & *Control*, *Divisions Management* is responsible for *Implementing Sales Strategy,* and *Marketing* and *R&D* define *Sales Policy* and *Technological Policy*.

Divisions Management groups managers that coordinate all managerial aspects of product and service sales. It relies on *Finance & Operations* for handling *Planning* and *Control* of products and services, it depends on *Marketing* for accurate *Market Studies* and on R&D for *Technological Awareness*.

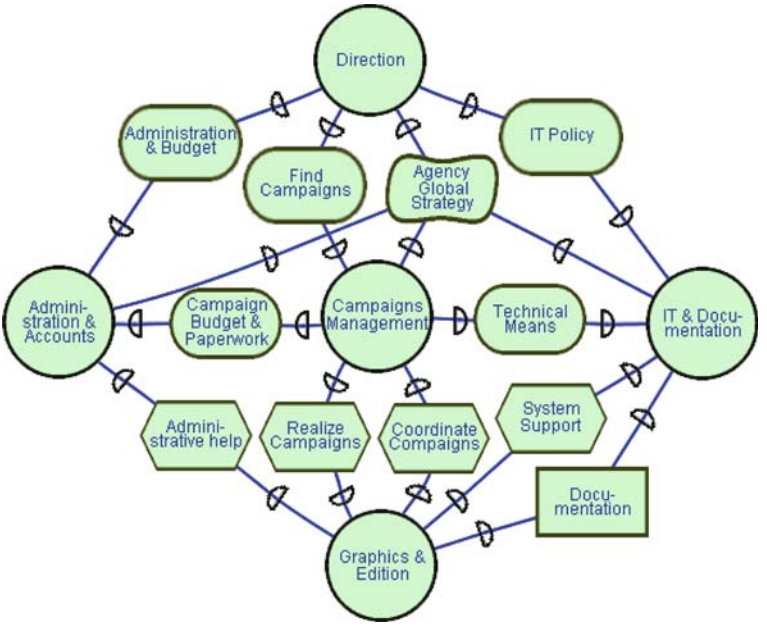

Fig. 1 Agate as a structure-in-5

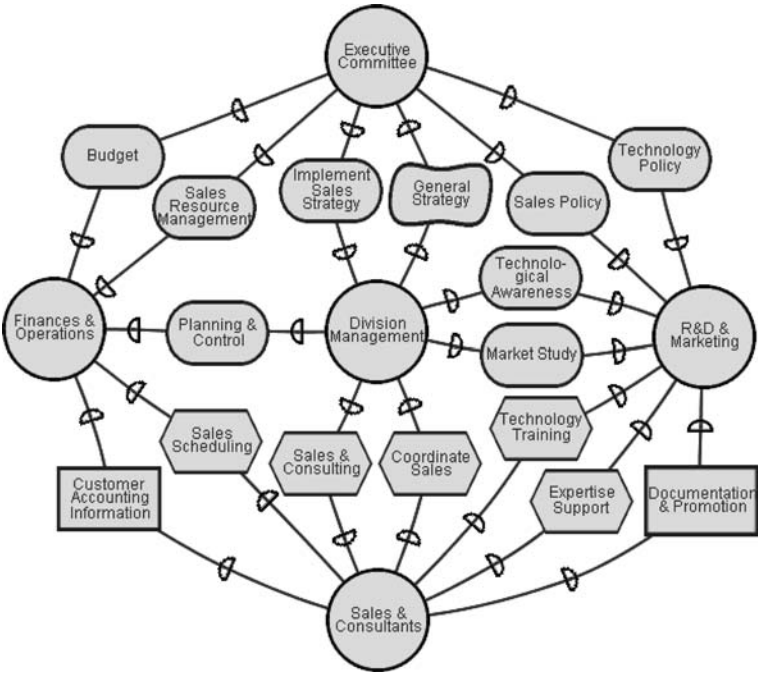

Fig. 2 GMT's sales organization as a structure-in-5

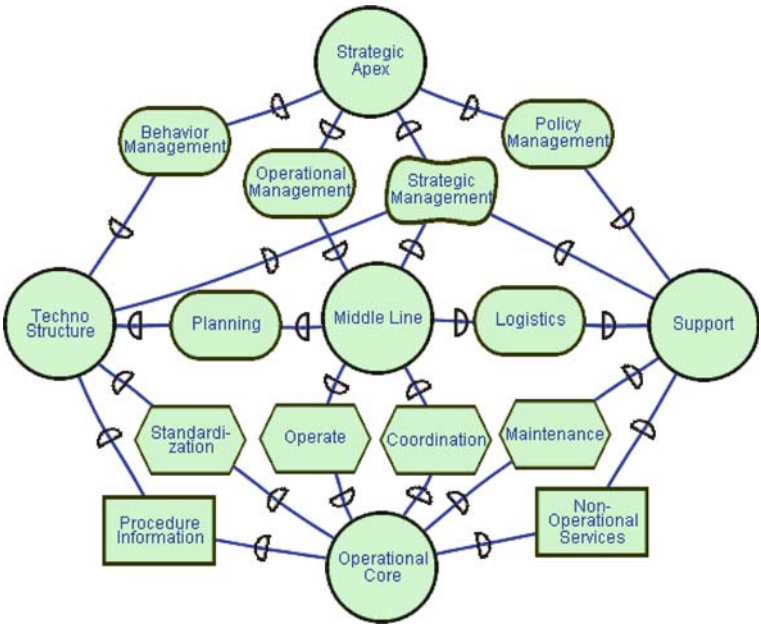

Fig. 3 The structure-in-5 style

Finances & Operations departments constitute the *technostructure* in charge of management *control* (financial and quality audit) and sales *planning* including *scheduling* and *resource management*.

The *Support* involves the staff of *Marketing* and *R&D*. Both departments jointly define and support the *Sales Policy*. The *Marketing* department coordinates *Market Studies* (customer positionment and segmentation, pricing, sales incentive, …) and provides the *Operational Core* with *Documentation* and *Promotion* services. The *R&D* staff is responsible for defining the technological policy such as *technological awareness services.* It also assists *Sales people* and *Consultants* with *Expertise Support* and *Technology Training*.

Finally, the *Operational Core* groups *the Sales people* and *Line consultants* under the supervision and coordination of *Divisions Managers*. They are in charge of selling products and services to actual and potential customers.

Figure 3 abstracts the structures explored in the case studies of Figs. 1 and 2 as a Structure-in-5 style composed of five actors. The case studies also suggested a number of constraints to supplement the basic style:

- the dependencies between the *Strategic Apex* as depender and the *Technostructure*, *Middle Line* and *Support* as dependees must be of type goal
- a softgoal dependency models the strategic dependence of the *Technostructure*, *Middle Line* and *Support* on the *Strategic Apex*
- the relationships between the *Middle Line* and *Technostructure* and *Support* must be of goal dependencies
- the *Operational Core* relies on the *Technostructure* and *Support* through task and resource dependencies
- only task dependencies are permitted between the *Middle Line* (as depender or dependee) and the *Operational Core* (as dependee or depender).

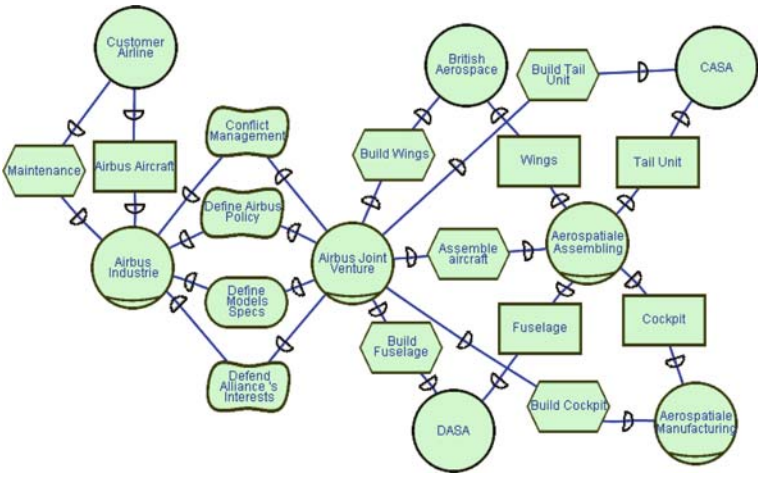

Fig. 4 The Airbus Industrie joint venture

3.2. Joint venture

We describe here two alliances – Airbus and Eurocopter – [15] that serve as a basis for proposing a meta-model the joint venture structure as an organizational style.

Airbus. The Airbus Industrie joint venture coordinates collaborative activities between European aeronautic manufacturers to built and market airbus aircrafts. The joint venture involves four partners: British Aerospace (UK), Aerospatiale (France), DASA (Daimler-Chrysler Aerospace, Germany) and CASA (Construcciones Aeronauticas SA, Spain). Research, development and production tasks have been distributed among the partners, avoiding any duplication. Aerospatiale is mainly responsible for developing and manufacturing the cockpit of the aircraft and for system integration. DASA develops and manufactures the fuselage, British Aerospace the wings and CASA the tail unit. Final assembly is carried out in Toulouse (France) by Aerospatiale. Unlike production, commercial and decisional activities have not been split between partners. All strategy, marketing, sales and after-sales operations are entrusted to the Airbus Industrie joint venture, which is the only interface with external stakeholders such as customers. To buy an Airbus, or to maintain their fleet, customer airlines could not approach one or other of the partner firms directly, but has to deal with Airbus Industrie. Airbus Industrie, which is a real manufacturing company, defines the alliance's product policy and elaborates the specifications of each new model of aircraft to be launched. Airbus defends the point of view and interests of the alliance as a whole, even against the partner companies themselves when the individual goals of the latter enter into conflict with the collective goals of the alliance.

Figure 4 models the organization of the Airbus Industrie joint venture using the *i** strategic dependency model. Airbus assumes two roles: Airbus Industrie and Airbus Joint Venture. *Airbus Industrie* deals with demands from customers, *Customer* depends on it to receive airbus aircrafts or maintenance services. The *Airbus Joint Venture* role ensures the interface for the four partners (*CASA, Aerospatiale*, *British Aerospace* and *DASA*) with *Airbus Industrie* defining Airbus strategic policy, managing conflicts between the four Airbus partners, defending the interests of the whole alliance and defining new aircrafts specifications. *Airbus*

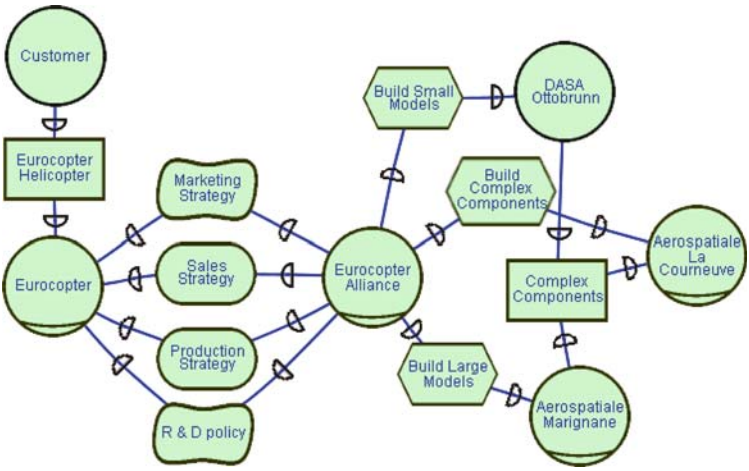

Fig. 5 The Eurocopter joint venture

Joint Venture coordinates the four partners ensuring that each of them assumes a specific task in the building of Airbus aircrafts: wings building for *British Aerospace*, tail unit building for *CASA*, cockpit building and aircraft assembling for *Aerospace* and fuselage building for *DASA*. Since Aerospatiale assumes two different tasks, it is modeled as two roles: *Aerospatiale Manufacturing* and *Aerospatiale Assembling*. *Aerospatiale Assembling* depends on each of the four partners to receive the different parts of the planes.

Eurocopter. In 1992, Aerospatiale and DASA decided to merge all their helicopter activities within a joint venture Eurocopter. Marketing, sales, R&D, management and production strategies, policies and staff were reorganized and merged immediately; all the helicopter models, irrespective of their origin, were marketed under the Eurocopter name. Eurocopter has inherited helicopter manufacturing and engineering facilities, two in France (La Courneuve and Marignane), one in Germany (Ottobrunn). For political and social reasons, each of them has been specialized rather than closed down to group production together at a single site. The Marignane plant manufactures large helicopters, Ottobrunn produces small helicopters and La Courneuve concentrates on the manufacture of some complex components requiring a specific expertise, such as rotors and blades.

Figure 5 models the organization of the Eurocopter joint venture in *i**. As in the Airbus joint venture, Eurocopter assumes two roles. The *Eurocopter* role handles helicopter orders from customers who depend on it to obtain the machines. It also defines marketing, sales, production and R & D strategies and policy. The *Eurocopter joint venture* role coordinates the manufacturing operations of the two partners – DASA and Aerospatiale – and depends on them for the production of small helicopters (*DASA Ottobrunn*), large ones (La Courneuve) and complex components (Marignane) such as rotors and blades. Since Aerospatiale assumes two different responsibilities, it is considered two roles: *Aerospatiale Marignane* and *Aerospatiale La Courneuve*. *DASA Ottobrunn* and *Aerospatiale Marignane* depends on *La Courneuve* to be supplied with complex helicopter parts.

Figure 6 abstracts the joint venture structures explored in the case studies of Figs. 4 and 5. The case studies suggest a number of constraints to supplement the basic style:

– Partners depend on each other for providing and receiving resources.

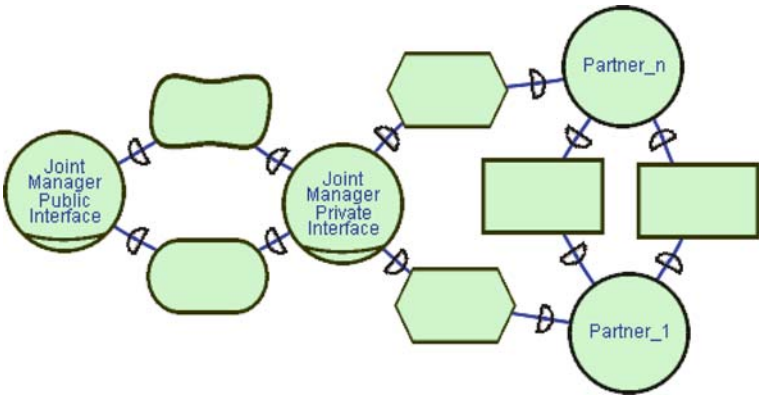

Fig. 6 The joint venture style

– Operation coordination is ensured by the joint manager actor which depends on partners for the accomplishment of these assigned tasks.
– The joint manager actor must assume two roles: a private interface role to coordinate partners of the alliance and a public interface role to take strategic decisions, define policy for the private interface and represents the interests of the whole partnership with respect to external stakeholders.

4. Software qualities for multi-agent systems

Software qualities constitute an essential element of software design theory. Qualities such as performance, maintainability, reliability, security and portability have been studied for decades and principles have been developed for generating designs that adhere to desirable qualities [7].

The choice of a software architecture depends critically on the qualities that stakeholders expect for a system-to-be. In landmark studies, [1] have evaluated alternative architectural styles with respect to different qualities. These evaluations can be used as a blueprint for choosing an architecture for a given set of desirable qualities.

We propose to adopt this line of research to offer guidelines for choosing among the architectural styles proposed here. But what are often-cited qualities for MAS? Below we offer an answer to this question based on a review of the MAS literature:

Predictability [59]. Autonomous software components – like agents – have many degrees of freedom [61] in the way that they undertake action in their respective domains. Consequently, it may be difficult to predict individual behaviors in order to determine the aggregate behavior of a distributed and open system. Generally, predictability of multi-agent systems impacts negatively on adaptability [26] and responsiveness [14].

Security. This quality measures the degree to which a system can protect from unauthorized use and ensure the integrity of its data and knowledge sources [59]. Important issues on multi-agent systems security concern [3,6,28] *authentication*, *network security*, *data security*, and *protection from malicious hosts*. Authentication makes it possible for an agent to ascertain the origin of received messages, so that intruders are not able to masquerade as

someone else. Network security services ensure that network packets do not get improperly read and modified by unauthorized intruders. Data security allows an agent to hide some of its data and capabilities from other agents. The problem of malicious hosts involves attacks on mobile agents from malicious hosts or intermediaries: if a host is to execute a process, the process can have no secrets from that host and there is nothing to prevent the host from analyzing the process and/or running it in altered form.

Adaptability. Agents may be required to adapt to changes in their environment. These may include changes to communication protocols, or the introduction of new kinds of agents. Generally, adaptability of multi-agent systems depends on the capabilities of individual agents to learn and predict the changes of the environments in which they act [58]. It also depends on their ability to make diagnosis [29] that determine the causes of a fault based on its symptoms. However, successful multi-agent systems tend to balance the degree of reactivity and predictability of individual agents against their ability to be adaptive.

Coordinability. Agents are not particularly useful unless they are able to coordinate with other agents (see [46] for recent contributions). Coordination is generally used to distribute expertise, resources or information among agents [31]. Coordinability measures the degree to which a system can coordinate by respecting interdependencies between agent actions, meet global constraints and optimize its operations.

Coordination can be realized in two ways:

– *Cooperativity*. Agents must be able to coordinate with other entities to achieve a common purpose or simply their own goals. Cooperation can be communicative in that the agents coordinate through communication. Alternatively, it can be non-communicative [11] where agents coordinate through observation. In deliberative communicating systems, agents jointly plan their actions so as to cooperate with each other.
– *Competitivity*. Deliberative negotiating systems [11] are like deliberative systems, except they have an added dose of competition. Here, the success of one agent implies the failure of others.

Availability. Agents that offer services to other agents (see for instance the FIPA standards [20]) must implicitly or explicitly guard against the interruption of their services. Availability can actually be considered a sub-attribute of security [7]. Nevertheless, we deal with it as a top-level software quality because of its increasing importance in multi-agent system design.

Fallibility-Tolerance. A failure of one agent (e.g., the inaccessibility to broker agents in [37]) need not lead to the failure of the whole system. In such a situation, however, the system needs to check the completeness and accuracy of data, transactions and flows. To prevent system failure, different agents can have similar or replicated capabilities and refer to more than one component for a specific behavior. Typically, in multi-agent systems failures of agents depends on coordination and interactions with external systems. For instance, interfering among agents' activities [31], increasing numbers of incoming agents [5], also aoption of different standards [20].

Modularity [53] increases efficiency of task execution, reduces communication overhead and usually results in high flexibility. On the other hand, it requires constraints on inter-module communication.

Aggregability. Measures the degree to which agents can become parts of composite agents. Aggregability entails surrendering of the individual to the control of the composite entity. This control results in efficient task execution and low communication overhead. However, it also impacts adversely on system flexibility [49].

5. Architectures for mobile robot control: A case study

This section presents the application of the structure-in-5 and joint venture styles and compares them to some conventional architectures. We illustrate the comparison with the classical mobile robot case study – often used in the software engineering literature (see e.g., [52]) – for its simplicity and pedagogical traits.

Mobile robot control systems must deal with external sensors and actuators. They must respond in time commensurate with the activities of the system within its environment. Consider the following activities [52] required for an office delivery mobile robot: acquiring input from a variety of sensors, controlling the motion of its wheels and other moveable parts, also planning its future paths. In addition, a number of factors complicate the execution of these 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 of its moves, the robot may encounter unpredictable events with little time for response.

5.1. Classical styles

For sample classical solutions, we examine three major conventional architectures – the layered architecture [54], control loops [43] and task trees [54] – that have been implemented on mobile robots.

- *Layered Architecture.* A classical layered architecture is depicted in Fig. 7. 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 (Fig. 8).
- *Task Trees.* This 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.

5.2. Organizational styles

We are developing organizational architectures for a miniature office delivery robot using the Lego® Mindstorms Robotics Invention Systems [38] and the Legolog programming platform [41] based on the Golog Planner [42]. Currently, we are testing two architectures

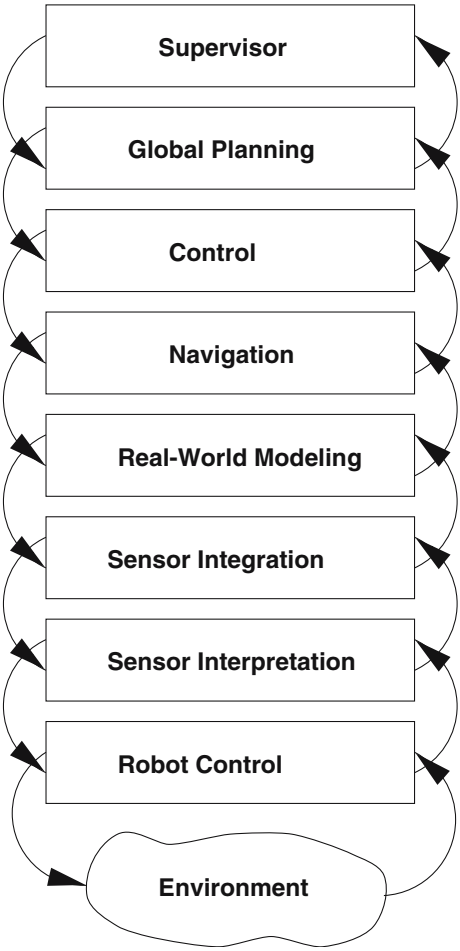

Fig. 7 Mobile robot layered architecture [52]

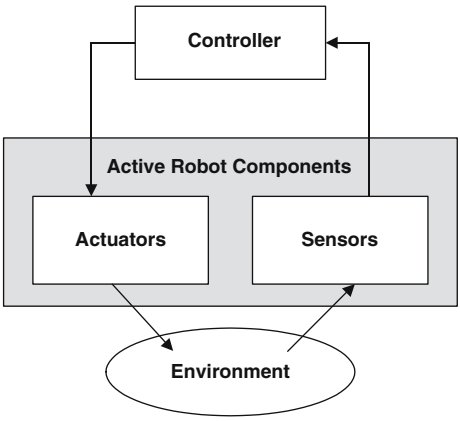

Fig. 8 Mobile robot control loop architecture [43]

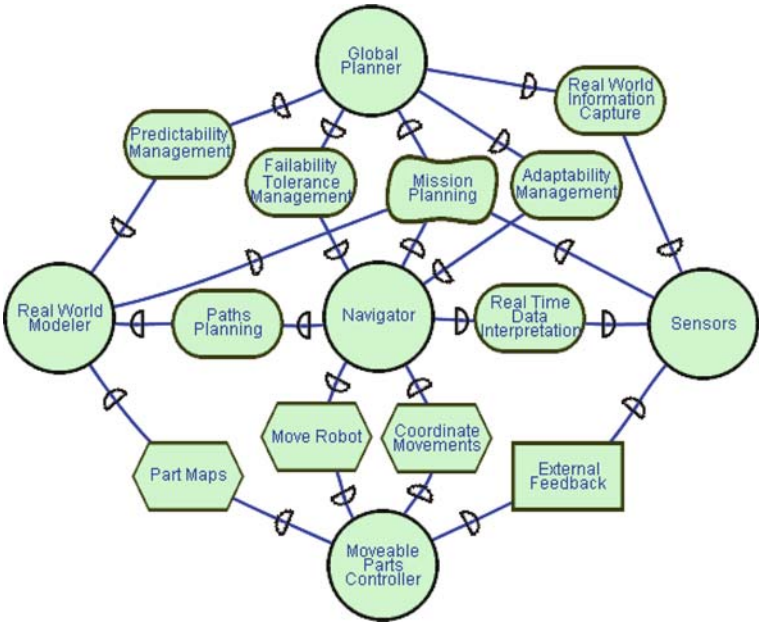

Fig. 9 A structure-in-5 mobile robot architecture

working with abstractions that are akin to those encountered in the layered architecture: the structure-in-5 and the joint-venture.

– *Structure-in-5.* Figure 9 depicts a structure-in-5 robot architecture in *i**. The moveable parts controller component is the operational core managing the robot motors, joints, wheels, etc. The Global Planner is the strategic apex planning and scheduling the robot's mission. The sensors compose the support component capturing real world raw information from hardware multiple sensors and integrating it into a coherent real-time interpretation for the Navigator component. It also gives direct external feedback to the Moveable Parts Controller. The Real World Modeler is the technostructure concerned with planning the mission paths, establishing and maintaining the robot's model of the world and checking the robot's mission environment to ensure predictability management. The Navigator is the middle agency component, the central intermediate module coordinating the movements of the robot to assume failability tolerance and adaptability management.
– *Joint Venture*. Following the style depicted in Fig. 6, the robot architecture in Fig. 10 is organized around a joint manager assuming two roles: the control interface role defines the robot's mission and quality strategies, i.e., predictability, adaptability and failability tolerance; the coordinator deals with coordinativity supervising the other agent components: a planner defining the mission planning, a monitors observing and checking the environment for landmarks, a motor controller to run the robot's parts and a perceptor subsystem that receives sensors data and interprets it. Each of these components also interact directly with each other to exchange information: the Motor Controller receives direct feedback from the perceptor, planning updates from the planner and adjust dynamically the robot's moves with respect to information provided by the Monitor that is also providing the perceptor with real-time mission information.

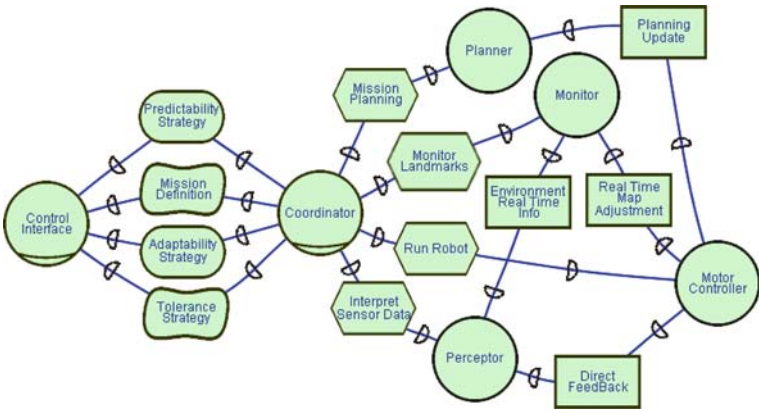

Fig. 10 A joint venture mobile robot architecture

5.3. Agent software qualities and evaluation

With respect to the activities and factors enumerated above, we adopt the following qualities for an office delivery mobile robot's architecture [52].

- *SQ1 – Coordinability.* 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.* All the circumstances of the robot's operation will never be fully predictable. The architecture must provide a framework wherein the robot can act even when faced with incomplete or unreliable information (e.g., contradictory sensor readings).
- *SQ3 – Failability-Tolerance.* The architecture must prevent the failure of the robot's operation and its environment. Local problems - such as reduced power supply, dangerous vapors, or unexpectedly opening doors – should not necessarily imply the failure of the mission.
- *SQ4 – Adaptability.* Application development for mobile robots frequently requires experimentation and reconfiguration. Moreover, changes in robot assignments may require regular modifications.

We evaluate each of the five styles – control loop, layered architecture, task trees, structure-in-5 and joint-venture described in Sections Classical styles and Organizational styles with respect to the four agent software quality attributes identified above.

- *Coordinability.* The simplicity of the control loop is a drawback when dealing with complex tasks because it offers no leverage for decomposing the software into more specific 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 coordinability 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 Fig. 9.
In the joint venture, each partner component interacts through the joint manager for strategic decisions. Components indicate their interest, and the joint manager provides them with needed information directly, or mediates requests to other partner components for such information.

– *Predictability.* The control loop reduces the unpredictable through iteration. Actions and reactions limit possible behaviors at each turn. Unfortunately, when 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 levels becomes clear with added knowledge at 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 (i.e., goal and softgoal) relationships.
In the joint-venture style, 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, this quality 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 by incorporating many checks and balances at different levels into the system. However, the drawback of this style is that control commands and transactions may often need to skip intermediate layers to check system state and behavior.
With 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 make incremental development and replacement of component straightforward through the use of implicit invocation: it is often sufficient to register new components, and existing ones need not feel the impact.
The structure-in-5 separates independently each typical component of the robot architecture isolating them from each other and allowing dynamic manipulation. Since the structure-in-5 is restricted to no more than 5 major components, as with the control loop, more refined tuning has to take place within these 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 pivotal role.

Table 2 summarizes the strengths and weaknesses of the five architectures included in the study. Following notations used by the NFR (non functional requirements) framework [7], $+, ++, -, --, +-$ respectively model partial/positive, sufficient/positive, partial/negative, sufficient/negative and positive-or-negative contributions.

The layered architecture offers a clear picture of 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. They also make clear the intentional and social dependencies governing these components. On the basis of these considerations, we can argue that the Structure-in-5 and the Joint-Venture fit better systems and applications that need open and cooperative components – such as the mobile robot example – because they are patterns governed by organizational principles.

6. Related work

The organizational perspective has been increasingly accepted within the MAS community, and mathematical and computational methods have been progressively used to develop a better understanding of the fundamental principles of organizing MAS [39].

In [21] Fox introduces the idea of using organization as a metaphor that can be useful in helping to describe, study, and design distributed software systems. In particular, he adopts results from Management Science to identify efficient agent organizations. Our work differs from his in that it focuses on how to use concepts from organization theory to model multi-agent organization and how to apply successful organization structures for MAS design.

The motivations of our work also have similarities with and differences from work by Zambonelli, Jennings & Wooldridge [64]. This work proposes to use agent roles and role models to specify and design multi-agent systems. Our approach is consistent with their idea of using organizational structures and organizational patterns, that are not just a collection of roles without any high-level structure as in [19]. However, the proposed styles are different.

Among others, TAEMS (Task Analysis, Environment Modeling and Simulation) (Decker, 1995) offers a modeling framework for representing coordination problems in a formal, domain-independent way. A TAEMS model can be used for both the analysis and simulation of coordination algorithms, and also to design organizational structures. Although, TAEMS can be extremely useful for detailed design (modeling sophisticated capabilities, alternative methods, activity-related effects, and complex interactions), it is not suitable for architectural design, where more abstract concepts, such as actor, goal and strategic dependencies are needed.

Table 2 Strengths and weaknesses of robot architectures

	Loop	Layers	Task Tree	S-in-5	Joint-Venture
Coordinability	−	−	+−	++	++
Predictability	+−	+	+−	+	++
Failability-Tolerence	+	+−	+	+	+
Adaptability	+−	+−	+	+	+−

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.

Aalaadin [19] uses concepts such as *agent*, *group*, and *role* to model the organizational structure of multi-agent systems. In this work, different types of organizational behavioral requirement patterns have been defined and formalized. Similarly, the Gaia methodology [60] uses role and interaction models are used for analyzing system structure. The organization of the system is viewed as a collection of roles, that stand in a certain relationship to each other, and take part in systematic, institutionalized patterns of interactions with other roles. The main difference with our approach is that in both Gaia and Aalaadin, goals are not part of the organization description.

Multi-agent organization theory [32] uses components such as scenario, organizational position, goal, plan, membership, and constraints to specify organizational structure. Although, the theory can be used for designing multi-agent systems, its principle aim is to extend organization theory to the design and control of multi-agent systems. In particular, a theory that allows the designer to embed the design of intelligent agents and multi-agents systems into organizational design.

Finally, other research results in multi-agent organizational design includes Self-organization Design [30] and Teamwork Models [40].

Self-organization design is based on the idea of having an organization where one or more members can monitor the organizational structure's effectiveness in directing organizational activities, design new organizational structures appropriate to new situations and evaluate possible organizations and select the best one, and implement and execute the new structure over the network while preserving the network's problem solving activities. So & Durfee [55,56] propose a predictive model of Task-Organization-Performance, which, given a task, allows one to generate the possible organizations to solve the problem, and evaluate each alternative.

Teamwork models [57], based on the joint intentions framework introduced in [40], allow the design of multi-agent systems where individual agents are provided with an explicit representation of the team goals and plans, an underlying explicit model of team activity.

7. Conclusions

We have proposed a set of organizational architectural styles for designing agent-based systems. Since the fundamental concepts of agent systems are intentional and social, rather than implementation-oriented, multi-agent systems (MAS) can be viewed as organizational structures composed of autonomous and proactive agents that interact to achieve common and/or

private goals. In proposing these styles, we have adopted social organizations as a metaphor, guided by organizational design theories.

To provide more details, we focused on the structure-in-5 – a well-understood organizational style used by organization theorists – and the joint venture – used to describe cooperative strategies in the business world. Both styles have been modeled in term of intentional and social primitives from case studies describing real world organizations.

The contributions also include the presentation and evaluation of software qualities that are specific to MAS. We have also offered a comparison of organizational and conventional styles, conducted through a mobile robot case study taken from the Software Engineering literature.

Future research directions include extending and formalizing the catalogue of organizational styles, also characterizing specifically how a particular architecture can be seen as an instance of a style.

We are also interpreting and formalizing in the same intentional and social way lower-level conventional architectural elements involving (software) components, ports, connectors, interfaces, libraries and configurations.

Organizational styles will eventually define an architectural macro level for MAS. At a micro level, we propose to focus on the notion of social agent patterns [13]. Many existing patterns can be incorporated into a system architecture, such as those identified in [22]. For agent software, patterns for distributed, and open architectures – such as the broker, matchmaker, embassy, mediator, wrapper, mediator – are more appropriate [27,59]. These detail how goals and dependencies identified in an organizational style can be refined and achieved. Such mappings from architectural to detailed design, from organizational styles to social patterns have been explored in [17,33,47].

References

1. Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice*. Addison-Wesley
2. Bennett, S., McRobb, S., & Farmer, S. (1999). *Object-oriented systems analysis and design using UML*. McGraw Hill.
3. Bonatti, P. A., Kraus, S., Salinas, S., & Subrahmanian, V. S. (1998). Data-Security in heterogeneous agent systems. In M. Klusch & G. Weiss (Eds.), *Cooperative information agents* (pp. 290–305). Springer Verlag.
4. Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: The Tropos Project. *Information systems*.
5. Cetnarowicz, K., Dobrowolski, G., Kisiel-Dorohinicki, M., & Nawarecki, E. (1998). Functional Integrity of MAS through the Dynamics of the Agents' Population In *Proceedings of the 3rd international conference on multi agent systems, ICMAS'98*.
6. Chess, D. M. (1998). Security Issues in Mobile Code Systems. In G. Vigna (ed.), *Mobile agents and security* (pp. 1–14). Springer Verlag.
7. Chung, L. K., Nixon, B., Yu, E., & Mylopoulos, J. (2000). *Non-functional requirements in software engineering*. Kluwer Publishing.
8. Coyette, A., Kolp, M., & Faulkner, S. (2004, December). Using Intelligent Agents to Build E-Business Software. In *Proceedings of the 4th international conference on electronic commerce. ICEB'04* Beijing, China.
9. Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-Directed Requirements Acquisition. *Science of Computer Programming, 20*(1–2), 3–50.
10. Decker, K. S. (1995). *Environment centered analysis and design of coordination mechanisms*. PhD thesis University of Massachusetts.
11. Doran, J. E., Franklin, S., Jennings, N. R., & Norman, T. J. (1997). On cooperation in multi-agent systems. *Knowledge Engineering Review, 12*(3), 309–314.
12. Do, T. T., Faulkner, S., & Kolp, M. (2003, April). Organizational Multi-Agent Architectures for Information Systems. In *Proceedings of the 5th international conference on enterprise information systems, ICEIS'03*, Angers, France.

13. Do, T. T., Kolp, M., & Pirotte, A. (2003, July). Social Patterns for Designing Multi-Agent Systems. In *Proceedings of the 15th international conference on software engineering and knowledge engineering, SEKE'03*, San Francisco. USA.
14. Durfee, E. H., & Lesser, V. R. (1988). Predictability Versus Responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of the 7th national conference on artificial intelligence*, (pp. 66–71).
15. Dussauge, P., & Garrette, B. (1999). *Cooperative strategy: Competing successfully through strategic alliances*. Wiley and Sons.
16. Faulkner, S., Kolp, M., Coyette, A., & Do, T. T. (2004a, April). Agent Oriented Design of E-Commerce System Architecture. In *Proceedings of the 6th international conference on enterprise information systems, ICEIS'04*, Porto, Portugal.
17. Faulkner, S., Kolp, M., Nguyen, T., & Coyette, A. (2004, June). Information Integration Architecture Development: A Multi-Agent Approach. In *Proceedings of the 16th international conference on software engineering and knowledge engineering, SEKE'04*, Banff, Canada.
18. Faulkner, S. (2004). *An architectural framework for describing BDI multi-agent information systems*. PhD thesis, University of Louvain, IAG-ISYS Information Systems Research Unit.
19. Ferber, J., & Gutknecht, O. (1998, June). A meta-model for the anlysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd international conference on multi agent systems, ICMAS'98*.
20. FIPA. (2001). The foundation for intelligent physical agents. At http://www.fipa.org.
21. Fox, M. (1981). An Organizational View of Distributed systems. *Transactions on Systems Man and Cybernetics, 11*(1), 70–80.
22. Gamma, E., Helm, R., Johnson, J., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
23. Giorgini, P., Kolp, M., & Mylopoulos, J. (2002, July). Multi-Agent and Software Architecture: A Comparative Case Study. In *Proceedings of the 3rd international workshop on agent software engineering, AOSE'02*, Bologna, Italy.
24. GMT. (2002). GMT Consulting Group. http://www.gmtgroup.com/
25. Gomes-Casseres, B. (1996). *The alliance revolution: The new shape of business rivalry*. Harvard University Press.
26. Gordon, D. (2000). APT Agents: Agents that are adaptive predictable and timely. In *Proceedings of the 1st goddard workshop on formal approaches to agent-based systems, FAABS'00*.
27. Hayden, S., Carrick, C., & Yang, Q. (1999, May). Architectural Design Patterns for multi-agent coordination. In *Proceedings of the 3rd international conference on autonomous agents, agents'99*, Seattle, USA.
28. He, Q., & Sycara, K. (1998). Towards a Secure Agent Society. In *Proceedings of the workshop on deception fraud and trust in agent societies, ACM AA'98*.
29. Horling, B., Lesser, V., Vincent, R., Bazzan, A., & Xuan, P. (1999). Diagnosis as an integral part of multi-agent adaptability. Technical Report UM-CS-1999-003, University of Massachusetts.
30. Ishida, T., Gasser, L., & Yokoo, M. (1992). Organization self-design of distributed production systems. *Transaction on knowledge and data engineering, 4*(2), 123–134.
31. Jennings, N. R. (1996). Coordination Techniques for Distributed Artificial Intelligence. In G. M. . O'Hare & N. R. Jennings (eds.), *Foundations of distributed artificial intelligenc*, (pp. 187–210). Wiley.
32. Kirn, S., & Gasser, L. (1998). Organizational Approaches to Coordination in Multi-Agent Systems. *Informationstechnik und Technische Informatik (IT+TI), 4*(40), 23–29.
33. Kolp, M., Do, T. T., Faulkner, S., & Hoang, T. T. H. (2005). Social-centric development of multi-agent architectures . *Journal of Organizational Computing and Electronic Commerce*.
34. Kolp, M., Giorgini, P., & Mylopoulos, J. (2001, August). A Goal-Based Organizational Perspective on Multi-Agents Architectures. In *Proceedings of the 8th international workshop on intelligent agents: Agent theories architectures and languages, ATAL'01*, Seattle, USA.
35. Kolp. M., Giorgini, P., & Mylopoulos, J. (2002, July). Information systems development through social structures. In *Proceedings of the 14th international conference on software engineering and knowledge engineering, SEKE'02,* Ishia, Italy.
36. Kolp, M., Giorgini, P., & Mylopoulos, J. (2003, June). Organizational Patterns for Early Requirements Analysis. In *Proceedings of the 15th international conference on advanced information systems, CAiSE'03*, Velden, Austria.
37. Kumar, S., & Cohen, P. R. (2000). The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams. In *Proceedings of the 4th international conference on Multi Agent Systems, ICMAS'00*.
38. Lego. (2002). Lego mindstorms robotics invention system. At http://mindstorms. lego.com.
39. Lesser, V. R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *Transaction on Knowledge and Data Engineering, 11*(1).

40. Levesque, H. J., Cohen, P. R., & Nunes, J. (1990). On acting together. In *Proceedings of the national conference on artificial intelligence,* Menlo Park, USA.
41. Levesque, H., & Pagnucco , M. (2000). Legolog. http://www.cs.toronto.edu/ cogrobo/Legolog/
42. Levesque, H., Reiter, R., Lesperance, Y., Lin, F., & Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, 31*(1–3).
43. Lozano-Perez, T. (1990). Preface. In L. Cox & G Wilfong (eds.) *Autonomous robot vehicles*. Springer Verlag.
44. Mintzberg, H. (1992). *Structure in fives : Designing effective organizations*. Prentice-Hall.
45. Morabito, J., Sack, I., & Bhate, A. (1999). *Organization modeling : Innovative architectures for the 21st century*. Prentice Hall.
46. Omicini, A., Zambonelli, F., Klusch, M., & Tolksdorf, R. (eds.), (2001). *Coordination of internet agents*. Springer Verlag.
47. Penserini, L., Kolp, M., Spalazzi, L., & Panti, M. (2004, September) Socially-Based Design meets Agent Capabilities. In *Proceedings of the 2004 IEEE/WIC/ACM international joint conference on web intelligence, WI'04 and intelligent agent technology, IAT'04*, Beijing, China .
48. Perini, A., Bresciani, P., Giunchiglia, F., Giorgini, P., & Mylopoulos, J. (2001, May). A knowledge level software engineering methodology for agent oriented programming. In *Proceedings of the 5th international conference on autonomous agents, Agents'01*, Montreal, Canada.
49. Sabater, J., Sierra, C., Parsons, S., & Jennings, N. R. (2001). Engineering executable agents using multi-context systems. *Journal of Logic and Computation*. (to appear).
50. Scott, W. R. (1998). *Organizations: Rational natural and open systems*. Prentice Hall.
51. Segil, L. (1996). *Intelligent business alliances : how to profit using today's most important strategic tool*. Times Business.
52. Shaw, M., & Garlan (1996) D. *Software Architecture: Perspectives on an emerging discipline*. Prentice Hall.
53. Shehory, O. (1998). Architectural Properties of Multi-Agent Systems. Technical Report CMU-RI-TR-98-28, Carnegie Mellon University.
54. Simmons, R., Goodwin, R., Haigh, K. Koenig, S., & O'Sullivan, J. (1997, February). A modular architecture for office delivery robots. In *Proceedings of the 1st international conference on autonomous agents, agents '97 agent'97* (pp. 245–252) Marian del Rey, USA.
55. So, Y. P., & Durfee, E. H. (1994). Modeling and Designing Computational Organizations. In *Working notes of the AAAI spring symposium on computational organization design*.
56. So, Y. P., & Durfee, E. H. (1996). Designing Tree-Structured Organizations for Computational Agents. *Computational and Mathematical Organization Theory, 2*(3), 219–246.
57. Tambe, M. (1996). Teamwork in Real-World dynamic Environments. In *Proceedings of the 2nd international conference on multi agents systems, ICMAS'96*, Kyoto, Japan.
58. Weiss, G. (ed.). (1997). *Learning in DAI Systems*. Springer Verlag.
59. Woods, S. G., & Barbacci, M. (1999). Architectural Evaluation of Collaborative Agent-Based Systems. Technical Report SEI-99-TR-025, SEI, Carnegie Mellon University, Pittsburgh, USA.
60. Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi Agent Systems, 3*(3), 285–312.
61. Wooldridge, M., & Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review, 2*(10).
62. Yoshino, M. Y., & Rangan, U. S. (1995). *Strategic alliances : an entrepreneurial approach to globalization*. Harvard Business School Press.
63. Yu, E. (1995). *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, Department of Computer Science.
64. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2000, June). Organisational Abstractions for the Analysis and Design of Multi Agent Systems. In *Proceedings of the 1st international workshop on agent-oriented software engineering*, Limerick, Ireland.