Information Systems Development through Social Structures

Manuel Kolp
IAG - Information Systems Unit,
University of Louvain
1348 Louvain-La-Neuve, Belgium
1, Place des Doyens
+32 10 47 83 95
kolp@isys.ucl.ac.be

Paolo Giorgini
Dept of Information & Communication Technology, University of Trento,
38100 Trento, Italy
14 Via Sommerive
+39 0461 88 2052
paolo.giorigni@dit.unitn.it

John Mylopoulos
Dept of Computer Science,
University of Toronto, M5S 3H5
Toronto, Canada
6 King's College Road
+1 416 978 5180
jm@cs.toronto.edu

ABSTRACT
Information systems for organizations such as e-business and knowledge management systems must continually evolve to adapt to their operational environment. Unfortunately, current development methodologies do not support system evolution well, making software an obstacle to organizational changes. The paper describes a framework that develops and evolves seamlessly a system-to-be within its organizational environment. We adopt a set of social structures – organizational styles and social patterns – based on concepts of organization theory and agent approaches, as a foundation to model early and late requirements as well as architectural and detailed design. We illustrate the use of the social structures through a case study, and we specify one of the styles in Formal Tropos language. This research has been conducted within the context of the Tropos project.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation methods, languages, methodologies*; D.2.11 [**Software Engineering**]: Software Architectures – *data abstraction, patterns*; K.6.1 [**Management of Computing and Information systems**]: Project and People Management – *systems analysis and design*; K.6.3 [**Management of Computing and Information systems**]: Software Management – *software development*.

General Terms
Design, Languages, Management.

Keywords
Requirements Engineering, *i** Framework, *Tropos* Methodology

1. INTRODUCTION
We are interested in narrowing the semantic gap between requirements analysis and system design. On one hand, requirements analysis techniques have been recognizing the modeling of the social and intentional context, within which a system will eventually operate, as an important part of the analysis process (e.g., [4, 7, 22]). On the other hand, software design techniques have traditionally been inspired and driven by the programming paradigm of the day (e.g., [3, 20]). This impedance mismatch between analysis and design is one of the main factors for the poor quality of system development projects.

One way to reduce this gap is adopting as much as possible the same concepts for all phases of the development process. In this paper, we propose a set of social structures – organizational styles and social patterns – as a foundation to model early and late requirements as well as architectural and detailed design. These social structures use primitives from *i** [22], a modeling framework for early requirements founded on the notions of actor, goal and social dependency.

This work continues the research in progress within the *Tropos* project [5, 10] and relies on material detailed in previous papers. In [5], we have presented *Tropos*, an information system development framework, which is requirements-driven in the sense that it adopts concepts used during early requirements analysis, especially those offered by *i**. The *Tropos* framework has also been applied for developing multi-agent systems [10]. *Tropos* spans four phases of software development:

• *early requirements analysis*, 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 inter-dependencies;

• *late requirements analysis*, 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 other dependencies;

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

In [8] we have detailed a social ontology for *Tropos* that views information systems as social structures. The ontology is described at three levels of granularity. At the lowest (finest granularity) level, *Tropos* adopts concepts offered by the *i** framework. 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.

In [13], we have described how to use our *Tropos* social ontology to design multi-agent architectures. As a matter of fact, multi-agent systems can be considered structured societies of coordinated autonomous agents that interact one another to achieve particular, possible common goals.

We argue that the development of methodologies for organizational information systems, like ERP, Knowledge Management, groupware and e-business systems, need to integrate organizational models and software system designs. This allows systems to better match their operational context. In this paper, we propose to reduce the impedance mismatch between phases of the development process by using social structures as building blocks all along the system life-cycle.

The paper is organized as follows. In Section 2, we present some of our social structures, firstly organization-inspired styles, and secondly social patterns based on agent approaches; then, we illustrate how social structures can be evaluated. Section 3 presents a case study in which social structures are used all along the information system life-cycle. It also proposes the *Formal Tropos* specification of one of our styles. Finally, Section 4 summarizes the contributions of the paper and points to further work.

2. SOCIAL STRUCTURES

For a detailed presentation of our organizational styles and social patterns, see [8, 13].

2.1 Organizational Styles

Organization theory (e.g., [14, 17]) and strategic alliances (e.g., [11, 19, 21]) study alternatives to model (business) organizations. An organizational style represents a possible way to structure the stakeholders – individuals, physical or social systems – of an organization in order to meet its strategic goals.

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

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

We propose a catalogue adopting (some of) the styles offered in organization theory for developing information systems. In the following, we present briefly some of these styles using the strategic dependency model of *i**.

A strategic dependency model is a graph, where each node represents an actor (an agent, a position, or a role within an organization) and each link between two actors indicates that one actor depends on another for a goal to be fulfilled, a task to be carried out, or a resource to be made available. We call the depending actor of a dependency the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers (goal, task or resource) is called the *dependum*. The model distinguishes among four types of dependencies – goal-, task-, resource-, and softgoal-dependency – based on the type of freedom that is allowed in the relationship between depender and dependee. Softgoals are distinguished from goals because they do not have a formal definition, and they are amenable to a different (more qualitative) kind of analysis [6].

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

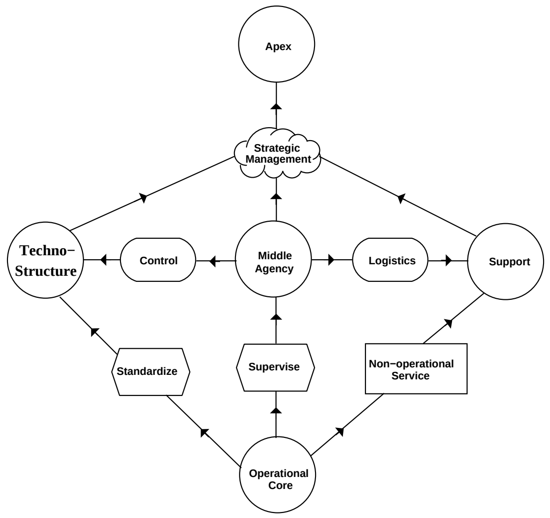

Figure 1. Structure-in-5

The **structure-in-5** (Figure 1) is a typical organizational style. At the base level, the *Operational Core* takes care of the basic tasks — the input, processing, output and direct support procedures — associated with running the organization. At the top lies the *Apex*, composed of strategic executive actors. Below it, sit the *Technostructure*, *Middle Agency* and *Support* actors, who are in charge of control/standardization, management and logistics procedures, respectively. The *Technostructure* component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the organization adapt to its environment. Actors joining the apex to the operational core make up the *Middle Agency*. The *Support* component assists the operational core for non-operational services that are outside the basic flow of operational tasks and procedures.

The **joint venture** style (Figure 2) is a more decentralized style that involves an agreement between two or more principal partners in order to obtain the benefits derived from operating at a larger scale and reusing the experience and knowledge of the partners. Each principal partner can manage and control itself on a local dimension and interact directly with other principal partners to exchange, provide and receive services, data and knowledge. However, the strategic operation and coordination is delegated to a *Joint Management* actor, who coordinates tasks and manages the sharing of knowledge and resources.

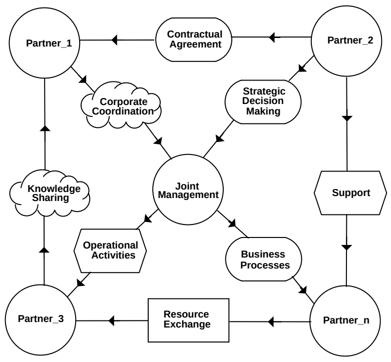

Figure 2. Joint Venture

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 3 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*.

For a more detailed presentation of organizational styles we have defined (takeover, hierarchical contracting, bidding, arm's-length, pyramid, flat structure, co-optation, …), see [8].

2.2 Social Patterns

A social pattern defines the actors (together with their roles and responsibilities) and the social dependencies that are necessary for the achievement of a goal. Considerable work has been done in software engineering for defining software patterns (see e.g., [9]), but unfortunately, they do not place emphasis on social aspects. On the other hand, proposals of patterns that address social issues (see e.g., [2]) 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 two of the social patterns that focus on social mechanisms recurrent in multi-agent and cooperative systems literature (e.g., [12]): mediator and embassy pattern.

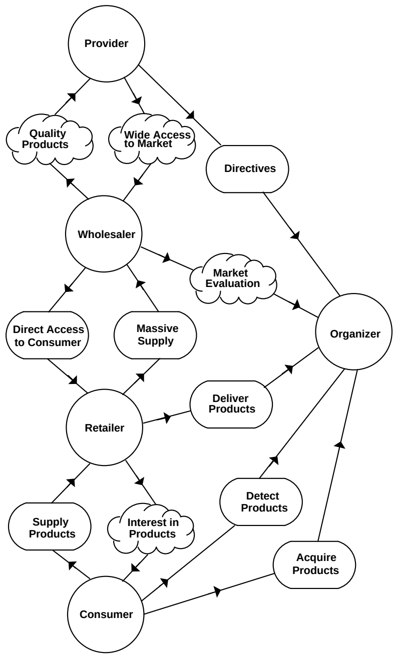

Figure 3. Vertical Integration

A **mediator** (Figure 4a) mediates interactions among different actors. An initiator addresses the mediator in place 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.

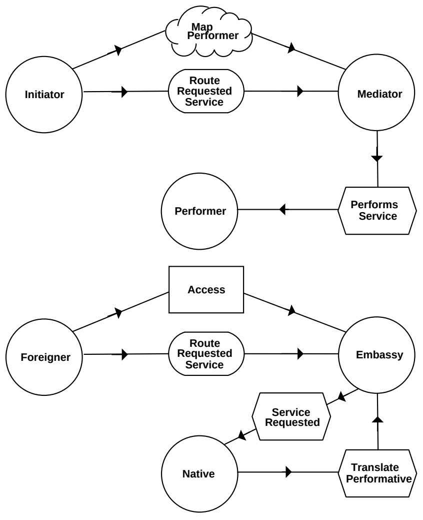

Figure 4. Mediator (a) and Embassy (b)

An **embassy** (Figure 4b) routes a service requested by a foreign actor to local ones and handles back the response. If the access is granted, the foreign actor 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 out to the foreign actor, translated in reverse.

For a more detailed presentation of the social patterns we have defined (broker, matchmaker, contract-net, facilitator, wrapper, …), see [13].

2.3 Evaluating Social Structures

Strenghts and weaknesses of styles and patterns can be evaluated and compared through quality attributes (or non-functional requirements) analysis. Quality attributes like *coordinativity*, *predictability*, *failability-tolerance* and *adaptability* have been found relevant for organizational constructs [18].

Coordinability: actors must be able to coordinate with other actors of the social structure to achieve a common purpose or simply their local goals.

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

Failability-Tolerance: a failure of one actor does not necessarily imply a failure of the whole structure. The structure then needs to check the completeness and the reliability of data, information and transactions. To prevent failure, different actors can, for instance, assume replicated capabilities.

Adaptability: actors must to adapt to modifications in their social environment. They may allow changes to the communication protocol, dynamic introduction of a new kind of actors previously unknown or manipulations of existing ones.

Due to the lack of space, we only consider the structure-in-5 and the joint venture with respect to the four qualities described above. Table 1 summarizes their strengths and weaknesses.

Table 1. Strengths and Weaknesses of some Social Structures

	Coordinat.	Predictab.	Failab-Tol.	Adaptab.
S-in-5	+	+	++	+-
Joint-Venture	+-	+	+-	+-

The ***structure-in-5*** improves *coordinativity* among components by differentiating the data hierarchy - supported by the support component – from the control hierarchy - supported by the operational core, technostructure, middle agency and strategic apex. The existence of different levels of abstraction in the structure-in-5 addresses the need for managing *predictability*. Besides, 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. Checks and control mechanisms can be integrated at different levels of abstraction assuming redundancy from different perspectives and increase considerably *failability-tolerance*. Since the structure-in-5 separates data and control hierarchies, integrity of these two hierarchies can also be verified independently. The structure-in-5 separates independently the typical components of an organization, isolating them from each other and allowing then dynamic *adaptability*. But since it is restricted to no more than 5 major components, more refinement has to take place inside the components.

The ***joint venture*** supports *coordinativity* in the sense that each partner interacts via the joint manager for strategic decisions. Partners indicate their interest, and the joint manager either returns them the strategic information immediately or mediates the request to some other partners. However, since partners are usually heterogeneous, it could be a drawback to define a common interaction background. The central position and role of the joint manager is a means for resolving conflicts and preventing *unpredictability*. Through its joint manager, the joint-venture proposes a central communication controller. How the joint venture style addresses *failability-tolerance*, notably *reliability*, is less clear. However, exceptions, wiretapping, supervising, and monitoring can improve it. Manipulation of partners can be done easily to *adapt* the structure by registering new ones to the joint manager. However, since partners can also communicate directly with each other, existing dependencies should be updated as well. The joint manager cannot be removed due to its central position.

To cope with these quality attributes and select the appropriate structure, more refined analysis and decomposition can be done with frameworks like KAOS [7] or the NFR framework [6]. In the NFR framework, we go through a means-ends refining of the identified quality attributes in more precise sub-attributes, and then, as shown partially in Figure 5, we evaluate the social structures against such sub-attributes.

The analysis is intended to make explicit the space of alternatives for fulfilling the top-level attributes. The social structures are represented as operationalized attributes (saying, roughly, "makes the structure *structure-in-5, joint-venture, vertical-integration*-based, …").

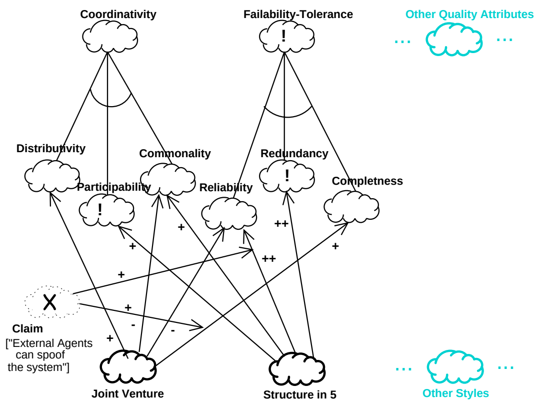

Figure 5. Partial Evaluation for Organizational Styles

The evaluation results in contribution relationships from the social structures to the quality attributes, labeled "+", "++", "-", "--" that mean *partially satisfied*, *satisfied*, *partially denied* and *denied*, respectively. Design rationale is represented by claims drawn as dashed clouds. They make it possible for domain characteristics such as priorities to be considered and properly reflected into the decision making process. Exclamation marks are used to mark priority attributes while a check-mark "✓" indicates an accepted attribute and a cross "✕" labels a denied attribute.

Relationships types (AND, OR, ++, +, -, and --) between quality attributes are formalized to offer a tractable proof procedure. For each attribute we consider two different variables: S for the satisfiability and D for the deniability. Such variables can assume three values: *null* (–) , *partial* (*p*), and *total* (*t*). For instance, when S=*t*, an attribute is totally satisfied, when S=*t* it is partially satisfied, and when S=– there is no evidence to say something about its satisfiability (analogously for D).

S and D are not required to be logically exclusive since there may be contradictory contributions, e.g., a softgoal is satisfied and partially denied. Table 2 shows propagation rules for ++, +, -, and -- relationships with respect to satisfiability (S). Notice that the null value does not produce any effect in the propagation. A dual table is given for the deniability and the partial deniability.

Table 2: Propagation rules for Satisfiability

S	++	+	-	--
t	**S**=*t*	**S**=*p*	**D**=*p*	**D**=*t*
p	**S**=*p*	**S**=*p*	**D**=*p*	**D**=*p*

Under the assumption that $– < p < t$, we use min-value and max-value functions respectively for AND and OR relationships. The basic algorithm for the labels propagation is presented in Figure 6. Initially, all the nodes are initialized with the available evidence, a null value is assigned to the nodes for which we do not have evidence. At each step the value of the two variables S and D of each node is calculated using the nodes' value of the previous step. The final value for D and S is given by the maximum value of all contributions of the incoming relations. The algorithm terminates when all the current nodes' values are the same of those calculated at the previous step. The use of the maximum value function guarantees the termination of the algorithm.

```
Initialize NODES'
do
NODES ← NODES'
 foreach node $n_i$
   foreach incoming relation $A_{ij}$
     $D_j$ ← ComputeD($A_{ij}$)
     $S_j$ ← ComputeS($A_{ij}$)
   $n_i$.D' ← $\mathrm{Max}_j(D_j)$
   $n_i$.S' ← $\mathrm{Max}_j(S_j)$
while(NODES≠NODES')
```

Figure 6. Basic propagation algorithm

We are currently working on a second approach to goal analysis, where numerical intervals are used to define the degree of satisfiability and deniability of a goal. Here, we are working along two different directions: one based on probability theory and the other on the Dempster-Shafer theory (see, e.g, [Par94]).

3. INFORMATION SYSTEM LIFE CYCLE WITH SOCIAL STRUCTURES

In order to illustrate the use of our social structures, we consider a business-to-business setting describing a typical media industry. *Media Retailer* is a specialized store selling and shipping different kinds of media items such as books, newspapers, audio CDs, videotapes, and the like. *Media Retailer* is supplied with the latest releases by *Media Supplier*. At the production level, *Editor* is specialized in the press and book business, *Movie Studio* makes movies while *Record Label* works in the music industry and *Games Design* creates video and computer games. All these actors have agreed to develop *Media System*, an internet-based information system supporting business-to-business capabilities to facilitate and improve business interactions, and to reduce costs and delays of traditional information and communication means. *Customers* will also be able to use the *Media System* to browse the catalogue, query the item database, and order on-line items.

3.1 Early Requirements

Early requirements analysis is 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 inter-dependencies. Like several media companies, *Media Producer* could be organized as a joint venture (Figure 7).

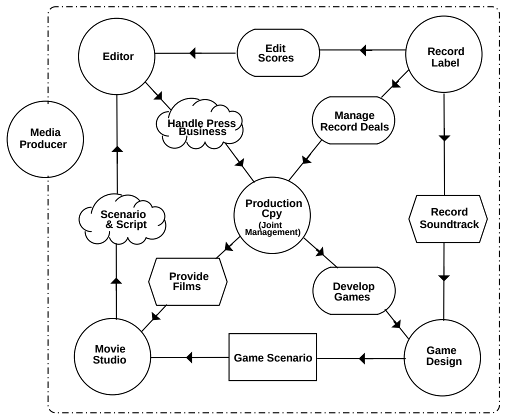

Figure 7. Media Producer using the Joint Venture Style

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 its own business management. However, only the joint management actor, *Production Cpy*, handles the corporate strategic management.

3.2 Late Requirements

Late requirements analysis describes the system-to-be as an actor within its operational environment, along with relevant functions and qualities. We introduce the *Media System* as a full-fledged social actor contributing to the fulfillment of stakeholder goals, along with other actors from the system's operational environment.

We use our organizational styles to guide the modeling of the system inside its organizational environment. For instance, the late requirements model of the system interacting with its environment might be represented as a *vertical integration* as shown in Figure 8. With respect to the vertical integration structure presented in Figure 3, the *Customer* takes the role of

Consumer, Media Producer 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 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, *Media Producer* wants to find information about the media market and stakeholders; *Media Supplier* wants to find and promote new ideas, projects and talents to increase her market share, while *Media Retailer* needs to be provided with e-commerce facilities to satisfy customers. Finally, *Customer* wants to consult product catalogues and place orders.

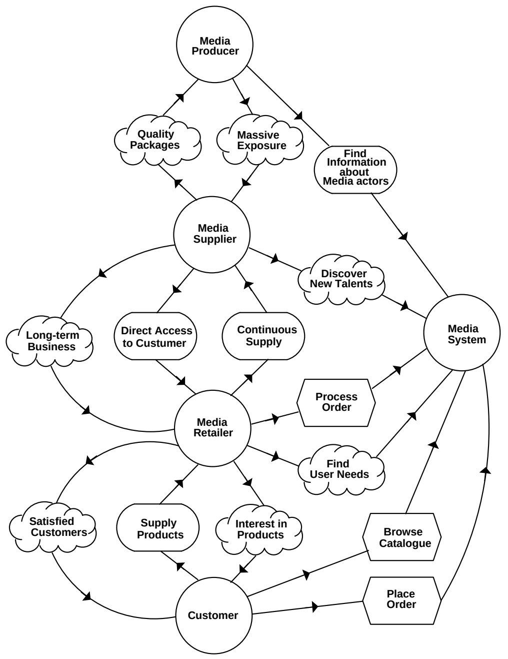

Figure 8. Introducing the system with the Vertical Integration

3.3 Architectural Design

Architectural design defines the system's global architecture in terms of subsystems, interconnected through data, control and other dependencies. We aim to apply our social structures not only to requirements models, but also to all levels of software design. In our example, the *joint venture* style is used to produce an architectural description of the *Media System*. A 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 *Media Retailer. Order Processor* is in charge for 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. These three software quality attributes (as well as sub-attributes *Authorization*, *Integrity*, *Usability*, *Updatability* and *Maintainability*) required for business-to-consumer applications are identified and evaluated in detail for the *Media system* example in [13].

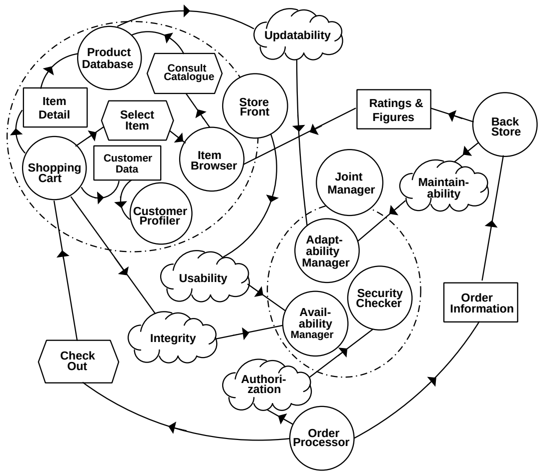

Figure 9. System Architecture with the Joint Venture Style.

All the system actors of Figure 9 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 the 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*.

3.4 Detailed Design

Detailed design is concerned with the definition in further detail of the behavior of each component identified during architectural design. Figure 10 shows a possible use of our social patterns in the e-business system shown in Figure 9. In particular, it shows how to solve the goal of managing catalogue navigation that the *Store Front* delegates to the *Item Browser*. The goal is decomposed into subgoals and solved with a combination of social 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*.

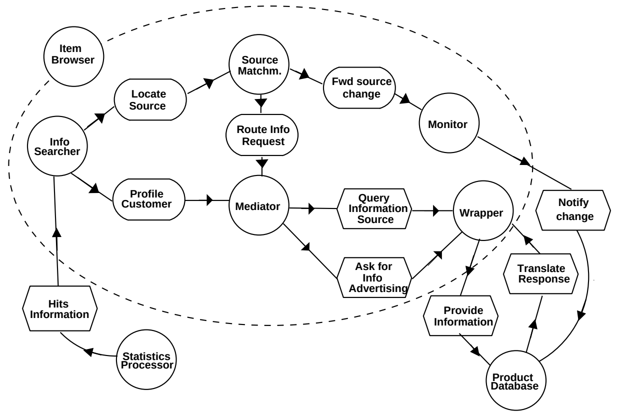

Figure 10. Social Patterns for Item Browser

3.5 Formalizing Social Structures

Formal Tropos [8] offers all the primitive concepts of *i** (such as actors, goals and dependencies), but supplements them with a rich specification language inspired by KAOS [7]: 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 6. In particular, we focus on the *Production Cpy* goal of applying a movie and game strategy.

In the following, 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

4. CONCLUSION

We have emphasized that the design of information systems should be based on the same organization concepts and models used in requirements analysis. This should help to reduce the impedance mismatch between analysis and design. Within the context of *Tropos*, a development methodology inspired by early requirements modeling techniques, we have proposed to use social structures not only for early but also late requirements analysis as well as architectural and detailed design. These social structures rely on concepts from organization theory and agent approaches [15].

We are continuing to work on the formalization of our organizational styles and social patterns. The idea is defining formally organization structures as metastructures that can be instantiated for particular information system designs. Moreover, we want to study and formalize when a particular design is an instance of such a metastructure. We are also contrasting our structures to conventional styles [18] and patterns [9] proposed in the software engineering literature. As mentioned, we are defining algorithms to propagate evidences of satisfaction and denial of each conventional or social structure with respect to a set of non-functional requirements. These should allow us to evaluate and compare more precisely the structures against them within the NFR framework.

5. REFERENCES

[1] A. I. Anton, "Goal-Based Requirements Analysis", In *Proceedings of the Second International Conference On Requirements Analysis* (ICRE'96), pp.136-144, 1996.

[2] Y. Aridor and D. B. Lange. "Agent Design Patterns: Elements of Agent Application Design" In *Proceedings of the Second International Conference on Autonomous Agents* (Agents'98), New York, USA, May 1998.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, The Addison-Wesley Object Technology Series, Addison-Wesley, 1999.

[4] J. A. Bubenko, "Next Generation Information Systems: an Organizational Perspective", In *Proceedings of the International Workshop on Development of Intelligent Information Systems*, Niagara-on-the-Lake, Ontario, pp. 22-31, Canada, April 1991.

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

[6] L. K. Chung, B. A. Nixon, E. Yu and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.

[7] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal–directed Requirements Acquisition", *Science of Computer Programming, 20*, pp. 3-50, 1993.

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

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

[10]P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia and P. Bresciani. "Agent-Oriented Software Development: A Case Study". In *Proceedings of the 13th International Conference on Software Engineering & Knowledge Engineering* (SEKE01), Buenos Aires, Argentina, June 2001.

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

[12] S. Hayden, C. Carrick, and Q. Yang. "Architectural Design Patterns for Multiagent Coordination". In *Proceedings of the International Conference on Autonomous Agents* (Agents'99), Seattle, USA, May 1999.

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

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

[15] J. Odell, H. Van Dyke Parunak, and B. Bauer, "Extending UML for Agents", In *Proceedings of the Agent-Oriented Information System Workshop at the 17th National Conference on Artificial Intelligence*, pp. 3-17, Austin, USA, July 2000.

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

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

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

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

[20] R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Englewood Cliffs, Prentice-Hall, 1990.

[21] M.Y. Yoshino, and U. S. Rangan, *Strategic Alliances: An Entrepreneurial Approach to Globalization*, Harvard Business School Press, 1995.

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