Journal of Web Engineering, Vol. 9, No. 4 (2010) 302–326
© Rinton Press

AN I*-BASED APPROACH FOR MODELING AND TESTING WEB REQUIREMENTS

ESTEBAN ROBLES LUNA
LIFIA, UNLP, Argentina
erobles@lifia.info.unlp.edu.ar

IRENE GARRIGÓS, JOSE-NORBERTO MAZÓN, JUAN TRUJILLO
Lucentia Research group, University of Alicante, Spain
{*igarrigos, jnmazon, jtrujillo*}*@dlsi.ua.es*

GUSTAVO ROSSI
LIFIA, UNLP, Argentina
gustavo@lifia.info.unlp.edu.ar

Received June 1, 2010
Revised November 11, 2010

Web designers usually ignore how to model real user expectations and goals, mainly due to the large and heterogeneous audience of the Web. This fact leads to websites which are difficult to comprehend by visitors and complex to maintain by designers; these problems could be ameliorated if users are able to evaluate the application under development providing their feedback. To this aim, in this paper we present an approach for using the i* framework for modeling users' goals with mockups and WebSpec diagrams for detailing the specification of Web requirements, in such a way that the process of evaluating i* models for Web applications can be automated thus improving users' feedback during the development process. Also, as part of our development approach, we derive the domain and navigational models by defining a set of automatic transformations to a specific Web modeling method. Finally, we illustrate our approach with a case study to show its applicability and describe a prototype tool that supports the process.

Keywords: Requirement engineering, Web requirements, i*, goal evaluation
Communicated by: M. Gaedke, M. Grossniklaus, and O. Diaz

1 Introduction

In the last decade, the number and complexity of websites and the amount of information they offer is rapidly growing. In this context, introduction of Web design methods and methodologies [1, 2, 3, 4, 5] have provided mechanisms to develop complex Web applications in a systematic way. To better accommodate the individual user, personalization of websites has been also introduced and studied [6, 7, 8, 9].

However, traditionally, methodologies for Web engineering have not taken into serious consideration the requirement analysis phase. Actually, one of the main characteristics of Web applications is that they typically serve a large and heterogeneous audience in which i) everybody can access to the website and ii) each user has different needs, goals and preferences.

Importantly, this scenario hinders Web designers from considering users and current efforts for requirement analysis in Web engineering are rather focused on the system. Therefore, the

needs of the users are figured out by the designer and their user browsing experience may not be successful. Consequently, there may appear development and maintenance problems for designers, since costly, time-consuming and rather non-realistic mechanisms (e.g. surveys among visitors) should be developed to improve the already implemented website *a posteriori*, thus increasing the initial project budget.

To solve these drawbacks, in this paper, our aim is to model which are the expectations, intentions and goals of the users when they are browsing the site, and determining how they can affect the definition of a suitable Web design. Moreover, as Web applications have a strong emphasis in interaction, UI (User Interface) and navigation aspects, requirements related to these aspects should be also captured. The main benefit of our point of view is that the designer will be able to make decisions from the very beginning of the development phase. These decisions could affect the structure of the envisioned website in order to satisfy needs, goals, interests and preferences of each user or user type. Our work is inspired by agile software development [10] that states that the continuous evaluation of the application under development helps to gather feedback from users during development thus ameliorating some maintenance time-consuming tasks.

To this aim, we propose to use the i* modeling framework [11, 12] for modeling requirements from the expectations and goals that users have (Fig. 1). The i* framework is one of the most valuable approaches for analyzing stakeholders' goals and how the intended system would meet them. This framework is also very useful for reasoning about how stakeholders' goals contribute to the selection of different design alternatives. However, although i* provides mechanisms to model stakeholders and relationships between them, it should be adapted for Web engineering, since the Web domain has special requirements that are not taken into account in traditional requirement analysis approaches. These requirements are related to the three main features of Web applications [13]: navigational structure, user interface and personalization capability.

Furthermore, because of the agile nature of Web applications there is a strong link between Web requirements and testing [14]. Specifically, defining test cases is needed to avoid erroneous implementations and deploying efficient Web applications meeting time constraints. Therefore, i* is complemented in this paper with mockups[a] and WebSpec diagrams [15]. We have chosen these artifacts because they help to agree on UI aspects, allow the specification of interactive Web requirements and provide automatic derivation of interactive tests to asses that the requirements are correctly implemented.

Once the requirements are specified, the next step is to obtain the conceptual models of the Web application (Fig. 1). To this aim, in this paper we also propose a set of QVT (Query/View/Transformation) rules [16] within a model-driven approach. In this way, designers will not have to create these models from scratch but they have a first tentative model satisfying the requirements specification and then they only have to refine these models, saving time and development effort. Though we use the A-OOH (Adaptive Object Oriented Hypermedia) [8] to illustrate our approach, any other Web engineering methodology could be used by only changing the QVT transformation rules. In the cases that the conceptual models of the Web methodology considered are based on UML, then the effort in updating the QVT

[a] A mockup is a sketch of the "possible" application which generally represents UI elements and helps to agree on broad aspects of the Look and Feel of the application under development

transformations would be minimal, since A-OOH is UML-based. In the case of using another modeling language (different from UML), the modifications would be achievable, since all Web methodologies share similar basic concepts.

During the aforementioned process of model refinement (see Fig. 1), it is interesting to evaluate if the requirements and the goals specified in the i* models are being satisfied. We propose that this model refinement is performed iteratively so that developers can tackle one requirement at a time, thus simplifying the process and in such a way that users can perceive the project's progress. From a user perspective it helps to ensure that time constraints are met and allow him to check (right after a task has been completed) if his expectations are actually satisfied in the developed application. In this way, the user will give its feedback during development like in agile development styles. In our approach i* models are automatically evaluated by means of the interactive tests obtained from WebSpec diagrams. As a consequence, users could look at a tagged i* model which states which goals are being satisfied by the application under development while developers refine the models.

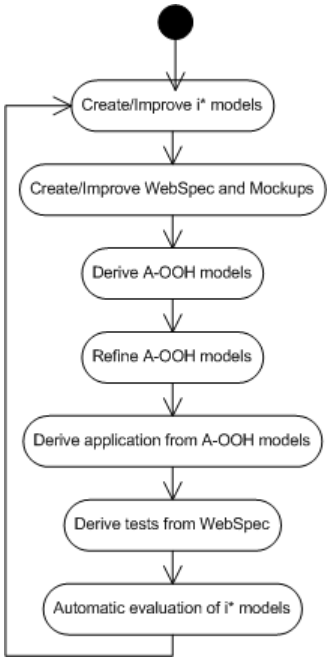


Fig. 1. Overview of our i*-based approach for Web application development

In [17] we have already presented our requirement analysis approach for using i* in Web engineering. In this paper we add a detailed analysis phase to specify interactive requirements in more depth. Also, from the elements used for requirement analysis, we add an automatic derivation of tests to evaluate the satisfiability of the requirements during the development process. Using our automatic evaluation method (Sect. 4) over the test results will serve to analyze if the application under development satisfies the goals described in the i* model.

The remainder of this paper is structured as follows: our approach for requirement analysis in Web engineering is presented in Sect. 2. In Sect. 3 we show how to trace these requirements to a Web design model. Sect. 4 shows how the stakeholder's goals are iteratively validated

during the development cycle thus improving stakeholders' feedback. Sect. 5 describes an example of applying our approach. Sect. 6 describes related work. Finally, in Sect. 7, we present our conclusions and sketch some future work.

2 Modeling Requirements in Web Engineering

In this section, we present an approach for modeling requirements in Web engineering. Our approach begins capturing the needs and goals of the stakeholders (Sect. 2.1) in i* models. Afterwards, a more detailed analysis is performed in order to capture interactive and UI aspects using a combination between mockups and WebSpec diagrams (Sect. 2.2). While we create the specifications in the diagrams, it is important to make explicit the relationships between the elements of the i* model and the WebSpec diagrams in order to automatically evaluate the satisfiability of the goals during development (this issue will be addressed in Sect. 4 after presenting how to derive Web models in Sect. 3).

2.1 Modeling stakeholders' needs and goals

The development of Web applications involves different kind of stakeholders with different needs and goals. Interestingly, these stakeholders depend on each other to achieve their goals, perform several tasks or obtain some resource, e.g. *the Web administrator relies on new clients for obtaining data in order to create new accounts*. In the requirements engineering community, goal-oriented techniques, such as the i* framework [11, 12], are useful for explicitly analyzing and modeling these relationships among multiple stakeholders (actors in the i* notation). The i* modeling framework provides mechanisms for representing (i) intentions of the stakeholders, i.e. their motivations and goals, (ii) dependencies between stakeholders to achieve their goals, and (iii) the (positive or negative) effects of these goals on each other in order to be able to select alternative designs for the system, thus maximizing goals fulfilment.

Next, we briefly describe an excerpt of the i* framework which is relevant for the present work. For a further explanation, we refer the reader to [11, 12]. The i* framework consists of two models: the strategic dependency (SD) model describes the dependency relationships (represented as) among various actors in an organizational context, and the strategic rationale (SR) model is used to describe actor's interests and concerns and how they might be addressed. The SR model (represented as) provides a detailed way of modeling internal intentional elements and relationships of each actor (). Intentional elements are goals (), tasks (), resources () and softgoals (). Intentional relationships are means-end links () representing alternative ways for fulfilling goals; task-decomposition links () representing the necessary elements for a task to be performed; or contribution links () in order to model how an intentional element contributes to the satisfaction or fulfillment of a softgoal.

Although i* provides good mechanisms to model actors and relationships between them, it needs to be adapted to the Web engineering domain to reflect special Web requirements that are not taken into account in traditional requirement analysis approaches, thus being able to assure the traceability to Web design. Web functional requirements are related to three main features of Web applications [13] (besides of the non-functional requirements): navigational structure, user interface and personalization capability. Furthermore, the required data structures of the website should be specified as well as the required (internal) functionality

provided by the system. Therefore, in this paper, we use the taxonomy of Web requirements presented in [13]:

Content Requirements With this type of requirements the content that the website presents to its users is defined. Some examples might be: "book information" or "product categories". Other kind of requirements may need to be related with one or more *content requirements*.

Service Requirements This type of requirement refers to the internal functionality the system should provide to its users. For instance: "register a new client", "add product", etc.

Navigational Requirements A Web system must also define the navigational paths available for the existing users. Some examples are: "consult products by category", "consult shopping cart", etc.

Layout Requirements Requirements can also define the visual interface for the users. For instance: "present a different style for teenagers", etc.

Personalization Requirements We also consider personalization requirements in this approach. The designer can specify the desired personalization actions to be performed in the final website (e.g. "show recommendations based on interest", "adapt font for visual impaired users", etc.)

Non-Functional Requirements In our approach the designer can also model non-functional requirements. These kind of requirements are related to quality criteria that the intended Web system should achieve and that can be affected by other requirements. Some examples can be "good user experience", "attract more users", "efficiency", etc.

Once this classification has been adopted, the i* framework needs to be adapted to the Web domain. As aforementioned, our proposal is presented in the context of A-OOH*(Adaptive Object Oriented Hypermedia method)* Web engineering method [8], an extension of the OO-H modeling method [2], which includes the definition of adaptation strategies.

As this approach (A-OOH) is UML-compliant, we have used the extension mechanisms of UML to (i) define a profile for using i* within UML; and (ii) extend this profile in order to adapt *i** to specific Web domain terminology. Therefore, new stereotypes have been added according to the different kind of Web requirements (see Fig. 2): *Navigational*, *Service*, *Personalization* and *Layout* stereotypes extend the *Task* stereotype and *Content* stereotype extends the *Resource* stereotype. It is worth noting that non-functional requirements can be modeled by directly using the softgoal stereotype.

Finally, several guidelines should be provided in order to support the designer in defining i* models for the Web domain.

1. Determine the kind of users for the intended Web and model them as actors. The website is also considered as an actor. Dependencies among these actors must be modeled in an SD model.

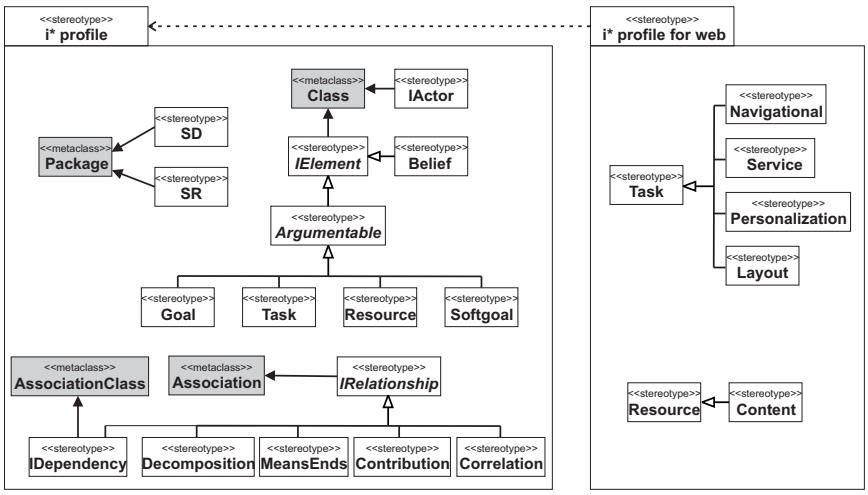


Fig. 2. Overview of the UML profiles for i^* modeling in the Web domain.

2. Define actors' intentions by using i* techniques in an SR model [34]: modeling goals, softgoals, tasks and resources, and the relationships between them.

3. Define the i* elements for the website actor and annotate tasks as navigational, service, personalization or layout requirements. Also, annotate resources as content requirements. It is worth noting that goals and softgoals should not be annotated.

To show the applicability of our approach, a case study is introduced. It is based on a company that sells books on-line. In this case study, a company would like to manage book sales via an online bookstore, thus attracting as many clients as possible.

A couple of actors are detected that depend on each other, namely "Client", and "Online Bookstore". A client depends on the online bookstore in order to "choose a book to buy". These dependencies are modeled by an SD model (see Fig. 3). Once the actors have been modeled in an SD model, their intentions are specified in SR models.

The SR model of the online bookstore is shown in Fig. 3. The main goal of this actor is to "manage book sales". To fulfill this goal the SR model specifies that a task should be performed: "books should be sold online". We can see in the SR model that the first of the tasks affects positively the softgoal "attract more users". Moreover, to complete this task three subtasks should be obtained: "provide book info" (which is a navigational requirement), "provide recommended books" (which is a personalization requirement), and "search engine for books". We can observe that some of these tasks affect positively or negatively to the non-functional requirement "easy to maintain": "Provide book information" is easy to maintain, unlike "provide recommended books" and "use a search engine for books". The navigational requirement "provide book information" can be decomposed into several navigational requirements according to the criteria used to sort the data. These data is specified by means of content requirements: "book", "author" and "category". The personalization requirement "provide recommended books" is related to the content requirement "book" because it needs the book information to be fulfilled. The task "search engine for books" is decomposed into

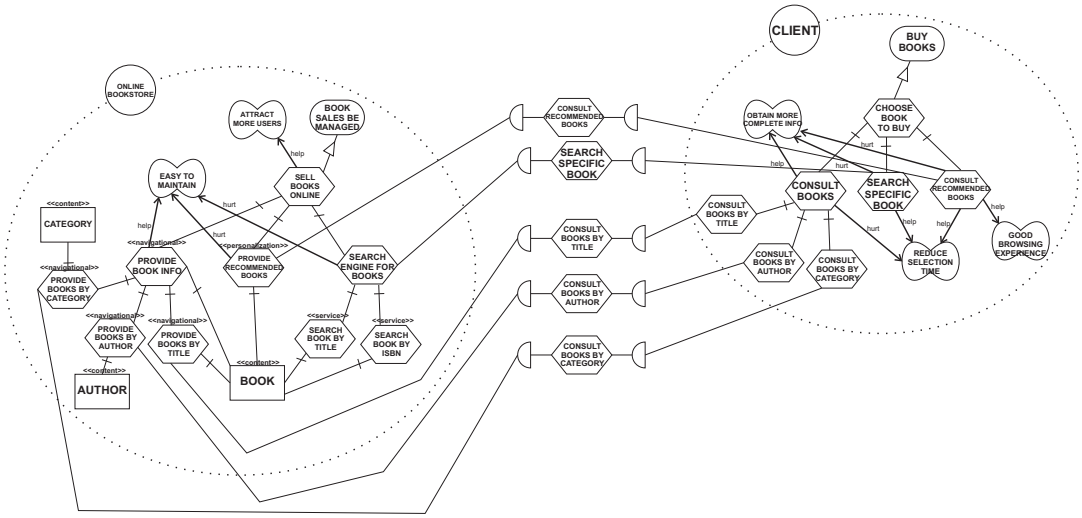


Fig. 3. Modeling the intentional elements with i*

a couple of service requirements: "search book by title" and "search book by ISBN", which are also related to the content requirement "book".

In the context of our case study the main goal of the client is to "buy books". In order to fulfill it, the client should be able to perform the "choose a book to buy" task. The task "choose a book to buy" should be decomposed in several subtasks: "consult books", "search specific book", "consult recommended books". These tasks can have positive or negative effects on some important softgoals. For example, "consult books" hurts the softgoal "reduce selection time".

Note that tasks of the online bookstore actor have been stereotyped according with our profile. In this figure we can see that tasks "provide books by title", "provide books by author", "provide recommended books" have been stereotyped with *Navigational* and that the "search books by ISBN" task has been stereotyped with *Service*.

2.2 Modeling detailed interactive requirements

Because of the idiosyncrasy of Web applications, there are certain parameters that need to be considered when they are developed (e.g. time constraints, fast evolution, etc). To efficiently meet these parameters, it is crucial that interactive requirements are specified in more detail and also validated in the early stages of the development process. In order to perform this validation, requirements need to be analyzed in deeper detail, including navigation, UI and interactive aspects which are of paramount importance in the context of Web applications. To this aim, we propose to use mockups and WebSpec diagrams because mockups are widely used to agree on UI aspects and WebSpec provides automatic validation of requirements independently of the development approach used [15].

A WebSpec diagram specifies a set of scenarios that must be satisfied by the application. In order to specify scenarios, a WebSpec diagram is composed of elements of two different types that capture the concepts involved in interactive Web applications: interactions and navigations (Fig. 4).

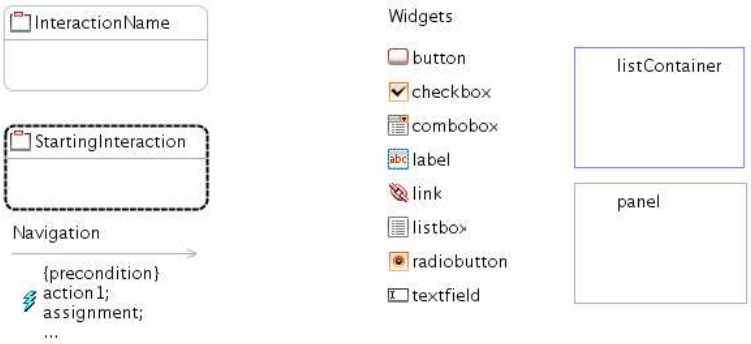


Fig. 4. WebSpec's basic concepts

An interaction (the counterpart of a Web page in the requirements stage) represents a point where the user can interact with the application by using its interface objects (widgets). Interactions have a name and may have widgets of different types such as: textfields, buttons, radiobuttons, panels, lists, etc. They are graphically represented with rounded rectangles containing the interaction's name and its widgets. The set of scenarios that the diagram specifies is obtained by following the different paths from a special interaction called "starting" denoted with dashed lines (Fig. 4).

To improve the communication between the different stakeholders, we can associate interactions with mockups and WebSpec widgets with their concrete UI elements to simulate the application [15]. There are several tools that could be used to create mockups, such as Balsamiq[b] or plain HTML. WebSpec allows using any of them as long as they provide a unique way to locate the interface elements. As an example, Fig. 5 presents two mockups created with Balsamiq that show how the user will search books by title and see the results of that search. Notice that a Mockup has several labels with constant values which provide an example of the application the we are trying to develop.

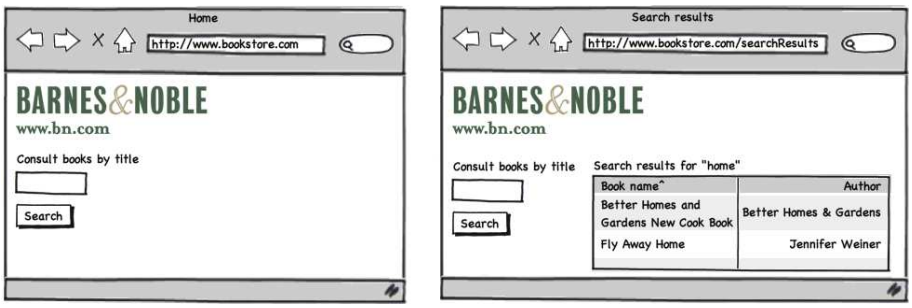


Fig. 5. "Consult books by title" mockup created with Balsamiq

To actually specify which properties must hold, we use invariants on each WebSpec interaction and in case that we do not define one, it is assumed that the invariant is true (it always hold independently of the interaction's state). Fig. 6 shows a simplified diagram of the "Consult books by title" of the Book store application. The diagram has 2 interactions

[b] http://www.balsamiq.com/products/mockups

named: Home and SearchResults. The Home interaction represents the starting point of the scenario and, for the interest of this requirement, we assume that it must have 2 widgets: a search field and a search button. The SearchResults defines an invariant (marked with the I icon near the interaction's name): that states: SearchResults.title = "Search results for:" + ${productName}. It means that the contents of the title label must be equal to the concatenation between "Search results for:" and the value of the productName variable (denoted as ${variableName}).

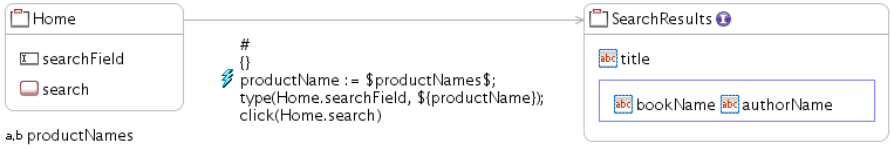


Fig. 6. "Consult books by title" Webspec diagram

A navigation from one interaction to another can be activated if its precondition holds by executing a sequence of actions such as: clicking a button, adding some text in a text field, etc. As well as invariants, preconditions can reference variables previously declared in the diagram. Navigations are graphically represented in the WebSpec diagrams with gray arrows while its name, precondition and actions are displayed as labels over them. Actions are written in an intuitive DSL conforming to the syntax: var := expr | actionName(arg1,... argn). For example the navigation from the Home interaction to the SearchResults performs three actions: first, it "generates" a productName (see [15] for further details of the use of generators), then this text is typed in the search field and finally the search button is clicked. We encourage the reader to look at [15] for further details about WebSpec.

After a diagram is created, a set of interaction tests can be derived from them [15]. These tests execute the actions and assert properties that are obtained from the navigations and invariants of the diagram which finally must be satisfied by the application. As the actions are performed directly over a Web browser, they are independent of the development approach used making the approach more appealing to be used within any Web engineering approach.

In order to evaluate the satisfiability of the i* model according to the application under development, each of the tasks in the i* model is specified in WebSpec diagrams that will specify the expected behavior of the application for fulfilling that task. It is worth noting that several WebSpec diagrams can be specified for each task as they represent different scenarios that must be satisfied. Consequently, tests derived from the diagrams are related with i* tasks thus helping to analyze which tasks and goals are satisfied automatically.

3 Deriving Web Models

Once the requirements have been defined they can be used to derive the conceptual models for the website. Typically, Web design methods comprise three main models to define a Web application: a *Domain model*, in which the structure of the domain data is defined, a *Navigation model*, in which the structure and behavior of the navigation view over the domain data is defined, and finally a *Presentation model*, in which the layout of the generated hypermedia presentation is defined. In this work, for the sake of a better understanding, the focus is on the Domain and Navigation models.

As aforementioned, in this paper, we consider the conceptual models of the A-OOH methodology. Once these models are derived from the specified requirements the designer has only to refine them, avoiding the task of having to create them from scratch.

Since the i* framework does not support generation to other design artifacts by its own, domain-oriented mechanisms should be considered to perform this task [18]. In our approach, the new stereotypes presented in the previous subsection allow us to prepare models for this generation phase. We have detected several i* patterns [19] in order to define a set of QVT transformation rules to map elements from the SR metamodel to their counterparts in the A-OOH metamodel. They are applied with a certain order as shown in Fig. 7, where the transformation workflow is summarized.

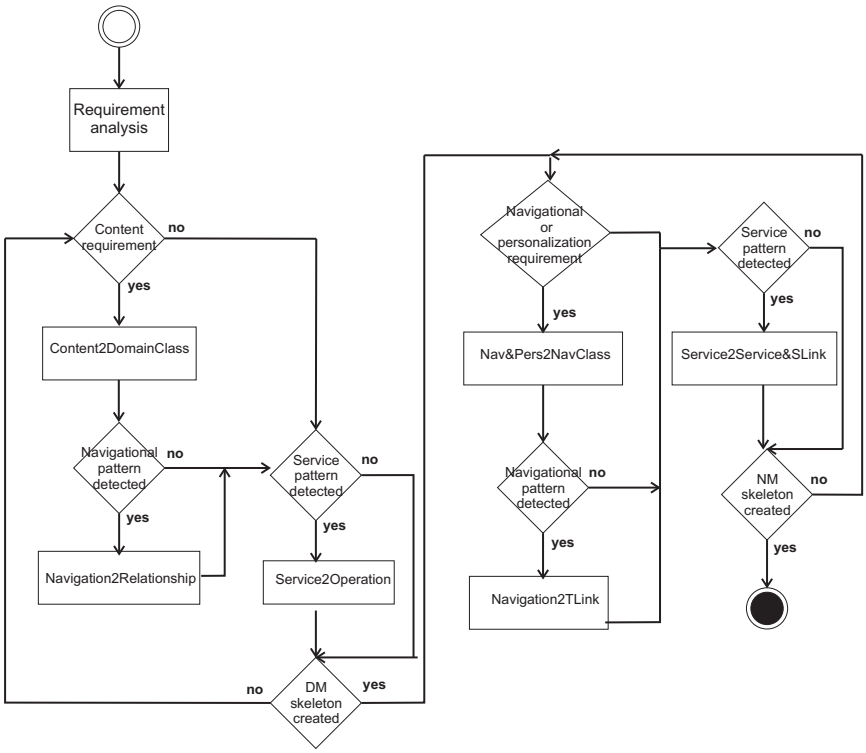


Fig. 7. Transformation workflow: from requirements to Web design

After analyzing and modeling the requirements of the website according to the guidelines presented in the previous subsection, the Domain model (DM) and Navigational model (NM) are generated from the specified requirements. Before explaining each of the derivations, we briefly introduce the QVT language, as well as the A-OOH DM and NM so the reader can easily follow the derivation of them.

Query/View/Transformation language Transformation between models can be defined in a formal way by using some transformation language [20]. These formal transformations must allow to automatically derive models assuring semantic correctness [21, 22]. Furthermore, they must be easily readable, understandable, adaptable, and maintainable [23]. To

this aim, OMG proposes the MOF 2.0 Query/View/Transformation (QVT) language [16], a standard approach for defining formal relations between MOF-compliant models.

QVT consists of two parts: declarative and imperative. The declarative part provides mechanisms to define relations that must hold between the model elements of a set of candidate models (source and target models). A set of these relations (or transformation rules) defines a transformation between models. The declarative part of QVT can be split into two layers according to the level of abstraction: the relational layer that provides graphical and textual notation for a declarative specification of relations, and the core layer that provides a simpler, but verbose, way of defining relations. The imperative part defines operational mappings that extend the declarative part with imperative implementations when it is difficult to provide a purely declarative specification of a relation.

In this paper, we focus on the relational layer of QVT. This layer supports the specification of relationships that must hold between MOF models by means of a relations language. A QVT relation (see Fig. 8) is defined by the following elements:

- **Two or more domains**: Each domain is a distinguished set of elements of a candidate model (source or target model). This set of elements (denoted by a `<<domain>>` label, see Fig. 8) must be matched in that model by means of patterns. A domain pattern can be considered as a template for elements, their properties and their associations that must be located, modified, or created in a candidate model in order to satisfy the relation. A relation between domains can be marked as check-only (labeled as C) or as enforced (labeled as E). When a relation is executed in the direction of a check-only domain, it is only checked if there exists a valid match in the model that satisfies the relationship (without modifying any model if the domains do not match); whereas for a domain that is enforced, when the domains do not match, model elements are created, deleted, or modified in the target model in order to satisfy the relationship. Moreover, for each domain the name of its underlying metamodel is specified (labels M1 and M2 in Fig. 8).

- **When clause**: This clause specifies the condition under which the relation needs to hold (i.e., it forms a precondition). This clause may contain arbitrary OCL (Object Constraint Language) [24] expressions in addition to the relation invocation expressions.

- **Where clause**: This clause specifies the condition that must be satisfied by all model elements participating in the relation (i.e., it forms a postcondition). This clause may also contain OCL expressions or relation invocation expressions.

Defining relations by using the QVT language has the following advantages:

1. QVT is a standard language.

2. Relations are formally specified, and transformation engines (e.g., Borland Together Architect,[c] SmartQVT,[d] mediniQVT,[e] or ATL[f]) can execute them automatically.

[c] `http://www.borland.com/together`
[d] `http://smartqvt.elibel.tm.fr`
[e] `http://projects.ikv.de/qvt`
[f] `http://www.eclipse.org/m2m/atl`

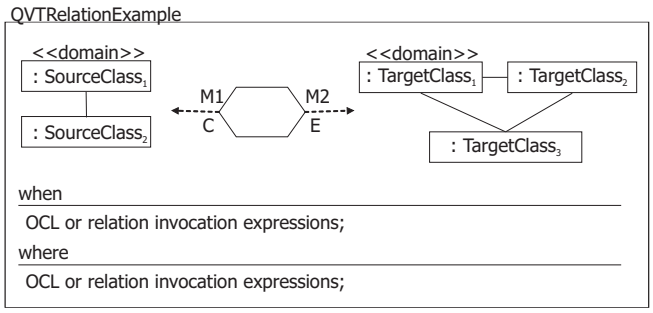


Fig. 8. Example of a QVT relation.

3. Relations can be easily integrated within any Web methodology (provided that metamodels are used).

Deriving the Domain model. The A-OOH DM is expressed as a UML-compliant class diagram. It encapsulates the structure and functionality required of the relevant concepts of the application and reflects the static part of the system. The main modeling elements of a class diagram are the classes (with their attributes and operations) and their relationships.

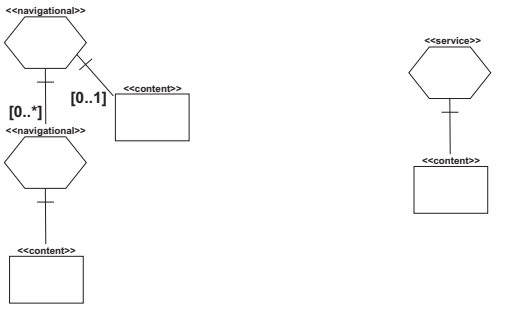
Table 1 summarizes how DM elements are mapped from the SR model. To derive a preliminary version of the DM we take into account two types of requirements defined in Sect. 2 content and service requirements. We have detected several patterns in the i* models and we have used these patterns to define several transformation rules in QVT. Specifically, three transformation rules are defined in order to derive the DM from the SR model:

- *Content2DomainClass* By using this transformation rule, each content requirement is detected and derived into one class of the DM.

- *Navigation2Relationship* Preliminar relations into classes are derived from the relations among goals/tasks with attached resources by applying this rule. To generate the associations in the DM we have to detect a *navigational pattern* in the SR model of the *website* stakeholder. In Fig. 9(a) we can see that the *navigational pattern* consists of a navigational root requirement (i.e. task) which can contain one or more navigational requirements attached. Each of the navigational requirement can have attached a resource (i.e. content requirement). The classes mapped from the resources we find in such pattern will have an association relation between them. The QVT rule which describes this transformation is shown in Fig. 10.

- *Service2Operation* This transformation rule detects a *service pattern*, i.e. a service requirement with an attached content requirement in the SR model (see Fig. 9(b)). In this case each service requirement is transformed into one operation of the corresponding class (represented by the content requirement). In this QVT rule (shown in Fig. 11), a service pattern is detected and transformed into the corresponding elements in the target model.

Once the DM skeleton has been obtained it is left to the designer to refine it, who will also have to specify the most relevant attributes of the classes, identify the cardinalities and



(a) Navigational pattern (b) Service pattern

Fig. 9. Patterns.

define (if existing) the hierarchical relationships.

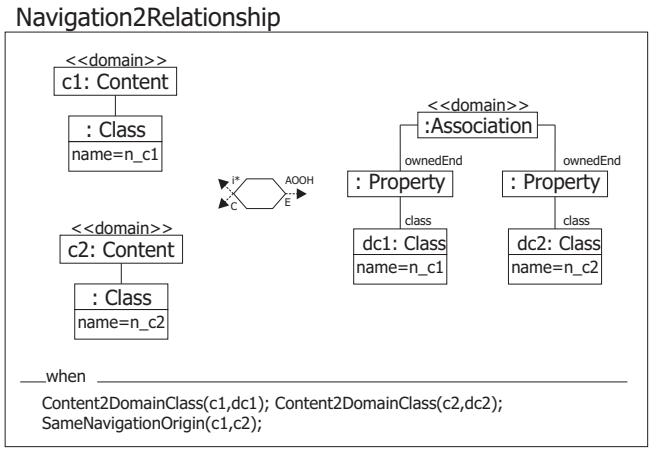


Fig. 10. QVT transformation rule for the navigation pattern

After the preliminary DM is created, a skeleton of the NM is also derived from the specified requirements. This diagram enriches the DM with navigation and interaction features. It is introduced next.

Deriving the Navigational model. The A-OOH Navigational model is composed of Navigational Nodes, and their relationships indicating the navigation paths the user can follow in the final website (Navigational Links).

There are three types of Nodes: (a) Navigational Classes (which are view of the domain classes), (b) Navigational Targets (which group the model elements which collaborate in the fulfillment of every navigation requirement of the user) and (c) Collections (which are (possible) hierarchical structures defined in Navigational Classes or Navigational Targets. The most common collection type is the C-collection (Classifier collection) that acts as an abstraction mechanism for the concept of menu grouping Navigational Links). Navigational

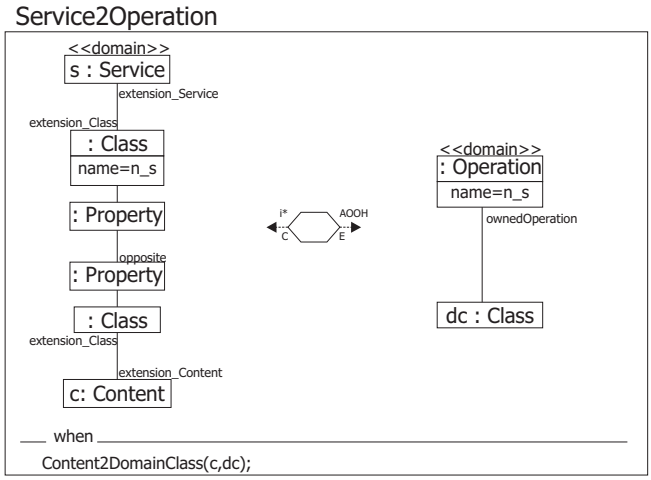


Fig. 11. QVT transformation rule for the service pattern in the DM

Table 1. Derivation of the Domain model

i* element	**A-OOH element**
Content Requirement	Class
Service Pattern	Operation
Navigational Pattern	Association between classes

Links (NL) define the navigational paths that the user can follow through the system. A-OOH defines two main types of links: Transversal links (which are defined between two navigational nodes) and Service Links(in this case navigation is performed to activate an operation which modifies the business logic and moreover implies the navigation to a node showing information when the execution of the service is finished).

To derive the NM we take into account the content requirements, service requirements and the navigation and personalization requirements. We also take into consideration the detected patterns (see Fig. 9) in order to develop several QVT transformation rules. In Tab. 2 we can see a summary showing how the different requirements are derived into elements of the NM. In the right part of Fig. 7 we can see the different transformation rules that are to be performed in order to derive a preliminary Navigation model. In this case we also define three transformation rules:

- *Nav&Pers2NavClass* By using this rule, a "home" navigational class is added to the model, which is a C-collection representing a Menu grouping navigational links. From each navigational and personalization requirement with an associated content requirement a navigational class (NC) is derived. From the "home" NC a transversal link is added to each of the generated NCs.

- *Navigation2TLink* This rule checks the *navigational pattern*, if it is detected, then a transversal link is added from the NC that represents the root navigational requirement to each of the NCs representing the associated navigational requirements.

- *Service2Service&SLink* Finally, the *service pattern* is checked by applying this transfor-

Table 2. Derivation of the Navigation model

i* element	**A-OOH element**
Navigation and Personalization Requirements	Navigational Class
Navigation Pattern	Transversal Links
Service pattern	Operation + Service Link with a target Navigational Class

mation rule. If a service pattern is found, then an operation to the class representing the resource is added and service link is created from each of the operations, with a target navigational class which shows the termination of the service execution. The QVT rule which describes this transformation is shown in Fig. 12.

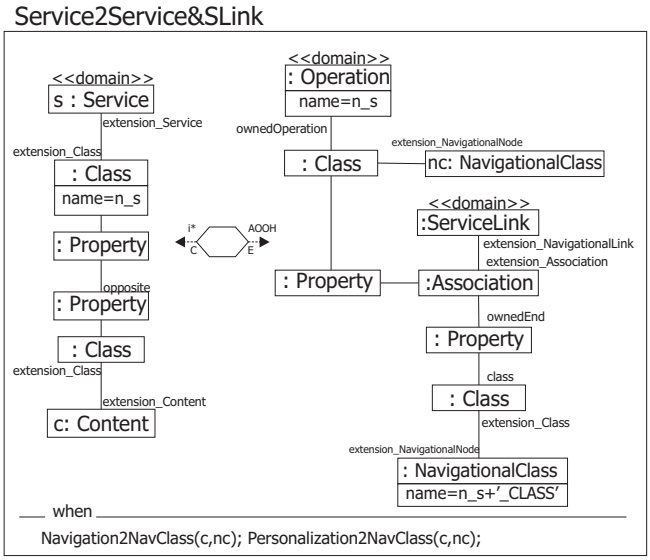


Fig. 12. QVT transformation rule for the service pattern in the NM

Finally, the derived NM could be refined by the designer in order to specify complementary elements for the desired navigation paths.

4 Automatic Goal Evaluation

In order to automatically evaluate whether the goals defined in the i* model for Web applications are satisfied by the application, our approach extends the approach presented in [25] where manual or semi automatic evaluation of general i* models is described. In that work, every task in the i* models is tagged with one of these possible labels:

- Satisfied (✔): the element is satisfied.
- Partially Satisfied (✔̇): represents the presence of evidence which is sufficient to satisfy an element.

- Partially denied (✗̇): represents the presence of negative evidence to satisfy an element.
- Denied (✗): has evidence that the element is not satisfied.
- Conflict (⧗): indicates the presence of both positive and negative evidence of roughly the same strength.
- Unknown (?̇): represents the situation where there is evidence, but its effect is unknown.
- None: lack of any label.

Once an initial configuration of labels are set, an algorithm is executed to evaluate which goals are satisfied. This algorithm consists of propagating the labels that are given to the initial elements to the other elements. The algorithm is iterative and may require the intervention of the user if we can not decide the resulting label value.

Our evaluation of i* models is done by using WebSpec diagrams to generate a set of test cases. In this way, an initial configuration of labels is obtained from test cases in order to verify and validate Web requirements. Our approach has a clear advantage: when users are involved in the development process of Web applications they want to know which goals are being satisfied while the application is under development (periodically, e.g. every hour). Our approach provides this automation without imposing any overhead to the development team.

In our i* models for Web requirements, the actor that represents the Web application may have several tasks (of course, every task can be further decomposed in other tasks). For some of these tasks, WebSpec diagrams and mockups have been developed (by following the process shown in Sect. 2.2) so that we specify in more detail the behavior of the application and agree on broad aspects of the UI before the development begins. By using WebSpec features we automatically derive a set of interaction tests from the diagrams. These tests will assess if the application correctly implements the requirements that they express. Thus, there is a transitive relationship between Goals ↔ Tasks ↔ WebSpec diagrams ↔ Interaction tests. Indeed, if every test that is transitively related with a specific task is satisfied, then we can say that the task is satisfied too.

Our process for automatically giving initial labels to elements of our i* models for Web requirements (assuming that we have a specific version of the application, an i* model, and WebSpec diagrams and their associations) is as follows:

1. Each test (t_i) associated with a WebSpec diagram (WS) is run. If it passes then the edge (w_i) that links the diagram and the test as a weight of 1, otherwise 0.
2. A WebSpec diagram is Y% satisfied where $\mathrm{Y} = \frac{\sum_{i=1}^{z} w_i}{z}$ and z is the size of tests that WS has.
3. A task (T) is X% done where $\mathrm{X} = \sum_{j=1}^{x} h_j * Y\%$, x is the number of diagrams associated with T, h_j is a weight defined (only once per diagram) such that $\sum_{j=1}^{x} h_j = 1$.

Once the initial labels are set, we can reuse the i* evaluation framework presented in [25] to automatically evaluate if the goals are satisfied based on the generated tags. However, tasks can now represent a percentile (instead of completely satisfied, partially satisfied, etc.)

thus we need to provide a mapping between our percentiles and the labels used to applied the algorithm. For example, we can define the following policy (note that the values for X, Y, and Z depend on the application under development and the actual characteristics of the project.):

- None: initial tag.
- None: if all the tests of a task have been failing since the beginning of the process.
- Satisfied: if the percentile of tests passed is $> X\%$.
- Partially Satisfied: if the percentile of the tests passed is between X% and Y%.
- Partially Denied: if the percentile of the tests passed is between Y% and Z%.
- Denied: if the percentile of tests passed is $< Z\%$.

One of the main advantages of our approach is that i* models are evaluated for Web engineering in an objective and straightforward manner, thus avoiding the problem of deciding if a task has been performed or not. Also, it is lightweight and does not impose any overheads during development. In the following section we show our approach in action in our case of study.

5 Sample Application of our Approach

In the next subsections we explain our process described in Fig. 1 by means of an example based on the i* model defined in Sect. 2.1 (see Fig. 3). We show how we specify in detail a specific requirement (Sect. 5.1), then a set of models is obtained (Sect. 5.2) which need to be refined. During the refinement process we can evaluate if our models satisfy the tasks by running the tests and evaluating their results (Sect. 5.3).

5.1 Detailed requirement specification

For the sake of understandability, we will show the mockups and diagrams corresponding to the "provide books by category" task. As shown in Fig. 13 the mockup for this task adds a combo-box in the home page that the user can change to filter the books according to the selected category. Also, the mockup shows its corresponding title and how many books have been found.

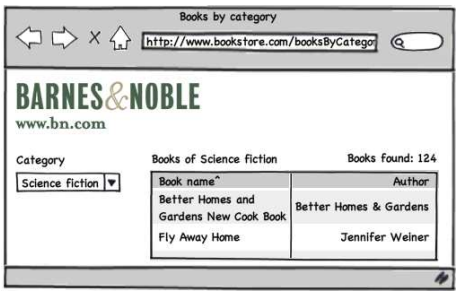


Fig. 13. "Provide books by category" mockup

In Fig. 14 we show a WebSpec diagram that specifies this scenario. Basically, the user starts being located in the Home page and can choose a category from a list of categories. After the user selects the category it should navigate to a different page that contains the title and a list of items. The invariant of this interaction is as follows: CategoryResult.title = "Books of " + ${category} && CategoryResult.bookSize > 0. The invariant states that the title should be valid according to the selection we have done on the previous combo-box and that there should be at least 1 product in the list.

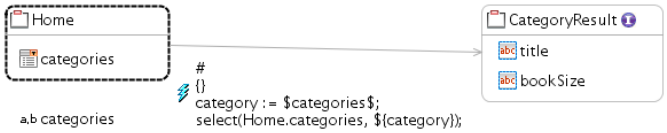


Fig. 14. "Provide books by category" WebSpec diagram

5.2 Generation of Domain and Navigational Models

In Fig. 15 we can see the Domain model which has been derived from the specified requirements. As explained in Sect. 3, to derive the Domain model we take into account the content and service requirements as well as the existence of service or navigational patterns. In this case we can see that five domain classes are created by applying the *Content2DomainClass* transformation rule: one class is generated for each content requirement specified in the SR model. Moreover, we detect three service patterns (see Fig. 9(b)), so operations are added to the classes *client, cart* and *book* by executing the *Service2Operation* rule. Finally we detect that the *Provide Book Info* requirement follows the navigational pattern as we can see in Fig. 9(a). In this case the rule *Navigation2Relationship* adds associations among all the resources found in this pattern. The generated Domain model is shown in Fig. 15.

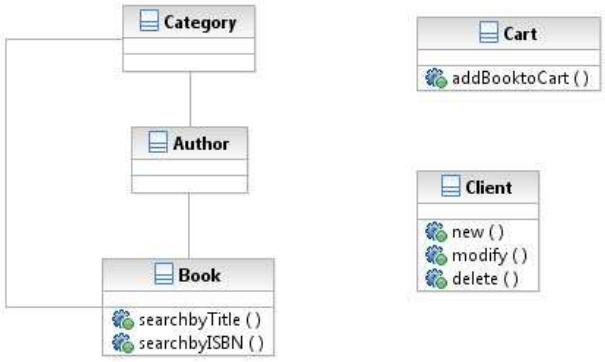


Fig. 15. Generating a Domain model

In the case of the Navigational model, the rule *Nav&Pers2NavClass* is performed adding a home page with a collection of links (i.e. menu). Afterwards, one NC is created for each nav-

igational and personalization requirement with an attached resource, in this case we have five NC created from navigational and personalization requirements. From the menu, a transversal link to each of the created NCs is added (L1 to L4).

The next step consists in checking the navigational and service patterns. In this example, we find a navigational pattern (see Fig.9(a)) where we apply the *Navigation2TLink* transformation creating a transversal link from the NCs created by the associated navigational requirements, to the NC that is represented by the root navigational requirement. In this case two links are added: L5 and L6.

Finally, as we are referring to the website stakeholder, we find three service patterns from which the operations of the NCs books and cart are added and the service links L7, L8 and L9 are created with an associated target NC by applying the *Service2Service&SLink*.

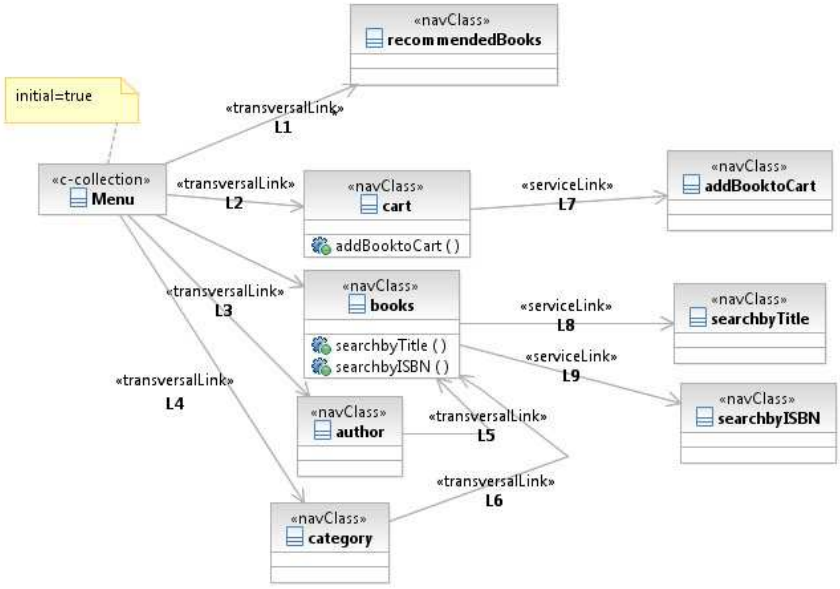


Fig. 16. Generating a Navigation model

5.3 Goal Evaluation

After we obtain and refine the models for our example, our evaluation algorithm should be applied. For our example, we have set X = 80%, Y = 60% and Z = 40%.

In our sample scenario up to 9 tests of 10 of a WebSpec diagram associated with the "provide books by title", "provide books by category", and "provide books by author" tasks are satisfied and only 1 test of 4 of a WebSpec diagram associated with the "provide recommended books" task is satisfied. Following the previously defined policy, the "provide books by title" task is satisfied and the "provide recommended books" is not satisfied (25% of the test passed). A sample of the i* model already evaluated with a starting configuration of labels from our test cases are shown in Fig. 17 (starting labels are given to the tasks in grey). To sum up, if we implement the Web application by taking into account these test cases, then we will obtain an application that achieves the softgoal"easy to maintain" but neglects the

main goal "Book sales be managed" (deduced by applying the algorithm in [25]).

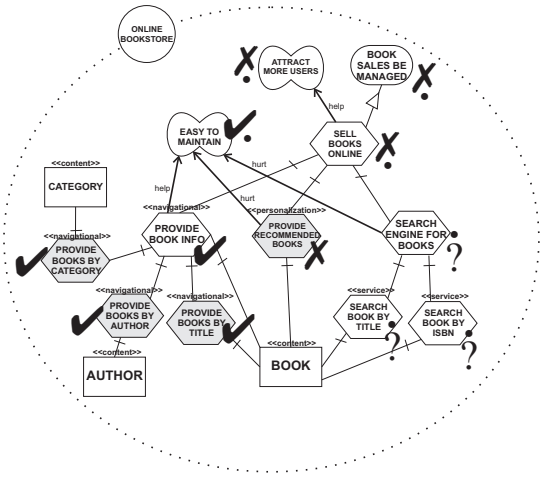


Fig. 17. The result of evaluating the i* model after running the tests

5.4 Implementation Framework

The presented approach has been implemented by using the *Eclipse development platform*[g] *Eclipse* is a framework which can be extended by means of plugins in order to add more features and new functionalities. A plugin that supports both defined UML profiles for i* has been developed. This new plugin implements several graphical and textual editors. Fig. 18 shows an overview of the tool: the palette for drawing the different elements of i* can be seen on the right-hand side of the figure, while a sample SR model is shown in the center of the figure. Generation rules are also being defined and tested in our prototype.

Implementation of our Web requirements i* model. Our implementation of the i* framework for Web requirements consists of a UML profile which incorporates a number of taxonomic features that enable Web requirements specification. With the implementation of this UML profile has been possible to implement the i* framework in Web to model the needs and expectations of the stakeholders of the Web application. The special features incorporated into the i* framework have allowed that elements of the model can be stereotyped using the requirements taxonomy presented in Sect. 2.

Implementation of A-OOH domain model. The domain model in A-OOH is represented by an UML class diagram, for this reason we have implemented the UML 2.0 metamodel using the Eclipse facilities to represent only the elements necessary to establish a UML class diagram.

Implementation of A-OOH navigational model. The A-OOH navigational metamodel represents the key to the derivation of the navigational model. The implementation was developed using UML profiles.

[g] `http://www.eclipse.org`

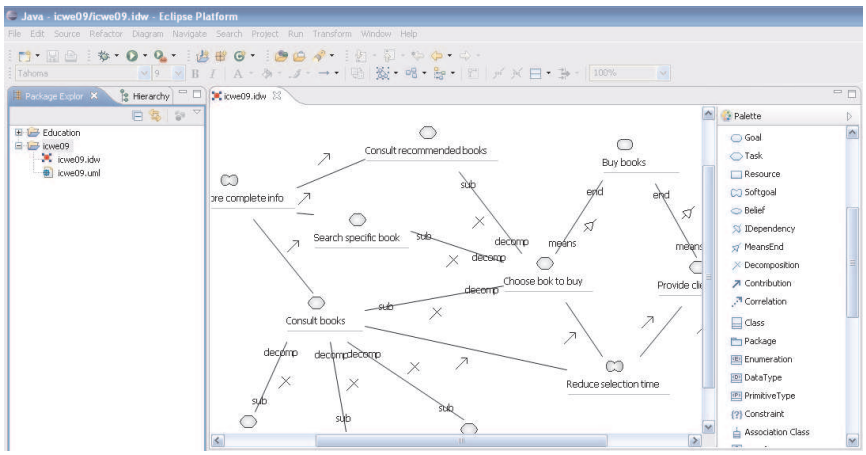


Fig. 18. Screenshot of our prototype

Implementation of the QVT transformation rules. Throughout the paper, QVT has been used as a language for formalizing transformations between models, thus ameliorating the understandability of the transformation process. However, once the transformations have been modeled, they have to be implemented. To this aim, the QVT transformation rules presented above has been implemented using the mediniQVT transformation engine.

Integration with WebSpec. The i* plugin can be easily used together with the WebSpec's Eclipse plugin, thus allowing us to seamless integrate the i* model and the WebSpec diagrams. In Fig. 19 a screenshot of this plugin is shown.

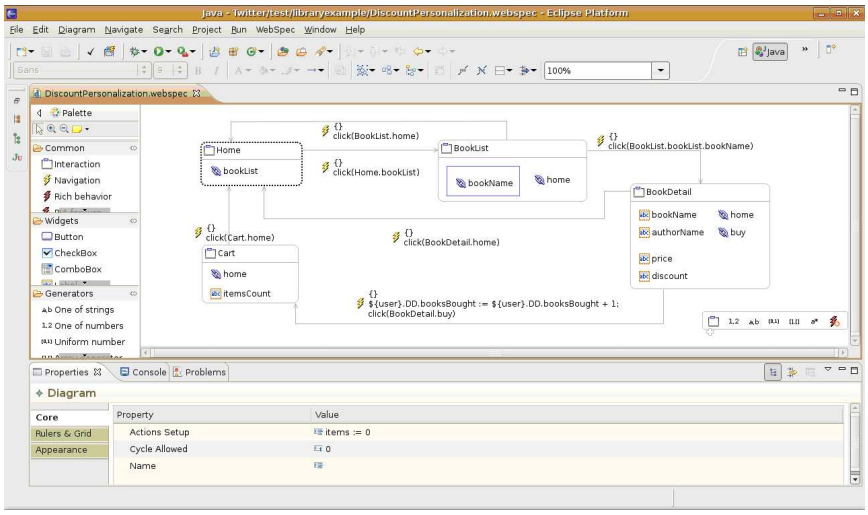


Fig. 19. WebSpec's Eclipse plugin

6 Related Work

Few approaches have focused on defining an explicit requirement analysis stage to model user needs. We can stress the following:

NDT [13] considers a complete taxonomy for the specification of Web requirements. It allows to specify requirements by means of use cases diagrams and templates. It uses a different template for each requirement type they consider, so requirements and objectives are described in a structured way. UWE [7] also describes a taxonomy for requirements related to the Web. It proposes extended use cases, scenarios and glossaries for specifying requirements. WebML [5] also proposes the use of use case diagrams combined with activity diagrams and semi-structured textual description. WSDM [3] is an audience driven approach in which they do a classification of the requirements and the audience. These classes are represented with a diagram in which they are related. Then they are modeled into detail in a Task model using concurrent task trees. OOHDM [26] captures the requirements in use case diagrams. They propose the use of UIDs (user interaction diagrams) for defining the requirements related to navigation which are derived from the Use cases. OOWS [27] focuses on the specification of tasks. They extend the activity diagrams with the concept of interaction point to describe the interaction of the user with the system.

Furthermore, generation of conceptual models from the requirements is an important issue to bridge the gap between users' needs and Web design. To the best of our knowledge, there are two approaches that support this in some way: OOWS provides automatic generation of (only) navigation models from the tasks description by means of graph transformation rules, while NDT [28] defines a requirement metamodel and allows to transform the requirements model into a content and a navigational model by means of QVT rules. Our approach for deriving conceptual models resembles NDT since we have also adopted QVT in order to obtain design artifacts from Web requirements, but we have kept the benefits of the i* framework by means of the defined profiles and patterns.

However, some of these approaches present the following drawbacks: (i) they do not take into consideration a complete taxonomy of requirements which is suitable in Web applications, or (ii) they consider non-functional requirements in an isolated manner, or (iii) they mainly focus on design aspects of the intended Web system without paying enough attention to Web requirements. Furthermore, none of them perform the analysis of the users' needs. Requirements are figured out by the designer, it may be needed to re-design the website after doing usability and satisfaction tests to the users. Modeling users allow us ensuring that the Web application satisfies real user needs and goals and the user is not overwhelmed with functionalities that he does not need or expect and he does not miss functionalities that were not implemented.

To the best of our knowledge, the only approaches that use goal oriented techniques have been presented in [29, 30]. They propose a complete taxonomy of requirements for the Web and use the i* notation to represent them. Unfortunately, they do not benefit from every i* feature, since they only use a metamodel that has some of its concepts, e.g. means-end, decomposition or contribution links from i* are not specified in the approach presented in [29]. Our approach not only benefits from i* features but also used with mockups and WebSpec diagrams can provide automatic evaluation of its models. This feature is extremely important to get feedback during the development process of a Web application.

On the other hand, in a goal-oriented requirement engineering approach, goal evaluation is important to check whether the goals and needs of the stakeholders are satisfied. Specially in the Web engineering field, where the continuos participation of the stakeholders during the process is vital to obtain feedback. There are some approaches that have goal evaluation such as the NFR Framework [31]. In this framework qualitative labels are propagated throughout a Softgoal Interdependency Graph (SIG), and similar to the goal evaluation procedure of [25], the user must resolve the conflicts [31]. In [32] there are some guidelines on how to extend this procedure to be used with i*. Despite this procedure could be applied to i* models, it should be adapted to be used to provide feedback to stakeholders. The resolution of conflicts and the integration with late requirement analysis artifacts like WebSpec are the main drawbacks we have found. Our approach integrates seemless with a detail requirement analysis and provides an automatic way of evaluating if the goals are been satisfied by the application under development.

GRL [33], a variant of i*, has a fully automated evaluation method but does not allow to make decisions in the presence of conflicting, partial or unknown information. The hard-coded rules used to resolve softgoals often result in the proliferation of conflicts or partial values. Moreover, this approach should be adapted to Web engineering to provide automatic generation of the models and automatic requirements validation as shown in this paper.

7 Conclusions and Future Work

Websites require special techniques for requirement analysis in order to reflect, from early stages of the development, specific needs, goals, interests and preferences of each user or user type. However, Web engineering field does not pay the attention needed to this issue. We have presented a goal oriented approach on the basis of the i* framework to specify Web requirements. It allows the designer to make decisions from the very beginning of the development phase that would affect the structure of the envision website in order to satisfy users.

We have improved the requirements phase by complementing i* models with mockups and WebSpec diagrams to provide a more detailed analysis of interactive requirements. Also, a first version of the domain and navigational models are obtained from the i* model allowing developers to have a starting point for model refinement. During the refinement process, users can observe and provide feedback of the progress by looking at the automatic evaluation of the i* model. This evaluation is performed by executing the automatic derived tests, generated from WebSpec, against the application under development.

Our short-term future work consists of completing the transformation rules in order to obtain the rest of the A-OOH models (i.e. presentation and personalization models). Finally, as long-term future work we plan to carry out a set of experiments to measure the effectiveness of our proposal.

Acknowledgements

This work has been partially supported by the MESOLAP project (TIN 2010-14860) from the Spanish Ministry of Education and Science, by the QUASIMODO project (PAC08-0157-0668) from the Castilla-La Mancha Ministry of Education and Science (Spain), and by the MANTRA project (GRE09-17) from the University of Alicante (Spain).

References

1. S. Casteleyn, W. V. Woensel, and G.-J. Houben. A semantics-based aspect-oriented approach to adaptation in Web engineering. In *Hypertext*, pages 189–198, 2007.
2. C. Cachero and J. Gómez. Advanced conceptual modeling of Web applications: Embedding operation interfaces in navigation design. In *JISBD*, pages 235–248, 2002.
3. S. Casteleyn, I. Garrigós, and O. D. Troyer. Automatic runtime validation and correction of the navigational design of Web sites. In *APWeb*, pages 453–463, 2005.
4. N. Koch. Software engineering for adaptive hypermedia systems: Reference model, modeling techniques and development process. *Softwaretechnik- Trends*, 21(1), 2001.
5. S. Ceri and I. Manolescu. Constructing and integrating data-centric web applications: Methods, tools, and techniques. In *VLDB*, page 1151, 2003.
6. G. Rossi, D. Schwabe, and R. Guimarães. Designing personalized Web applications. In *WWW*, pages 275–284, 2001.
7. N. Koch. Reference model, modeling techniques and development process software engineering for adaptive hypermedia systems. *KI*, 16(3):40–41, 2002.
8. I. Garrigós. *A-OOH: Extending Web Application Design with Dynamic Personalization.* PhD thesis, University of Alicante, Spain, 2008.
9. F. Daniel, M. Matera, A. Morandi, M. Mortari, and G. Pozzi. Active rules for runtime adaptivity management. In *AEWSE*, 2007.
10. 10R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.
11. E. Yu. *Modelling Strategic Relationships for Process Reenginering.* PhD thesis, University of Toronto, Canada, 1995.
12. E. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *RE*, pages 226–235, 1997.
13. M. J. Escalona and N. Koch. Requirements engineering for Web applications - a comparative study. *J. Web Eng.*, 2(3):193–212, 2004.
14. D. C. Nguyen, A. Perini, and P. Tonella. A goal-oriented software testing methodology. In *AOSE*, pages 58–72, 2007.
15. E. Robles, I. Garrigós, J. Grigera, and M. Winckler. Capture and evolution of Web requirements using WebSpec. In B. Benatallah, F. Casati, G. Kappel, and G. Rossi, editors, *ICWE*, volume 6189 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2010.
16. QVT Language. `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`.
17. I. Garrigós, J.-N. Mazón, and J. Trujillo. A requirement analysis approach for using i* in Web engineering. In *ICWE*, pages 151–165, 2009.
18. H. Estrada, A. M. Rebollar, O. Pastor, and J. Mylopoulos. An empirical evaluation of the i* framework in a model-based software generation environment. In *CAiSE*, pages 513–527, 2006.
19. M. Strohmaier, J. Horkoff, E. S. K. Yu, J. Aranda, and S. M. Easterbrook. Can patterns improve i* modeling? two exploratory studies. In *REFSQ*, pages 153–167, 2008.
20. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained. The Practice and Promise of The Model Driven Architecture.* Addison Wesley, 2003.
21. K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Technique in the Context of the Model Driven Architecture*, Anaheim, October 2003.
22. A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The missing link of MDA. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *ICGT*, volume 2505 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2002.
23. S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.
24. OCL. `http://www.omg.org/cgi-bin/doc?ptc/03-10-14`.
25. J. Horkoff and E. Yu. Evaluating goal achievement in enterprise modeling an interactive procedure

and experiences. In W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, A. Persson, and J. Stirna, editors, *The Practice of Enterprise Modeling*, volume 39 of *Lecture Notes in Business Information Processing*, pages 145–160. Springer Berlin Heidelberg, 2009. 10.1007/978-3-642-05352-812.

26. D. Schwabe and G. Rossi. An object oriented approach to Web-based applications design. *TAPOS*, 4(4):207–225, 1998.
27. P. Valderas, V. Pelechano, and O. Pastor. A transformational approach to produce Web application prototypes from a web requirements model. *Int. J. Web Eng. Technol.*, 3(1):4–42, 2007.
28. N. Koch, G. Zhang, and M. J. Escalona. Model transformations from requirements to Web system design. In *ICWE*, pages 281–288, 2006.
29. D. Bolchini and P. Paolini. Goal-driven requirements analysis for hypermedia-intensive Web applications. *Requir. Eng.*, 9(2):85–103, 2004.
30. F. M. Molina, J. Pardillo, and J. A. Toval. Modelling Web-based systems requirements using WRM. In *WISE Workshops*, pages 122–131, 2008.
31. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering (THE KLUWER INTERNATIONAL SERIES IN SOFTWARE ENGINEERING Volume 5)*. Springer, 1st edition, October 1999.
32. L. Liu and E. Yu. Designing information systems in social context: a goal and scenario modelling approach. *Inf. Syst.*, 29(2):187–203, 2004.
33. D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu. Evaluating goal models within the goal-oriented requirement language. *Int. J. Intell. Syst*, 25(8).
34. i* wiki. `http://istar.rwth-aachen.de`.