Dealing with Complexity Using Conceptual Models Based on *Tropos*

Jaelson Castro[1], Manuel Kolp[2], Lin Liu[3], and Anna Perini[4]

[1] Universidade Federal de Pernambuco, Recife, Brazil
jbc@cin.ufpe.br
[2] University of Louvain, LSM-ISYS, Louvain-la-Neuve, Belgium
manuel.kolp@uclouvain.be
[3] Tsinghua University, Beijing, China
linliu@tsinghua.edu.cn
[4] Fondazione Bruno Kessler – Irst, CIT, Trento, Italy
perini@fbk.eu

Abstract. Since its establishment, the major objective of the Tropos methodology has been to develop an approach for the systematic engineering of agent-oriented information systems. In this chapter we illustrate a number of approaches to deal with complexity, which address different activities in software development and are deemed to be used in combination. We begin with handling complexity at requirements levels. In particular we examine how modularization can be improved using some of Aspect Oriented Software Development Principles. We then examine how model-based testing applied in parallel to requirements analysis and design can support incremental validation and testing of software components, as well as help to clarify ambiguities in requirements. We also look at how Tropos can help to address complexity in social context when making design decisions. Last but not least, we show how to tackle complexity at the process modelling level. We explore iterative development extension to Tropos as well as perspectives taken from software project management. This allows us to deal with the complexity of large real world projects.

Keywords: requirements engineering, complexity, aspect, goal modelling, testing, process modelling.

1 Introduction

Enterprises are continually changing their internal structures and business processes, as well as their external alliances, as they strive to improve and grow. Software systems that operate within such a setting have to evolve continuously to accommodate new technologies and meet new requirements. Indeed, it is well known that the latest generation of software systems, such as Enterprise Resource Planning (ERP), groupware, knowledge management and e-business systems, should be designed to perform within ever-changing organizational environments.

These features will characterize more and more future software systems [14], which will be employed by an increasing number of people for different purposes and using a variety of devices. These systems will use a growing amount of data stored, accessed, manipulated, and refined in a distributed way, and rest on a set of interdependencies among software components (services), which may dynamically change. This increase in size (scale) and dynamicity are considered major sources of complexity of software systems, which calls for new solution approaches and new concepts for system design, development, operation, and evolution. A rich research agenda is proposed in [14] that highlights, for example, the need of more expressive modeling languages and of revisiting model-based, aspect-oriented, and other generative methods for helping to validate and certify requirements.

The Tropos project [24], [5] was launched with the objective to develop a methodology for building agent-oriented information systems, in competition with existing methodologies founded on structured and object-oriented concepts. Agent-based technologies are considered a promising solution towards the realization of software having flexibility and (self-)adaptive properties [23], moreover the agent paradigm offers a suitable set of concepts to model large-scale systems in terms of socio-technical ecosystems [14].

The Tropos methodology rests on the idea of starting by building a model of the organizational context within which the system-to-be will eventually function, then the system-to-be is introduced and the model is incrementally refined and extended with a definition of the functional and non-functional requirements of the system-to-be. This model provides a common interface to the various software development activities. The model also serves as a basis for documentation and evolution of the software system. The approach is requirements-driven in the sense that the concepts used to define requirements for a software system are also used later on during design and implementation. To this end, Tropos adopts the concepts offered by *i** [37], a modelling framework proposing concepts such as *actor* (actors can be *agents*, *positions* or *roles*), social dependency among actors, including *goal*, *softgoal*, *task* and *resource* dependencies. Thus, an actor can depend upon another one to satisfy a goal, execute a task, and provide a resource or satisfice a softgoal. Softgoals are associated to non-functional requirements, while goals, tasks and resources are associated to system functionalities. The i* framework offers two models: the Strategic Dependency (SD) and Strategic Rationale (SR). The SD model consists of a set of nodes and links connecting them, where nodes represent actors and each link indicates a dependency between two actors (dependum). The depending actor is called depender, and the actor who is depended upon is called the dependee. The SR model provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships.

Researches on how to manage software complexity with Tropos have been conducted in parallel, by different research groups. They attack the problem from a number of perspectives, involving technical (e.g. related to issues in modeling and validation of requirements), as well as managerial and automation issues.

In this chapter, we illustrate some of them, with the objective to show their complementarities and the potential to be used in combination.

It is well known that requirements models may become cluttered, compromising their evolution and scalability. In fact, empirical evaluation has shown that there is a lack of modularity in the *i** framework [13]. This is a serious drawback for large and complex projects. In fact, i* models tend to include scattered and tangled representations, i.e. crosscutting, resulting in models with poor modularization and, therefore, harder to understand and maintain [1]. In Section 3, we present an approach to create modular *i*/Tropos* models, where a desired concern is separated as an individual actor. Consequently, improved modularization mechanisms are required to avoid the crosscutting representations in *i* /Tropos* models.

Developing complex software systems requires that single components, as well as the overall system, are incrementally validated and certified against requirements and user expectations, along the whole development process. This motivated the adoption of a V-model[1] approach to software development in Tropos that complement analysis and design with validation phases, which is called Goal-Oriented Software Testing (GOST) [26]. Section 4 recalls basic elements of GOST and illustrates how it works when validating early requirements against user goals. A further benefit of the GOST methodology will also emerge since early test specification will require clarifying ambiguities in the requirements model, then improving the requirements model itself.

In order to design a better information system, a designer would like to have notations to visualize how design experts' know-how can be applied according to one's specific social and technology situation. Section 5 proposes the combined use of *Tropos* and a scenario-oriented notation *UCM* for representing design knowledge of information systems. So that goal models are combined with scenarios descriptions to complement each other to handle complexity in design decision making. The combined use of GRL (a variant of *i**/Tropos) and a scenario mapping approach is part of the Users Requirements Notation (URN), a newly approved ITU-T standard [17] [1][22].

Section 6 proposes *I-Tropos*, a software project management framework dedicated to extend *Tropos* with an iterative life cycle. The process fills the project and product life cycle gaps of Tropos and offers a goal-oriented project management perspective to support project stakeholders for applying Tropos on large information systems. It is supported by *DesCARTES* a specific CASE tool.

2 Running Example

In order to illustrate Tropos requirements models, let us consider the Media Shop example presented in [10]. *Media Shop* is a store which sells and ships different kinds of media items, such as books, newspapers, magazines. To increase market share, *Media Shop* has decided to use the *Medi@* system, a business to customer retail sales front-end on the Internet to allow an on-line customer to examine the items in its catalogue and place orders.

The system uses communication facilities provided by *Telecom Cpy*. 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. 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

[1] The V-Model gets its name from the fact that the process is often mapped out as a flowchart that takes the form of the letter V: the left edge defines a sequence of analysis and design activities, the right edge the corresponding set of validation and testing activities.

search. If the item is not available in the catalogue, the customer has the option of asking *Media Shop* to order it.

In Figure 1 you can find an expanded description of the *Medi@ actor*. In this actor, a root task *Manage Internet Shop* is specified (located at the centre-top of the larger circle that represents the *Medi@* actor's boundary). That task is firstly refined into *Item Searching Handled* goal, *Secure, Adaptable* and *Available* softgoals, and *Produce Statistics* task. These intentional elements are further refined by using task-decomposition, means-end and contribution links to define the *Medi@* system requirements. These three new types of relationships are explained as follows: (i) task-decomposition links describe what should be done to perform a certain task (e.g., the relationship between the *Provide Access Link* task and the *Provided Internet Service* task inside the *Telecom Cpy* actor); (ii) means-end links suggest that one model element can be offered as a means to achieve another model element (e.g., relationship between the *Chose Non-Available Item* task and the *Item Selection* goal inside the Medi@ actor); (iii) contributions links suggest how a task can contribute (positively or negatively) to satisfy a softgoal (e.g., the relationship between the *Use Fault-Tolerant Strategies* task and the *Available* softgoal inside the Medi@ actor).

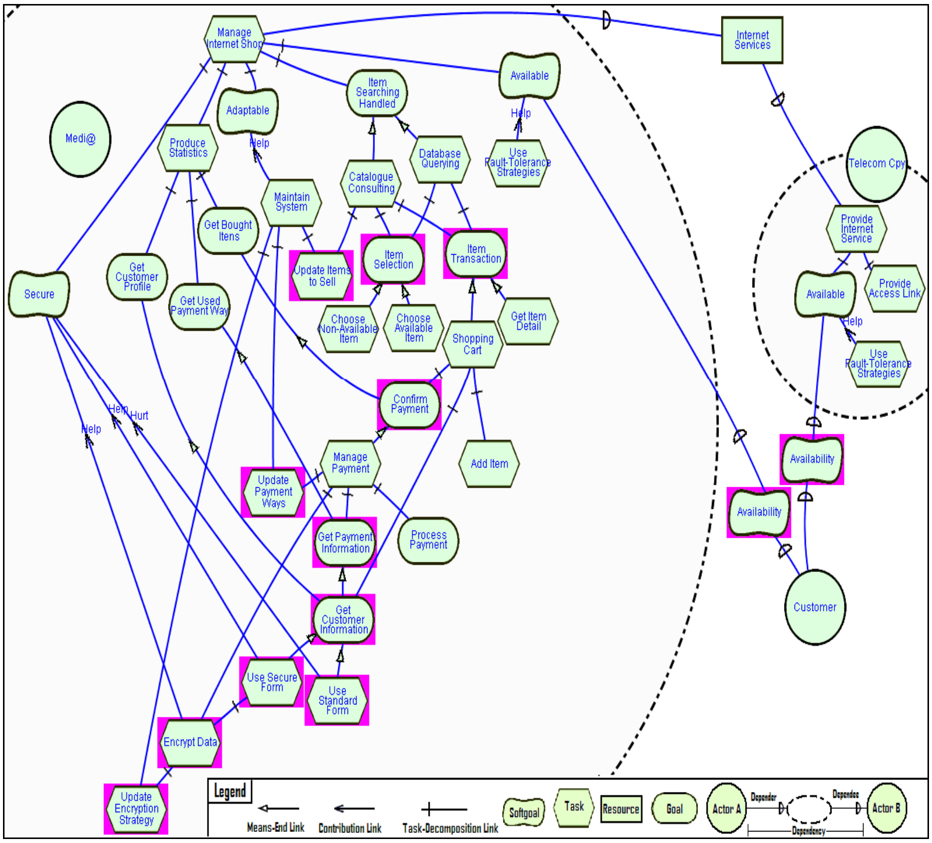


Fig. 1. The Medi@ Strategic Rationale Model

Apart from the previous three types of relationships, there are intentional dependencies between actors, which can be of four types: goal, task, resource or softgoal. For example, the *Customer* actor (depender) is related to *Medi@* (dependee) actor through *Availability* goal (dependum).

3 Modularization of Requirements Models

As the problem at hand grows, *i*/Tropos* models may become cluttered, compromising their evolution and scalability. This is a serious drawback for large and complex projects. In fact, *i** models tend to include scattered and tangled representations, i.e. crosscutting, resulting in models with poor modularization and, therefore, harder to understand and maintain [1]. This problem could be avoided if some approach was available to create modular *i*/Tropos* models, where a desired concern is separated as an individual actor. Consequently, improved modularization mechanisms are required to avoid the crosscutting representations in i*/Tropos models.

In the sequel we introduce (i) a set of guidelines to identify crosscutting concerns in i* models; and (ii) propose an extension of the *i** modelling language [37] by adding aspectual constructors to modularize crosscutting concerns and to allow its graphical composition with other system modules.

3.1 Identifying and Modularizing Aspects

The following three guidelines described helps to identify aspectual elements.

Guideline G1 (Repeated dependum): *if a dependum (i.e. a goal, a task, a resource or a softgoal) is provided by at least two dependee actors, and the subgraph operationalizing that dependum is handled equally by all dependee actors, then this operationalization is part of an aspectual element.*

This guideline aims at identifying dependencies in a SD model that have been repeated and addressed in similar ways. Thus, if a dependency has multiple similar occurrences, i.e. different dependee actors can handle (operationalize) it in the same way, the elements contained in the operationalization sub-graph can be relocated to an aspectual element. For example, in Figure 1, the *Availability* softgoal dependum has multiple occurrences in the model. But this repetition is not sufficient. It is also necessary to check if their respective operationalizations (inside the respective dependee actors) are the same. Notice that in *Telecom Cpy a*ctor, the *Availability* dependum is related to the softgoal *Available*, operationalized by *Use Fault-Tolerance Strategies* task. The *Medi@* actor treats *Availability* dependum similarly, since it is also operationalized by *Use Fault-Tolerance Strategies* task. This means that the entire sub-graph operationalizing the *Availability* dependum will be part of an aspectual element, because it is repeated in different actors. This element will be the *Availability Manager* aspectual element (Figure 3).

Guideline G2 (Repeated intentional element): This guideline is subdivided into three sub-guidelines: one deals with the task decomposition link; another deals with means-ends links (which includes the contribution link); and the last one deals with both links (task-decomposition and means-ends links). These sub-guidelines are applied to the intentional elements that are internal to the actor's boundary presented in the SR model.

Guideline G2.1 (Intentional element in a task-decomposition link): *if an intentional element (goal, softgoal or task) is required by (i.e., is a decomposition element of) two or more internal tasks, indicating a sharing of information, then the subgraph that contains this element as the root is part of an aspectual element.*

In Figure 1, the *Item Selection* goal is simultaneously required through task-decomposition by the tasks: *Database Querying* and *Catalogue Consulting*. Thus, the *Item Selection* goal is part of an aspectual element.

Guideline G2.2 (Intentional element in a means-end link): *if an intentional element (goal, softgoal or task) is a means element which is required by two or more end elements (indicating a sharing of information) then the sub-graph containing this element as a root will be part of an aspectual element.*

According to Figure 1, *Use Secure Form* task is simultaneously a means to *Get Customer Information* goal (end) and contributes to *Secure* softgoal. Hence, if we consider a contribution link as a means-end link (such as in [37]), the *Use Secure Form* is part of an aspectual element.

Guideline G2.3 (Intentional element is found simultaneously in a task-decomposition link and in a means-end link): *if an intentional element (goal, softgoal or task) is a means element and also is a sub-element in a task-decomposition (indicating a sharing of information) then the sub-graph containing this element as the root is part of an aspectual element.*

In Figure 1, this guideline captures the *Get Payment Information* goal which is a means to achieve *Get Used Payment Way* goal (through a means-end link) and a sub-element of *Manage Payment* task (through a task-decomposition link). Then the sub-graph with *Get Payment Information* goal as the root is part of an aspectual element.

Guideline G3 (Redundancy): *the aspectual elements identified by guidelines G1 and G2 are now merged together to remove multiple occurrences.*

In the example, we have captured *Encrypt Data* task simultaneously by the guidelines G2.2 and G2.3. To increase cohesion of the aspectual element, along with each intentional element identified by the guidelines, it is also required to extract other intentional elements related to the same concern and locate them all into the same aspectual element.

For example, in Figure 1, the tasks *Update Encryption Strategy* and *Encrypt Data* were identified by the guidelines and should be modularized by an aspectual element. However, these tasks are part of the sub-graph that operationalizes the *Secure* softgoal which was first identified, separated and located into an aspectual element. Therefore, those tasks can be seen as related to the *Security* concern. Thus, all these intentional elements can be extracted and located into the aspectual element *Security Manager* (Figure 2 (b)).

3.2 Identifying Relationship among Aspectual Elements

In parallel with the aspect identification, we may store the information of the relationships of all elements captured by the guidelines (for example into a table) to allow the automation of the process From the model in Figure 1 and the guidelines proposed in Section 3.1 we list in Table 1 some elements (for lack of space, not all captured elements are listed in the Table 1) that must be made persistent: (i) the dependee actor, which provides an intentional element, in both SD and SR models; (ii) the aspect which is the identified crosscutting intentional element; (iii) the concern addressed by intentional element; (iv) the related elements which represent the elements of the subgraph provided that they have not already been captured as aspect; (v) the chosen name for the identified aspectual element.

For example, in the case study the *Customer* depends on the *Medi@* and *Telecom Cpy* for the intentional *Availability* element (a softgoal). According to guideline G1 the characteristics of this dependency indicated the need to identify an aspect to deal with the availability concern. Hence, we need to define an appropriate name for it. For example it could be called *Availability Manager*. Its related intentional element is only the *Use Fault-Tolerance Strategies* task that is present in both original dependee actors (*Medi@* and *Telecom Cpy*). Hence, it should be transferred to that new element (*Availability Manager*). Similar analysis can be performed for the *Confirm Payment* goal (by G2.3), *Encrypt Data* task (by G2.3), and *Item Selection* task (by G2.1).

Table 1. Modularization of Aspects

Actor	**Crosscutting Element**	**Concern**	**Related Elements**	**Aspectual Element Name**
Medi@	*Confirm Payment* goal	*Payment*	*Manage Payment* task, and *Process Payment* and *Get Payment Information* goals	*Payment Processor*
Medi@	*Encrypt Data* task	Security	*Update Encryption Strategy task,* Secure softgoal	*Security Manager*
Medi@	*Item Selection* goal	*Item Selection*	*Choose Available Item* and *Choose Non-Available Item* tasks	*Item Selector*
Medi@, Telecom Cpy	*Available softgoal*	*Availability*	*Use Fault-Tolerance Strategies*	*Avalability Manager*

3.3 Representing Aspectual Elements Using the Aspectual i* Notation

A specific notation has been created to represent aspectual i* models. This leads to the addition of two new concepts in the *i** modeling language, namely aspectual element and crosscut relationship. Aspectual elements modularize crosscutting concerns and the crosscut relationship captures the information of source and target model elements, as well as, when and how an aspectual element crosscuts other model elements. For modularization purposes and following the principles of AOSD, we should extract and modularize the aspects, removing them from the original actors, and placing them in a new type of model element, the so called *Aspectual Element*. This new element is graphically represented by an actor with a vertical line crossing it (see, for example, the Security Manager element in Figure 2). An aspectual element, as well as an actor, is composed of intentional elements, whereas an intentional element can be a goal, a softgoal, a resource or a task. An aspectual element can be composed with an actor or another aspectual element through a *Crosscut Relationship*. This relationship specifies how an intentional element, located inside an aspectual element, is related with another intentional element, which is located inside an actor or another aspectual

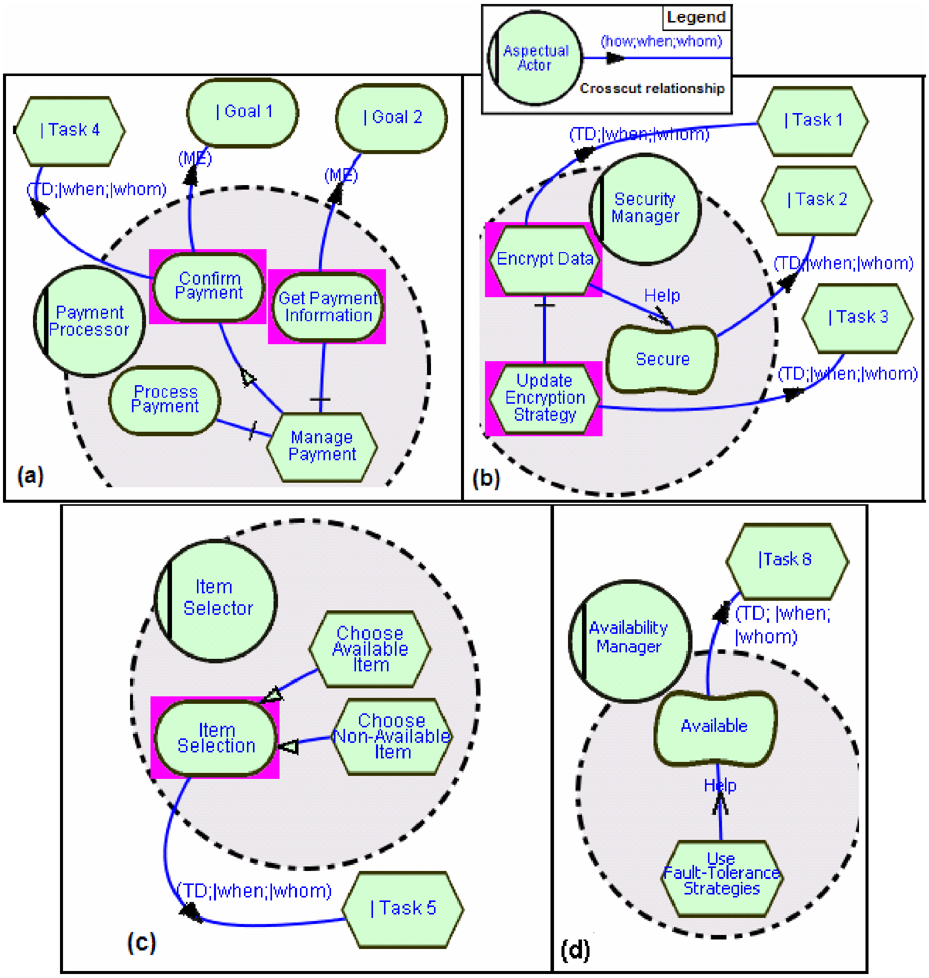


Fig. 2. Aspect Modelling for: (a) Payment Processor; (b)Security Manager; (c) Item Selector; (d) Availability Manager

element. The *how* attribute present in the crosscut relationship means the type of i* relationship (*Task-Decomposition (TD)*, *Means-End (ME)* and *Contribution*) that will be recovered with the weaving. The crosscut relationship between each aspectual element and other model elements are shown as arcs, with a dark triangle (Figure 2).

The direction indicated by the triangle suggests the way of the composition, meaning that the source element's behavior needs to be composed with the target elements' behaviors. The crosscut relationship also contains a when attribute, which can assume the values before or after, to specify when an element inside the aspectual element will be composed with an element inside another aspectual element or Actor. This composition rule must be defined taking into account the intentional element in relation to whom the composition must occur, described by the attribute whom (see, for example, the legend in Figure 2).

In order to describe the aspectual elements and to systematically compose them with other model elements, we use the concept of model roles [19] which have been used to describe object-oriented patterns, as proposed in [15], and agent-oriented patterns, as presented in [32]. They facilitate the graphical composition of concern and improve the reuse of aspectual elements.

In particular, to describe aspectual elements, it is necessary to specialize each target intentional element in a crosscut relationship and the attribute of the crosscut relationship: how, when and whom. Model roles are identified by preceding the intentional elements (goals, task, softgoals) identifiers with a "|" (see Figure 2). In practice they work as variables to be instantiated to concrete model elements.

Let us concentrate on *Payment Processor, Security Manager*, *Item Selector*, and *Availability Manager* depicted in Figure 2(a), Figure 2(b), Figure 2(c) and Figure 2(d), respectively. The composition of the aspectual elements with the original model requires the instantiation (or binding) of the model roles present in the crosscut relationship and in the target element related with the crosscut relationship. Thus, to compose the aspectual element *Payment Processor* with the *Medi@* actor, we need to bind |Goal 2 to *Get Used Payment Way* (see Figure 3). Since the relationship from a goal to another goal can only be a means-end link, the properties of that crosscut relationship do not have any model roles. Observe that the *when* and *whom* properties are used just in case we need to insert the ordering of composition. If the *when* (and, therefore, the *whom*) property is empty, then the order of the task-decomposition weaving does not matter. Finally, the composition of *Payment Processor* aspectual element with the *Medi@* Actor needs also to bind |Goal 1 to *Get Bought Items* and |Task 4 to *Shopping Cart*. For the crosscut relationship properties of |Task4, we need to bind |when to after and |whom to none (this means after all sub-elements of *Shopping Cart* Task).

Notice also that the *how* property of this crosscut relationship is already stated as TD (Task Decomposition) because the relationship from a goal to a task can only be a task-decomposition link. As a result, in Figure 3, *Payment Processor* is composed with the *Item Transactor* aspectual element by adding a task-decomposition link from *Confirm Payment* goal to *Shopping Cart* task after all intentional elements. It is also composed with the *Medi@* actor by adding both a means-end link from *Confirm Payment* goal to *Get Bought Items* task and a means-end link from *Get Payment Information* goal to *Get Used Payment Way* goal.

3.4 Performing Trade-Off Analysis

After composing the aspectual elements with the i* models using the graphical composition rules, we should identify and resolve conflicting situations that may exist in composition points [30].

A trade-off analysis method could be considered, as for example [7], when we have two or more aspectual elements composed with the same element in a base module. We start by analyzing if these aspects influence negatively on each other. In such cases, we need to choose proper trade-off analysis methods to guide the conflict resolution.

In Figure 3, one conflicting situation could be identified in the *Manage Internet Shop* task at the *Medi@* actor. In this composition point three aspects, the *Security Manager*, *Availability Manager* and *Adaptability Manager*, are composed through a task-decomposition. Therefore, it is necessary to establish their order of composition.

In general, conflict resolution might lead to a revision of the requirements specification (stakeholders' requirements, aspectual requirements or composition rules). If this happens, the requirements are recomposed, the i* models are restructured and any further conflicts arising are resolved.

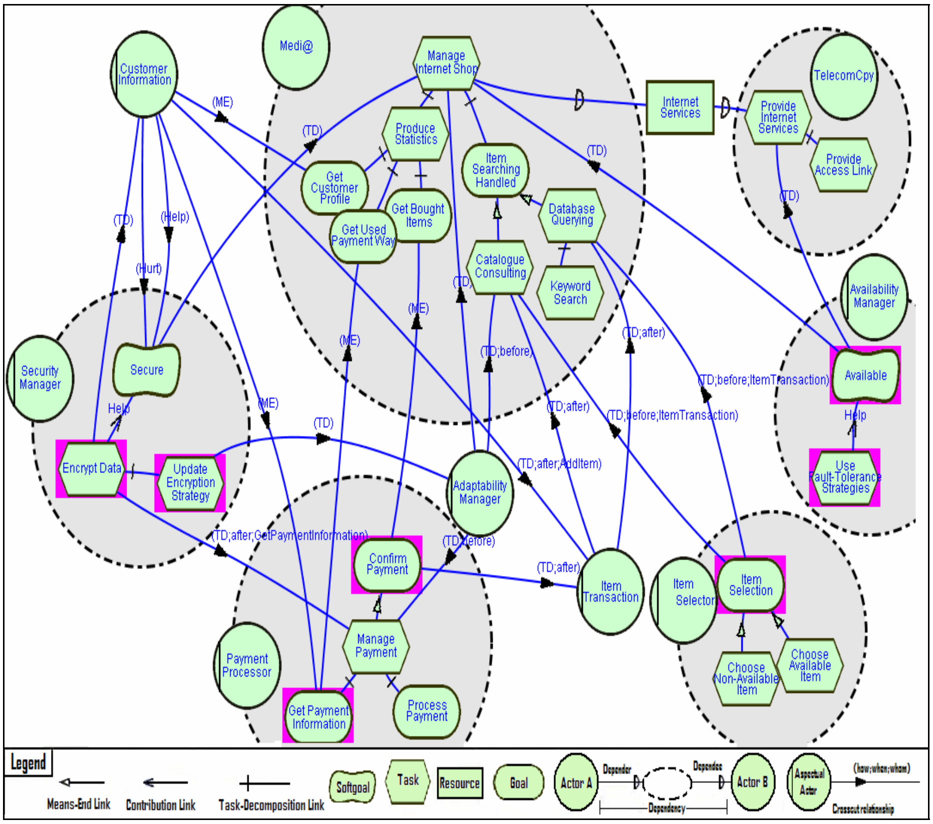


Fig. 3. The Medi@ Aspectual Strategic Rationale (SR) Model

4 Early Validation of Requirements Models

The V-model defines a software development process that supports incremental validation of software artefacts as well as code testing, according to a test-first perspective in software development that is becoming more and more compelling while the complexity of the system-to-be increases. In the V-model, validation and testing activities start at the beginning of the project, and complement requirements and design activities [12].

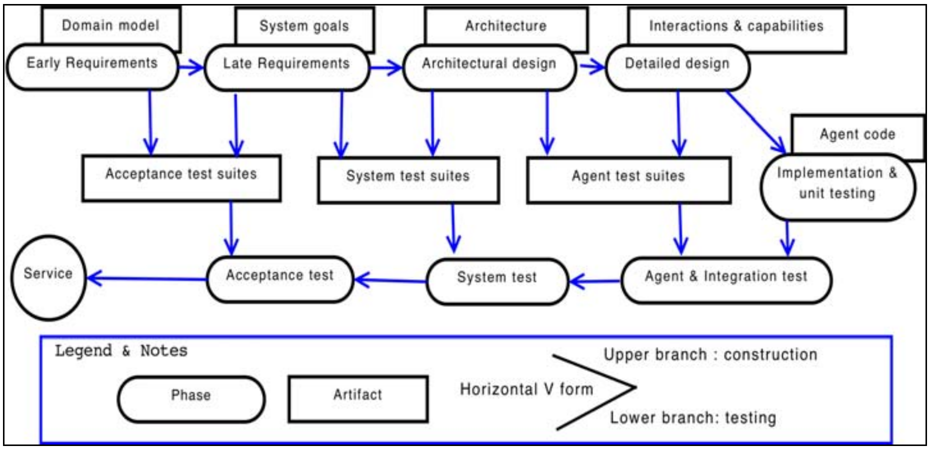


Fig. 4. The V-Model in GOST [26]

The Goal-Oriented Software Testing (GOST) approach proposed in [26], applies the V-model to the Tropos methodology [6]. In GOST test cases are derived from goal-oriented analysis and design models. GOST identifies five different validation and testing levels, each one addressing a specific objective, namely, acceptance, system, integration, agent, and unit testing. It provides also detailed procedures for deriving test suites from Tropos design artefacts, based on the relationship between design and testing artefacts depicted in Figure 4: the acceptance test's artefact is in relationship with early and late requirements models; system test with late requirements and architectural design models; agent test with architectural and detailed design; and unit testing is in relationship with detailed design and agent code.

In this section, we illustrate how the GOST approach works focusing on acceptance testing and show how we can derive test suites from early and late-requirements models using an excerpt of the Tropos requirements model of the *Medi@* system, depicted in Figure 5. Test cases for the other validation and testing levels can be derived following analogous procedures.

Acceptance testing aims at testing the software system in the customer execution environment (with the involvement of the stakeholders), and at verifying that the system meets the original stakeholder goals. In GOST, acceptance test suites (that is set of test cases) can be derived from early and late requirements models applying the following procedure[2]:

[2] The procedure to derive acceptance test suites has been here adapted since the original i* modelling language is used [37], instead of the Tropos variant described in [33].

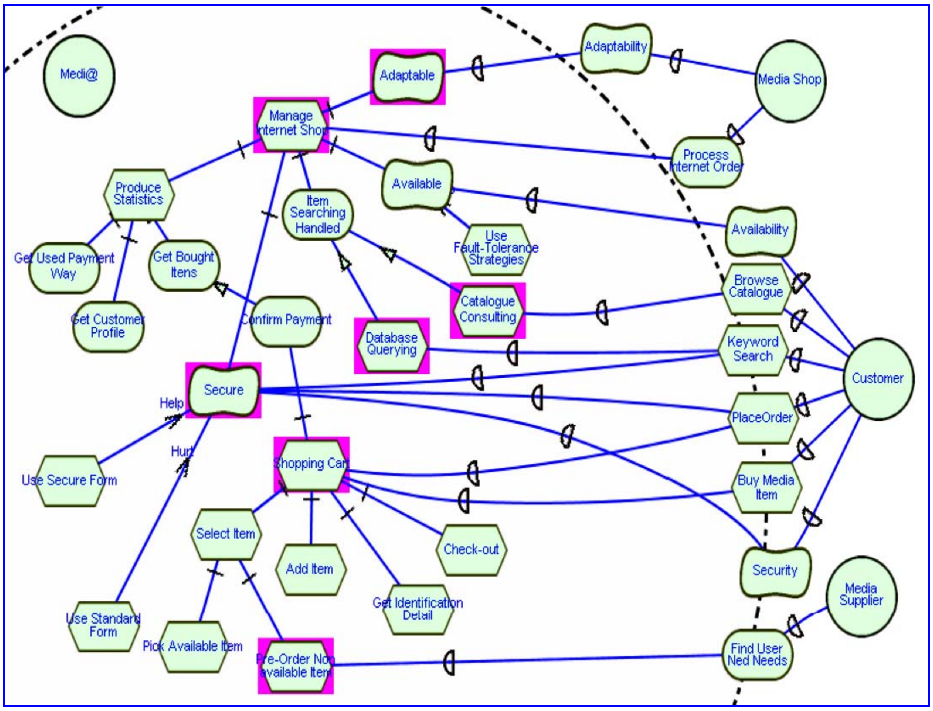


Fig. 5. An excerpt of the Medi@ Early and Late Requirements Model [10]

Acceptance test suites derivation consists of the following steps:

```
1: forall actor ∈ {stakeholder actors}do
2:    forall d ∈ {actor's dependencies towards the system}do
3:       analyze the corresponding system goal/task/softgoal (in the SR model of the system)
4:       for all lt ∈ {leaf task in the means-end / decomposition tree}
                       and sg ∈ {softgoal} do
5:          /*create a test suite for lt and sg */
6:          step1: identify operational or usage scenarios related to lt
7:          step2: identify fulfillment criteria (oracle) for each scenario
8:          step3: create one test suite with at least one test case for each scenario
9:       endfor
10:   endfor
11: endfor
```

While deriving test cases we may discover underspecified or potentially conflicting situations that need to refine the original requirements specification. That is, the test-first perspective brings as additional benefits the possibility of preventing defects and faults and of improving requirements specifications [3].

This emerged also when applying the above procedure to the early- and late-requirements models of *Media@* which is partially reported in Figure 5, as discussed in the following.

Along step #2 in the test suite derivation procedure, we first identify the *Medi@* system requirements that derive from domain stakeholders dependencies. The result is illustrated in Table 2, which lists: the domain stakeholders (the *Media Supplier*, the *Customer* and the *Media Shop*); their dependencies to the system-to-be, namely the *Medi@* system; and the associated requirements that may be expressed in terms of goals, tasks, softgoals.

A first observation is that we have two different dependencies from the same domain stakeholder (the *Customer*) that define the same requirement of *Medi@*, which is represented by the *Shopping* task. This observation raises the following questions:

– is there any real difference between the task dependency and the goal dependency, or shall we consider the *Place Order* task as the intended way by the *Customer* to pursue its *Buy Media item* goal?
– why does the *Place Order* task dependency induce the *Secure* quality while the *Buy Media item* goal does not?

These questions should be posed back to the requirements analyst that can refine the model. A possible refinement could be that of considering only one of the two dependencies, maintaining the link to the two requirements expressed by the *Medi@*´s *Shopping Cart* task and *Secure* softgoal.

Let's focus now on the *Medi@*'s *Shopping Cart* task and apply steps from #4 to #9 to derive a test suite for it. Table 3, illustrates examples of test suites for the leaf tasks *Pick available item* and *Pre-Order non available item*, *Add Item* and *Check Out*[3].

Acceptance tests assume that an URL for *Medi@* is available for internet access through a web browser, and it serves a (set) of *Media Shop*(s) which sells DVD, Book, Video concerning a variety of categories, such as Sport, Music of different

Table 2. System-to-be Requirements Derived from Domain Stakeholders Dependencies.

Domain Stakeholder	*Dependency*	*Medi@ requirements*		
		Root Goal	*Root Task*	*Softgoal*
Media Shop	[G] Process Internet Order	-	Manage Internet Shop	-
	[SG] Adaptability	-	-	Adaptable
MediaSupplier	[G] Find User New Needs		Pre-Order Non Available Item	
Customer	[SG] Availability			Available
	[G] Buy Media Item	-	Shopping Cart	-
	[T] Place Order	-	Shopping Cart	Secure
	[T] Keyword Search	-	Database Querying	Secure
	[T] Browse Catalogue	-	Catalogue Consulting	
	[SG] Security	-		Secure

[3] The complete application of the steps #4-#9 to the Shopping cart task will derive test suites also for the other leaf tasks, namely Check Out, Get Identification Detail. They are not shown here for space reasons.

Table 3. Examples of Test Suite derived by applying the Acceptance Test Suite Derivation Procedure

TS	Leaf task	TC	Scenario	Oracle
TS1	Pick available item	TC1.1	Given a list of 4 items each one identified by a unique name (or a short description) the user can point and click on the name of the third item with the mouse or the pen-stick.	An instance of selected-item with the ID of item #3 is created and ready to be added to the cart.
		TC1.2	Given a list of 4 items, two having similar or equal names, the user can point it with the mouse (or the pen-stick) to get a short description. A further click on it will define its selection.	An instance of selected-item with the right ID is created and ready to be added to the cart.
		TC1.3	The available item list is empty	The customer can switch to the Pre-Order non available item function or perform another query
		TC1.4	The session expires while the customer pick an item from the list	Resuming the session the customer is informed about the current selected items
TS2	Pre-order non avail-able item	TC2.1	The customer can select an item marked as not available	An instance of selected-item with the right ID is created and ready to be added to the cart in the pre-order set.
TS3	Add Item	TC3.1	The customer add to the cart one item from the list of selected items (both available or not)	The cart set is updated upon the inclusion of the added item
		TC3.2	The customer add to the cart all the selected items (both available or not)	The cart set is updated upon the addition of the selected items
TS4	Check Out	TC4.1	The customer is shown the list of items put in the cart and confirm the order	The order of the selected items is ready to be completed with the customer payment info
		TC4.2	The session expires	The order info are saved and ready to be resubmitted to the customer for confirmation

genres. These *Media Shops* rest on (a) *Media Supplier(s)* to have the ordered items available to be sent to the customers. *Customers* access to the *Medi@* system with a laptop equipped with a mouse or with a PDA equipped with a pen-stick.

Focusing on TS2 we may notice that the Pre-order non available item leaf task results from the analysis of the *Medi@´*s *Shopping Cart* root task as well as from the dependency from the *Media Supplier* domain stakeholder to achieve the *Find User New Need* goal. The requirements model seems to assume that the Medi@ system is able to provide to the customers a list of items which fit their current needs, and are marked as available or not. This requires that the *Media Suppliers*, the *Media Shop* is working with, allow the system to access their product databases, which should be dynamically updated with respect to the (non)/availability of their products.

This requirement should be made explicit, for instance in the analysis of the *Item Searching Handled* goal, or in the associated *Database Querying* and *Catalogue Consulting* tasks. Here the database or the catalogue the two tasks refer to, may correspond to a distributed database (or catalogue) that can be built dynamically by accessing to the catalogues of the media suppliers that work for the *Media Shop*.

5 Dealing with Complexity Using a Combined Goal and Scenario Approach

The combined use of goals and scenarios has been explored within requirements engineering, primarily for eliciting, validating and documenting software requirements. Van Lamsweerde and Willement studied the use of scenarios for requirements elicitation and explored the process of inferring formal specifications of goals and requirements from scenario descriptions in [21].

In the CREWS project, Rolland et al. have proposed the coupling of goals and scenarios in requirements engineering with CREWS-L'Ecritoire [31]. In CREWS-L'Ecritoire, scenarios are used as a means to elicit requirements/goals of the system-to-be. Both goals and scenarios are represented as structured text. The coupling of goal and scenario could be considered as a "tight" coupling, as goals and scenarios are structured into <Goal, Scenario> pairs, which are called "requirement chunks". Their work focuses mainly on the elicitation of functional requirements/goals.

The Software Architecture Analysis Method (SAAM) [17] is a scenario-based method for evaluating architectures. It provides a means to characterize how well a particular architectural design responds to the demands placed on it by a particular set of scenarios. Based on the notion of context-based evaluation of quality attributes, scenarios are used as a descriptive means of specifying and evaluating quality attributes. SAAM scenarios are use-oriented scenarios, which are designed specifically to evaluate certain quality attributes of architecture. The evaluations are done using simulations or tests on a finished design.

This section first introduces the goal and scenario model integrated design process based on *Tropos* and UCM [8]. Then the running example is used to illustrate how to deal with design decision making problems with the Tropos concepts. Part of this work is based on [22] and the URN [17] notation, new development of this work is that we aim at using the joint goal and scenario analysis to cope with the complexity in the organizational environment – mutual social dependencies, conflicting intentions and interests, hard to express operational scenarios.

5.1 Coping with Complexity Using TROPOS

Based on the Tropos concepts, we now explore how to capture the organizational and environmental complexity with strategic dependency network, and how agents can respond to the complexity according to their own needs and capabilities based on strategic rationale analysis.

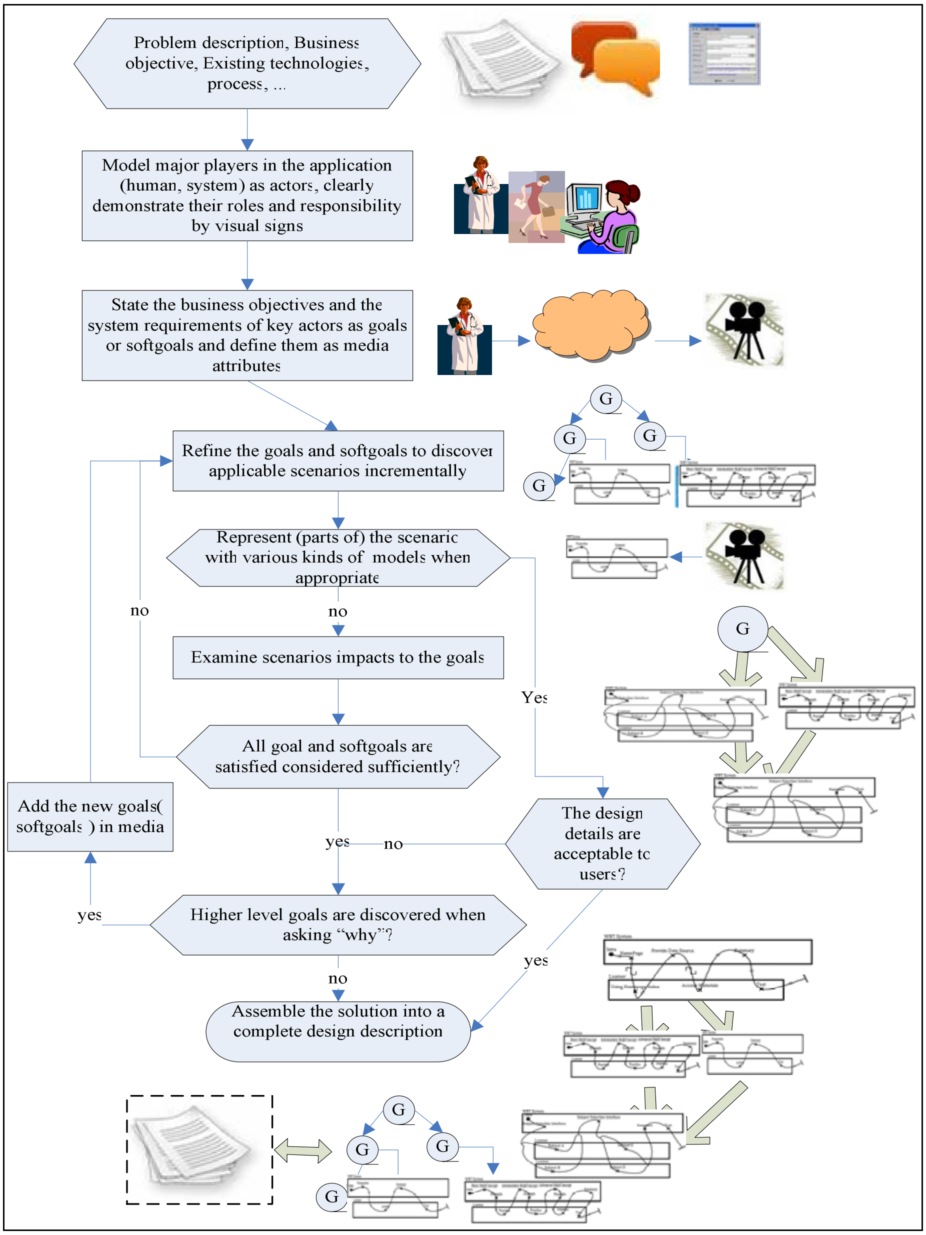

Problem description, Business objective, Existing technologies, process, ...
Model major players in the application (human, system) as actors, clearly demonstrate their roles and responsibility by visual signs
State the business objectives and the system requirements of key actors as goals or softgoals and define them as media attributes
Refine the goals and softgoals to discover applicable scenarios incrementally
Represent (parts of) the scenario with various kinds of models when appropriate
no
no
Examine scenarios impacts to the goals
Yes
All goal and softgoals are satisfied considered sufficiently?
Add the new goals(softgoals) in media
yes
no
The design details are acceptable to users?
yes
Higher level goals are discovered when asking "why"?
yes
no
Assemble the solution into a complete design description
G

Fig. 6. Goal and Scenario Model Integrated Design Process

We use the running example to illustrate the complementary application of Tropos and UCM. The approach is applicable to information systems in general, where there are conflicting goals and tradeoffs during design. Starting from the identification of the major stakeholders of the domain, we explain in sequence how to capture the original business objectives of the stakeholders, refine and operationalize these objectives into applicable design alternatives with *Tropos* and how to visualize and concretize some solutions with UCM.

Step 1: Placing system design within its broader social context, the proposed modeling approach can help to address the following questions systematically: Who are the major players in the business domain? What kinds of relationships exist among them? What are the business objectives and criteria of success for these players? The various dependency links in the model depict that in the Media example, the *Medi@* is a key player, who provides media products and services to *Customers* through the Internet. At the same time, it depends on the support of *Telecom Company, Media Supplier* and *Bank.*

Step 2: After the main players are identified, we ask them what their business objectives are, i.e., what they hope to accomplish for their organization, their sponsors, or their financial backers. Assume that, in our specific e-commerce system, the *Medi@* is playing the role of "Media Service Provider", who should then have two things in mind:

Attract new customer by selling media products online
Improve availability, adaptability and security of the service
They are represented as softgoals in the *Tropos* model in Figure 7.

Step 3: Explore the alternative business processes, methods or technologies used in this industry or business. Evaluate how these alternatives are serving the specific business objectives and the quality expectations of stakeholders.

In Figure 7, we see how the two solutions *Medi@* and Conventional *Media Shop* contribute differently to the goals. By using contribution links labeled with numbers or different symbolic types, the model portrayed that *Medi@* ***makes*** the goal of *Availability* satisficable, while *media shop* method ***hurts*** the fulfillment of this goal. Furthermore, the fulfilling of this goal ***helps*** the achievement of *Attract New Customer* goal. The result of this analysis suggests that *Medi@* may be a better option for current stakeholder. Part of this model (the two softgoals and the help relationship between them) is only applicable to current system, while other part (the structure showing the different resource consumption of the two solutions) depicts generic domain knowledge reusable to all service providers of *Media Products.*

Step 4: The advantages and disadvantages of the candidate solution are further investigated by evaluating its contributions to other concerned softgoals. For each disadvantage, mitigation plans are considered to complement the current solution.

The corresponding goal model shows that the advantages of *Media@* include *Availability*, *Increase Market Share* and *Adaptablity* is satisfied. Consequently, the overall quality of service improved. It also contributions positively to *Globalization*, *Flexibility*, both of which contribute positively (helps) to the customer's satisfactory.

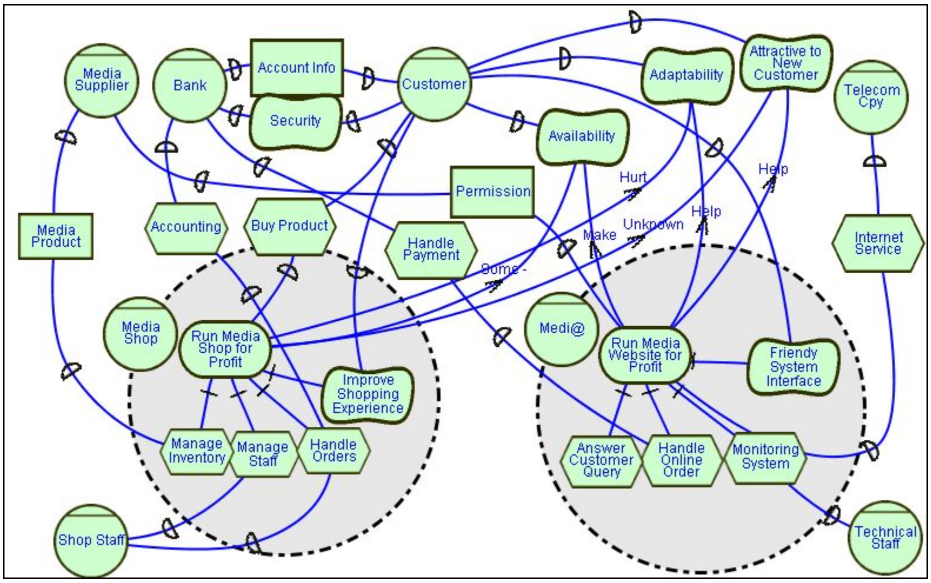


Fig. 7. Two Alternative Solutions in the Medi@ Example

However, there are also disadvantages e.g., the inherent *Security* and *More Efforts* on *Electronic Delivery* of *Media@* hurts the high level goals of the stakeholder. These disadvantages can be mitigated by countermeasures such as "DRM", which is represented as tasks connected with a negative correlation links (the dotted lines with arrows) to the unfavorable contributions links in the graph.

To identify the best design solutions, goal-reasoning techniques such as qualitative goal labeling algorithms are used. Quantitative techniques, such as probability or other quantitative measures, are used. With the help of *i** model, we are able to explore a space of design alternatives of considerable size. If there are *m* decision points (goals/softgoals with black rectangle shadow) and average *n* options at each point, there will be about n^m alternatives to be chosen from. Considering the presence of some external domain constraints, not all of alternatives are workable. When there is a large space of alternatives to choose from, system designers will greatly appreciate automated support such as an approximate ranking according to some criteria. The ranking of design alternatives is determined by the contributions to the softgoals of concern. In order to rank design alternatives, various criteria can be adopted. We can then either rank alternatives according to their overall contributions to all softgoals, or rank according to user's specific preferences.

Step 5: Identify the alternative essential sub-processes/components to implement the candidate solution. Next, we build model to elaborate the generic knowledge about *Media Shop Managed*. First of all, a media shop manager needs to *Choose a Business Front-end*, decide whether to use *bricks* or *clicks*, and *order handling Process* for the business. As all of these sub-processes are necessary steps for the finishing of the root task, they are represented as subgoals connected to the root task with decomposition links.

Step 6: As the goal-oriented design proceeds, finer-grained analysis needs to be conducted; hence the scenario-based notation comes into use. To elaborate the goal *Pick Ordering Process*, alternative processes are denoted in the *i** model as task nodes having different usage. For instance, *Media Shop* provides physical shopping experience, as they provide a *Safe* and *convenient* solution.

Each of the alternative processes can be described as a UCM scenario. *Medi@* system and *Customer* are represented as agent components (rectangles), holders of responsibilities (small crosses along on the wiggle lines). In the scenario, the use case path shows that different *Customers* can have different routines if they choose different subjects in the Web Interface. The *Customer* and the *Medi@* system collaborates on searching on the web for materials of interest, so they are sharing responsibilities (denoted by adding a square *S* between the shared responsibilities).

Having analyzed the benefits and tradeoffs of these structures, we can see that UCM is a useful counterpart to Tropos in the process from requirements to high-level design, because it provides a concrete model of each design alternative. Based on the features in such a model, new non-functional requirements may be detected and added to the Tropos model. At the same time, in the Tropos model, new means to achieve the functional requirements can always be explored and concretized in a UCM model. Thus the above design process may iterate several rounds until an acceptable design is made.

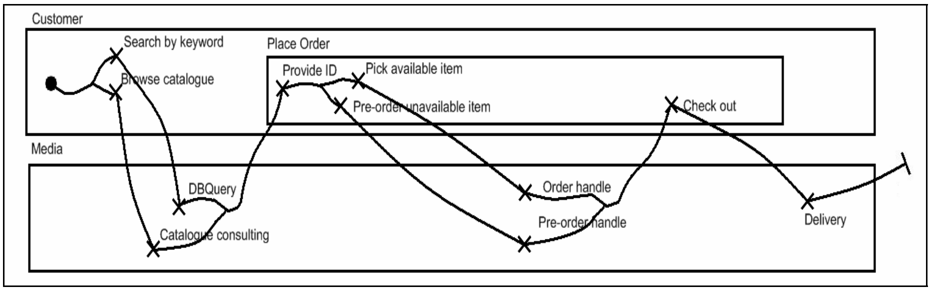


Fig. 8. Medi@ Scenario Model in UCM

6 Software Project Management Process

Due to benefits and perspectives such as efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, iterative development is more and more used by software engineering professionals especially through methodologies such as the *Unified Process* [34].

Most agent-oriented software development and requirements-driven methodologies only use a waterfall system development life cycle (SDLC) or advice their users to proceed iteratively without offering a strong project management framework to support that way of proceeding. Consequently they are not suited for the development of huge and complex user-intensive applications. The aim of this section is to present a research dedicated to extend *Tropos* with an iterative life cycle called *I-Tropos*[4]. This

[4] *I-Tropos* stands for *Iterative Tropos*.

methodology fills the project and product life cycle gaps of Tropos and offers a goal-oriented project management perspective to support project stakeholders for applying the methodology.

The *I-Tropos* project management framework covers several dimensions including risk, quality, time and process management. Contributions include, among others, taking threats and quality factors' evaluation directly in account for planning the goals realization over multiple iterations. The process is exposed in this section and illustrated on the *Medi@* case study using *DesCARTES*, a CASE-tool designed to support I-Tropos.

6.1 Process Engineering Concepts

An *I-Tropos development* is made of disciplines[5] iteratively repeated while the relative effort spend on each one is variable from one iteration to the other. The Organizational Modeling and Requirements Engineering disciplines are respectively largely inspired by Tropos' Early and Late Requirements disciplines. The Architectural and Detailed Design disciplines correspond to the same stages of traditional Tropos. *I-Tropos* includes core activities i.e. *Organizational Modeling*, *Requirements Engineering*, *Architectural Design*, *Detailed Design*, *Implementation*, *Test* and *Deployment* but also support disciplines to handle the project development called *Risk Management*, *Time Management*, *Quality Management* and *Software Process Management*. There is little need for support activities in a process using a waterfall SDLC since the core disciplines are sequentially achieved one for all. When dealing with a process using an iterative SDLC, the need for support disciplines for managing the whole software project is from primary importance. I-Tropos process' disciplines are described in detail in [35].

Using an iterative SDLC implies repeating process' disciplines many times during the software project. Each iteration belongs to one of the four phases inspired by the Unified Process (UP); a complementary documentation can be found in [20] while a summary of each phase objective is depicted into the next section. These phases are achieved sequentially and have different goals evaluated at milestones through knowledge and achievement oriented metrics, those are informally described into the next section. Figure 9 offers a two dimensional view of the I-Tropos process: it shows the disciplines and the four different phases they belong to.

[5] The phase and discipline notions are often presented as synonyms in software engineering literature. In [24], Tropos is described as composed of five phases (Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation). However [29] defines disciplines as "*a particular specialization of Package that partitions the Activities within a process according to a common "theme".*", while the phase is defined as "*a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal (often called a "milestone") defines the phase exit criteria*". In order to be compliant with the most generic terminology, traditional Tropos phases will be called disciplines in our software process description since they partition Activities under a common theme. In the same way, phases will be considered as groups of iterations which are workflows with a minor milestone.

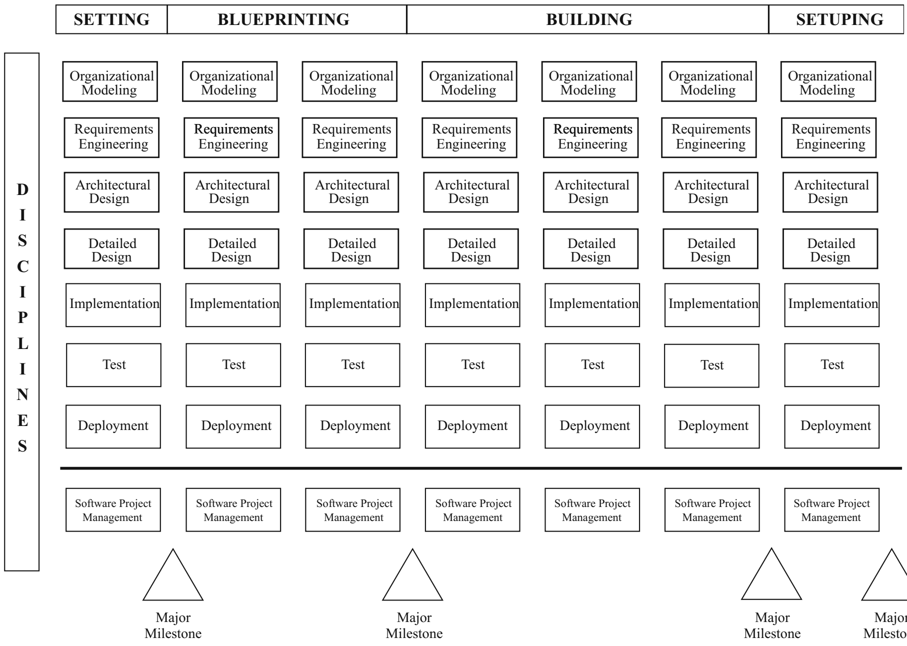


Fig. 9. *I-Tropos*: Iterative Perspective

6.2 Process Phases

I-Tropos phases are inspired by the UP phases[6]; each one is made of one or more iterations. Disciplines are gone through sequentially; as stressed before the phases are separated by major milestones. Each of them has its own goal:

The *setting* phase is designed to identify and specify most stakeholders requirements, have a first approach of the environment scope, identify and evaluate project's threats and identify and evaluate quality factors.

The *blueprinting* phase is designed to produce a consistent architecture for the system on the basis of the identified requirements, eliminate most risky features in priority and evaluate blueprints/prototypes to stakeholders; feedback will feed next iterations.

The *building* phase is designed to build a working application and validate developments.

The *setuping* phase is designed to finalize production, train users and document the system.

6.3 Process Core Disciplines

The *I-Tropos* process has been fully described using the Software Engineering Process Metamodel in [35]. That technical report describes each process' *Discipline*, *Activity*,

[6] The phases milestones expressed hereafter are based on the metrics expressed in the Unified Process (see [20]).

Role, *WorkDefinition* and *WorkProduct*, so that it can be used as reference or guide to the methodology. A lightened overview of the process is given in this section.

The Organizational Modeling discipline, strongly inspired from the Tropos *Early Requirements* stage, aims to understand the problem by studying the existing organizational setting.

The Requirements Engineering discipline, inspired from the Tropos *Late Requirements* stage, extends models created previously by including the system to-be, modeled as one or more actors.

The Architectural Design discipline, inspired by the Tropos *Architectural Design* stage, aims to build the system's architecture specification, by organizing the dependencies between the various sub-actors identified so far, in order to meet functional and non-functional requirements of the system.

The Detailed Design discipline, inspired by Tropos *Detailed Design*, aims at defining the behavior of each architectural component in further detail.

The Implementation discipline aims to produce an executable release of the application on the basis of the detailed design specification.

The Test discipline aims on evaluating the quality of the executable release.

The Deployment discipline aims to test the software in its final operational environment.

6.4 Process Support Disciplines

These disciplines provide features to support software development i.e. tools to manage risks, quality levels, time, resources allocation but also the software process itself. All those features can be regrouped onto the term *software project management*.

Risk Management is the process of identifying, analyzing, assessing risk as well as developing strategies to manage it. Strategies include transferring risk to another party, avoiding risk, reducing its negative effects or accepting some or all of the consequences of a particular one. Technical answers are available to manage risky issues. Choosing the right mean to deal with particular risk is a matter of compromise between level of security and cost. This compromise requires an accurate identification of the threats as well as their adequate evaluation.

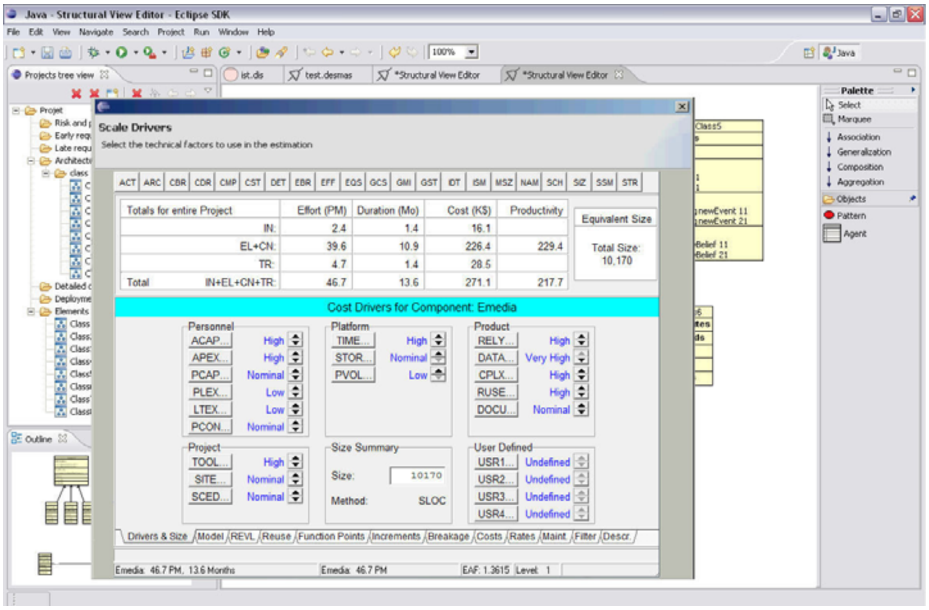
Quality Management is the process of ensuring that quality expected and contracted with clients is achieved throughout the project. Strategies include defining quality issues and the minimum quality level for those issues. Technical answers are available to reach quality benchmarks. Choosing the right mean to deal with quality issues is a matter of compromise between level of quality and cost. This compromise requires an accurate identification of the quality benchmarks as well as their adequate evaluation.

Time Management is the process of monitoring and controlling the resources (time, human and material) spent on the activities and tasks of a project. This discipline is of primary importance since, on the basis of the risk and quality analyses, the global iterations time and human resources allocation are computed; they are revised during each iteration.

Software Process Management is the use of process engineering concepts, techniques, and practices to explicitly monitor, control, and improve the systems engineering process. The objective of systems engineering process management is to enable an organization to produce system/segment products according to plan while simultaneously improving its ability to produce better products [9]. In this context, *Software Process Management* regroups the activities aimed to tailor the generic process onto a specific project as well as improving the software process.

6.5 Applying I-Tropos on Medi@

Figure 10 depicts *DesCARTES* [11], more specifically the cost and effort estimation interface provided by the module supporting the Software Project/Time Management Disciplines from *I-Tropos*. Project Management Features such as scale drivers, cost factors, increment settings, labor rates, breakage, etc., can directly be tuned through this kind of interfaces.


Java - Structural View Editor - Eclipse SDK
File Edit View Navigate Search Project Run Window Help
Projects tree view
ist.dis
test.desmas
*Structural View Editor
*Structural View Editor
Palette
Select
Marquee
Association
Generalization
Composition
Aggregation
Objects
Pattern
Agent
Outline
ACT ARC CBR CDR CMP CST DET EBR EFF EQS GCS GMI GST IDT ISM MSZ NAM SCH SIZ SSM STR
Drivers & Size Model REVL Reuse Function Points Increments Breakage Costs Rates Maint Filter Descr.

Scale Drivers

Select the technical factors to use in the estimation

Totals for entire Project		Effort (PM)	Duration (Mo)	Cost (K$)	Productivity
	IN:	2.4	1.4	16.1	
	EL+CN:	39.6	10.9	226.4	229.4
	TR:	4.7	1.4	28.5	
Total	IN+EL+CN+TR:	46.7	13.6	271.1	217.7

Equivalent Size

Total Size: 10,170

Cost Drivers for Component: Emedia

Personnel		Platform		Product	
ACAP...	High	TIME...	High	RELY...	High
APEX...	High	STOR...	Nominal	DATA...	Very High
PCAP...	Nominal	PVOL...	Low	CPLX...	High
PLEX...	Low			RUSE...	High
LTEX...	Low			DOCU...	Nominal
PCON...	Nominal				

Project		Size Summary		User Defined	
TOOL...	High	Size:	10170	USR1...	Undefined
SITE...	Nominal	Method:	SLOC	USR2...	Undefined
SCED...	Nominal			USR3...	Undefined
				USR4...	Undefined

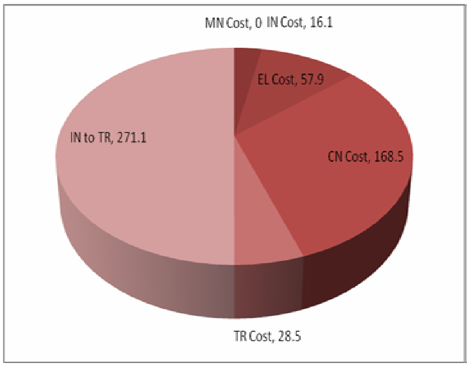
Emedia: 46.7 PM, 13.6 Months   Emedia: 46.7 PM   EAF: 1.3615 Level: 1

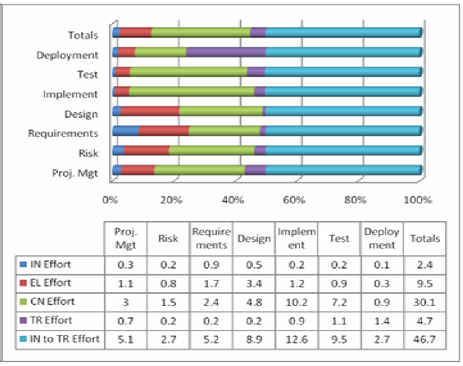
Fig. 10. *DesCARTES*: Estimating the Medi@ Application with the I-Tropos Software Project/Time Management Disciplines Module

DesCARTES (Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems) Architect is a Computer-Aided Software Engineering Tool developed as a plug-in for the Eclipse Platform by the Information Systems Unit at the University of Louvain. It is designed to support various models edition: i* models (Strategic Dependency and Strategic Rationale models), NFR models, UML models, AUML models in the context of *I-Tropos* or *Unified Process*-like developments. The originality of the tool is that it allows the development of the methodology models throughout iterative software life cycle processes as well as forward engineering capabilities and integrated software project management, time and risk/quality management modules.

Figure 11 provides graphical reporting outputs directly produced by *DesCARTES* related to the cost, effort, activities and schedule estimation for Medi@.

Instantiated to Medi@, these outputs applying regression models based on CO-COMO or SLIM [4] and factor scales supported by maturity models such as CMM-I give the following estimation figures. Total size is estimated to 101700 Java SLOC while the total cost will be 271 400 $. The duration of the project will be 13.6 months with 46.7 actual person-months (PM) and a nominal PM at 29.1. Productivity is estimated to 256.9 SLOC per PM at the unit cost at 22.26$ per line. Average staffing during the project is 3.43 persons with a high at 4.87 during *Building* a low at 1.71 during *Setting*.





	Proj. Mgt	Risk	Require ments	Design	Implem ent	Test	Deploy ment	Totals
🟦 IN Effort	0.3	0.2	0.9	0.5	0.2	0.2	0.1	2.4
🟥 EL Effort	1.1	0.8	1.7	3.4	1.2	0.9	0.3	9.5
🟩 CN Effort	3	1.5	2.4	4.8	10.2	7.2	0.9	30.1
🟪 TR Effort	0.7	0.2	0.2	0.2	0.9	1.1	1.4	4.7
🟦 IN to TR Effort	5.1	2.7	5.2	8.9	12.6	9.5	2.7	46.7

Fig. 11. Graphical Outputs from *DesCARTES*: Cost, Effort, Activities and Schedule Estimation for Medi@

7 Conclusion

This chapter presents a set of approaches to deal with complexity, which address various activities in software development, namely requirements modeling, testing and project management.

More specifically, we outlined an approach to improve modularization of requirements models described in i*, by identifying, separating and composing crosscutting concerns. A specific notation has been created to represent aspectual *i** models. This leads to the addition of two new concepts in the *i*/Tropos* modeling language, namely aspectual element and crosscut relationship. Aspectual elements modularize crosscutting concerns while the crosscut relationship captures the information of source and target model elements, as well as, when and how an aspectual element crosscuts other model elements. The approach introduces modularity (it creates units that are strongly cohesive and loosely coupled), reduces the scalability (removing the redundant elements and links) and improves the reusability. Work in under way to evaluate the resulting models by means of metrics to assess well-known attributes in software engineering, such as separation of concerns, size, cohesion and coupling. In the near future we plan to define a trade-off analysis method to complement the proposed process as well as to provide tool support for the approach.

The application of a V-model software development process in Tropos, namely the GOST methodology [26], has been introduced, as an approach to enable incremental validation and testing of artifacts while building complex software system. Further benefits of using this test-first perspective for clarifying ambiguities in requirements models has also been illustrated by applying GOST to a fragment of the early- and late-requirements models of the *Media@* system. The systematic application of the GOST approach can be supported by the eCAT tool, which automatically generates test suite skeletons from goal models [25]. Extensions of GOST with automated test case generation techniques are described in [27] and [28]. This is a necessary step towards supporting a continuous validation and testing approach for the development of complex software systems.

The combined use of Tropos and UCM enables the description of functional and non-functional requirements, abstract requirements and concrete system models, intentional strategic design rationales and non-intentional details of concurrent, temporal aspects of the future system. It is natural to adopt Tropos as a basic requirements knowledge representation language, and try to find how other existing requirements modeling languages relate and complement to it. So following the attempt in integrating i*(GRL) with UCM, we move to integrate *i** with the Problem Frames. The ultimate objective is to build a requirement ontology that incorporates as many perspectives as possible.

In terms of software process management, *I-Tropos* represents an evolution of the Tropos process. It constitutes an operationalization of the Tropos methodology in order to be used in large software developments. *I-Tropos* mainly fills up the gap of project management which is, for now, seldom approached in MAS literature. The main contributions include meta-level process documentation, a full software and product life cycle coverage, a project management framework for the inclusion of Tropos developments into an iterative and incremental process template and the support of a specific CASE tool, *DesCARTES*. *I-Tropos* is an adequate project management for building large-scale enterprise systems from scratch. However, many firms have turned to the reuse of existing software or using commercial off-the-shelf (COTS) software as an option due to lower cost and time of development. Work for enlarging its scope including COTS software customization onto specific case and adequate project management is in progress. The basic adaptation of *I-Tropos* to support this paradigm of software development is described in [36].

Acknowledgements

This work was supported by several research grants: National Natural Science Foundation of China (Grant No.60873064), National Basic Research and Development 973 Program of China (Grant No.2009CB320706), CNPq Grant 308587/2007-3, BIT Initiative, FUR-PAT Trento (STAMPS project). We thank our colleagues at Universidade Federal de Pernambuco and Universidade Nova de Lisboa, in particular Fernanda Alencar, João Araújo and Ana moreira for their contributions to the aspectualization of i* models.

References

1. Amyot, D.: Introduction to the User Requriements Notation: Learning by Example. Computer Networks~42(3), 285--301 (2003)
2. Alencar, F., Castro, J., Moreira, A., Araújo, J., Silva, C., Ramos, R., Mylopoulos, J.: Integration of aspects with i* models. In: Kolp, M., Henderson-Sellers, B., Mouratidis, H., Garcia, A., Ghose, A.K., Bresciani, P. (eds.) AOIS 2006. LNCS, vol. 4898, pp. 183–201. Springer, Heidelberg (2008)
3. Beck, K.: Test Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
4. Boehm, B., et al.: Software cost estimation with COCOMO II. Prentice-Hall, Englewood Cliffs (2000)
5. Bresciani, P., Perini, A., Giunchiglia, F., Giorgini, P., Mylopoulos, J.: A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In: Proc. of the 5th Int. Conference on Autonomous Agents (Agents 2001), Montreal, pp. 648–655 (2001)
6. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
7. Brito, I.S., Vieira, F., Moreira, A., Ribeiro, R.A.: Handling conflicts in aspectual requirements compositions. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 144–166. Springer, Heidelberg (2007)
8. Buhr, R.J.A.: Use Case Maps as Architectural Entities for Complex Systems. Transactions on Software Engineering 24(12), 1131–1155 (1998)
9. Capability Assessment Working Group on Systems Engineering, Systems Engineering Capability Assessment Model, INCOSE-TP-1996-002-01, Version 1.50a (June 1996)
10. Castro, J., Kolp, M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems Journal 27(6), 365–389 (2002)
11. DesCARTES Architect: Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems (2008), `http://www.isys.ucl.ac.be/descartes/`
12. Development Standards for IT Systems of the Federal Republic of Germany, The V-Model (2005), `http://www.v-modell-xt.de`
13. Estrada, H., Rebollar, A.M., Pastor, Ó., Mylopoulos, J.: An empirical evaluation of the i* framework in a model-based software generation environment. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 513–527. Springer, Heidelberg (2006)
14. Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longsta, T., Sullivan, K., Wallnau, K.: Ultra-large-scale systems: The software challenge of the future. Technical report, Software Engineering Institute (July 2006), `http://www.sei.cmu.edu/uls/`
15. France, F., Kim, D., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique. IEEE Transactions on Software Engineering 30(3), 193–206 (2004)
16. Graham, D.R.: Requirements and testing: Seven missing-link myths. IEEE Software 19(5), 15–17 (2002)
17. International Telecommunications Union (ITU-T) Recommendation Z.151: User Requirements Notation (URN) - Language Definition (2008)

18. Kazman, R., Bass, L., Abowd, G., Webb, M.: SAAM: A Method for Analyzing the Properties of Software Architectures. In: Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81–90 (1994)
19. Kim, D., France, R., Ghosh, S., Song, E.: Using Role-Based Modeling Language as Precise Characterizations of Model Families. In: 8th Intl. Conf. on Engineering of Complex Computer Systems, IEEE, USA (2002)
20. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley, Reading (2003)
21. van Lamsweerde, A., Willemet, L.: Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering, Special Issue on Scenario Management (December 1998)
22. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. Information Systems 29(2), 187–203 (2004)
23. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink, Liverpool, UK (2005)
24. Mylopoulos, J., Castro, J.: Tropos: A Framework for Requirements-Driven Software Development. In: Brinkkemper, S., Lindencrona, E., Sølvberg, A. (eds.) Information Systems Engineering: State of the Art and Research Themes, pp. 261–273. Springer, Heidelberg (2000)
25. Nguyen, C.D., Perini, A., Tonella, P.: eCAT: a Tool for Automating Test Cases Generation and Execution in Testing Multi-Agent Systems (Demo Paper). In: Proc. Of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Demo Proceedings, Estoril, Portugal, May 12-16, 2008, pp. 1669–1670 (2008)
26. Nguyen, C.D., Perini, A., Tonella, P.: Goal-Oriented Testing for MAS. Int. Journal of Agent-Oriented Software Engineering (submitted, 2008)
27. Nguyen, C.D., Perini, A., Tonella, P.: Automated Continuous Testing of Autonomous Distributed Systems. In: 1st International Workshop on Search-Based Software Testing, in conjunction with the IEEE International Conference on Software Testing, Verification and Validation, ICST 2008 (2008)
28. Nguyen, C.D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M.: Evolutionary Testing of Autonomous Software Agents. In: Decker, Sichman, Sierra, Castelfranchi (eds.) Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15 (2009)
29. OMG: The Software Process Engineering Metamodel Specification. Version 1.1 (January 2005)
30. Rashid, A., Moreira, A., Araujo, J.: Modularisation and Composition of Aspectual Requirements. In: Proc. of the 2nd Intl. Conf. on Aspect-Oriented Software Development, pp. 11–20. ACM Press, New York (2003)
31. Rolland, C., Grosz, G., Kla, R.: Experience With Goal-Scenario Couplingin Requirements Engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering 1998, Limerick, Ireland (1998)
32. Silva, C., Araújo, J., Moreira, A., Castro, J.: Designing Social Patterns using Advanced Separation of Concerns. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 309–323. Springer, Heidelberg (2007)

33. Susi, A., Perini, A., Giorgini, P., Mylopoulos, J.: The Tropos metamodel and its use. Informatica 29(4), 401–408 (2005)
34. Royce, W.: Software Project Management. A Unified Framework. Addison-Wesley, Reading (1998)
35. Wautelet, Y., Kolp, M., Achbany, Y.: S-Tropos: An Iterative SPEM-Centric Project Management Process. Working Paper IAG 06/01, Université catholique de Louvain (2006)
36. Wautelet, Y., Achbany, Y., Kiv, S., Kolp, M.: A Service-Oriented Framework for Component-Based Software Development: An i* Driven Approach. In: Proceedings of the 11th International Conference on Enterprise Information Systems, ICEIS 2009, Milan (2009)
37. Yu, E.: Modelling Strategic Relationships for Process Reengineering. Ph.D Thesis, Department of Computer Science, University of Toronto, Canada (1995)