A Social Ontology for Information System Development

Manuel Kolp[1] **Paolo Giorgini**[2]

[1] IAG - Information Systems Research Unit - University of Louvain, 1 Place des Doyens, B-1348 Louvain-La-Neuve, Belgium, tel.: 32-10 47 83 95, kolp@isys.ucl.ac.be

[2] Department of Mathematics - University of Trento, 14 via Sommarive, I-3850, Trento, Italy, pgiorgini@science.unitn.it

Abstract

Organizations are changing at an ever-faster pace, as they try to keep up with globalization and the information revolution. Unfortunately, information systems technologies do not support system evolution well, making information systems a roadblock to organizational change. We propose to view information systems as social structures and define methodologies which develop and evolve seamlessly an information system within its operational environment.

To this end, this paper proposes an ontology for information systems that is inspired by social and organizational structures. The ontology adopts components of the *i** organizational modeling framework, which is founded on the notions of *actor*, *goal* and *social dependency*. Social patterns, drawn from research on cooperative and distributed architectures, offer a more macroscopic level of social structure description. Finally, the proposed ontology includes organizational styles inspired from organization theory. These are used not only to model the overall organizational context of an information system, but also its architecture. Social patterns and organizational styles are defined in terms of configurations of *i** concepts. The research has been conducted in the context of the *Tropos* project.

Keywords
Information systems, software development methodology, organization modeling, software architectures.

1 Introduction

Information systems have traditionally suffered from an impedance mismatch. Their operational environment is understood in terms of actors, responsibilities, dependencies, social structures, organizational entities, objectives, tasks and resources, while the information system itself is usually conceived as a collection of (software) modules, entities (e.g., objects, agents), data structures and interfaces. This mismatch is one of the main factors for the poor quality of information systems, and for the frequent failure of system development projects.

We are interested in developing an information system methodology, called *Tropos* [Cas01], which views information systems as social structures thereby reducing the impedance mismatch alluded to earlier. *Tropos* is intended as a seamless methodology tailored to describe both the organizational environment of a system and the system itself in terms of the same concepts. By social structures, we mean a collection of social actors, human or software, which act as agents,

positions (e.g., the department chair), or roles (e.g., the meeting chair) and have social dependencies among them (e.g., the meeting chair depends on the meeting participants to show up, while they depend on the chair to conduct an effective meeting).

The *Tropos* ontology is described at three levels of granularity. At the lowest (finest granularity) level, *Tropos* adopts concepts offered by the *i** organizational modeling framework [Yu95], such as *actor*, *agent*, *position*, *role*, and *social dependency*. At a second, coarser-grain level the ontology includes possible *social patterns*, such as mediator, broker and embassy. At a third, more macroscopic level the ontology offers a set of *organizational styles* inspired by organization theory and strategic alliances literature. All three levels are defined in terms of the *i** concepts.

The *Tropos* methodology 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, where the system-to-be is described within its operational environment, along with relevant functions and qualities.

• Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies.

• Detailed design, where behavior of each architectural component is defined in further detail.

For purposes of presentation, we describe first *i**, then the organizational styles and finally the social patterns. The rest of the paper is organized as follows. Section 2 shows how *Tropos* can be used to produce and formalize an initial *i** organization model. Section 3 presents the organization-inspired styles, and their application to the kind of models presented in Section 2. Section 4 proposes a number of social goal-based patterns. Section 5 proposes a *Formal Tropos* specification of one of our styles. Finally, Section 6 summarizes the contributions and points to further work.

2 Initial Organizational Models

Tropos adopts a goal- and actor-oriented ontology for modeling organizational settings based on *i** [Yu95]. It assumes that an organization involves *actors* who have *strategic dependencies* among each other. 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 are used to 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 the fulfillment can occur 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 respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form *depender* → *dependum* → *dependee*.

These elements are sufficient for producing a first model of an organizational environment. For instance, Figure 1 depicts an *i** model of a business organization – *Media Producer* -- producing

media items (books, newspapers, CDs, movies, games, …). The main actors are *Editor*, *Record Label*, *Movie Studio* and *Game Design*. Each partner actor composing *Media Producer* is specialized in one or several specific media production areas: *Editor* handles press business and contributes to provide film scenarios and script ideas, *Movie Studio* makes films and video clips for *Record Label* which handles record deals and records soundtracks for *Movie Studio* and *Game Design*. This last actor develops games and relies on *Movie Studio* for game plots. Each actor is responsible for his own business management. The joint management actor, *Production Cpy*, handles the corporate strategic management. *Production Cpy* depends on *Game Design* to fulfill her goal: Develop Games. *Editor* depends on *Production Cpy* to "*handle efficiently press business*". Since the dependum *handle efficiently press business* cannot be defined precisely, it is represented as a softgoal.

Game Design depends on *Record Label* to have the soundtrack of the game recorded (task dependency). It also depends on *Movie Studio* to get the game scenario (resource dependency) of the movie from which a video game can be derived.

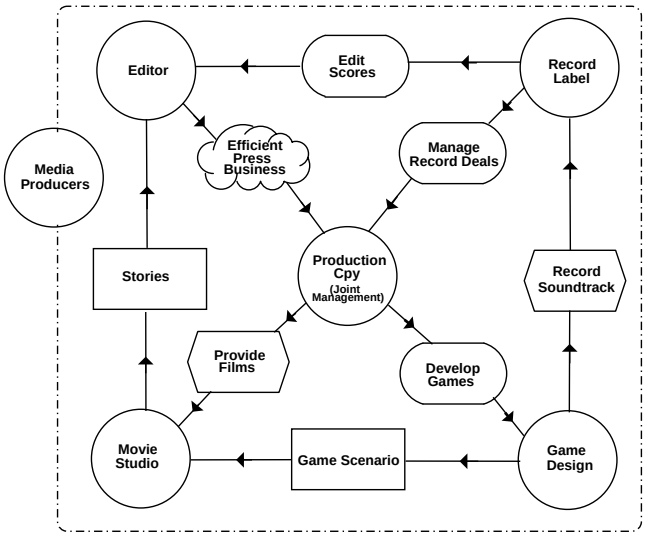

Figure 1 : *i** Model for a Media Producer

We have defined a formal language, called *Formal Tropos* [Fux01], that offers all the primitive concepts of *i** (such as actors, goals and dependencies), but supplements them with a rich specification language inspired by KAOS [Dar93]: it provides a textual notation for *i** models and allow us to describe dynamic constraints among the different elements of the specification in a first order linear-time temporal logic. It has also a precisely defined semantics that is amenable to formal analysis. In the following we present a part of the *Formal Tropos* formalization of the *Media Producer* depicted in Figure 1. In particular, we focus on the *Production Cpy* goal of applying a movie and game strategy.

We specify that there will be just one *ProductionCpy*, which has the goals of *drawing up a Game Movie Contract* and *applying the Game Movie Strategy*. The goal *DrawUpGameMovieContract* is fulfilled when there is a contract signed by a *Movie Studio* actor and a *Game Design* actor. The goal of *applying the game movie strategy* is fulfilled when for each action movie there is a game about the movie, which will be delivered within a month from the date of the movie delivery.

Entity Movie
Attribute constant type:{action, love, thriller, comedy , drama, sciences-fiction}
delivery_date : Date;
Entity Game
Attribute constant movie_ref : Movie, delivery_date: Date;

Entity Scenario
Attribute constant movie_ref : Movie, game_ref: Game;

Entity GameMovieContract
Attribute constant conditions: Conditions, ms : MovieStudio,
gd : GameDesign, signature_date :Date
Actor ProductionCpy
Creation condition ¬∃ pCpy: ProductionCpy
Goal DrawUpGameMovieContract
Mode maintain
Fulfilment
definition ∃ contract: GameMovieContract(
∃ ms:MovieStudio(contract.ms=ms) ^
∃ gd:GameDesign(contract.gd=gd))
Goal ApplyGameMovieStrategy
Mode maintain
Fulfilment
condition ◆Fulfilled(DrawUpGameMovieContract)
definition ∀movie:Movie (movie.type=action → (
∃ game:Game(game.movie_ref=movie ^
game.delivery_date≥movie.delivery_date ^
game.delivery_date ≤ (1 month + movie.delivery_date)))

The following describes a goal and a resource dependency. The *DevelopGame* dependency is created when there is a new action movie for which there is no games, and it is fulfilled when there will be at least one game for such a movie.

Dependency DevelopGame
Type goal
Mode Achieve
Depender ProductionCpy
Dependee GameDesign
Attribute constant movie: Movie, contract: GameMovieContract
Creation
condition movie.type=action ^ ¬∃ game:Game(game.movie_ref=movie) ^
contract.gd=dependee
trigger JustCreated(movie)
Fulfilment
condition for depender
∃ game:Game(game.movie_ref=movie)

Here, the *GameScenario* dependency applies when a new game has to be developed and no scenario for such game exists. The dependency is fulfilled when the Movie Studio provides the scenario.

Dependency GameScenario
Type resource **Mode maintain**
Depender GameDesign

Dependee MovieStudio
Attribute constant game: Game, scenario: Scenario, contract: GameMovieContract
Creation
condition ¬∃ scenario: Scenario(scenario.game_ref=game) ^
(contract.gd=depender ^ contract.ms=dependee)
trigger JustCreated(game)
Fulfilment
condition for depender
scenario.game_ref=game

3 Organizational Styles

Organizational theory [Min92, Sco98] and strategic alliances literature [Gom96, Seg96, Yos95] study alternative styles for (business) organizations. These styles are used to model how business stakeholders -- individuals, physical or social systems – coordinate in order to achieve common goals. *Tropos* adopts (some of these) *organizational styles* at the macroscopic level of its ontology in order to describe the overall structure of the organizational context of the system or its architecture. In this section, we explain some of these styles in terms of the basic ontology introduced in the previous section.

The **structure-in-5** (Figure 2) is a typical organizational style. At the base level, the *Operational Core* takes care of the basic tasks -- the input, processing, output and direct support procedures -- associated with running the organization. At the top lies the *Apex*, composed of strategic executive actors. Below it, sit the *Coordination*, *Middle Agency* and *Support* actors, who are in charge of control/standardization, management and logistics procedures, respectively. The *Coordination* component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the organization adapt to its environment. Actors joining the apex to the operational core make up the *Middle Agency*. The *Support* component assists the operational core for non-operational services that are outside the basic flow of operational tasks and procedures.

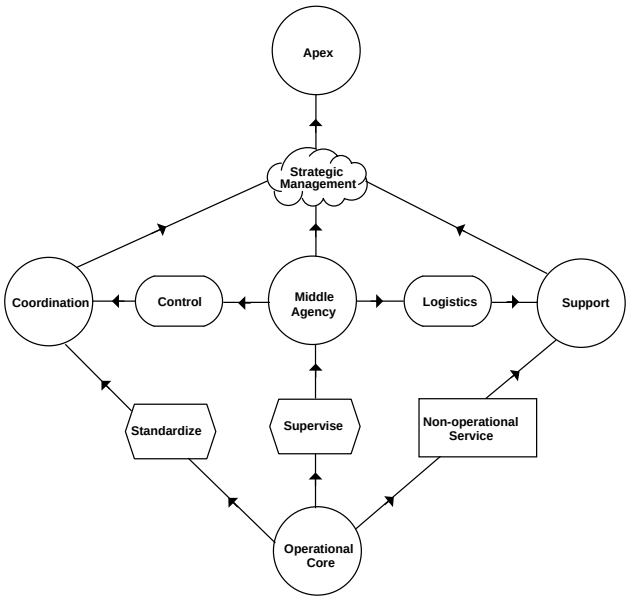

Figure 2 : Structure-in-5

The **joint venture** style (Figure 3) is a more decentralized style that involves an agreement between two or more principal partners in order to obtain the benefits derived from operating at a larger scale and reusing the experience and knowledge of the partners.

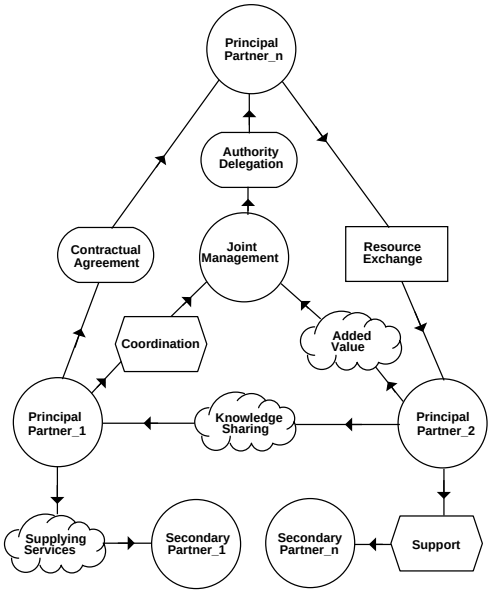

Figure 3 : Joint Venture

Each principal partner can manage and control itself on a local dimension and interact directly with other principal partners to exchange, provide and receive services, data and knowledge. However, the strategic operation and coordination is delegated to a *Joint Management* actor, who coordinates tasks and manages the sharing of knowledge and resources. Outside the joint

venture, secondary partners supply services or support tasks for the organization core.

The **takeover** style involves the total delegation of authority and management from two or more partners to a single collective *takeover* actor. It is similar in many ways to the joint venture style. The major and crucial difference is that while in a joint venture identities and autonomies of the separate units are preserved, the takeover absorbs these critical units in the sense that no direct relationships, dependencies or communications are tolerated except those involving the takeover.

The **pyramid** style is the well-know hierarchical authority structure. Actors at lower levels depend on those at higher levels for supervision. The crucial mechanism is 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 **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 **vertical integration** style merges, backward or forward, several actors engaged in achieving or realizing related goals or tasks at *different stages* of a production process. An *Organizer* merges and synchronizes interactions/dependences between participants, who act as intermediaries. Figure 4 presents a vertical integration style for the domain of goods distribution. *Provider* is expected to supply quality products, *Wholesaler* is responsible for ensuring their massive exposure, while *Retailer* takes care of the direct delivery to the *Consumers*.

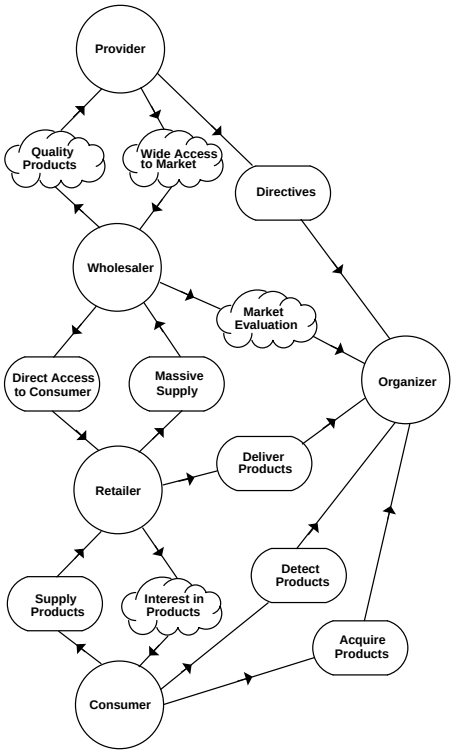

Figure 4 : Vertical Integration

The **hierarchical contracting** style (Figure 5) identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority.

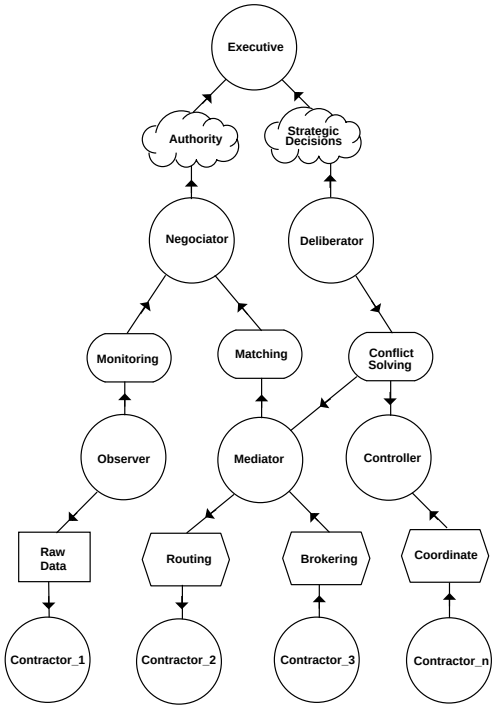

Figure 5 : Hierarchical Contracting

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 (top to bottom) restrict autonomy and underlie a cooperative venture between the contracting parties.

The **bidding** style (Figure 6) involves competitivity mechanisms, and actors behave as if they were taking part in an auction. The *Auctioneer* actor runs the show, advertises the auction issued by the auction *Issuer*, receives bids from *Bidder* actors and ensures communication and feedback with the auction *Issuer*. The auction *Issuer* is responsible for issuing the bidding.

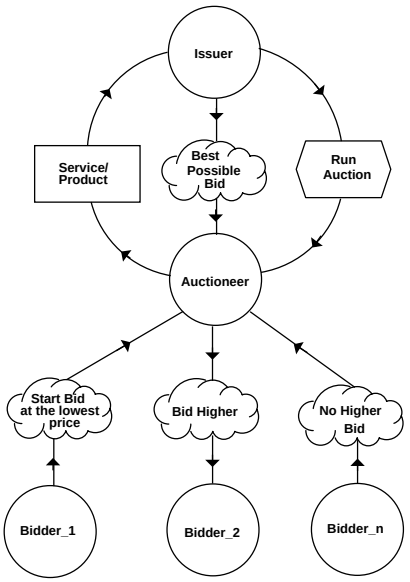

Figure 6 : Bidding

The **co-optation** style (Figure 7) involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of an initiating organization. By co-opting representatives of external systems, organizations are, in effect, trading confidentiality and authority for resource, knowledge assets and support.

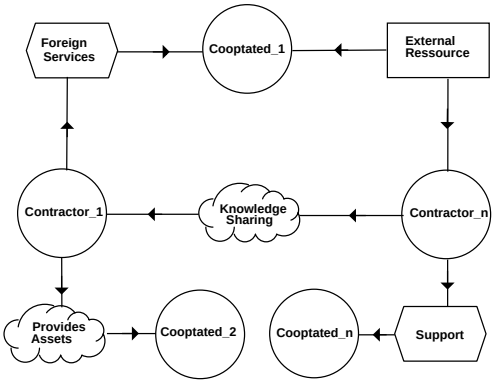

Figure 7 : Co-optation

The initiating system has to come to terms with the contractors 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.

Organizational styles guide the development of the organizational model for a system. For instance, suppose that we detect that the organizational style for the Media Company example of the previous Section can be represented as a *vertical integration.* Then, the initial organizational model of Figure 1 can be refined and completed as shown in Figure 8.

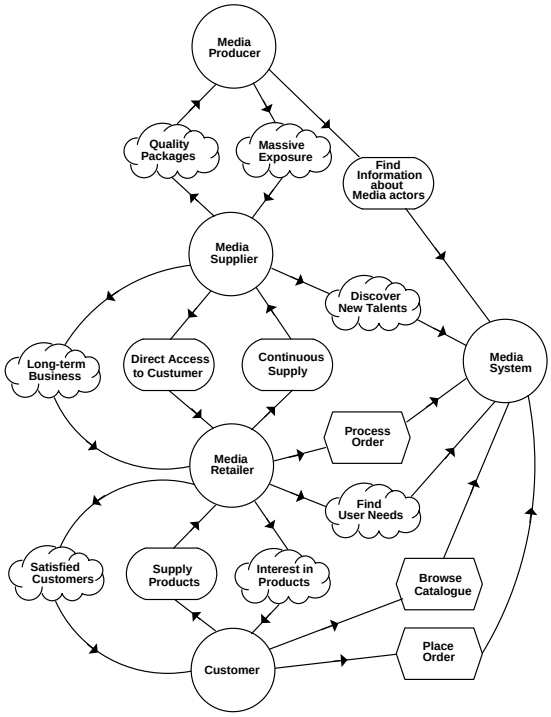

Figure 8 : Modeling the Organizational Context of the Media Company with the Vertical Integration Style

The model is an instantiation of the *vertical integration* style of Figure 4. The *Customer* takes the role of *Consumer*, *MediaProducer* assumes the position of *Provider*, and *Media System* the role of *Organizer*. *Media Producer* is expected to provide quality products, *Media Supplier* ensures massive exposure of media items while *Media Retailer* interacts with the *Customer*. The information system is also introduced as a full-fledged organizational actor, and each of the human stakholders uses the *Media system* for her particular needs and goals. For instance, *MediaProducer* wants to find information about the media market and stakeholders; *MediaSupplier* would like to find and promote new ideas, projects and talents to increase its market share while *MediaRetailer* needs to be provided with e-commerce facilities to satisfy customers. Finally, *Customer* would like to consult product catalogues and place orders.

Tropos aims to apply its social ontology not only to organizational models, but also to all levels of software development (most notably, architectural design). For instance, the *joint venture* style can be used to produce an architectural description of the *Media System*. A more detailed description of this particular architecture can be found in [Kol01]. Figure 9 suggests a possible assignment of system responsibilities for the business-to-consumer (B2C) part of the *Media System*. Following the joint venture style, the architecture is decomposed into three principal partner actors (*Store Front, Order Processor* and *Back Store*). They control themselves on a local dimension for exchanging, providing and receiving services, data and resources with each other.

Each of the three system actors delegates authority to and is controlled and coordinated by the joint management actor (*Joint Manager*), managing the system on a global dimension. *Store Front* interacts primarily with *Customer* and provides her with a usable front-end Web application. *Back Store* keeps track of all Web information about customer orders, product sales, bills and other data of strategic importance to *MediaRetailer. Order Processor* is in charge of

the secure management of orders and bills, and other financial data. *Joint Manager* manages all of them handling *Security* gaps, *Availability* bottlenecks and *Adaptability* issues, three software quality attributes (as well as sub-attributes *Authorization*, *Integrity*, *Usability*, *Updatability* and *Maintainability*) required for business-to-consumer applications identified and evaluated in detail for our *Media system* example in [Kol01].

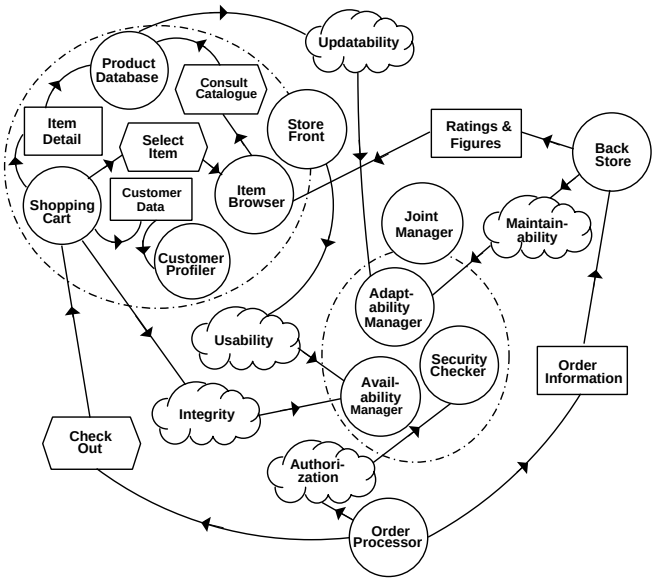

Figure 10 : Designing the System Architecture with the Joint Venture Style

All the system actors of Figure 10 will eventually be further specified into subactors, and delegated with specific responsibilities. For instance, in the *Store Front*, *Item Browser* is delegated the task of managing catalogue navigation; *Shopping Cart*, the selection and customization of items; *Customer Profiler*, the tracking of customer data and the production of client profiles; and *Product Database*, the management of media items information. Similarly, to cope with *Security*, *Availability* and, *Adaptability*, *Joint Manager* is further refined into three new system sub-actors *Security Checker*, *Availability Manager* and *Adaptability Manager*. Further decomposition details can be found in [Kol01].

4 Social Patterns

The last element of our ontology are the *social patterns*. Unlike organizational styles, they focus on the social structure necessary to achieve *one* particular goal, instead of the overall goals of the organization.

A social pattern defines the actors (together with their roles and reponsibilites) and the social dependencies that are necessary for the achievement of the goal.

Considerable work has been done in software engineering for defining software patterns (see e.g., [Gam95, Pre95, Bus96]); unfortunately, they do not place emphasis on social aspects. On

the other hand, proposals of patterns that address social issues (see e.g., [Ari98, Deu99, Ken98]) are not intended to be used at an organizational level, but rather during implementation phases by addressing issues such as agent communication, information gathering from information sources, or connection setup.

In the following, we present some social patterns that focus on social mechanisms recurrent in multi-agent and cooperative systems literature; in particular, the following structures are inspired by the federated patterns introduced in [Hay99, Woo99]. As with organizational styles, patterns are also metastructures that can be instantiated to model/design a specific application context/architecture (See Figure 17).

A **broker** (Figure 11) is an arbiter and intermediary who has access services of an actor (*Provider*) in order to satisfy the request of a *Consumer*. This pattern is especially used in the hierarchical contracting and joint venture styles. Notice that roles are established in the context of a particular interaction. For instance, *Consumers* may be in turn *Providers*, and vice versa.

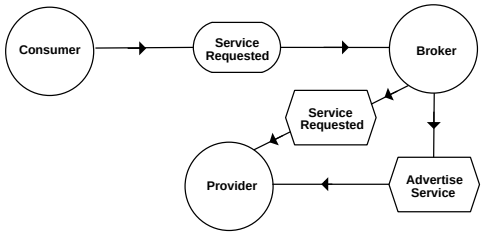

Figure 11 : Broker

A **matchmaker** (Figure 12) locates a *Provider* that can handle a *Consumer*'s request for service, and then directs the *Consumer* to the chosen *Provider*. As opposed to the *Broker* who handles all interactions between the *Consumer* and the *Provider*, the *Matchmaker* only makes the connection, and leaves all further interaction to be done directly between the intervening actors. It can also be used in hierarchical contracting and joint ventures.

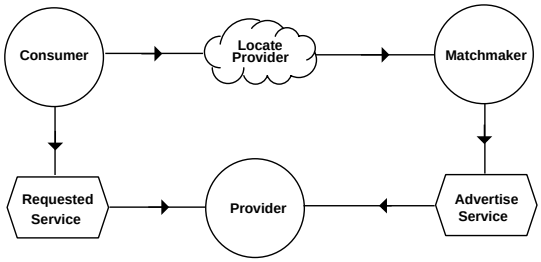

Figure 12 : Matchmaker

A **mediator** (Figure 13) mediates interactions among different actors. An *Initiator* addresses the *Mediator* instead of asking directly another colleague, the *Performer*. It has acquaintance models of colleagues and coordinates the cooperation between them. Inversely, each colleague has an acquaintance model of the *Mediator*. While a broker simply matches providers with consumers, a *Mediator* encapsulates interactions and maintains models of initiators and performers behaviors over time. It is used in the pyramid, vertical integration and hierarchical contracting styles since it underlies direct cooperation and encapsulation features reinforcing

authority.

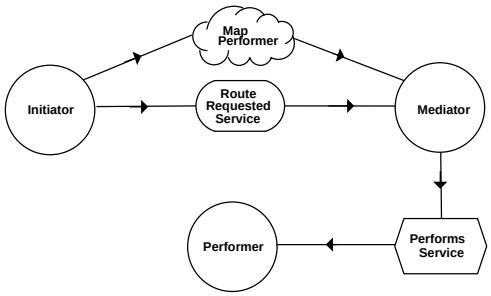

Figure 13 : Mediator

A **monitor** (Figure 14) alerts a *Subscriber* about relevant events. It accepts subscriptions, requests notifications for subjects of interest, receives such notifications, and alerts subscribers of relevant events. The *Subject* provides notifications of state changes as requested. The *Subscriber* registers for notification of state changes to distributed subjects, receives notifications with current state information, and updates its local state information. This pattern is used in the hierarchical contracting, vertical integration, arm's-length and bidding styles implying observation activities.

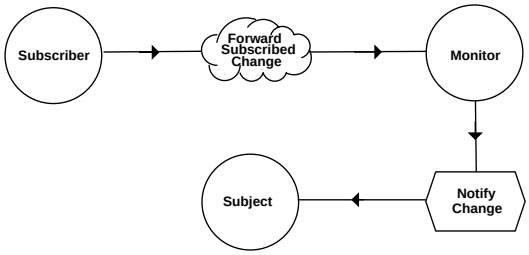

Figure 14 : Monitor

An **embassy** (Figure 15) routes a service requested by a foreign actor (*Foreigner*) to a local one, and handles back the response. If the access is granted, the *Foreigner* can submit messages to the *Embassy* for translation. The content is translated in accordance with a standard ontology. Translated messages are forwarded to target local actors. The results of the query are passed back to the *Foreigner*, and translated in reverse. This pattern can be used in the structure-in-5, arm's-length, bidding and co-optation styles to take in charge security aspects between systems component related to the competitivity mechanisms inherent to these styles.

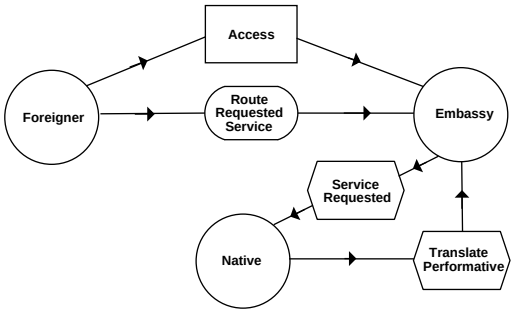

Figure 15 : Embassy

A **wrapper** is an embassy that incorporates a legacy system into the organization. The wrapper interfaces the clients to the legacy by acting as a translator between them. This ensures that communication protocols are respected and the legacy system remains decoupled from the clients. This pattern can be used in the co-optation style when one of the co-optated actor is a representative for a legacy system.

The **contract-net** pattern (Figure 16) selects an actor to which to assign a task. The pattern involves a manager (*Contractor*) and any number of participants (*Clients*). The manager issues a request for proposal for a particular service to all participants, and then accepts "proposals" to meet the service request at a particular "cost". The manager selects one participant who performs the contracted work and informs the manager upon completion. This pattern is especially used in the arm's-length and bidding and co-optation styles due to their inherent competitive features.

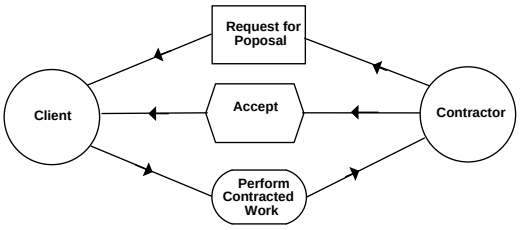

Figure 16 : Contract-net

A detailed analysis of a pattern allows to define a set of capabilities associated with the roles assigned to the actors of the pattern. Due to the lack of space, Table 1 only presents the set of capabilities for the broker pattern.

A capability states that an actor is able to act in order to achieve a given goal. In particular, for each capability the actor has a set of plans that may apply in different situations. A plan describes the sequence of actions to perform and the conditions under which the plan is applicable. It is important to notice that we have common capabilities for different actors; for instance, the capability "handle services ontology" is common to the three actors of the *Broker* pattern. Capabilities are collected in a catalogue and associated to the pattern. This allows to define the actors' role and capabilities suitable for a particular domain.

BROKER	
Actor	**Capabilities**
Customer	- Build a request to query the Broker - Handle services ontology - Query the broker for a service - Find alternative brokers - Manage possible broker failures - Monitor the broker's ongoing processes - Ask the broker to stop the requested service
Provider	- Handle services ontology - Advertise a service to the appropriate broker - Withdraw the advertisement - Use an agenda for managing the requests - Inform the broker of the acceptance of the request service - Inform the broker of a service failure

	- Inform the broker of success of a service
Broker	- Update the local database - Handle services ontology - Use an agenda for managing the customer requests - Follow the status of the requested services - Search the name of an agent to ask a service - Inform the customer of the impossibility for a service - Inform the customer of a service failure - Inform the customer of the success of a service - Request a service to a provider - Manage possible provider failures - Monitor the provider's ongoing processes - Ask the provider to stop a requested service

Table 1 : Capabilities for the Broker Pattern

Figure 17 shows a possible use of the patterns in the e-business system shown in Figure 10. In particular, it shows how to solve the goal of managing catalogue navigation that the *Store Front* has delegated to the *Item Browser*. The goal is decomposed into different subgoals and solved with a combination of patterns.

The broker pattern is applied to the *Info Searcher*, which satisfies requests of searching information by accessing *Product Database*. The Source *Matchmaker* applies the matchmaker pattern locating the appropriate source for the *Info Searcher*, and the monitor pattern is used to check any possible change in the *Product Database*. Finally, the mediator pattern is applied to mediate the interaction among the *Info Searcher*, the *Source Matchmaker*, and the *Wrapper*, while the wrapper pattern makes the interaction between the *Item Browser* and the *Product Database* possible. Of course, other patterns can be applied. For instance, we could use the contract-net pattern to select a wrapper to which delegate the interaction with the *Product Database*, or the embassy to route the request of a wrapper to the *Product Database*.

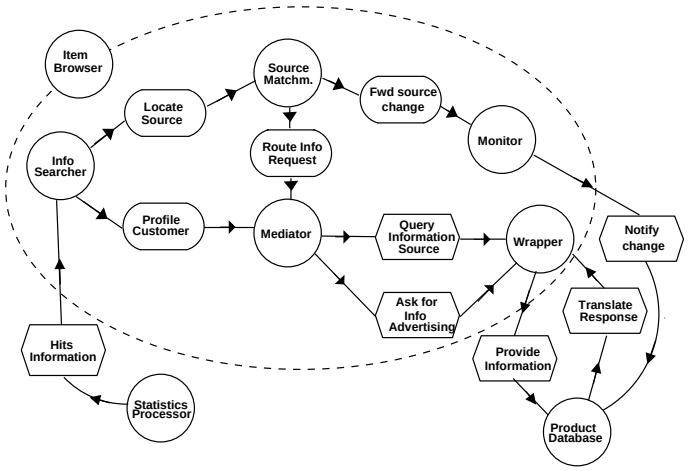

Figure 17 : Social Patterns for Item Browser

5 Formalizing Social Structures

In this section we describe in more detail one of our social structures: the structure-in-5 style. To specify the structure and formal properties of the style, we use *Formal Tropos* [Fux01].

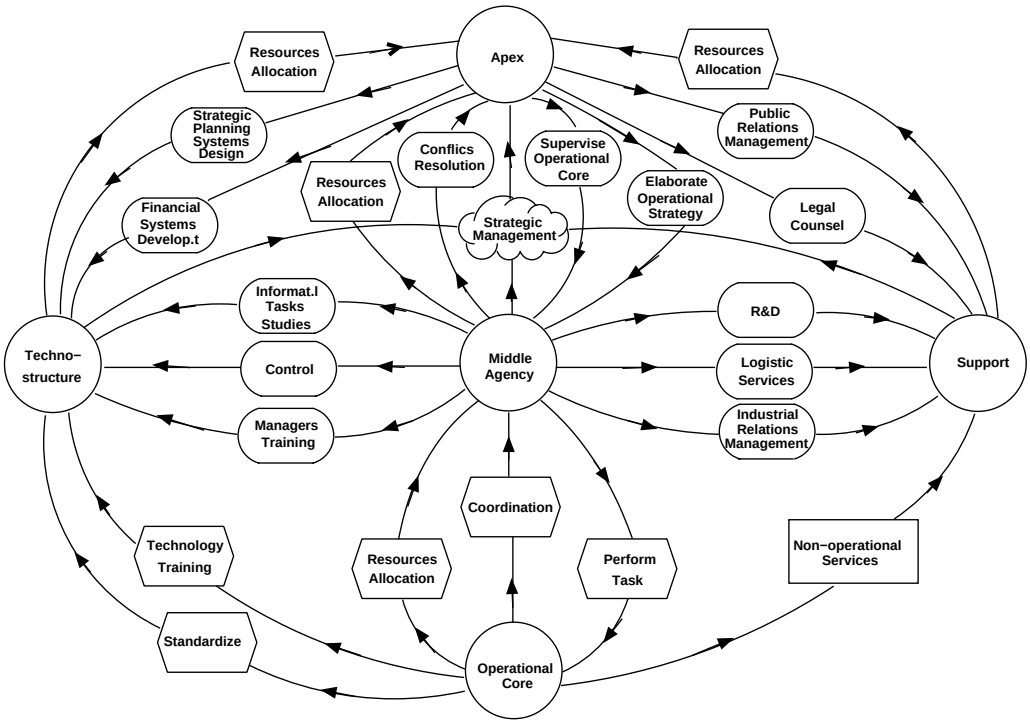

Figure 18. Structure-in-5 in detail

As Minztberg points out [Min92], the structure of an organization can be defined as the sum total of the way in which its labor is divided into distinct tasks, along with the coordination mechanisms used to achieve these tasks. The elements of the structure should be selected to achieve an internal consistency as well as a basic consistency with the organization's situation, namely its size, its age, the kind of the environment in which it functions, the technical systems it uses, and the like. Minztberg proposes a basic structure for organizations (for us, organizational style) based on fives subunits, hence its name *structure-in-5* (Figure 1). This decomposition allows one to apply alternative coordination mechanisms (such as mutual adjustment, direct supervision, standardization of skills, outputs and work processes) and design parameters (such as job specialization, behavior formalization, decentralization, unit size, and unit grouping) in order to analyze the different behaviors of the organization.

Basically, this structure defines a hierarchy of roles inside the organization, the responsibilities associated with each subunits, and inter-dependencies among them. Figure 18 shows a more detailed *i** strategic dependency model for this style.

At the base level one finds the Operational Core where the basic tasks and operations are carried out. Basic tasks include securing inputs for production, also transforming these into outputs. For example, in a manufacturing firm, the purchasing department buys raw materials, while the production department transform these to products. In addition, outputs need to be distributed

and support functions need to be performed (e.g., production machine maintenance, inventory control and the like.) In the following, we focus on the Operational Core specification with respect to the performance of basic tasks. The following specification says that each basic task must be performed within a precise time period that depends on the type of the task. For instance, providing raw materials is a task that must be performed before the production process begins.

Entity BasicTask
Attribute constant *taskType: TaskType, resourceNeed: Resource, performed: Boolean, timePeriod: Time, output: OutputType*

Entity Resource
Attribute constant *resourceType:ResourceType*

Actor OperationalCore
Attribute optional resource: Resource
Goal PerformBasicTasks
Mode achieve
Fulfillment definition
$\forall task{:}BasicTask\ (Perform(self,task) \wedge TimePerforming(task) \leq task.time)$
[each basic task in the organization will be performed by the Operational Core within the allotted time period for that type of task]

The Operational Core is the heart of the organization and depends directly on a middle level, which depends in turn on a top level (Strategic Apex). At the middle level we have three main actors: Middle Agency, Technostructure, and Support.

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

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

Dependency PerformTask
Type task
Mode achieve
Depender MiddleAgency
Depndee OperationalCore
Attribute constant task: BasicTask
Creation condition $\neg task.performed$

trigger *JustCreated(task)*

Fulfillment

condition for depender *task.performed*

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

Dependency ResourceAllocation
Type task
Mode achieve
Depender OperationalCore
Depndee MiddleAgency
Attribute constant task :BasicTask
Creation **condition** *¬(task.resourceNeed =depender.resource)*
trigger *JustCreated(task)*
Fulfillment
condition for depender *Assign(task.resourceNeed,depender)*
[a ResourceAllocation dependency is created when there is a basic task to be performed and a needed resource has not been allocated; the dependency is fulfilled when the resource is allocated]

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

Actor Technostructure
Goal StandardizeBasicTasks
Mode achieve
Fulfillment definition
∀task:BasicTask (Standardize(task.output))
[for each basic task in the organization, the Technostructure will standardize the task output]

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

Dependency Standardize
Type task
Mode achieve
Depender OperationalCore
Depndee Technostructure
Attribute constant task :BasicTask
Creation
condition *¬∃standard: Standard (standard.output=task.output)*
trigger *JustCreated(task)*

Fulfillment

condition for depender *∃standard: Standard (standard.output=task.output)*

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

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

At the top lies the Apex composed of strategic executive actors responsible for ensuring that the organization serves its mission in a effective way. Their major goals include direct supervision (e.g., allocate resources to the middle level, resolve conflicts within the Middle Agency, and monitor performances), and management of the relations with the environment (e.g., inform influential external actors of organizational activities, develop high-level contact, and negotiating major agreements). They also develop organizational strategies consistent with the interpretation of the environment. In the following we focus on the Strategic Management goal and the respective dependency with Middle Agency. In particular, we specify that for each objective of the organization the Apex defines a specific strategy consistent with the environment and that the strategies of the Middle Agency must be in turn consistent with those of the Apex.

Actor Apex
Goal StrategicManagement

Mode achieve

Fulfillment definition

∀obj:OrgObjective (∃strategy: Strategy
(strategy.objective=obj ^ env-consistent(strategy) ^
Apply(self,strategy))

[for each objective of the organization, the Apex applies a strategy consistent with the environment]

Dependency StrategicManagement
Type SoftGoal
Mode achieve
Depender MiddleAegncy
Depndee Apex
Attribute constant objective : MiddleAgencyObejective
Creation

condition *¬∃objective.strategy*

trigger *Pursue(objective)*

Fulfillment

condition for depender

∃ma-strategy: MiddleAgencyStrategy
(∀ org -strategy: OrgStrategy
(objective.strategy=strategy ^
consistent(ma-strategy,org-strategy))

[the StrategicManagement dependency is created when there is no strategy for a given middle agency objective, and it is fulfilled when there exists a middle agency strategy consistent with all the strategies of the organization]

Figure 19 details the Technostructure actor in terms of sub-actors. These include Financial Analysts who develop financial systems for the Apex, also Management and Technology

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

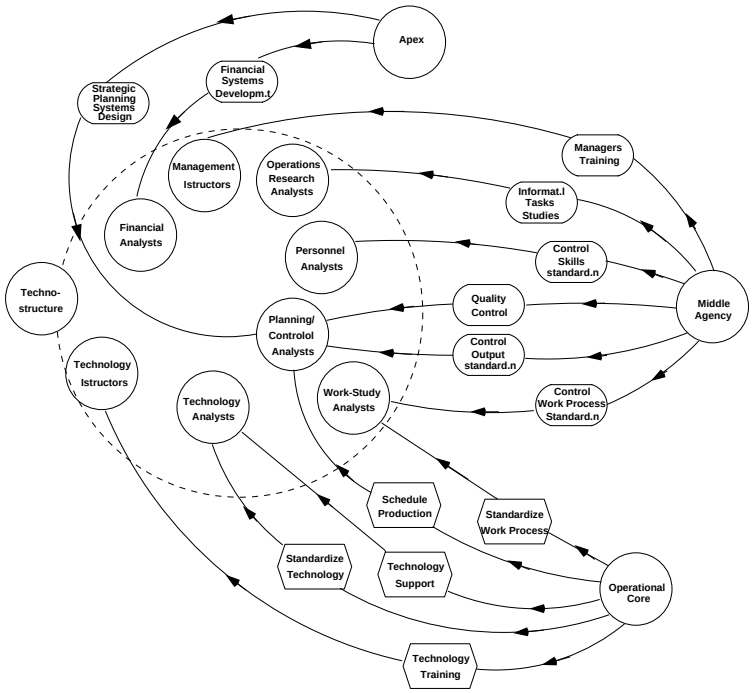

Figure 19. The Technostructure actor

6 Conclusion

We have proposed an ontology which views information systems as social structures. The ontology has been inspired by organizational modeling frameworks and theories, also by multi-agent and cooperative system research.

Obviously, this social perspective on software systems is best suited for software which operates within an open, dynamic, and distributed environment, such as those that are becoming prevalent with Web, Internet, agent, and peer-to-peer software technologies.

We are continuing work on formalizing the organizational styles and social patterns that have been presented. In particular, we propose to define formally the patterns and styles as metaclasses which are instantiated for particular information system designs. To this end, we are improving the syntax and semantics of *Formal Tropos* especially to support metalevel specifications. We also propose to compare and contrast our styles and patterns to classical software architectural styles and patterns proposed in the software engineering literature and

relate them to implementation-inspired architectural components such as ports, connectors, interfaces, libraries and configurations. Finally, we are working on formalizing the "code of ethics" for the different patterns, answering the question: what can one expect from a broker, mediator, embassy, etc.?

References

[Ari98] Y. Aridor and D. B. Lange. "Agent Design Patterns: Elements of Agent Application Design". In *Proceeding of Autonomous Agents* (Agents'98), ACM Press, 1998.

[Cas01] J. Castro, M. Kolp, and J. Mylopoulos. "A Requirements-Driven Development Methodology". In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering* (CAiSE'01), Interlaken, Switzerland, June 2001.

[Cla99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*, MIT Press, 1999.

[Deu99] D.Deugo, F. Oppacher, J.Kuester, and I. V. Otte. "Patterns as a Means for Intelligent Software Engineering". In *Proceedings of the International Conference of Artificial Intelligence* (IC-AI'99), Vol II, CSRA Press, 605-611, 1999.

[Feb98] J. Ferber and O. Gutknecht. "A meta-model for the analysis and design of organizations in multi-agent systems". In *Proceedings of the 3rd International Conference on Multi-Agent Systems* (ICMAS'98), IEEE CS Press, June, 1998.

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

[Fux01] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. "Model Checking Early Requirements Specification in Tropos". In *Proc. of the 5th Int. Symposium on Requirements Engineering, RE'01*, Toronto, Canada, Aug. 2001.

[Fux01a] A. Fuxman, P. Giorgini, M. Kolp, and J. Mylopoulos. "Information systems as social structures". In *Proc. of the 2nd Int. Conf. on Formal Ontologies for Information Systems* (FOIS'01), Ogunquit, USA, October 2001.

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

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

[Hay99] S. Hayden, C. Carrick, and Q. Yang. "Architectural Design Patterns for Multiagent Coordination". In *Proc. of the International Conference on Agent Systems* (Agents'99), Seattle, WA, May 1999.

[Ken98] E. Kendall, P.V. Murali Krishna, C. V. Pathak, and C.B. Suersh. "Patterns of Intelligent and Mobile Agents". In K. Sycara and M. Wooldridge, eds., *Proceedings of the 2nd International Conference on Autonomous Agents* (Agents'98), pages 92—98, New York, May 1998, ACM Press.

[Kol01] M. Kolp, P. Giorgini, and J. Mylopoulos. "An Organizational Perspective on Multi-agent Architectures". In *Proc. of the Eighth International Workshop on Agent Theories, architectures, and languages, ATAL'01*, Seattle, USA, August 1-3, 2001.

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

[Mal88] T.W. Malone. "Organizing Information Processing Systems: Paralles Between Human Organizations and Computer Systems". In W. Zachry, S. Robertson, and J. Black, eds. *Cognition, Cooperation and Computation*, Norwood, N.J., Ablex, 1988.

[Mot93] R. Motschnig-Pitrik. "The Semantics of PartsVersus Aggregates in Data/Knowledge Modeling". In *Proc. of the 5th Int. Conference on Advanced Information Systems Engineering* (CAiSE'93), Paris, June 1993, pp 352-372.

[Myl90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. "Telos: Representing Knowledge About Information Systems". In *ACM Trans. Info. Sys.*, 8 (4), October 1990, pp. 325 – 362.

[Pre95] W. Pree. *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.

[Sha96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Upper Saddle River, N.J., Prentice Hall, 1996.

[Sco98] W. Richard Scott. *Organizations : rational, natural, and open systems*, Upper Saddle River, N.J., Prentice Hall, 1998.

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

[Yos95] M.Y. Yoshino and U. Srinivasa Rangan. *Strategic alliances: an entrepreneurial approach to globalization*, Boston, Mass., Harvard Business School Press, 1995.

[Yu95] E. Yu. Modelling *Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[Woo99] S. G. Woods and M. Barbacci. "Architectural Evaluation of Collaborative Agent-Based Systems". Technical Report, CMU/SEI-99-TR-025, SEI, Carnegie Mellon University, PA, USA, 1999.

[Zam00] F. Zambonelli, N.R. Jennings, and M. Wooldridge. "Organisational Abstractions for the Analysis and Design of Multi-Agent Systems". In *Proc. of the 1st International Workshop on Agent-Oriented Software Engineering* at ICSE 2000, Limerick, Ireland, June 2000.