Tropos: A Requirements-Driven Methodology for Agent-Oriented Software

Jaelson Castro[1], Paolo Giorgini[2], Manuel Kolp[3], and John Mylopoulos[4]

[1] Federal University of Pernambuco - Brazil
[2] University of Trento – Italy
[3] University of Louvain – Belgium
[4] University of Toronto – Canada

Software systems of the future will have to perform well within ever-changing organizational environments. Unfortunately, existing software development methodologies (object-oriented, structured or otherwise) have traditionally been inspired by programming concepts, rather than organizational ones, leading to a semantic gap between system and its operational environment. To reduce this gap, we propose a software development methodology named *Tropos* which is founded on the *i** organizational modeling framework. *i** offers the notions of *actor*, *goal* and (actor) *dependency*. Tropos uses these concepts as a basis to model early and late requirements, architectural and detailed design for a software system. The paper outlines *Tropos* phases through an e-business example. The methodology complements well proposals for agent-oriented programming platforms.

1 INTRODUCTION

Organizational software systems have traditionally suffered from an impedance mismatch between their outer and inner environment: their operational environment is understood in terms of actors, responsibilities, objectives, tasks and resources, while their inner environment is conceived as a collection of (software) modules, entities (e.g., objects, components), data structures and interfaces. This mismatch contributes greatly to poor software quality, also to frequent failures of software system development projects.

One reason for this mismatch is that development methodologies have traditionally been inspired and driven by the programming paradigm of the day. So, during the era of structured programming, structured analysis and design techniques were proposed [10,37], while object-oriented programming has given rise more recently to object-oriented design and analysis [3,34]. For structured development techniques this meant that throughout software development, the developer could conceptualize the system in terms of functions and processes, inputs and outputs. For object-oriented development, on the other hand, conceptualizations consist of objects, classes, methods, inheritance and the like throughout.

Using the same concepts to align requirements analysis with system design and implementation makes perfect sense. For one thing, such an alignment reduces impedance mismatches between different development phases. Moreover, such an alignment can lead to coherent toolsets and techniques for developing software. As well, it can streamline the development process itself.

But, why base such an alignment on implementation concepts? Requirements analysis is arguably the most important stage of software development. This is the phase where technical considerations have to be balanced against social and organizational ones. This is also the phase where the operational environment of the system is modeled and analyzed. Not surprisingly, this is also the phase where most and costliest errors are introduced compared to other development phases. Even if (or rather, when) the importance of design and implementation phases wanes sometime in the future, requirements analysis will remain a critical phase for the development of any software system, answering the most fundamental of all design questions: "what is the system intended for?"

Latest generation software systems, such as enterprise resource planning (ERP), groupware, knowledge management and e-business systems, should be designed to match their operational environment. For instance, ERP systems have to implement a process view of the enterprise to meet business goals, tightly integrating all relevant functions of their operational environment. To reduce as much as possible the impedance mismatch between the system and its environment, we outline in this paper a development framework, named *Tropos*[1], which is requirements-driven in the sense that it is based on concepts used during early requirements analysis. To this end, we adopt the concepts offered by *i** [40], a modeling framework proposing concepts such as *actor* (actors can be *agents*, *positions* or *roles*), as well as social dependencies among actors, including *goal*, *softgoal*, *task* and *resource* dependencies. These concepts are used for an e-commerce example[2] to model not just early requirements, but also late requirements, as well as architectural and detailed design. The proposed methodology spans four phases that can be used either following the waterfall or the spiral model respectively for sequential and iterative development [24]:

- *Early requirements*, concerned with the understanding of a problem by studying an organizational setting.
- *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 other dependencies.
- *Detailed design*, where the behavior of each architectural component is further refined.

The proposed methodology includes techniques for generating an implementation from a *Tropos* detailed design. It is very natural to use an agent-oriented programming platform for the implementation, given that the detailed design is defined in terms of (system) actors, goals and inter-dependencies among them.

The paper is organized as follows. Section 2 provides and overview of the methodology while Section 3 describes the case study to be used for illustration purposes. Sections 4 to 7 present the application of Tropos to the case study. Section 8 concludes the paper discussing strengths and weaknesses of the methodology.

2 THE METHODOLOGY

[1] For further detail and information about the \textit{Tropos} project, see http://www.troposproject.org.

[2] Based on a realistic e-commerce system development exercise of moderate complexity.

Tropos rests on the idea of using requirements modeling concepts to build a model of the system-to-be within its operational environment. This model is incrementally refined and extended, providing a common interface to the various software development activities. The model also serves as a basis for documentation and evolution of the software system.

In the following, we describe and illustrate the four development phases of the Tropos methodology: *Requirements Analysis* (early and late), *Architectural Desig*n, and *Detailed Design*.

Requirements Analysis

Requirement analysis represents the initial phase in most software engineering methodologies. Requirements analysis in Tropos consists of two phases: *Early Requirements* and *Late Requirements* analysis. Early requirements is concerned with understanding the organizational context within which the system-to-be will eventually function. Late requirements analysis, on the other hand, is concerned with a definition of the functional and non-functional requirements of the system-to-be.

Tropos adopts the *i** [40] modeling framework for analyzing requirements. In *i** (which stands for "distributed intentionality"), stakeholders are represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The *i** framework includes the *strategic dependency model* (actor diagram in Tropos) for describing the network of inter-dependencies among actors, as well as the *strategic rationale model* (rationale diagram in Tropos) for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors. These
models have been formalized using intentional concepts from Artificial Intelligence, such as goal, belief, ability, and commitment (e.g., [6]). The framework has been presented in detail in [40] and has been related to different application areas, including requirements engineering [38], software processes [39], and business process reengineering [41].

During early requirements analysis, the requirements engineer identifies the domain stakeholders and models them as social actors, who depend on one another for goals to be fulfilled, tasks to be performed, and resources to be furnished. Through these dependencies, one can answer *why* questions, besides *what* and *how*, regarding system functionality. Answers to *why* questions ultimately link system functionality to stakeholder needs, preferences and objectives. Actor diagrams and rationale diagrams are used in this phase.

An actor diagram is a graph involving *actors* who have *strategic dependencies* among each other. A dependency represents an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* depends on the *dependee*, to deliver on the dependum. The dependum can be a *goal* to be fulfilled, a *task* to be performed, or a *resource* to be delivered. In addition, the depender may depend on the dependee for a *softgoal* to be fulfilled. Softgoals represent vaguely defined goals, with no clear-cut criteria for their fulfillment. Graphically, 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* $\rightarrow$ *dependum* $\rightarrow$ *dependee*.

Actor diagrams are extended during early requirements analysis by incrementally adding more specific actor dependencies which come out from a means-ends analysis of each goal. This analysis is specified using rationale diagrams.

A rationale diagram appears as a balloon within which goals of a specific actor are analyzed and dependencies with other actors are established. Goals are decomposed into subgoals and positive/negative contributions of subgoals to goals are specified.

During *late requirements* analysis, the conceptual model developed during early requirements is extended to include the system-to-be as a new actor, along with dependencies between this actor and others in its environment. These dependencies define functional and non-functional requirements for the system-to-be. Actor diagrams and rationale diagrams are used also in this phase.

Architectural Design

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 [33]. In Tropos, 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 that result from a design process, namely *Organization Theory* and *Strategic Alliances*. Organization Theory (e.g., [32]) describes the structure and design of an organization; *Strategic Alliances* (e.g., [27]) models the strategic collaborations of independent organizational stakeholders who have agreed to pursue a set of business goals.

We define an organizational style as a metaclass of organizational structures offering a set of design parameters to coordinate the assignment of organizational objectives and processes, thereby affecting how the organization itself functions [22]. Design parameters include, among others, goal an task assignments, standardization, supervision and control dependencies and strategy definitions.

For instance the **Structure-in-5** [26] specifies that an organization is an aggregate of five sub-structures. 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 fulfills 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). For further details about architectural styles in Tropos, see [11,23].

Styles can be compared and evaluated with quality attributes [33], also called non-functional requirements [5] such as predictability, security, adaptability, coordinability, availability, fallibility-tolerance, or modularity.

To cope with non-functional requirements and select the style for the organizational setting, we go through a means-ends analysis using the non functional requirements (NFRs) framework [5]. We refine the identified requirements to sub-requirements that are more precise and evaluates alternative organizational styles against them.

The analysis for selecting an organizational setting that meets the requirements of the system to build is based on propagation algorithms. Basically, the idea is to assign a set of initial labels for some requirements of the graph, about their satisfiability and deniability, and
see how this assignment leads to the labels propagation for other requirements. In particular, we adopt from [16] both qualitative and a numerical axiomatization for goal (requirements) modeling primitives and label propagation algorithms that are shown to be sound and complete with respect to their respective axiomatization. See [16] for more details.

Detailed Design

The detailed design phase is intended to introduce additional detail for each architectural component of a system. It consists of defining how the goals assigned to each actor are fulfilled by agents with respect to social patterns.

For this step, designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Considerable work has been done in software engineering for defining software patterns (see e.g., [15]). Unfortunately, little emphasis has been put on social and intentional aspects. Moreover, proposals for agent patterns that do address these aspects (see e.g., [1]) are not intended for use at a design level. Instead, such proposals seem to aim at the implementation phase, when issues such as agent communication, information gathering, or connection setup are addressed.

Social patterns in Tropos [11] are design patterns focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems. In particular, the structures
are inspired by the federated patterns introduced in [17,21]. We have classified them into two categories.

The *Pair* patterns -- such as booking, call-for-proposal, subscription, or bidding -- describe direct interactions between negotiating agents. For instance, the Bidding pattern involves an initiator and a number of participants. The initiator organizes and leads the bidding process. He publishes the bid to the participants and receives various proposals. At every iteration, the initiator can accept an offer, raise the bid, or cancel the process.

The *Mediation* patterns -- such as monitor, broker, matchmaker, mediator, embassy, or wrapper -- feature intermediary agents that help other agents to reach an agreement on an exchange of services. For instance, in the Broker pattern, the broker agent is an arbiter and intermediary that requests services from a provider to satisfy the request of a consumer.

Detailed design also includes actor communication and actor behavior. To support it, we propose to adopt existing agent communication languages like FIPA-ACL [25] or KQML [13], message transportation mechanisms and other concepts and tools. One possibility is to adopt extensions to UML [3], like AUML, the Agent Unified Modeling Language [2,29] proposed by the FIPA (Foundation for Physical Intelligent Agents) [14] and the OMG Agent Work group.

We have also proposed and defined a set of stereotypes, tagged values, and constraints to accommodate Tropos concepts within UML [28].

3 CASE STUDY

Media Shop is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. *Media Shop* customers (on-site or

remote) can use a periodically updated catalogue describing available media items to specify their order. *Media Shop* is supplied with the latest releases from *Media Producer* and in-catalogue items by *Media Supplier*. To increase market share, *Media Shop* has decided to open up a B2C retail sales front on the internet. With the new setup, a customer can order *Media Shop* items in person, by phone, or through the internet. The system has been *Medi@* and is available on the world-wide-web using communication facilities provided by *Telecom Cpy*. It also uses financial services supplied by *Bank Cpy*, which specializes on on-line transactions. The basic objective for the new system is to allow an on-line customer to examine the items in the *Medi@* internet catalogue, and place orders.

There are no registration restrictions, or identification procedures for *Medi@* users. Potential customers can search the on-line store by either browsing the catalogue or querying the item database. The catalogue groups media items of the same type into (sub)hierarchies and genres (e.g., audio CDs are classified into pop, rock, jazz, opera, world, classical music, soundtrack,…) so that customers can browse only (sub)categories of interest. An on-line search engine allows customers with particular items in mind to search title, author/artist and description fields through keywords or full-text search. If the item is not available in the catalogue, the customer has the option of asking *Media Shop* to order it, provided the customer has editor/publisher references (e.g., ISBN, ISSN), and identifies herself (in terms of name and credit card number). Details about media items include title, media category (e.g., book) and genre (e.g., science-fiction), author/artist, short description, editor/publisher international references and information, date, cost, and sometimes pictures (when available).

The main interface of the system is shown in Figure 1.

Figure 1. *Interface of the System*

4 EARLY REQUIREMENTS ANALYSIS

These elements described in the previous section are sufficient for producing a first model of an organizational environment. For instance, Figure 2 depicts the actor diagram of our *Medi@* example. The main actors are *Customer*, *Media Shop*, *Media Supplier* and *Media Producer*. *Customer* depends on *Media Shop* to fulfill her goal: *Buy Media Items*. Conversely, *Media Shop*

depends on *Customer* to *increase market share* and make". Since the dependum *Happy Customers* cannot be defined precisely, it is represented as a softgoal. The *Customer* also depends on *Media Shop* to *consult the catalogue* (task dependency). Furthermore, *Media Shop* depends on *Media Supplier* to supply media items in a continuous way and get a *Media Item* (resource dependency). The items are expected to be of good quality because, otherwise, the *Continuing Business* dependency would not be fulfilled. Finally, *Media Producer* is expected to provide *Media Supplier* with *Quality Packages*.

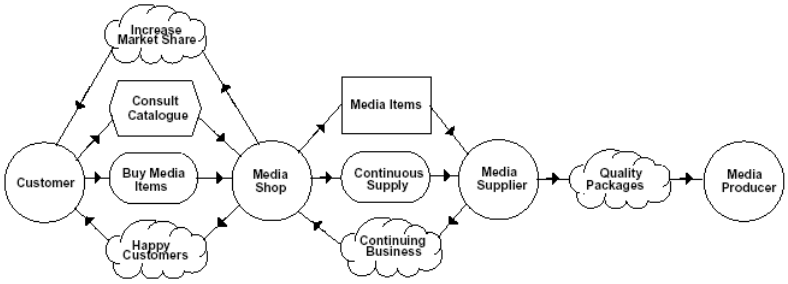

Figure 2. *Actor diagram for a Media Shop*

Figure 3 focuses on one of the (soft)goal dependency identified for \textit{Media Shop,} namely *Increase Market Share*. To achieve that softgoal, the analysis postulates a goal \textit{Run Shop} that can be fulfilled by means of a task *Run Shop*. Tasks are partially ordered sequences of steps intended to accomplish some (soft)goal. Tasks can be decomposed into goals and/or subtasks, whose collective fulfillment completes the task. In the figure, *Run Shop* is decomposed into goals *Handle Billing* and *Handle Customer Orders*, tasks *Manage Staff* and *Manage Inventory*, and softgoal *Improve Service* which together accomplish the top-level task. Subgoals and subtasks can be specified more precisely through refinement. For
instance, the goal *Handle Customer Orders* is fulfilled either through tasks *Order By Phone*, *Order In Person* or *Order By Internet* while the task *Manage Inventory* would be collectively accomplished by tasks *Sell Stock* and *Enhance Catalogue*. These decompositions eventually allow us to identify actors who can accomplish a goal, carry out a task, or deliver on some needed resource for *Media Shop*. Such dependencies in Figure 3 are, among others, the goal and resource dependencies on *Media Supplier* for supplying, in a continuous way, media items to enhance the catalogue and sell products, the softgoal dependencies on *Customer* for increasing market share (by running the shop) and making customers happy (by improving service), and the task dependency *Accounting* on *Bank Cpy* to keep track of business transactions.

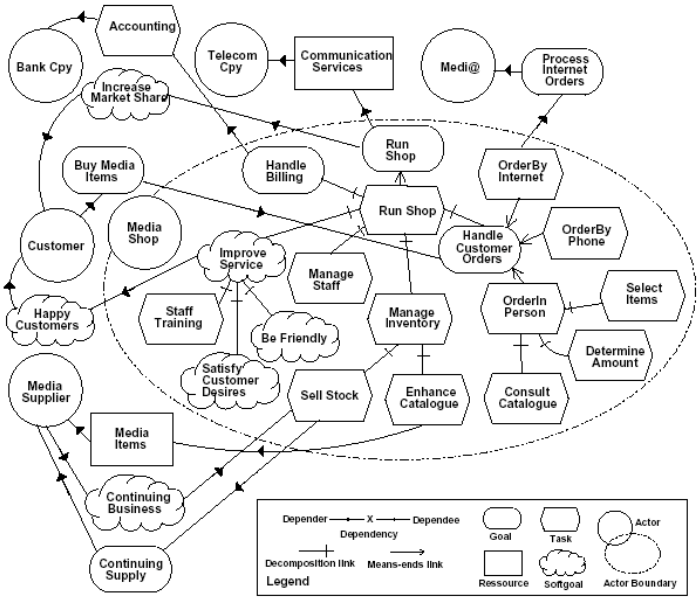

Figure 3. *Means-Ends Analysis for the Softgoal*

4 LATE REQUIREMENTS ANALYSIS

For our example, the *Medi@* system is viewed as a full-fledge actor in the actor diagram depicted in Figure 4. With respect to the actors previously identified, *Customer* depends on *Media Shop* to buy media items while *Media Shop* depends on *Customer* to increase market share and make them happy (with *Media Shop* service). *Media Supplier* is expected to supply *Media Shop* with media items in a continuous way since depending on the latter for continuing business. It can also use *Medi@* to determine new needs from customers, such as media items not available in the catalogue while expecting *Media Producer* to provide her with *quality packages*. As indicated earlier, *Media Shop* depends on *Medi@* for processing internet orders and on *Bank Cpy* to process business transactions. *Customer*, in turn, depends on *Medi@* to place orders through the internet, to search the database for keywords, or simply to browse the on-line catalogue. With respect to relevant qualities, *Customer* requires that transaction services be secure and available, while *Media Shop* expects *Medi@* to be easily adaptable (e.g., catalogue enhancing, item database evolution, user interface update,…).

Finally, *Medi@* relies on internet services provided by *Telecom Cpy* and on secure on-line financial transactions handled by *Bank Cpy*.

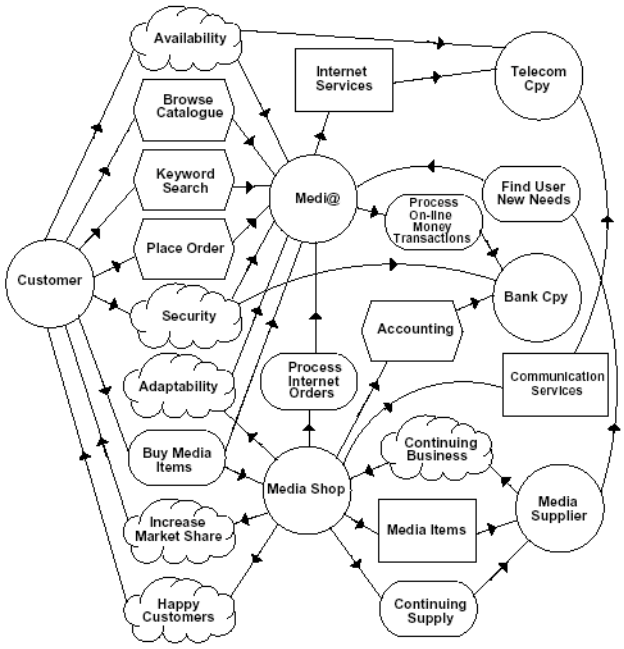

Figure 4. *Actor diagram for a Media Shop*

Although an actor diagram provides hints about why processes are structured in a certain way, it does not sufficiently support the process of suggesting, exploring, and evaluating alternative solutions. As late requirements analysis proceeds, *Medi@* is given additional responsibilities, and ends up as the dependee of several dependencies. Moreover, the system is decomposed into several sub-actors which take on some of these responsibilities. This decomposition and responsibility assignment is realized using the same kind of means-ends analysis illustrated in Figure 3. Hence, the analysis in Figure 5 focuses on the system itself, instead of an external stakeholder.

Figure 5 postulates a root task *Internet Shop Managed* providing sufficient support (++) [5] to the softgoal *Increase Market Share*. That task is firstly refined into goals *Internet Order Handled* and *Item Searching Handled*, softgoals *Attract New Customer*, *Secure* and *Available*, and tasks *Produce Statistics* and *Adaptation*. To manage internet orders, *Internet Order Handled* is achieved through the task *Shopping Cart* which is decomposed into subtasks *Select Item*, *Add Item*, *Check Out*, and *Get Identification Detail*. These are the main process activities required to design an operational on-line shopping cart [7]. The latter (task) is achieved either through sub-goal *Classic Communication Handled* dealing with phone and fax orders or *Internet Handled* managing secure or standard form orderings. To allow for the ordering of new items not listed in the catalogue, *Select Item* is also further refined into two alternative subtasks, one dedicated to select catalogued items, the other to preorder unavailable products. To provide sufficient support (++) to the *Adaptable* softgoal, *Adaptation* is refined into four subtasks dealing with catalogue updates, system evolution, interface updates and system monitoring. The goal *Item Searching Handled* might alternatively be fulfilled through tasks *Database Querying* or *Catalogue Consulting* with respect to customers' navigating desiderata, i.e., searching with particular items in mind by using search functions or simply browsing the catalogued products.

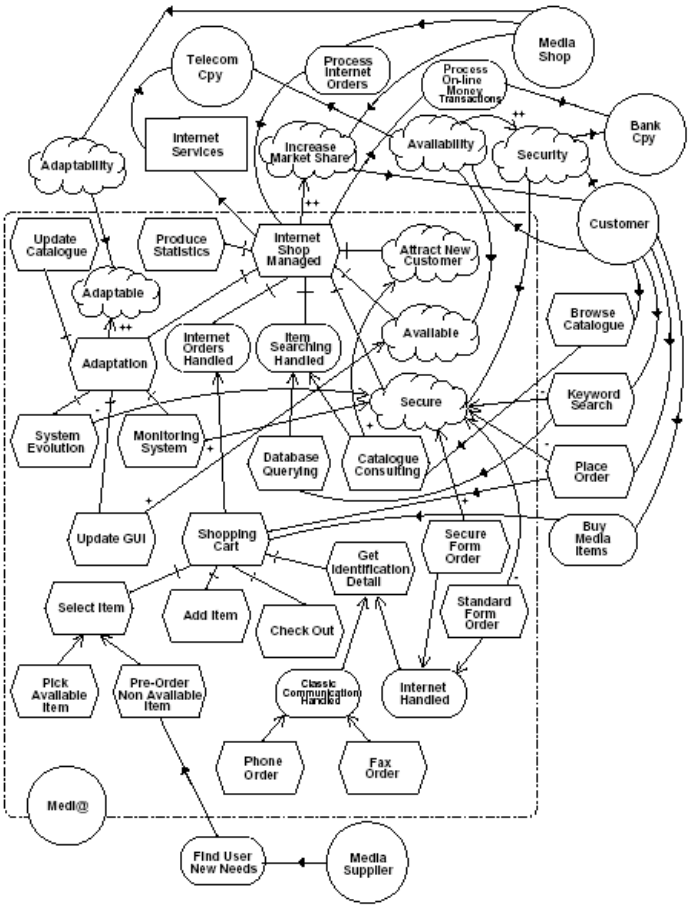

Figure 5. *Rationale diagram for* Medi@

In addition, as already pointed out, Figure 5 introduces softgoal contributions to model sufficient/partial positive (respectively ++ and +) or negative (respectively -- and -) support to softgoals *Secure*, *Available*, *Adaptable*, *Attract New Customers* and *Increase Market Share*. The result of this means-ends analysis is a set of (system and human) actors who are dependees for some of the dependencies that have been postulated.

Resource, task and softgoal dependencies correspond naturally to functional and non-functional requirements. Leaving (some) goal dependencies between system actors and other actors is a novelty. Traditionally, functional goals are "operationalized" during late requirements [8], while quality softgoals are either operationalized or "metricized" [9]. For example, *Billing Processor* may be operationalized during late requirements analysis into particular business processes for processing bills and orders. Likewise, a security softgoal might be operationalized by defining interfaces which minimize input/output between the system and its environment, or by limiting access to sensitive information. Alternatively, the security requirement may be metricized into something like "No more than X unauthorized operations in the system-to-be per year".

Leaving goal dependencies with system actors as dependees makes sense whenever there is a foreseeable need for flexibility in the performance of a task on the part of the system. For example, consider a communication goal "communicate X to Y". According to conventional development techniques, such a goal needs to be operationalized before the end of late requirements analysis, perhaps into some sort of a user interface through which user Y will receive message X from the system. The problem with this approach is that the steps through which this goal is to be fulfilled (along with a host of background assumptions) are frozen into

the requirements of the system-to-be. This early translation of goals into concrete plans for their fulfillment makes systems fragile and less reusable.

In our example, we have left three (soft)goals *Availability*, *Security*, *Adaptability*) in the late requirements model. The first goal is *Availability* because we propose to allow system agents to automatically decide at run-time which catalogue browser, shopping cart and order processor architecture fit best customer needs or navigator/platform specifications. Moreover, we would like to include different search engines, reflecting different search techniques, and let the system dynamically choose the most appropriate. The second key softgoal in the late requirements specification is *Security*. To fulfil it, we propose to support in the system's architecture a number of security strategies and let the system decide at run-time which one is the most appropriate, taking into account environment configurations, web browser specifications and network protocols used. The third goal is *Adaptability*, meaning that catalogue content, database schema, and architectural model can be dynamically extended or modified to integrate new and future web-related technologies.

6 ARCHITECTURAL DESIGN

Figure 6 suggests a possible assignment of system responsibilities for *E-Media* following the structure-in-5 style [11]. It is decomposed into five principal actors *Store Front*, *Coordinator*, *Billing Processor*, *Back Store* and *Decision Maker*. *Store Front* serves as the *Operational Core*. It interacts primarily with Customer and provides her with a usable front-end web application for consulting and shopping media items. *Back Store* constitutes the *Support* component. It manages the product database and communicates to the *Store Front* information on products selected by the user. It *stores* and *backs up* all web information from the *Store Front* about customers, products, sales, orders and bills to produce *statistical information* to the *Coordinator*. It provides the *Decision Maker* with *strategic information* (analyses, historical charts and sales reports).

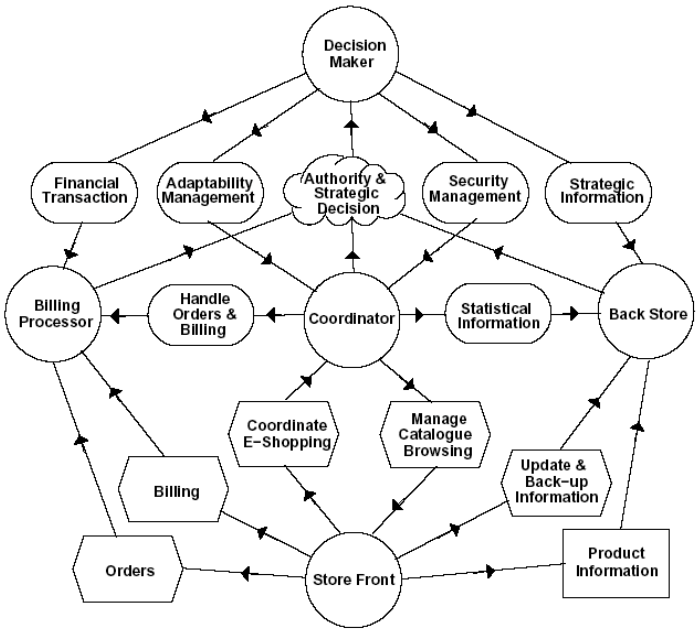

Figure 6. *The Medi@ Architecture in Structure-in-5*

The *Billing Processor* is in charge of handling orders and bills for the *Coordinator* and implementing the corresponding procedures for the *Store Front*. It also ensures the secure

management of financial transactions for the *Decision Maker*. As the *Middle Line*, the *Coordinator* assumes the central position of the architecture. It ensures the coordination of *e-shopping* services provided by the *Operational Core* including the management of conflicts between itself, the *Billing Processor*, the *Back Store* and the *Store Front*. To this end, it also handles and implements strategies to manage and prevent *security* gaps and *adaptability* issues. The *Decision Maker* assumes the *Strategic Apex* role. To this end, it defines the *Strategic Behavior* of the architecture ensuring that objectives and responsibilities delegated to the *Billing Processor*, *Coordinator* and *Back Store* are consistent with that global functionality.

Three software quality attributes have been identified as being particularly strategic for e-business systems [12]:

Adaptability deals with the way the system can be designed using generic mechanisms to allow web pages to be dynamically changed. It also concerns the catalogue update for inventory consistency.

The **structure-in-5** separates independently each typical component of the *E-Media* architecture isolating them from each other and allowing dynamic manipulation.

Security. Clients, exposed to the internet are, like servers, at risk in web applications. It is possible for web browsers and application servers to download or upload content and programs that could open up the client system to crackers and automated agents. JavaScript, Java applets, ActiveX controls, and plug-ins represent a certain risk to the system and the information it manages. Equally important are the procedures checking the consistency of data transactions.

In the *structure-in-5*, checks and control mechanisms can be integrated at different levels assuming redundancy from different perspectives. Contrary to the classical layered architecture [33], checks and controls are not restricted to adjacent levels. Besides, since the structure-in-5 permits the separation of process (*Store Front*, *Billing Processor* and *Back Store*) from control (*Decision Maker* and *Monitor*), security and consistency of these two hierarchies can also be verified independently.

Availability. Network communication may not be very reliable causing sporadic loss of the server. There are data integrity concerns with the capability of the e-business system to do what needs to be done, as quickly and efficiently as possible in particular with the ability of the system to respond in time to client requests for its services.

The *structure-in-5* architecture prevents availability problems by differentiating process from control. Besides, contrary to the classical layered architecture [33], 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.

7 DETAILED DESIGN

Figure 7 shows a possible use of the patterns for the Store Front component of the e-business system of Figure 6. In particular, it shows how to realize the dependencies *Manage catalogue browsing*, *Update Information* and *Product Information* from the point of view of the Store Front. The Store Front and the dependencies are decomposed into a combination of social patterns [11] involving agents, pattern agents, subgoals and subtasks.

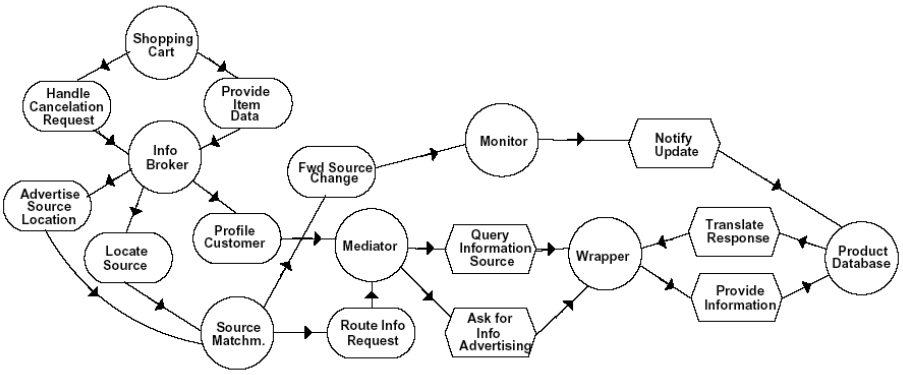

***Figure 7**. Decomposing the Store Front with Social Patterns*

The booking pattern is applied between the *Shopping Cart* and the *Information Broker* to reserve available items. The broker pattern is applied to the *Information Broker*, which satisfies the Shopping Cart's requests of information by accessing the *Product Database*. The *Source Matchmaker* applies the matchmaker pattern to locate the appropriate source for the *Information Broker*, and the monitor pattern is used to check any possible change in the *Product Database*. Finally, the mediator pattern is applied to dispatch the interactions between the *Information Broker*, the *Source Matchmaker*, and the *Wrapper*, while the wrapper pattern makes the interaction between the *Information Broker* and the *Product Database*.

Figure 8 shows the information broker of Figure 7. The customer sends a service request to the broker asking for buying or selling DVDs. He chooses which DVDs to sell or buy, selects the corresponding DVD titles, the quantity and the deadline (the time-out before which the broker has to realize the requested service). When receiving the customer's request, the broker interacts with the media shops. The interactions between the broker and the media shops are shown in the bottom-right corner of the figure.

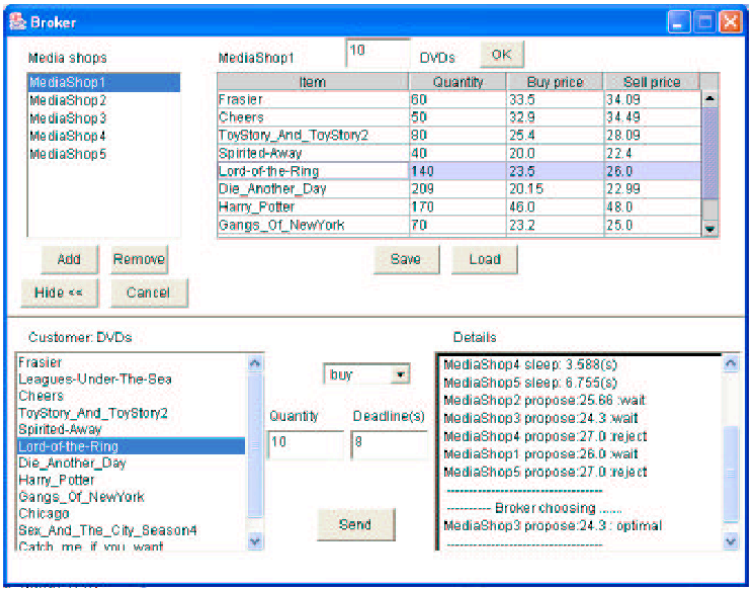

***Figure 8**. The Information Broker of* Medi@

To illustrate the integration of AUML into Tropos, the rest of the section concentrates on the *Store Front* actor. Figure 9 depicts a partial UML class diagram focusing on that actor that will be implemented as an aggregation of several *CartForm*s and *ItemLine*s.

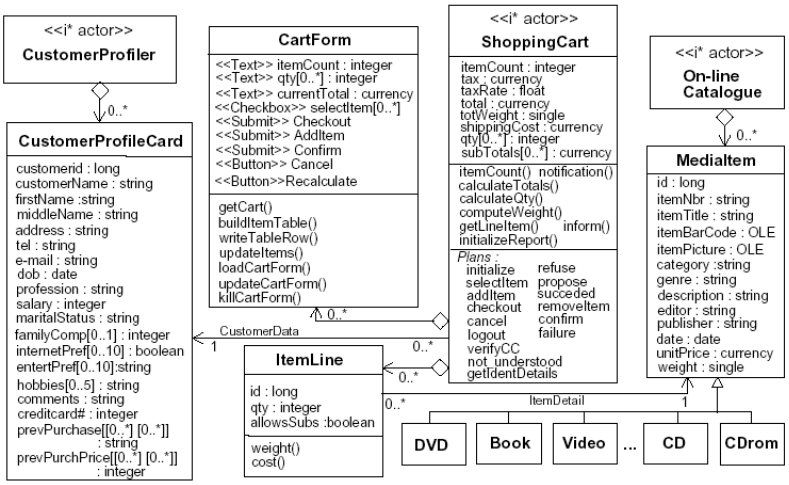

***Figure** 9*. Partial Class Diagram for *Store Front*

To specify the *checkout* operation identified in Figure 9, AUML allows us to use templates and packages to represent *checkout* as an object, but also in terms of sequence and collaborations diagrams.

Figure 10 focuses on the protocol between *Customer* and *Shopping Cart* which consists of a customization of the FIPA Contract Net protocol [29]. Such a protocol describes a communication pattern among actors, as well as constraints on the contents of the messages they exchange. When a *Customer* wants to check out, a request-for-proposal message is sent to *Shopping Cart*, which must respond before a given timeout (for network security and integrity reasons). The response may refuse to provide a proposal, submit a proposal, or express miscomprehension. The diamond symbol with an "X" indicates an "exclusive or" decision. If a proposal is offered, *Customer* has a choice of either accepting or canceling the proposal.

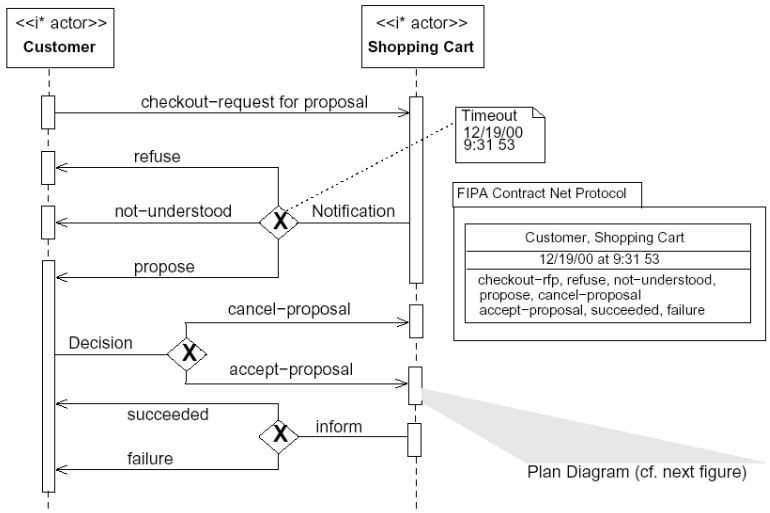

***Figure 10**. Agent Interaction Protocol Focusing on a* Checkout *Dialogue*

At the lowest level, we use plan diagrams [20], to specify the internal processing of atomic actors. Each identified plan is specified as a plan diagram, which is denoted by a rectangular box. The lower section, the plan graph, is a state transition diagram. However, plan graphs are not just descriptions of system behavior developed during design. Rather, they are directly executable prescriptions of how a BDI agent should behave (execute identified plans) to achieve a goal or respond to an event.

The initial transition of the plan diagram is labeled with an activation event (*Press checkout button*) and activation condition (*[checkout button activated]*) which determine when and in what context the plan should be activated. Transitions from a state automatically occur when exiting the state and no event is associated (e.g., when exiting *Fields Checking*) or when the associated event occurs (e.g., *Press cancel button*), provided in all cases that the associated condition is true (e.g., *[Mandatory fields filled]*). When the transition occurs any associated action is performed (e.g., *verifyCC()*).

The elements of the plan graph are three types of node; start states, end states and internal states, and one type of directed edge; transitions. Start states are denoted by small filled circles. End states may be pass or fail states, denoted respectively by a small target or a small no entry sign. Internal states may be passive or active. Passive states have no substructure and are denoted by a small open circle. Active states have an associated activity and are denoted by rectangular boxes with rounded corners. An important feature of plan diagrams is
their notion of failure. Failure can occur when an action upon a transition fails, when an explicit transition to a fail state occurs, or when the activity of an active state terminates in
failure and no outgoing transition is enabled.

Figure 11 depicts the plan diagram for *checkout*, triggered by pushing the checkout button. Mandatory fields are first checked. If any mandatory fields are not filled, an iteration allows the customer to update them. For security reasons, the loop exits after 5 tries ([I<5]) and causes the plan to fail. Credit Card validity is then checked. Again for security reasons, when not valid, the CC# can only be corrected 3 times. Otherwise, the plan terminates in failure. The customer is then asked to confirm the CC# to allow item registration. If the CC|# is not confirmed, the plan fails. Otherwise, the plan continues: each item is iteratively registered, final amounts are calculated, stock records and customer profiles are updated and a report is displayed. When finally the whole plan succeeds, the *Shopping Cart* automatically logs out and asks the *Order Processor* to initialize the order. When, for any reason, the plan fails, the *Shopping Cart* automatically logs out. At anytime, if the cancel button is pressed, or the timeout is more than 90 seconds (e.g., due to a network bottleneck), the plan fails and the *Shopping Cart* is reinitialized.

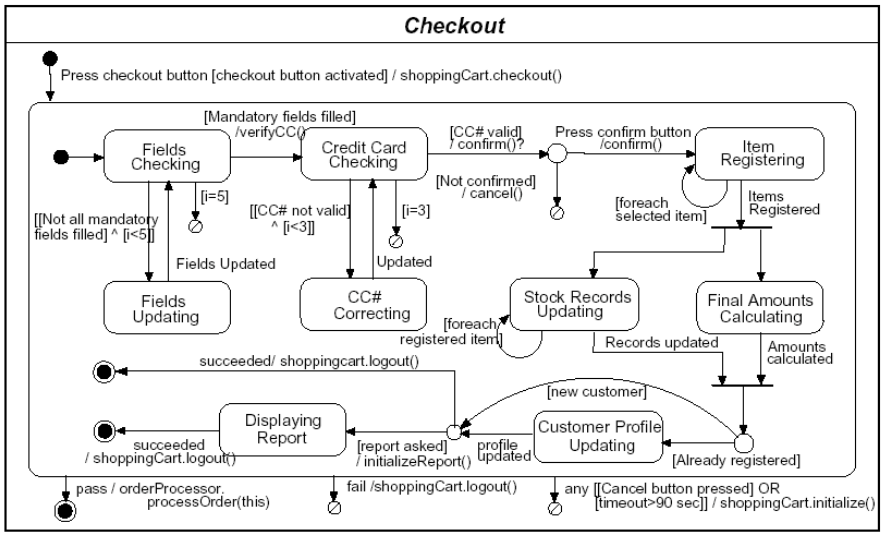

***Figure 11**. A Plan Diagram for* Checkout

8 CONCLUSIONS

We have proposed a development methodology named *Tropos*, founded on intentional and social concepts, and inspired by early requirements analysis. The modeling framework views software from five complementary perspectives:

- **Social** - who are relevant actors, what do they want? What are their obligations? What are their capabilities?
- **Intentional** - relevant goals and how they interact? How are they being fulfilled, and by whom?
- **Communicational** - how do actors communicate with each other?
- **Process-oriented** - what are relevant business/computer processes? Who is responsible for what?
- **Object-oriented** - what are relevant objects and classes, along with their inter-relationships?

We believe that the methodology is particularly appropriate for generic, component-based systems for e-business applications that can be downloaded and used in a variety of operating environments and computing platforms. The requirements-driven approach, on which Tropos is based, suggests that the methodology complements well proposals for agent-oriented programming environments [4,30] given that the software is defined in terms of (system) actors, goals and social dependencies among them. Moreover, it does not force the developer to operationalize these intentional and social structures early on during the development process, thereby avoiding the hardwiring of solutions into software requirements.

There are other notable proposals for agent-oriented software development, such as [2,18,19,20,31,35,36]. Such proposals are mostly extensions to known object-oriented and/or knowledge engineering methodologies. Moreover, all these proposals focus on design -as opposed to requirements analysis -- and are therefore considerably narrower in scope than *Tropos*. Indeed, *Tropos* proposes to adopt the same concepts, inspired by requirements modeling research, for describing requirements *and* system design models in order to narrow the semantic gap between them. The architecture and software design models produced within our framework

are intentional in the sense that system actors have associated goals that are supposed to be fulfilled. They are also social in the sense that each component (actor) has obligations/expectations towards/from other components. Obviously, such models are best suited for cooperative, dynamic and distributed applications.

The Tropos methodology is not intended for any type of software. For system software (such as a compiler) or embedded software, the operating environment of the system-to-be is an engineering artifact, with no identifiable stakeholders. In such cases, traditional software development techniques may be most appropriate. However, a large and growing percentage of software does operate within open, dynamic organizational environments. For such software, the Tropos methodology and others in the same family apply and promise to deliver more robust, reliable and usable software systems. The Tropos methodology in its current form is also not suitable for sophisticated software agents requiring advanced reasoning mechanisms for plans, goals and negotiations. Further extensions will be required, mostly at in detailed design phase, to address this class of software applications.

Much remains to be done to further refine the proposal and validate its usefulness with very large case studies. We are currently working on the development of additional formal analysis techniques for Tropos including goal analysis and social structures engineering. We are also developing tools that support different phases of the methodology.

Acknowledgments We thank all contributors to the Tropos project -- in Trento, Toronto, Louvain, Recife and elsewhere - for useful comments, discussions and feedback.

REFERENCES

[1] Y. Aridor and D. Lange. Agent design patterns: Elements of agent application design. In *Proc. of the 2nd Int. Conf. on Autonomous Agents, Agents'98*, pages 108–115, St. Paul, USA, May 1998.

[2] B. Bauer, J. Muller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In *Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering, AOSE'00*, pages 91–104, Limerick, Ireland, 2001.

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

[4] J. Castro, M. Kolp, and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems* Vol. 27, Elsevier, Amsterdam, the Netherlands, 2002.

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

[6] P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 32(3):213–261, 1990.

[7] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000.

[8] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[9] A. Davis. *Software Requirements: Objects, Functions and States*. Prentice Hall, 1993.

[10] T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press, 1978.

[11] T. Do, M. Kolp, and A. Pirotte. Social patterns for designing multi-agent systems. In *Proc. of the 15th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'03*, 2003.

[12] T. T. Do, S. Faulkner, and M. Kolp. Organizational multi-agent architectures for information systems. In *Proc. of the 5th Int. Conf. on Enterprise Information Systems, ICEIS'03*, Angers, France, April 2003.

[13] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1997.

[14] FIPA. The Foundation for Intelligent Physical Agents. At http://www.fipa.org, 2001.

[15] E. Gamma, R. Helm, J. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
[16] 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.
[17] 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.
[18] C. Iglesias, M. Garrijo, and J. Gonzalez. A survey of agent-oriented methodologies. In *Proc. of the 5th Int. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages, ATAL'98*, pages 317–330, Paris, France, Oct. 1999.
[19] N. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
[20] D. Kinny and M. Georgeff. Modelling and design of multi-agent systems. In *Proc. of the 3rd Int.Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages, ATAL'96*, pages 1–20, Budapest, Hungary, Aug. 1997.
[21] M. Kolp, P. Giorgini, and J. Mylopoulos. A goal-based organizational perspective on multi-agents architectures. In *Proc. of the 8th Int. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages, ATAL'01*, Seattle, USA, Aug. 2001. [22] M. Kolp, P. Giorgini, and J. Mylopoulos. Organizational multi-agent architectures: A mobile robot example. In *Proc. of the 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems, AAMAS'02*, Bologna, Italy, July 2002.
[23] M. Kolp, P. Giorgini, and J. Mylopoulos. Organizational patterns for early requirements analysis. In *Proc. of the 15th Int. Conf. on Advanced Information Systems, CAiSE'03*, Velden, Austria, June 2003.
[24] P. Kruchten. *The Rational Unified Process: An introduction*. Addison-Wesley, 2003.
[25] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages. *Intelligent Systems*, 14(2):45–52, 1999.
[26] H. Mintzberg. *Structure in fives : designing effective organizations*. Prentice-Hall, 1992.
[27] J. Morabito, I. Sack, and A. Bhate. *Organization modeling : innovative architectures for the 21st century*. Prentice Hall, 1999.
[28] J. Mylopoulos, M. Kolp, and J. Castro. UML for agent-oriented software development: The Tropos proposal. In *Proc. of the 4th Int. Conf. on the Unified Modeling Language UML'01*, Toronto, Canada, Oct. 2001.
[29] J. Odell, H. Van Dyke Parunak, and B. Bauer. ExtendingUMLfor agents. In *Proc. of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS'00*, pages 3–17, Austin, USA, July 2000.
[30] 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.
[31] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge engineering and management: the CommonKADS methodology*. MIT Press, 2000.
[32] W. Scott. *Organizations: rational, natural, and open systems*. Prentice Hall, 1998.
[33] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
[34] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall, 1990.
[35] M. Wood and S. DeLoach. An overview of the multiagent systems engineering methodology. In *Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering, AOSE'00*, pages 207–222, Limerick, Ireland, 2001.
[36] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
[37] E. Yourdon and L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, 1979.
[38] E. Yu. Modeling organizations for information systems requirements engineering. In *Proc. of the 1st Int. Symposium on Requirements Engineering, RE'93*, pages 34–41, San Jose, USA, Jan. 1993.
[39] E. Yu. Understanding 'why' in software process modeling, analysis and design. In *Proc. of the 16th Int. Conf. on Software Engineering, ICSE'94*, pages 159–168, Sorrento, Italy, May 1994.
[40] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of

Toronto, Department of Computer Science, 1995. [41] E. Yu and J. Mylopoulos. Using goals, rules, and methods to support reasoning in business process reengineering. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 5(1):1–13, 1996.

HOW TO USE THIS IGI CHAPTER TEMPLATE

Please use this template when you are creating your paper/ chapter to submit to an Idea Group Inc. publication. By having all paper/chapter authors using the same format, there will be less chance for formatting errors to occur throughout the book, thus making a more presentable, readable final book.

Below you will see examples of several items that have been problem areas in IGI books/journals in the past. Please browse through the samples before you begin your chapter on the following page. For the body of your chapter, use Styles such as Heading 1-3, Body Text, Block Quotation, and List Bullet from the Style control on the Formatting toolbar.

FOR BOOK CHAPTERS ONLY--This chapter template is complete with style for Index entries. From the Insert menu, choose Index and Tables. Click on Index and "Mark". Please only select the first instance of each term you want in the index. Be sure to choose the Custom Format or use the shortcut below:

> *Index entries. To insert an index entry field, select the text to be indexed, and press ALT+SHIFT+X.. On the dialog click "mark".*

PAGE BREAK

Since your full chapter will be typeset by the IGI staff in Adobe InDesign, there is no need to for you to worry about orphan/widow lines in your submitted chapter. This will be taken care of during the typesetting phase. Therefore, please DO NOT insert page breaks in your chapter.

FORMATTING REFERENCES IN APA STYLE

APA (American Psychological Association) style should be followed for the references. References should relate only to material cited within the manuscript and be listed in alphabetical order at the end of the chapter/paper. When you use the source in the text, author's name and year of publication should appear [An example of this is, (Travers, 1995)]. Please do not include any abbreviations. See the following examples:

Example 1: Single author periodical publication.
Smith, A.J. (2003). Databases and organizations. *Database Ideology Review*. 16(2), 1-15.
Example 2: Multiple authors periodical publication.
Smith, A.J., & Brown, CJ. (2003). Organizations and Database Management. *Data Source*, 10(4), 77-88.
Example 3: Books:
Smith, A.J. (2003). *Database Booklet*. New York: J.J. Press.

State author's name and year of publication where you use the source in the text. See the following examples:
Example 1: In most organizations, data resources are considered to be a major resource (Brown, 2002; Smith, 2001). **Example 2:** Brown (2003) states that the value of data is recognized by most organizations.
The author's name, date of publication, and the page(s) on which the quotation appears in the original text should follow direct quotations of another author's work.
Example 1: Brown (2002) states that "the value of data is realized by most organizations" (p. 45).

Example 2: "In most organizations, data resources are considered to be a major organization asset" (Smith, 2003, pp. 35-36) and must be carefully monitored by the senior management.

FIGURE AND TABLES IN IGI PUBLICATION

For the best final results of figures/tables in your paper, please submit them in the actual size in which they will appear in the final version of your chapter. The printable-page size (the image area) of all IGI publications is 5" x 7 1/4" so DO NOT submit figures or tables that are larger than this because during reduction, details are often lost or type becomes so small that it is hard to read in the printed book. Be advised, if your figures look blurred or unreadable in your Word copy of your manuscript, this is how they will look in the final typeset version of the book. So please be sure to send high quality images, saved at a 112 dpi minimum setting. If the figures are embedded in the Word document please also include as a separate tif, jpeg, or eps file on your disk.

THE IMPORTANCE OF YOUR ABSTRACT

IGI, as a scholarly publishing company, provides abstracts for book chapters, journal articles, etc. to a variety of scholarly indexes for inclusion. Therefore, it is extremely important that your final abstract clearly describes the essence of your work in your chapter. Below is a sample of an abstract that clearly states the purpose of the chapter and summarizes the content. Please follow the sample to create a clear description of your work for better recognition within the indexes.

Sample Abstract

This chapter introduces the Chaos Theory as a means of studying information systems. It argues that the Chaos Theory, combined with new techniques for discovering patterns in complex quantitative and qualitative evidence, offers a potentially more substantive approach to understanding the nature of information systems in a variety of contexts. Furthermore, the authors hope that understanding the underlying assumptions and theoretical constructs through the use of the Chaos Theory will not only inform researchers of a better design for studying information systems, but also assist in the understanding of intricate relationships between different factors.

First, the authors describe what the chapter is about. (*This chapter introduces the Chaos Theory as a means of studying information systems.*)
They summarize the content of the chapter. (*It argues that the Chaos Theory, combined with new techniques for discovering patterns in complex quantitative and qualitative evidence, offers a potentially more substantive approach to understanding the nature of information systems in a variety of contexts.*)
Then, they explain their purpose or objectives for writing the chapter. (*Furthermore, the authors hope that understanding the underlying assumptions and theoretical constructs through the use of the Chaos Theory will not only inform researchers of a better design for studying information systems, but also assist in the understanding of intricate relationships between different factors.*)

Note: Your abstract does not necessarily need to be three sentences like the sample above – but it will need to be between 100-150 words, nor does it need to be worded the same way. Use your own words, but capture the idea behind this sample abstract.

Begin your paper/chapter here