Multi-Agent Architectures as Organizational Structures

Paolo Giorgini (paolo.giorgini@dit.unitn.it)
Department of Information and Communication Technology - University of Trento Via Sommarive, 14, I-38050 Povo - Trento, Italy

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

John Mylopoulos (jm@cs.toronto.edu)
Department of Computer Science - University of Toronto - 6, King's College Road, M5S 3H5 Toronto, Canada

Abstract.

A Multi-Agent System is an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real world organizations as an analogy, this paper proposes architectural styles for MAS which adopt concepts from organizational theories. The styles are modeled using the *i** framework which offers the notions of actor, goal and actor dependency and specified in *Formal Tropos*. They are evaluated with respect to a set of software quality attributes, such as predictability and adaptability. In addition, we conduct a comparative study of organizational and conventional software architectures using the mobile robot control example from the Software Engineering literature. The research is conducted in the context of *Tropos*, a comprehensive software system development methodology.

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.

System architectural design has been the focus of considerable research during the last fifteen years that 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 [21]. Examples of software qualities include maintainability, modifiability, portability etc [1]. We are interested in developing a suitable set of architectural styles for multi-agent software systems. Since 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. 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* and *Strategic Alliances*. Organizational styles from organization theory describe the internal structure and design of an organization, while strategic alliances model the strategic cooperation of independent organizational actors that pursue shared goals.

In this paper, we describe some of these styles, adapt two of them – the structure-in-5 and the joint venture – for multi-agent architectural design, model them using the strategic dependency model of i^* [64], and propose a specification in *Formal Tropos* [19].

This research is being conducted in the context of *Tropos* [5, 24, 45], a project developing a requirements-driven methodology for software systems. The *Tropos* methodology adopts ideas from MAS technologies, mostly to define the detailed design and implementation phase, and ideas from requirements engineering, where agents/actors and goals have been used heavily for early requirements analysis [9, 64]. 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 modelling an application during early requirements analysis. The key assumption which distinguishes *Tropos* from other methodologies is that actors and goals are used as fundamental concepts for analysis and design during *all phases of software development*, not just requirements analysis.

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 – based on real-word examples of organizations. These two styles are modeled in terms of social and intentional concepts using the i^* framework. A semi-formal specification language founded on i^* is also proposed for each style. Section 4 presents a set of software quality attributes for these styles in terms of which one can evaluate architectural alternatives. Section 5 proposes fragments of the mobile robot case study[1] adopted from the Software Engineering literature to illustrates the use of our styles and compares them to conventional ones. Section 6 presents a framework for selecting architectural styles with respect to software quality attributes while Section 7 overviews the *Tropos* methodology. Finally, Section 8 discusses related research while Section 9 summarizes the contributions and outlines future work.

[1] Although we have worked out other, e-business case studies using the styles describes in this paper (see, e.g., [5, 45]), we have decided to use a simpler and more pedagogical example in this paper for understandability purposes.

2. Structuring Organizations

Since the origins of civilization, people have been designing, participating in, and sharing the burdens and rewards of organizations. The early organizations were primarily military or governmental in nature. In the *Art of War*, Sun Tzu describes the need for hierarchical structure, communications, and strategy. In the *Politics*, Aristotle wrote of governmental administration and its association with culture. To the would-be-leader, Machiavelli advocated in the *Prince* power over morality. The roots of organizational theories, then, can be traced to antiquity, including thinkers from around the world who studied alternative organizational structures. Such structures consist of stakeholders – individuals, groups, physical or social systems – that coordinate and interact with each other to achieve common goals. Today, organizational structures are primarily studied by two disciplines: *Organization Theory* (e.g., [41, 49, 63]), that describes the structure and design of an organization and *Strategic Alliances* (e.g., [26, 42, 50, 14]), that model the strategic collaborations of independent organizational stakeholders who have agreed to pursue a set of agreed upon business goals.

Both disciplines aim to identify and study organizational patterns. These are not just modeling abstractions or structures, rather they can be seen, felt, handled, and operated upon. They have a manifest form and lie in the objective domain of reality as part of the concrete world. A pattern is however not solely a set of execution behaviors. Rather, it exists in various forms at every stage of crystallization (e.g., specification), and at every level of granularity in the organization. The more manifest is its representation, the more the pattern emerges and becomes recognizable – whether at a high or low level of granularity.

At the lowest level of granularity, we find *information patterns* and *service patterns* that represent the "nitty-gritty" of business that an organization must deal with on a day-to-day basis. When we move to an upper level, we find *business patterns* – the mix of products and markets that flows from organizational styles. The highest level of granularity is the *organizational styles* that addresses the mix of socio-technical context and organizational constructs: they are manifestation of organization invariants, layers of organizational constructs, organization molecules, and complex arrangements of molecules, the collection of which constitutes organizational structures.

Many organizational styles are fully formed patterns with definite characteristics as the ones we present in the rest of this section. In contrast, many other organizational styles are not very explicit, that is, not easily specified, operationalized, and measured. Michael Porter's generic strategies [46] are examples of such patterns. Each strategy type

is characterized by general properties that distinguish one strategy from another. For the most part, however, the distinguishing characteristics of each style are only partially described in terms of an organization's architecture.

In this paper, we are interested to identify and use as system architectural styles, organizational styles that have been already well-understood and precisely defined in organizational theories. Our purpose is not to categorize them exhaustively nor to study them on a managerial point of view. The following sections will thus only insist on styles that have been found, due to their nature, interesting candidates for MAS architectural design also considering the fact that they have been studied in great detail in the organizational literature and presented as fully formed patterns.

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" [42]. Organization theory is the discipline that studies both structure and design in such social entities. Structure deals with the descriptive aspects while design refers to the prescriptive aspects of a social entity. Organization theory describes how practical organizations are actually structured, offers suggestions on how new ones can be constructed, and how old ones can change to improve effectiveness. To this end, since Adam Smith, schools of organization theory have proposed models patterns to try to find and formalize recurring organizational structures and behaviors.

In the following, we briefly present organizational styles identified in Organization Theory. The structure-in-5 will be studied in detail in Section 3.

The Structure-in-5. An organization can be considered an aggregate of five sub-structures, as proposed by Minztberg [41]. 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 of the organization in 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 the well-know hierarchical authority structure. Actors at lower levels depend on those at higher levels. The crucial mechanism 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 (low) level. They can coordinate behaviors or take decisions by 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. 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. 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, the 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 competitivity mechanisms, and actors behave as if they were taking part in an auction. 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 who is responsible for issuing the bidding.

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 the participant organizations 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 necessary and sufficient characteristics:

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

In the following, we briefly present organizational styles identified in Strategic Alliances. The joint venture will be studied in details in Section 3.

The joint venture style involves agreement between two or more intra-industry partners to obtain the benefits of larger scale, partial investment and lower maintenance costs. A specific joint management actor coordinates tasks and manages the sharing of resources between partner actors. Each partner can manage and control itself on a local dimension and interact 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 in which the original actors possess equity participations. We describe and model examples of joint ventures in Section 3.

The arm's-length style implies agreements between independent and competitive, but partner actors. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise common goals. No authority is 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. Coordination mechanisms developed for arm's-length (independent) characteristics involve a variety of negotiators, mediators and observers at different levels handling conditional clauses to monitor and manage possible contingencies, negotiate and resolve conflicts and finally deliberate and take decisions. Hierarchical relationships, from the executive apex to the arm's-length contractors restrict autonomy and underlie a cooperative venture between the parties.

The co-optation style involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of an initiating organization. By co-opting representatives of external systems, organizations are, in effect, trading confidentiality and authority for resource, knowledge assets and support. The initiating system has to come to terms with the contractors for what is being done on its behalf; and each co-optated actor has to reconcile and adjust its own views with the policy of the system it has to communicate.

3. Modeling Organizational Styles

We will 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 the organization itself functions. Design parameters include, among others, goal and task assignments, standardization, supervision and control dependencies and strategy definitions.

This section describes two of the organizational styles presented in Section 2: the structure-in-5 and the joint-venture.

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 [25]. They will serve to propose a model and a semi-formal specification of the structure-in-5.

Agate. Agate Ltd is an advertising agency located in Birmingham, UK, that employs about fifty staff, as detailed in Table I [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* groups the *IT and Documentation* departments. It defines the *IT policy* of Agate, provides *technical means* required for the management of campaigns, and ensures services for *system support* as well as information retrieval (*documentation* resources). The *Operational Core* includes the *Graphics and Edition* staff in charge of the creative and artistic aspects of *realizing campaign* (texts, photographs, drawings, layout, design, logos).

Table I. Organization of Agate Ltd

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

Figure 1 models Agate in structure-in-5 using the i^* strategic dependency model. i^* is a modeling framework for early requirements analysis [64], which offers goal- and actor-based notions such as *actor*, *agent, role, position, goal, softgoal*, *task*, *resource, belief* and different kinds of social *dependency* between actors. Its strategic dependency model describes the network of social dependencies among actors. It is a graph, where each node represents an *actor* and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, and the *dependee,* the actor who is depended upon. 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 (for instance, the appreciation is subjective or fulfillment is obtained only to a given extent); *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. As shown in Figure 1, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are represented as ovals, clouds, hexagons and rectangles; respectively, and dependencies have the form *depender* → *dependum* → *dependee*.

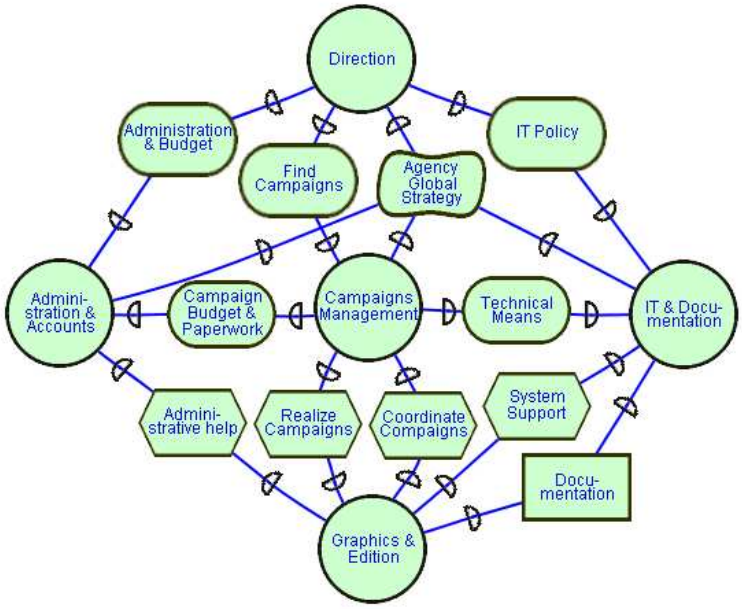

Figure 1. Agate as a Structure-in-5

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 Figure 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*.

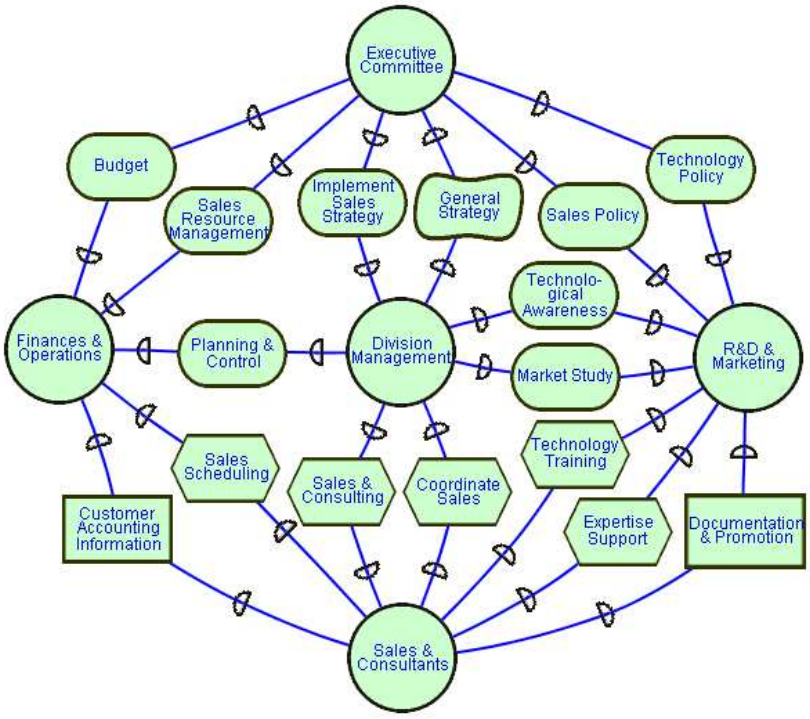

Figure 2. GMT's Sales Organization as a Structure-in-5

The *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.*

The *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 Man-*

agers. They are in charge of selling products and services to actual and potential customers.

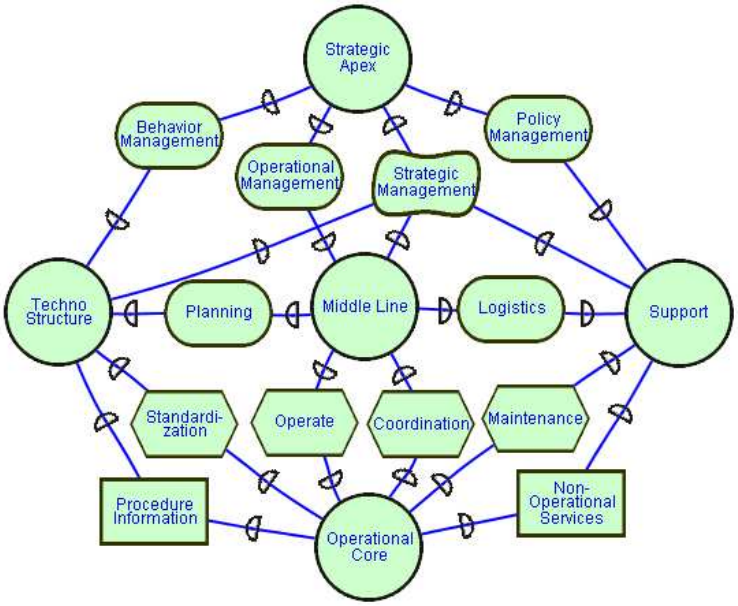

Figure 3. The Structure-in-5 Style

Figure 3 abstracts the structures explored in the case studies of Figures 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).

To specify the formal properties of the style, we use *Formal Tropos* [19], which extends the primitives of i^* with a formal language comparable to that of KAOS [9]. Constraints on i^* specifications are thus formalized in a first-order linear-time temporal logic. *Formal Tropos* provides three basic types of metaclasses: *actor, dependency, and entity* [22]. The attributes of a *Formal Tropos* class denote relationships among different objects being modeled.

Metaclasses

Actor := **Actor** name [attributes] [creation-properties] [invar-properties] [actor-goal]

With subclasses:

Agent(with attributes occupies: Position, play: Role)
Position(with attributes cover: Role)
Role

Dependency := **Dependency** name type mode **Depender** name **Dependee** name [attributes] [creation-properties] [invar-properties] [fulfill-properties]

Entity:=**Entity** name [attribute] [creation-properties][invar-properties]

Actor-Goal := **(Goal**|**Softgoal)** name mode **FulFillment**(actor-fulfill-property)

Classes: Classes are instances of Metaclasses.

In Formal Tropos, constraints on the lifetime of the (meta)class instances are given in a first-order linear-time temporal logic (see [19] for more details). Special predicates can appear in the temporal logic formulas: predicate $JustCreated(x)$ holds in a state if element x exists in this state but not in the previous one; predicate $Fulfilled(x)$ holds if x has been fulfilled; and predicate $JustFulfilled(x)$ holds if $Fulfilled(x)$ holds in this state, but not in the previous one.

In the following, we only present some specifications for the *Strategic Management* and *Operational Management* dependencies.

Actor StrategicApex
Actor MiddleLine
Actor Support
Actor Technostructure
Actor OperationalCore

Dependency StrategicManagement
Type SoftGoal
Depender te: Technostructure, ml: MiddleLine, su: Support
Dependee sa: StrategicApex
Invariant

$\forall dep : Dependency\ (JustCreated(dep) \rightarrow Consistent(self, dep))$
$\forall ag : Actor - Goal\ (JustCreated(ag) \rightarrow Consistent(self, ag))$

Fulfillment

$\forall dep : Dependency\ (dep.type = goal \wedge dep.depender = sa \wedge$
$(dep.dependee = te \vee dep.dependee = ml \vee dep.dependee = su)) \wedge$
$Fulfilled(self) \rightarrow \blacklozenge Fulfilled(dep)$

[*Invariant properties specify, respectively, that the strategic management softgoal must be consistent with any other dependency of the organization and with any other goal of the actors in the organization. The predicate Consistent depends on the particular organization we are considering and it is specified in terms of goals' properties to be satisfied. The fulfillment of the dependency necessarily implies that the goal dependencies between the Middle Line, the Technostructure, and the Support as dependees, and the Strategic Apex as depender have been achieved some time in the past]*

Dependency OperationalManagement
Type Goal
Mode achieve
Depender sa: StrategicApex
Dependee ml: MiddleLine
Invariant

$Consistent(self, StrategicManagement)$
$\exists c : Coordination\ (c.type = task \wedge c.dependee = ml \wedge c.depender =$
$OperationalCore \wedge ImplementedBy(self, c))$

Fulfillment

$\forall ts : Technostructure, dep : Dependency\ (dep.type = goal \wedge$
$dep.depender = ml \wedge dep.dependee = ts) \wedge Fulfilled(self))$
$\rightarrow \blacklozenge Fulfilled(dep)$

[The fulfillment of the Operational management goal implies that all goal dependencies between the Middle Line as depender and the Technostructure as dependee have been achieved some time in the past. Invariant properties specifies that Operational Management goal has to be consistent with Starategic Management softgoal and that there exists a coordination task (a task dependency between MiddleLine and Operational Core) that implement (ImplementedBy) the OperationalManagaemnt goal.]

In addition, the following structural (global) properties must be satisfied for the Structure-in-5 style:

- $\forall\ inst1, inst2 : StrategicApex \rightarrow inst1 = inst2$

  [There is a single instance of the Strategic Apex (the same constraint also holds for the Middle Line, the Technostructure, the Support and the Operational Core)]

- $\forall sa : StrategicApex,\ te : Technostructure,\ ml : MiddleLine,$
  $su : Support,\ dep : Dependency$
  $(dep.dependee = sa \wedge (dep.depender = te \vee dep.depender = ml$
  $\vee dep.depender = su) \rightarrow dep.type = softgoal)$

  [Only softgoal dependencies are permitted between the Strategic Apex as dependee and the Technostructure, the Middle Line, and the Support as dependers]

- $\forall sa : StrategicApex,\ te : Technostructure,\ ml : MiddleLine,$
  $su : Support,\ dep : Dependency :$
  $(dep.depender = sa \wedge (dep.dependee = te \vee dep.dependee =$
  $ml \vee dep.dependee = su) \rightarrow dep.type = goal)$

  [Only goal dependencies are permitted between the Technostructure, the Middle Line, and the Support as dependee, and the Stategic Apex as depender]

- $\forall su : Support,\ ml : MiddleLine,\ dep : Dependency$
  $((dep.dependee = su \wedge dep.depender = ml) \rightarrow dep.type = goal)$

  [Only task dependencies are permitted between the Middle Agency and the Operational Core]

- $\forall te : Technostructure,\ oc : OperationalCore,\ dep : Dependency$
  $((dep.dependee = te \wedge dep.depender = oc) \rightarrow$
  $(dep.type = task \vee dep.type = resource))$

  [Only resource or task dependencies are permitted between the Technostructure and the Operational Core (the same constraint also holds for the Support)]

- $\forall a : Actor,\ ml : MiddleLine,$
  $(\exists dep : Dependency(dep.depender = a \wedge dep.dependee =$
  $ml) \vee (dep.dependee = a \wedge dep.depender = ml) \rightarrow$
  $((\exists sa : StrategicApex(a = sa)) \vee (\exists su : Support(a = su) \vee$
  $(\exists te : Technostructure(a = te)) \vee (\exists op : OperationalCore$
  $(a = op))$

  [No dependency is permitted between an external actor and the Middle Agency (the same constraint also holds for the Operational Core)]

This specification can be used to establish that a certain *i** model does constitute an instance of the Structure-in-5 style. For example, the *i** model of Figure 1 can be shown to be such an instance, in which the actors are instances of the Structure-in-5 actor classes (e.g., *Direction* and *IT&Documentation* are instances of the *Strategic Apex* and the *Support*, respectively), dependencies are instances of Structure-in-5 dependencies classes (e.g., *Agency Global Strategy* is an instance the *Strategic Management*), and all above global properties are enforced (e.g., since there are only two task dependencies between *Campaigns Management* and *Graphics&Edition*, the fourth property holds).

3.2. Joint Venture

We describe here two alliances – Airbus and Eurocopter – [14] that will serve to model the joint venture structure as an organizational style and propose a semi-formal specification.

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-Benz 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.

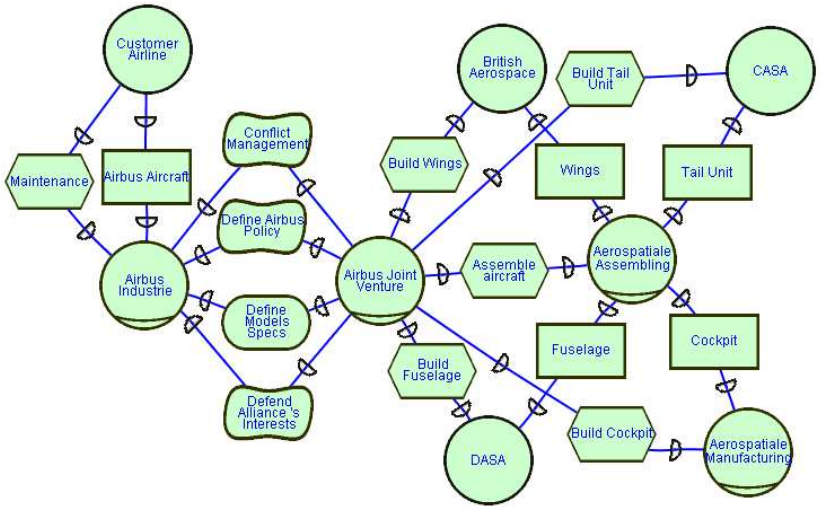

Figure 4. The Airbus Industrie Joint Venture

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 en-

sures 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 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 Figures 4 and 5. The case studies suggest a number of constraints to supplement the basic style:

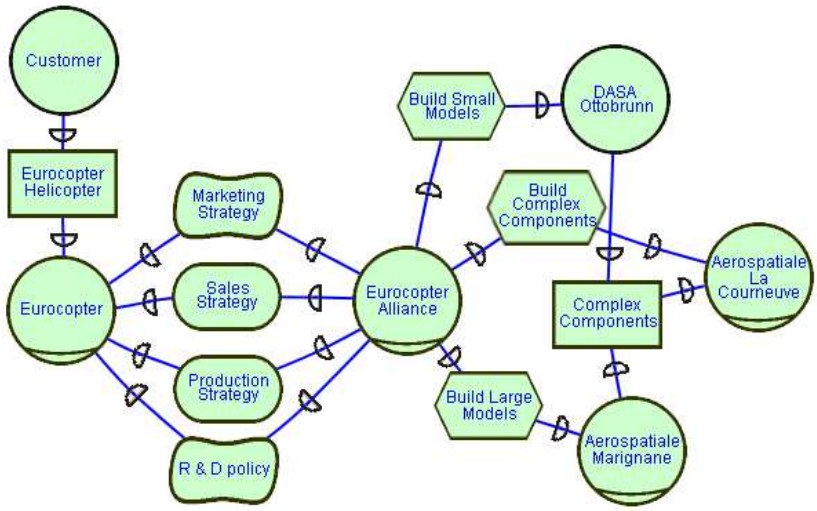

Figure 5. The Eurocopter Joint Venture

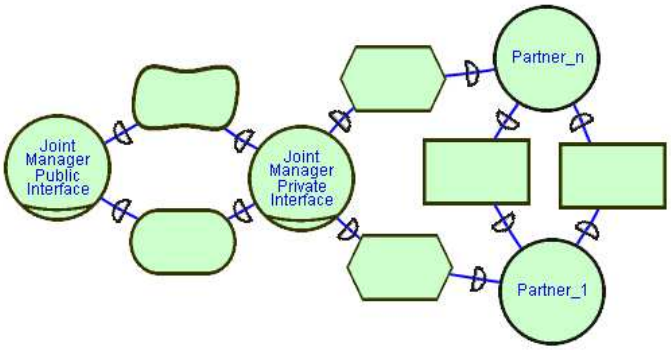

Figure 6. The Joint Venture Style

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

Part of the Joint Venture style specification is in the following:

Role JointManagerPrivateInterface
  Goal CoordinateStyles

Role JointManagerPublicInterface
  Goal TakeStrategicDecision
  SoftGoal RepresentPartnershipInterests

Actor Partner

and the following structural (global) properties must be satisfied:

- $\forall jmpri1, jmpri2 : JointManagerPrivateInterface$
  $(jmpri1 = jmpri2)$

  [Only one instance of the joint manager]

- $\forall p1, p2 : Partner, dep : Dependency$
  $(((dep.depender = p1 \wedge dep.dependee = p2) \vee (dep.depender =$
  $p2 \wedge dep.dependee = p1)) \rightarrow (dep.type = resource))$

  [Only resource dependencies between partners]

- $\forall jmpri : JointManagerPrivateInterface, p : Partner,$
  $dep : Dependency((dep.dependee = p \wedge dep.depender = jmpri)$
  $\rightarrow dep.type = task)$

  [Only task dependencies between partners and the joint manager, with the joint manager as depender]

- $\forall jmpri : JointManagerPrivateInterface,$
  $jmpui : JointManagerPublicInterface, dep : Dependency$
  $((dep.depender = jmpri \wedge dep.dependee = jmpui)$
  $\rightarrow (dep.type = goal \vee dep.type = softgoal))$

  [Only goal or softgoal dependencies between the joint manager roles]

- $\forall dep : Dependency, p1 : Partner$
  $((dep.depender = p1 \vee dep.dependee = p1) \rightarrow$
  $((\exists p2 : Partner(p1 \neq p2$
  $\wedge(dep.depender = p2 \vee dep.dependee = p2))$
  $\vee(\exists jmpi : JointManagerPrivateInterface$
  $((dep.depender = jmpi \vee dep.dependee = jmpi))))$

  [Partners only have relationships with other partners or the joint manager private interface]

- $\forall dep : Dependency, jmpi : JointManagerPrivateInterface$
  $((dep.depender = jpmi \vee dep.dependee = jpmi) \rightarrow$
  $((\exists p : Partner((dep.depender = p \vee dep.dependee = p)))\vee$
  $(\exists jmpui : JointManagerPublicInterface$
  $((dep.depender = jmpui \vee dep.dependee = jmpui))))$

  [The joint manager private interface only has relationships with the joint manager public interface or partners]

4. Software Qualities for Multi-Agent Systems

Generally the following software qualities are addressed to characterized multi-agent system architectures:

Predictability [60]. Autonomous components like agents have a high degree of autonomy [61] in the way that they undertake action and communication in their domains. It can be then difficult to predict individual characteristics as part of determining the behavior of a distributed and open system at large. Generally, predictability of multi-agent systems is in contrast with the agents capabilities to be adaptive [27] and responsive [13]: agents must be predictable enough to anticipate and plan actions while being responsive and adaptive to unexpected situations.

Security. Agents are often able to identify their own data and knowledge sources and they may undertake additional actions based on these sources [60]. Protocols and strategies for verifying authenticity for these data sources by individual agents are an important concern in the evaluation of overall system quality since, in addition to possibly misleading information acquired by agents, there is the danger of hostile external entities spoofing the system to acquire information accorded to trusted domain agents.

Important issues on multi-agent systems security concern [3, 29, 7]: *authentication*, *network security*, *data security*, and *protection from malicious hosts*. Authentication of agents makes it possible for an agent to ascertain an origin of received messages, so that intruders are not able to masquerade as some one else. Network security services ensure that network packets do not get improperly read and modified by unauthorized intruders: an agent is able to verify that a received messages has not been modified in transit. Data security allows an agent to hide some of its data and capabilities from the other agents. The problem of malicious hosts involves attacks on the 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 program and running an altered form of the program.

Adaptability. Agents may be required to adapt to modifications in their environment. They may include changes to the component's communication protocol or possibly the dynamic introduction of a new kind of component previously unknown or the manipulations of existing agents.

Generally, adaptability of multi-agent systems depends on the capabilities of the single agents to learn and predict the changes of the environments in which they act [59], and also their capability to make diagnosis [30], that is being able to detect and 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 the single agents with their capabilities to be adaptive.

Coordinability. Agents are not particularly useful unless they are able to coordinate with other agents (see [43] for recent contributions). In multi-agent systems coordination is generally [32] used to distribute expertise, resources or information among the agents (agents may have different capabilities, specialized knowledge, different sources of information, resources, responsibilities, limitations, charges for services, etc.), solve interdependencies between agents' actions (interdependence occur when goal undertaken by individual agents are related), meet global constraints (when the solution being developed by a group of agents must satisfy certain conditions if is to be deemed successful), and to make the system efficient (even when individuals can function independently, thereby obviating the need for coordination, information discovered by one agent can be of sufficient use to another agent that both agents can solve the problem twice as fast).

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 local goals. Cooperation can either be communicative in that the agents communicate (the intentional sending and receiving of signals) with each other in order to cooperate or it can be non-communicative [12]. In the latter case, agents coordinate their cooperative activity by each observing and reacting to the behaviour of the other. In deliberative communicating systems, agents jointly plan their actions so as to cooperate with each other.
- **Competitivity**. Deliberative negotiating systems [12] are like deliberative systems, except that they have an added dose of competition. The success of one agent implies the failure of others.

Availability. Agents that offer services to other agents/humans (see for instance the FIPA standards [17]) must implicitly or explicitly guard against the interruption of offered services. Availability must actually be considered a sub-attribute of security [8]. Nevertheless, we deal with it as a top-level software quality attribute due to its increasing importance in multi-agent system design.

Fallibility-Tolerance. A failure of one agent (e.g., the inaccessibility to broker agents in [34]) does not necessarily imply a failure of the whole system. The system then needs to check the completeness and the accuracy of data, information and knowledge transactions and flows. To prevent system failure, different agents can have similar or replicated capabilities and refer to more than one component for a specific behavior.

Typically, in multi-agent systems failures of agents depends on coordination and interaction with external systems. For instance, interfering among agents' activities [32], the increasing number of incoming agents [6], and different standards adopted [17].

Modularity [52] increases efficiency of task execution, reduces communication overhead and usually enables high flexibility. On the other hand, it implies constraints on inter-module communication.

Aggregability. Some agent components are parts of other agent components. They surrender to the control of the composite entity. This control results in efficient tasks execution and low communication overhead, however prevents the system to benefit from flexibility [48].

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., [51]) for its simplicity and pedagogical aspects.

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 [51] 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.

5.1. Agent Software Qualities for Mobile Robot

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 [51].

- **SQ1 - Coordinativity.** 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.** 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.** 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.** Application development for mobile robots frequently requires experimentation and reconfiguration. Moreover, changes in robot assignments may require regular modification.

5.2. Classical Styles

For sample classical solutions, we examine three major conventional architectures – the layered architecture [54], control loops [40] and task trees [53] – that have been implemented on mobile robots.

- **Layered Architecture.** A classical layered architecture is depicted in Figure 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.

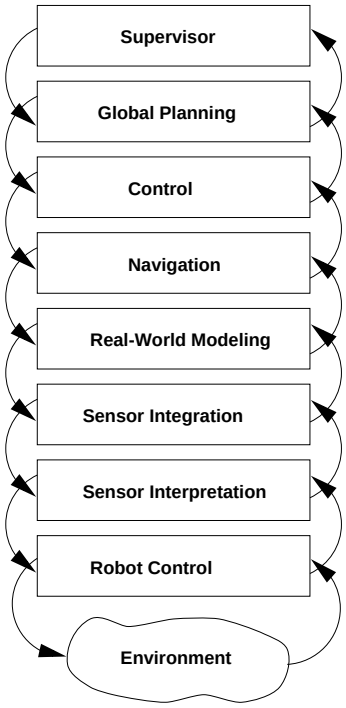

Figure 7. Mobile robot layered architecture [51]

– **Control loop.** A controller component initiates the robot actions. Since mobile robots have responsibilities with respect to their operational environment, the controller also monitors the consequences of the robot actions adjusting the future plans based on the return information (Figure 8).

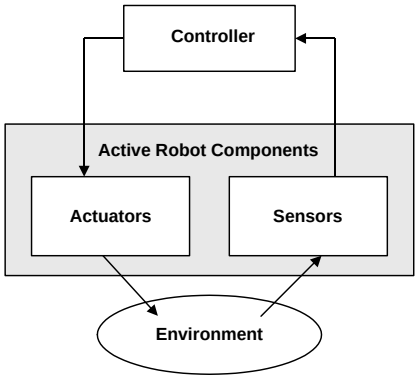

Figure 8. Mobile robot control loop architecture [40]

– **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.

5.3. Organizational Styles

We are developing organizational architectures for a miniature office delivery robot using the Lego ® Mindstorms Robotics Invention Systems [35] and the Legolog programming platform [37] based on the Golog Planner [38]. Currently, we are testing two architectures working with abstractions reminiscent of those encountered in the layered architecture: the structure-in-5 and the joint-venture.

– **Structure-in-5.** Figure 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 Figure 6, the robot architecture in Figure 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

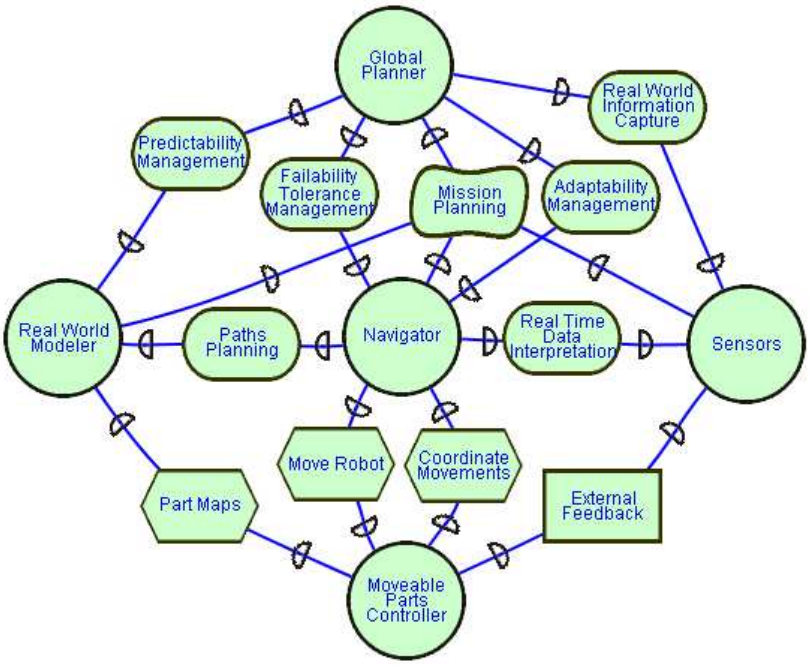

Figure 9. A structure-in-5 mobile robot architecture

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.

5.4. Evaluation

In this section, we evaluate each of the five styles - control loop, layered architecture, task trees, structure-in-5 and joint-venture described in Sections 5.2 and 5.3 with respect to the four agent software quality attributes identified in Section 5.1.

– **Coordinativity.**

The simplicity of the control loop is a drawback when dealing with complex tasks since it gives no leverage for decomposing the

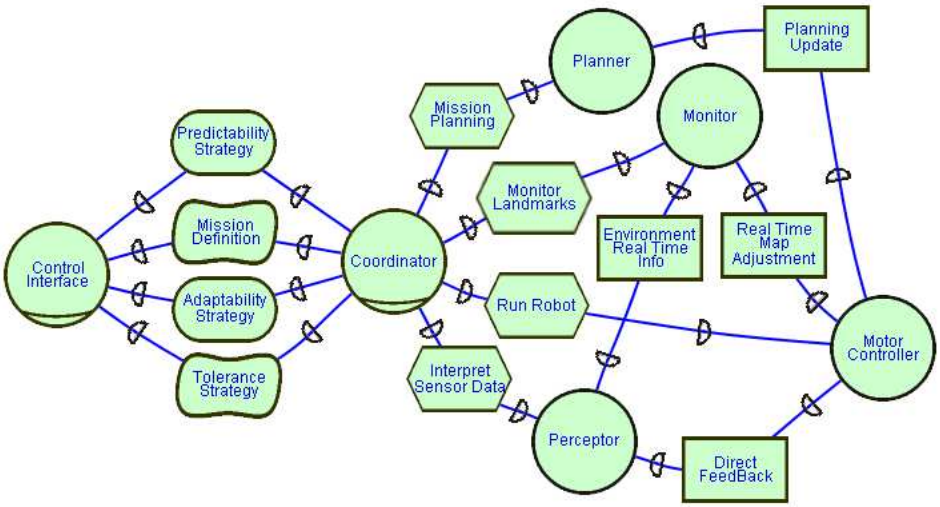

Figure 10. A joint venture mobile robot architecture

software into more precise cooperative agent components.
The layered architecture style suggests that services and requests are passed between adjacent agent layers. However, information exchange is actually not always straight-forward. Commands and transactions may often need to skip intermediate layers to establish direct communication and coordinate behavior.
A task tree permits a clear-cut separation of action and reaction. It also allows incorporation of concurrent agents in its model that can proceed at the same time to. Unfortunately, components have little interaction with each other.
Unlike the previous architectures, the structure-in-5 separates the data (sensor control, interpreted results, world model) from control (motor control, navigation, scheduling, planning and user-level control). The architecture improves coordinativity among components by differentiating both hierarchies - data is implemented by the support component, while control is implemented by the operational core, technostructure, middle agency and strategic apex - as shown in Figure 9.
In the joint venture, each partner component interacts via the joint manager for strategic decisions. Components indicate their interest, and the joint manager returns them such strategic information immediately or mediates the request to some other partner component.

– **Predictability.**

The control loop only reduces the unpredictable through iteration. Actions and reactions eliminate possibilities at each turn. Unfortunately, more subtle steps are needed, the architecture offers no framework for delegating them to separate agent components.
In the layered architecture, the existence of abstraction layers addresses the need for managing unpredictability. What is uncertain at the lowest level become clear with the added knowledge in the higher layers. How task trees address predictability is less clear. If imponderables exist, a tentative task tree can be built, to be adapted by exception handlers when the assumptions it is based on turn out to be erroneous. Like in the layered architecture, the existence of different abstraction levels in the structure-in-5 addresses the need for managing unpredictability. Besides, contrary to the layered architecture, higher levels are more abstract than lower levels: lower levels only involve resources and task dependencies while higher ones propose intentional (goals and softgoals) relationships. In the joint-venture, the central position and role of the joint manager is a means for resolving conflicts and prevent unpredictability in the robot's world view and sensor data interpretation.

– **Failability-Tolerance.**

In the control loop, it is supported in the sense that its simplicity makes duplication of components and behavior easy and reduces the chance of errors creeping into the system.
In the layered architecture, failability-tolerance could be served, when the robot architect strives not do something, by incorporating many checks and balances at different levels into the system. Again the drawback is that control commands and transactions may often need to skip intermediate layers to check the system behavior.
In the task trees, exception, wiretapping and monitoring features can be integrated to take into account the needs for integrity, reliability and completeness of data.
In the structure-in-5, checks and control mechanisms can be integrated at different abstractions levels assuming redundancy from different perspectives. Contrary to the layered architecture, checks and controls are not restricted to adjacent layers. Besides, since the structure-in-5 permits to separate the data and control hierarchies, integrity of these two hierarchies can also be verified independently.
The jointure venture, through its joint manager, proposes a central

message server/controller. Like in the task trees, exception mechanism, wiretapping supervising or monitoring can be supported by the joint manager to guarantee non-failability, reliability and completeness.

– **Adaptability.**

In the control loop, the robot components are separated from each other and can be replaced or added independently. Unfortunately, precise manipulation has to take place inside the components, at a level detail the architecture does not show.
In the layered architecture, the interdependencies between layers prevent the addition of new components or deletion of existing ones. The fragile relationships between the layers can become more difficult to decipher with change.
Task trees, through the use of implicit invocation, make incremental development and replacement of component straightforward: it is often sufficient to register new components, no existing one feels the impact.
The structure-in-5 separates independently each typical component of the robot architecture isolating them from each other and allowing dynamic manipulation. The structure-in-5 is restricted to no more than 5 major components then, as in the control loop, more refined tuning has to take place inside the components.
In the joint venture, manipulation of partner components can be done easily by registering new components to the joint manager. However, since partners can also communicate directly with each other, existing dependencies should be updated as well. The joint manager cannot be removed due to its central position.

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

Table II. Strengths and Weaknesses of Robot Architectures

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

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 II, 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.

6. Selecting an Architecture

To cope with software quality attributes and select the architecture of the system, we go through a means-ends analysis using the non functional requirements (NFRs) framework[2]. We refine the identified attributes to sub-attributes that are more precise and evaluates alternative architectural styles against them, as shown in Figure 11. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level attributes. The styles are represented as operationalized attributes (saying, roughly, "make the architecture of the multi agent system *pyramid, structure-in-5, joint venture, arm's-length*-based, ...").

The evaluation results in contribution relationships from the architectural styles to the quality attributes, labeled "+", "++", "–", "––". Design rationale is represented by claims drawn as dashed clouds. They make it possible for domain characteristics (such as priorities) to be considered and properly reflected into the decision making process, e.g., to provide reasons for selecting or rejecting possible solutions (+, –). Exclamation marks (! and !!) are used to mark priority attributes while a check-mark "$\surd$" indicates an accepted attribute and a cross "$\times$" labels a denied attribute.

Eventually, the analysis shown in Figure 11 allows us to choose the structure-in-5 architectural style for our mobile robot example (the operationalized attribute is marked with a "$\surd$"). The analysis uses the correlation catalogue depicted in Table II and the top level quality attributes identified.

[2] In the NFR framework, software quality attributes are called non functional requirements represented as softgoals (cloudy shapes).

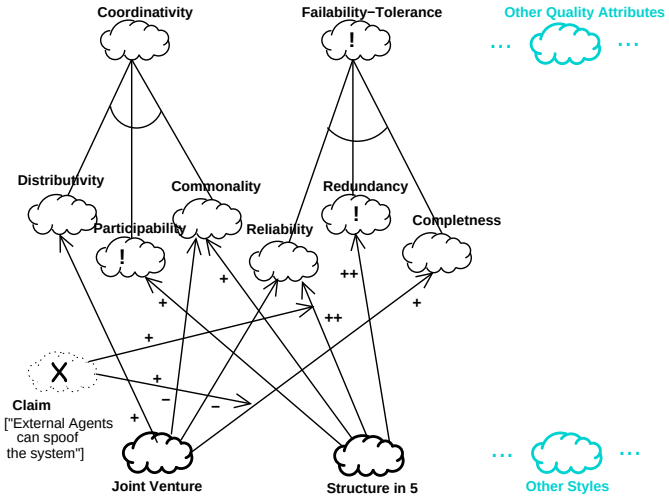

Figure 11. Partial Architecture Evaluation for Organizational Styles

Relationships types (AND, OR, ++, +, –, and – –) between quality attributes are formalized to offer a tractable proof procedure. AND/OR relationships corresponds to the classical AND/OR decomposition relationships: if attribute A_0 is AND-decomposed (respectively, OR-decomposed) into $A_1, A_2, \ldots, A_n$ then all (at least one) of the attributes must be satisfied for the attribute A_0 to be satisfied. So for instance, in Figure 11 attribute Coordinativity is AND-decomposed into attributes Distributivity, Participability, and Commonality. Relationships "+" and "– " model respectively a situation where an attribute contributes positively or negatively towards the satisfaction of another attribute. for instance, in Figure 11, Joint Venture contributes positively to the satisfaction of Distributivity attribute and negatively to the Reliability. In addition, relationships "++" and "– –" model a situation where the satisfaction of an attribute implies the satisfaction or denial of another goal. In Figure 11, for instance, the satisfaction of attribute Structure in 5 implies the satisfaction of attributes Reliability and Redundancy.

The analysis for selecting an architecture is based on propagation algorithms presented in [23]. Basically, the idea is to assign a set of initial labels for some attributes of the graph, about their satisfiability and deniability, and see how this assignment leads to the labels propagation for other attributes. In particular, we adopt from [23] both qualitative and a numerical axiomatization for goal (attributes) modeling primitives and label propagation algorithms that are shown to be sound and complete with respect to their respective axiomatization. In the following, a brief description of the qualitative algorithm.

To each attribute A, we associate two variables $Sat(A), Den(A)$ ranging in $\{F, P, N\}$ (full, partial, none) such that $F > P > N$, representing the current evidence of satisfiability and deniability of the attribute A. E.g., $Sat(A_i) \geq P$ states there is at least a partial evidence that A_i is satisfiable. Starting from assigning an initial set of input values for $Sat(A_i)$, $Den(A_i)$ to (a subset of) the attributes in the graph, we propagate the values through the propagation rules of Table III. Propagation rules for AND (respectively OR) relationship are min-value function for satisfiability (max-value function) and max-value function (min-value function) for deniability. A dual table is given for deniability propagation.

Table III. Propagation rules for satisfiability in the qualitative framework. A dual table is given for deniability propagation.

	$G_2 \overset{+}{\longmapsto} G_1$	$G_2 \overset{-}{\longmapsto} G_1$	$G_2 \overset{++}{\longmapsto} G_1$	$G_2 \overset{--}{\longmapsto} G_1$
$Sat(G_1)$	$min\{Sat(G_2), P\}$	N	$Sat(G_2)$	N
$Den(G_1)$	N	$min\{Sat(G_2), P\}$	N	$Sat(G_2)$

The schema of the algorithm is described in Figure 12. *Initial*, *Current* and *Old* are arrays of pairs $\langle Sat(A_i), Den(A_i)\rangle$, one for each A_i of the graph, representing respectively the initial, current and previous labeling status of the graph.

The array *Current* is first initialized to the initial values *Initial* given in input by the user. At each step, for every attribute A_i, $\langle Sat(A_i), Den(A_i)\rangle$ is updated by propagating the values of the previous step. This is done until a fixpoint is reached, that is, no updating is mode possible ($Current == Old$).

The updating of $\langle Sat(A_i), Den(A_i)\rangle$ works as follows. For each relation R_i incoming in G_i, the satisfiability and deniability values sat_{ij} and den_{ij} derived from the old values of the source attributes are computed by applying the rules of Table III. Then, it is returned the maximum value between those computed and the old values.

We are currently working on applying different techniques, such as Dempster Shafer theory [44], to take into consideration the sources of information in the propagation algorithms. This will allow us to consider, for instance, the reliability and/or the competence of a evidence source.

```
1       Current=Initial;
2       do
3           Old=Current;
4           for each A_i do
5               Current[i] = Update_label(i, Old);
6       until not (Current==Old);
7       return Current;
8       for each R_j s.t. target(R_j) == A_i do
9           sat_ij = Apply_Rules_Sat(i, R_j, Old);
10          den_ij = Apply_Rules_Den(A_i, R_j, Old);
11      return ⟨ max(max_j(sat_ij), Old[i].sat),
12               max(max_j(den_ij), Old[i].den) ⟩;
```

Figure 12. Schema of the label propagation algorithm.

7. A Requirements-Driven Methodology

This research is conducted in the context of the *architectural design* phase of *Tropos* [4], a software system development methodology which is founded on the concepts of actor and goal.

Tropos describes in terms of the same concepts the organizational environment within which a system will eventually operate, as well as the system itself. The proposed methodology supersedes traditional development techniques, such as structured and object-oriented ones in the sense that it is tailored to systems that will operate within an organizational context and is founded on concepts used during early requirements analysis. To this end, we adopt the concepts offered by *i** [64]. *Tropos* spans four phases of software development:

- Early requirements, concerned with the understanding of a problem by studying an organizational setting; the output is an organizational model which includes relevant actors, their goals and dependencies.
- Late requirements, in which the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design, in which the system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies.
- Detailed design, in which behaviour of each architectural component is defined in further detail.

8. Related Work

Literature on MAS offers many contributions on using organization theory concepts to model and design MAS.

In [18] 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 studies results from managements science to identify efficient agent organizations. The motivation of our work is similar: we focus on how to use concepts from organization theory to model multi-agent organization and how to apply successful organization structures for designing MAS.

The organizational perspective has been increasingly accepted within MAS community, and mathematical and computational methods have been progressively used to develop a better understanding of the fundamental principles of organizing MAS [36].

Among others, TAEMS (Task Analysis, Environment Modeling and Simulation) [11, 10] is 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.

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 [15] uses concepts such as *agent*, *group*, and *role* to model the organizational structure of multi-agent systems. By this model, different types of organizational behavioral requirement patterns have been defined and formalized [16]. Similarly, in Gaia methodology [62], role and interaction models are used for analyzing the understanding of the system and its structure. The organization of the system is viewed as a collection of roles, that stand in certain relationship on another, 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, the organization description does not includes the goals associated to the agents, groups, and roles, namely goals are not part of the organization description.

Multi-agent organization theory [33] uses components such as scenario, organizational position, goals, plans, membership, and constraints

to specify organizational structure. Although, the theory can be used for designing multi-agent systems, its principle aim is extending 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 the theory of organizational design.

Finally, other research results in multi-agent organizational design includes Self-organization Design [31] and Teamwork Models [39].

Self-organization Design is based on the idea to have an organization in which one or more members can monitor the organizational structure's effectiveness in directing organizational activities, design new organizational structures appropriate to new situation, 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. [55, 56, 57] 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 organization.

Teamwork models [58], based on the joint intentions framework introduced by Levesque et al. [39], allow to design multi-agent systems in which individual agents are provided with an explicit representation of the team goals and plans, an underlying explicit model of team activity.

9. Conclusion

We are working towards the definition of a collection of specific organizational architecture styles for designing agent-based systems. Since the fundamental concepts of agent systems are intentional and organizational, 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 or private goals.

We propose to use human organizations as a metaphor to suggest a set of generic styles for agent systems, with a preference for organizational design theories over social emergence theories.

To this end, the paper has proposed architectural styles for MAS inspired from organization theory that describes the internal structure and design of organizations and from theories for strategic alliances that model the collaboration of independent organizations that pursue agreed goals.

In particular we have detailed and adapted 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. A semi-formal specification has been also given for each of them in Formal Tropos.

The contribution also includes the presentation and evaluation of software qualities identified for these styles and a comparison of organizational and conventional styles conducted on a mobile robot case study taken from the Software Engineering literature. We have also presented a framework to select architectural styles with respect to identified software quality attributes.

Future research directions will extend and formalize precisely the catalogue of organizational styles and characterize specifically how a particular model can be seen as an instance of a style.

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

The organizational styles should eventually constitute an architectural macro level. At a micro level we will focus on the notion of social agent patterns. Many existing patterns can be incorporated into system architecture, such as those identified in [20, 47]. For agent inherent characteristics, patterns for distributed, and open architectures like the broker, matchmaker, embassy, mediator, wrapper, mediator are more appropriate [28, 60]. They will detail how goals and dependencies identified in an organizational style can be refined and achieved.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.
2. S. Bennett, S. McRobb, and R. Farmer. *Object-Oriented Systems Analysis and Design using UML*. McGraw Hill, 1999.
3. P. A. Bonatti, S. Kraus, J. Salinas, and V. S. Subrahmanian. Data-security in heterogeneous agent systems. In M. Klusch and G. Weiss, editors, *Cooperative Information Agents*, pages 290–305. Springer Verlag, 1998.
4. J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering, CAiSE'01*, pages 108–123, Interlaken, Switzerland, June 2001.
5. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: The Tropos Project. *Information Systems*, 2002.
6. K. Cetnarowicz, G. Dobrowolski, M. Kisiel-Dorohinicki, and E. Nawarecki. Functional integrity of mas through the dynamics of the agents' population. In *Proc. of the 3rd Int. Conf. on Multi Agent Systems, ICMAS'98*, 1998.

7. D. M. Chess. Security issues in mobile code systems. In G. Vigna, editor, *Mobile Agents and Security*, pages 1–14. Springer Verlag, 1998.
8. L.K. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
9. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.
10. K. S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.
11. K. S. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 217–224, 1993.
12. J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *Knowledge Engineering Review*, 12(3):309–314, 1997.
13. E. H. Durfee and V. R. Lesser. Predictability versus responsiveness: Coordinating problem solvers in dynamic domains. In *Proc. of the 7th National Conf. on Artificial Intelligence*, pages 66–71, 1988.
14. P. Dussauge and B. Garrette. *Cooperative Strategy: Competing Successfully Through Strategic Alliances*. Wiley and Sons, 1999.
15. J. Ferber and O. Gutknecht. A meta-model for the anlysis and design of organizations in multi-agent systems. In *Proc. of the 3rd Int. Conf. on Multi Agent Systems, ICMAS'98*, June 1998.
16. J. Ferber, O. Gutknecht, C. M. Jonker, J. P. Müller, and J. Treur. Organization models and behavioual requirements specification for multi agent systems. In *Proc. of the ECAI2001*, 2000.
17. FIPA. The Foundation for Intelligent Physical Agents. At http://www.fipa.org, 2001.
18. M. Fox. An organizational view of distributed systems. *Transactions on Systems, Man and Cybernetics*, 11(1):70–80, 1981.
19. A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specification in Tropos. In *Proc. of the 5th Int. Symposium on Requirements Engineering, RE'01*, Toronto, Canada, August 2001.
20. E. Gamma, R. Helm, J. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
21. D. Garlan and M. Shaw. An introduction to software architectures. *Advances in Software Engineering and Knowledge Engineering*, 1, 1993.
22. P. Giorgini, M. Kolp, and J. Mylopoulos. Organizational patterns for early requirements analysis. Submitted to the *IEEE Joint International Requirements Engineering Conference 2002, RE 2002*, August 2002, Essen, Germany, 2002.
23. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002)*, Tampere, Finland, October 2002.
24. P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, and P. Bresciani. Agent-Oriented Software Development: A Case Study. In S. Sen J.P. Müller, E. Andre and C. Frassen, editors, *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE01)*, Buenos Aires - ARGENTINA, June 13 - 15 2001.
25. GMT. Gmt consulting group. http://www.gmtgroup.com/, 2002.
26. B. Gomes-Casseres. *The alliance revolution: the new shape of business rivalry*. Harvard University Press, 1996.

27. D. Gordon. APT agents: Agents that are adaptive, predictable, and timely. In *Proc. of the 1st Goddard Workshop on Formal Approaches to Agent-Based Systems, FAABS'00*, 2000.
28. S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multiagent coordination. In *Proc. of the 3rd Int. Conf. on Autonomous Agents, Agents'99*, Seattle, USA, May 1999.
29. Q. He and K. Sycara. Towards a secure agent society. In *Proc. of the Workshop on Deception, Fraud and Trust in Agent Societies, ACM AA'98*, 1998.
30. B. Horling, V. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Technical Report UM-CS-1999-003, University of Massachusetts, 1999.
31. T. Ishida, L. Gasser, and M. Yokoo. Organization self-design of distributed production systems. *Transaction on knowledge and data engineering*, 4, 1992.
32. N. R. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligenc*, pages 187–210. Wiley, 1996.
33. S. Kirn and L. Gasser. Organizational approaches to coordination in multiagent systems. *Informationstechnik und Technische Informatik (IT+TI)*, 4(40), 1998.
34. S. Kumar and P. R. Cohen. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proc. of the 4th Int. Conf. on Multi Agent Systems, ICMAS'00*, 2000.
35. Lego. Lego Mindstorms Robotics Invention System. At http://mindstorms.lego.com, 2002.
36. V. R. Lesser. Cooperative multiagent systems: a personal view of the state of the art. *Transaction on Knowledge and Data Engineering*, 11(1), 1999.
37. H. Levesque and M. Pagnucco. Legolog. http://www.cs.toronto.edu/cogrobo/Legolog/, 2000.
38. H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 1997.
39. H.J. Levesque, P.R. Cohen, and J. Nunes. On acting together. In *Proc. of the National Conf. on Artificial Intelligence*, Menlo Park, USA, 1990.
40. T. Lozano-Perez. Preface. In L. Cox and G Wilfong, editors, *Autonomous Robot Vehicles*. Springer Verlag, 1990.
41. H. Mintzberg. *Structure in fives : designing effective organizations*. Prentice-Hall, 1992.
42. J. Morabito, I. Sack, and A. Bhate. *Organization modeling : innovative architectures for the 21st century*. Prentice Hall, 1999.
43. A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents*. Springer Verlag, 2001.
44. S. Parsons. Some qualitative approaches to applying the Dempster-Shafer theory. *Information and Decision technologies*, 19:321–337, 1994.
45. A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proc. of the 5th Int. Conf on Autonomous Agents, Agents'01*, Montreal, Canada, May 2001.
46. M.E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. New York, The Free Press, 1985.
47. W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.

48. J. Sabater, C. Sierra, S. Parsons, and N. R. Jennings. Engineering executable agents using multi-context systems. *Journal of Logic and Computation*, 2001. (to appear).
49. W.R. Scott. *Organizations: rational, natural, and open systems.* Prentice Hall, 1998.
50. L. Segil. *Intelligent business alliances : how to profit using today's most important strategic tool.* Times Business, 1996.
51. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.
52. O. Shehory. Architectural properties of multi-agent systems. Technical Report CMU-RI-TR-98-28, Carnegie Mellon University, 1998.
53. R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan. A modular architecture for office delivery robots. *IEEE Control Systems*, 1:49–56, 1992.
54. R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan. A modular architecture for office delivery robots. In *Proc. of the 1st Int. Conf. on Autonomous Agents, Agents '97 Agent'97*, pages 245–252, Marian del Rey, USA, February 1997.
55. Y. P. So and E. H. Durfee. An organizational self-design model for organizational change. In *Working Notes of the AAAI-93 Workshop on AI and Theories of Groups and Organizations*, 1993.
56. Y. P. So and E. H. Durfee. Modeling and designing computational organizations. In *Working Notes of the AAAI Spring Symposium on Computational Organization Design*, 1994.
57. Y. P. So and E. H. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219–246, 1996.
58. M. Tambe. Teamwork in real-world, dynamic environments. In *Proc. of the 2nd Int. Conf. on Multi Agents Systems, ICMAS'96*, Kyoto, Japan, 1996.
59. G. Weiss, editor. *Learning in DAI Systems.* Springer Verlag, 1997.
60. S.G. Woods and M. Barbacci. Architectural evaluation of collaborative agent-based systems. Technical Report CMU/SEI-99-TR-025, SEI, Carnegie Mellon University, Pittsburgh, USA, 1999, 1999.
61. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practic. *Knowledge Engineering Review*, 2(10), 1995.
62. M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi Agent Systems*, 3(3):285–312, 2000.
63. M.Y. Yoshino and U. Srinivasa Rangan. *Strategic alliances : an entrepreneurial approach to globalization.* Harvard Business School Press, 1995.
64. E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, 1995.