TRIAD: Triad-based Rich Internet Application Design

F. J. Martínez-Ruiz[1], Jean Vanderdonckt[1], Jaime Muñoz Arteaga[2]

[1]Université catholique de Louvain, Louvain School of Management
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{Francisco.martinez, jean.vanderdonckt@uclouvain.be} – www.isys.ucl.ac.be/bchi/members/jva
[2]Universidad Autónoma de Aguascalientes Aguascalientes, Mexico
jmunozar@correo.uaa.mx

ABSTRACT

Current trends in web development still are attached to the web page paradigm. Nevertheless, new uses of already available technology and recent development in terms of concepts, as the asynchronous communication, have produced a new generation of web applications: Rich Internet Applications (RIAs). These web applications essays to fulfill user expectations in terms of usability, reliability, quality, maintainability and performance. In this paper, we are going to present a designing methodology that pursued as goal describing and developing User Interfaces of RIAs in a standardized way. The name of this ensemble of models and meta-descriptions is, TRIAD (Triad-based Rich internet Application Design).

Keywords

Rich internet applications, web engineering, Model driven engineering, USIXML

INTRODUCTION

Current trends in web development still are attached to the web page metaphor. The last generation of web applications is called, Rich Internet Application (RIA). These web applications break this paradigm and promise to fulfill user expectations in terms of usability, reliability, quality, maintainability and performance [18]. For this, a set of models based on UsiXML language (User Interface Description Language) supported by the CAMELEON Reference Framework are presented. Our work extends UsiXML User Interface Description Language with the purpose of designing UIs of Rich Internet Applications. This work presents these notations and the general method of design.

Indeed, software design includes a set of notations and models in order to specify different aspects of the software system. These notations specify applications in terms of various abstraction levels. The modeling of web applications has been treated in several works [18][19]. Roughly, we could classify web modeling in four categories: (1) hypertext models where organization and navigation is treated in a single model. [18](2) Data-driven models where exploitation of databases and a query language is translated into web applications [20]. (3) UI is defined in independent representations (e.g., UML) in a Model Driven Approach [21],[22] and Finally, (4) Task based modeling, where web application is modeled in terms of tasks needed to complete the application goal. Our proposal takes a mixed approach since we use two categories: (3) and (4).

The rest of the paper is organized as follows: First, Core elements of the method are explained. Second, a running example shows the application of notations. And Third, Conclusions and Future work.

WHAT IS TRIAD?

In this paper, we are going to present a designing methodology that pursued as goal describing and developing User Interfaces of RIAs in a standardized way. The name of this ensemble of models and meta-descriptions is, TRIAD (**T**riad-based **R**ich **I**nternet Application **D**esign). TRIAD is a method for developing User Interfaces for RIAs. We start with an abstract definition of the UI. Then, in an iterative process more details are included until arriving to a concrete definition. Various features make TRIAD a viable choice: Extensibility (supports composition of structures), a set of Visual Patterns and the separation of concerns (data, logic and presentation).

CORE CONCEPTS OF THE METHOD

Now, we present the core concepts of our method which offers a scalable and model driven engineering approach [16] that is supported by the CAMELEON Reference Framework [5] and UsiXML language [4]. These concepts are: (1) Zoomable User Interfaces [15], which are applied to Task hierarchies (ZUIT concept) [13], (2) the triplet-based design concept and (3) the concept of location.

Zoomable User Interface Task Hierarchy (ZUIT)

TRIAD defines the application as a hierarchy of tasks. This hierarchy is composed by multiple levels where inner nodes are gathering elements and leaf nodes are atomic tasks. Tasks are connected by temporal operators of three types (Sequential, concurrent and choice). Nevertheless, there are some shortcomings in these models (discussed in [13]). For instance, the complexity is directly proportional to the size of the application and at some point, icons and texts become unintelligible. Another problem is that, models as ConcurTaskTrees notation (CTTs) (see Figure) which is discussed in [13], do not provide any semantic information through its structure

In Faure, D., Vanderdonckt, J., (Eds.), *Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010* (Berlin, 20 June 2010), Thales Research and Technology France, Paris, 2010. ISBN 978-2-9536757-0-2

since the structure is simply replicated at all levels. For instance, in a minimal Sign in application (see, Figure a) we have two tasks (Get information and Submit). First you have to recollect user data before submit it. Then, tasks are related by a sequential operator ([]>>). Get information is subdivided into two atomic tasks (Input User Name, Input Password) which are executed at the same time (|[]|).We use a treemap-like representation [14], instead of using an arborescence representation (Figure 1a) with an important variation: indeed, it is a Zoomable User Interface. The representation of the Sign in application is shown in Figure 1b. All tasks under the area of Login rectangle are her children which would be executed in sequential order (from left to right). Under the first child, **Get information** we have two other children. Here, for the sake of simplicity all the tasks are labeled but the only information visible at the beginning is the color (which is associated to operator and task types). We have chosen Piccolo toolkit [4] for taking advantage of its zooming predefined libraries. The benefits expand in three aspects. First, the navigation to inner hierarchies is done in a more intuitive way. Second, the visual overload is reduced since we are using coding schemes in order to identify task types and task relationships (Note: Here, for the sake of simplicity is not expressed). And third, more coding schemes could be integrated. For instance, weight metrics in order to adjust task area of the elements in the ZUIT.

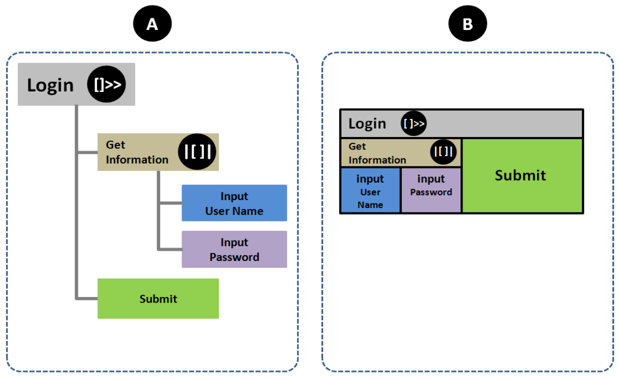


Figure 1. Login example in ZUIT format.

Triplet-based Design

The idea behind our method is simple: Keep designers focused on their tasks so that they do not become distracted when they go though the development process. Their objective is to define the sequence of tasks that is needed to accomplish the application goal. For each task **T** introduced by the designer, task triplets (triads) are introduced in the model. **T** is substituted by **T'** under which **R** (Robustness) and **D** (Decorative) set of tasks are added (Figure a). These groups exchange information between them and with the original task (which is part of the Utilitarian task set[17]). In order to update the model in terms of validation and presentation. In Figure 2b, this structure is represented as a ZUIT. Note: the triad is defined by the task sets: R(T), U(T) and D(T).That means that each set (R, U or D) is populated, with pertinent tasks, in function of the necessities of T.

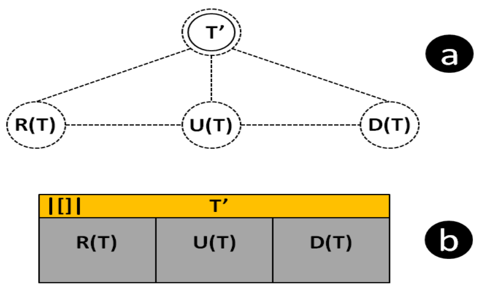


Figure 2. Triplet of task model.

Our model is loosely based in the triad of Vitruvius [8]. He stated that every building design should have three qualities: It must be strong, useful, and beautiful. We translate these features for UI design into: (1) Robustness tasks. This set agglutinates all tasks related to prevention, validation and recuperation. (2) Utilitarian tasks. This set includes Input, Output, Control and navigation tasks. (Besides CRUD operations to be applied on associated data [8]). That is, all the possible operations to be executed over tasks. (3) Decorative tasks. This set groups all tasks related to aesthetic aspects of the presentation of both, robustness and utilitarian tasks of UI. An overview of triads is shown in Tables 1, 2, and 3. Note that these are examples of the tasks in each category; it is not an exhaustive enumeration of them.

Task Operations	
	Description
Input	Entry of data
Output	Describes what information may be shown to the user
Control	Trigger a method of business logic
Navigation	Describes a container transition

Table 1. Utilitarian Tasks.

Prevention (task is affected a priori)	Validation (task is affected during interaction)	Recuperation (task is affected a posteriori)
Static User Help	Data Type check	Roll back
hidden tasks until needed (Displaying code at the last possible moment)	Order check	Error User help (Acknowledge)
	Auto-complete	Reset
	Dynamic User Help	

Table 2. Robustness tasks.

Decorative Tasks		
	Description	**Example**
Semantic	Affects semantic presentation, in general this means the value to be used.	A error signal icon is changed by other in term of internationalization

[17] This set include CRUD and Task operations.

Syntactic	Affects order and presentation of task sequence to be done by the user	Change a text box for a date picker to simply the introduction of a date
Lexical	Affects order and presentation of lexical elements	A button could include a label and or icon or both
Alphabetic	Only affects alphabetic representation	Change from centimeter to millimeter values of a component
Physical	only affects physical appearance	Color, size

Table 3. Decorative Tasks.

Now, we are going to update the Login example (Figure 2). Each task in the Task Hierarchy is wrapped by the triad. This process is automated by XLST transformations in the UsiXML code.

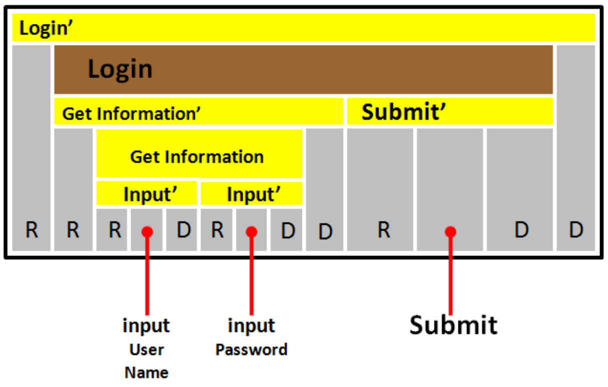


Figure 2. Inclusion of task Triad in Login Example.

The incorporation of the sense of location

It is time to focus in the Web Realm and for this, TRIAD introduces a pattern set in order to model **distributed nature** of RIAs. The original tasks are wrapped by these pattern structures which are added hierarchies that take into account location of tasks (see Table 4). This location is not related to physical places (e.g., client and server) at AUI level. The sense of location has two goals: (1) add hint labels to be used in the next step of the method and (2) populate the TH with tasks related to manipulation and/or system-oriented [17] (which are going to remain hidden until next step). An intuitive definition of location is that web applications could be divided into ***n*** distributed blocks. Nevertheless, we could generalize these arrangements into a two-block model. That is, some components of web application (presentation, logic and data) are located in the client (e.g., browser or local machine). This location is called Local Realm (see W1 in Figure 3b and c). The rest of components are assigned to the Foreign Realm (see Figure 3e). This last one includes server-side components, such as web services, web applications hosted in alternative places than user interaction point (usually his/her web browser). Also, it is possible to extend this definition to include event a local web server or application server. The logic to define the two blocks implies as local the application executed in the client host environment, generally a web browser. It is worth notice that Figure3d describes a mixed realm task. This implies a compound task is internally composed by some elements in local and others in foreign realms. A concrete example is a submit operation which include client and server validation. Nevertheless, This type of task is included in foreign location in order to simplify the notation. Task types used in the Task Hierarchy level are wrapped up and divided in terms of their scope (local or foreign). In Table 3, it is described the set of labels to be added. Their work is adding hints or indicators in order to assist designers and the method in the translation into concrete web components. For instance, if the developer wishes to introduce an interaction task, for him/her the process is straightforward. But the real model will be updated with task triads that take into account the location.

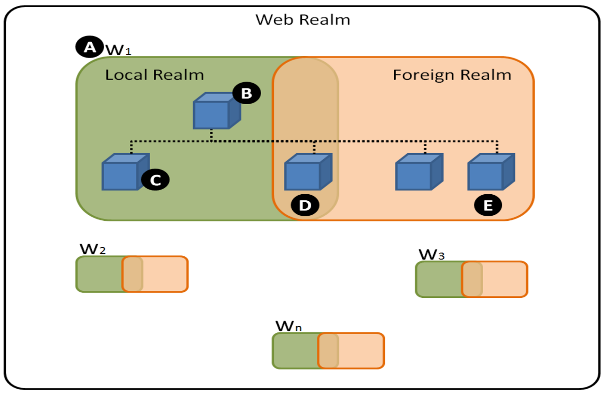


Figure 3. The Web Realm.

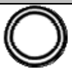
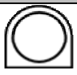
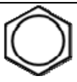
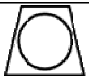
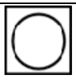
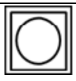
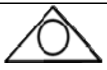
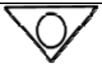
Description	Local Realm		Foreign Realm
Interaction task			
Application task			
Abstract task	 Implicit	 Explicit	
Link	 Explicit	 Implicit	

Table 4. Wrapper notation for RIAs.

GETTING STARTED

This section describes our method step by step modifying a case study in order to show all the updates.

Features of the Method

The development of web applications needs to be very flexible. Changes could appear in all the steps. In order to deal with this characteristic, our method includes two important features: First, the separation of concerns principle is integrated to the method and second is based in a Model Driven Approach.

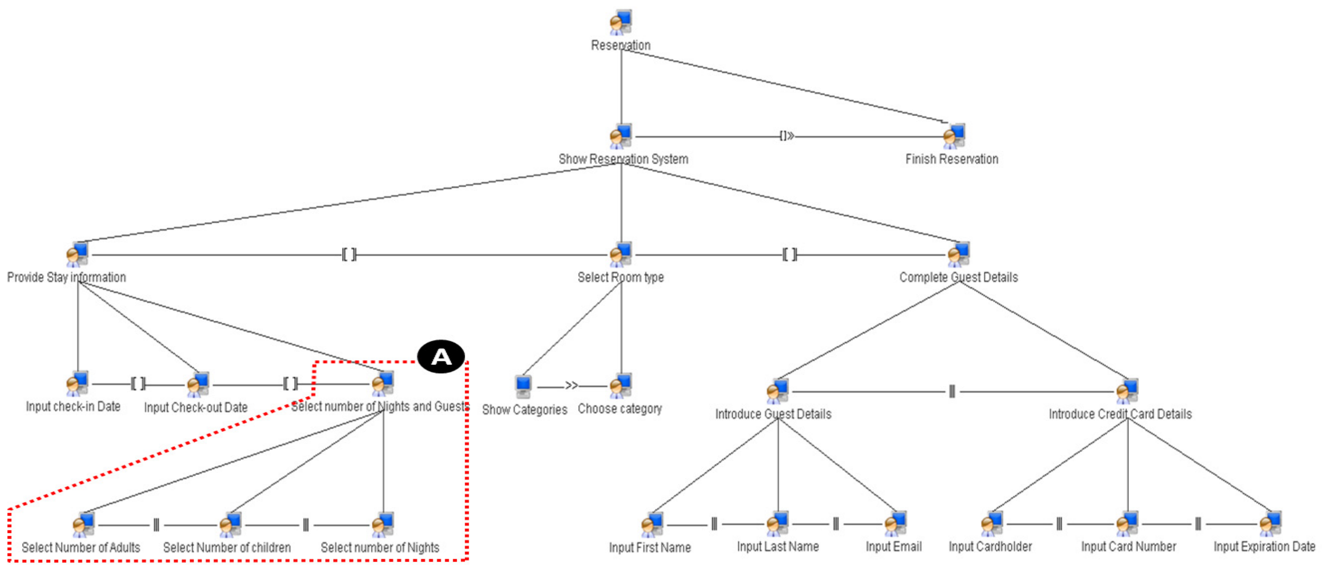


Figure 4. Example of an application for booking a hotelr oom in CTT model.

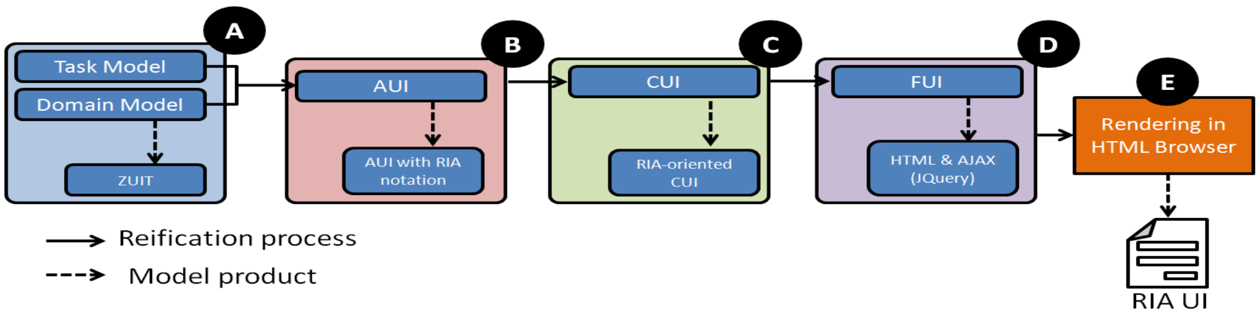


Figure 5. TRIAD Method.

The general Method

Our method is a process of translation abstract models (Figure 4) into more concrete ones (see Figure 5) [3]. The general process could be synthesized in the following steps: In the first phase, we create two models: a Task hierarchy model which describes User's goal and a Domain model to represent the data needed (T&D). The second phase implies the production of an Abstract User Interface (AUI) without any context or compromise with any technological platform. Then, in a Third phase a Concrete User Interface (CUI) is derived from the previous model. There, modality and platform widgets are decided. In the last phase, a Final User Interface (FUI) is obtained for a specific technology (For instance AJAX, .NET, LZX, SWF among others).

CASE STUDY: BOOK A ROOM

We use a Reservation System example. This simple UI allows users to book a hotel room. Check-in and Check–out dates, as well as, the room type and number of guests are selected. The UI is presented in a single interface over three sections. Validation is done during submission and during the recollection of values. This running example is named: Reservation System Case Study (RSC). We are going to focus our attention on the models produced for the different phases of the method. A final remark is that model translation is based on XSLT in all steps.

Step 1: Creation of Task Hierarchy and Domain models

The recollection of User requirements are out of the scope of this work. They could be recollected by UML use cases and activity diagrams. These requirements are expressed in a hierarchy of tasks (TH) and Domain models (UML Class Diagrams).

Task Hierarchy model

The goal of the application is described by a TH. This TH is decomposed in successive levels until arriving to atomic tasks related to system manipulation. In RSC, the goal is booking a room. Then, this task could be decomposed in three subtasks (Pick a Date, Select Room Type and Complete Guest Details) which could be decomposed as well in other sub tasks. The result of this is presented in Figure 4. Also, the temporal relationships of tasks are modeled by temporal operators of three types (Sequential, concurrent and choice). For instance, in the first level of the TH, We have to reserve a room before confirm the transaction. This relationship is expressed with a sequential operator ([]>>). This representation (a variant of CTT) was created with the IDEALXML tool which the current tool used one in UsiXML [11]. Also we have in Figure 6 the proposed Domain model. Then, TH is translated into ZUIT format (see Figure 7) using XLST. In Figure 4a, we marked a fragment in order to show its UsiXML representation (Figure 8). This fragment allows presenting the code which will be refined in following sections.

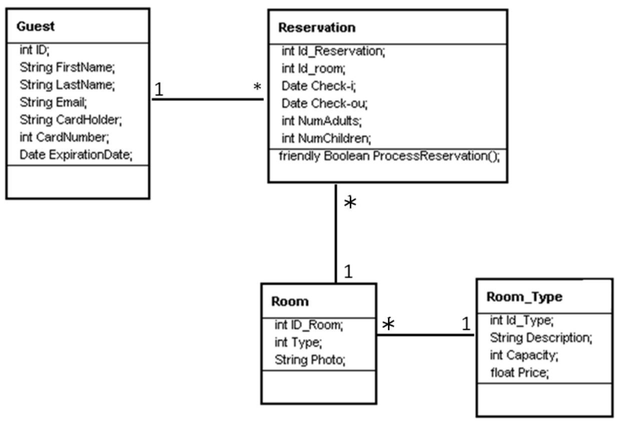


Figure 6. Domain model.

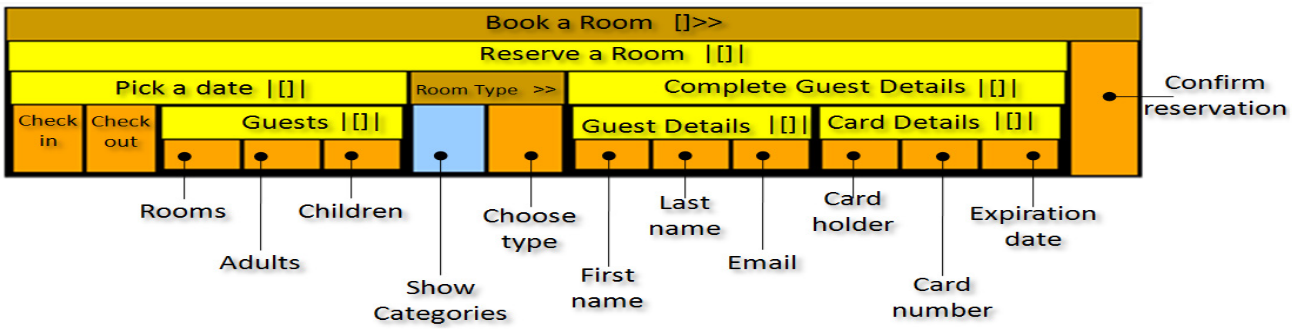


Figure 7. ZUIT representation of Task Hierarchy model.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskmodel>

…
<task id="st0task8" name="Select number of Nights and Guests" type="interaction">
            <task id="st0task19" name="Select Number of Adults" type="interaction" />
            <task id="st0task20" name="Select Number of children" type="interaction" />
            <task id="st0task21" name="Select number of Nights" type="interaction" />
 </task>
<orderIndependence>
            <source sourceId="st0task19" />
            <target targetId="st0task20" />
 </orderIndependence>
 <orderIndependence>
            <source sourceId="st0task20" />
            <target targetId="st0task21" />
 </orderIndependence>
…
</taskmodel>
```

Figure 8. Fragment of UsiXML code of TH.

Mapping Model

The models in TRIAD are connected through a mapping model that aids to define diverse relationships (this model is defined in UsiXML). These relationships allow features as derivation from models from one level to another, helping in the successive process of adding component features and selection of elements (depending on the level, abstract components or concrete ones), also for addressing context aspects (e.g., if the context defines a java-based platform which version of virtual machine is needed).

Available Relationships between domain model and other used models are: **triggers, observes, updates, isReifiedBy, isAbstractedInto, isExecutedIn, isTranslatedInto, manipulates** and **hasContext** (see [7] for more information about them). In the case of RSC, a relationship of manipulates type is established between the Domain and Task Models. For instance, Figure 9 depicts "manipulates" relationships between task and domain model as pointed arrows. **Pick a Date** is mapped onto **Reservation** class. **Complete Guest Details** is mapped onto **Guest** class. **Show Categories** is mapped onto **Room_Type** class. And finally, **Confirm reservation** is mapped onto the method **ProcesReservation** of the class **Reservation**.

Manipulation of TRIAD tasks

The TH begins with the tasks needed to fulfill the goal of the application. The next step is manipulating triad tasks. That is, the developer should select decorative, robustness and utilitarian tasks, see in Figure 10a a mock up of the development environment that we are developing. Here, **FirstName** is selected to change its features. A possible visualization of these sets is shown in Figure 10b. In this case, instantiated attributes are colored in green (Figure c) when you move over Figure 10a. New features or modifications could be done by Triad Management Pattern inside a development environment (Figure 11). Note: We use this pattern to describe the *modus operandi* of the triad task repository.

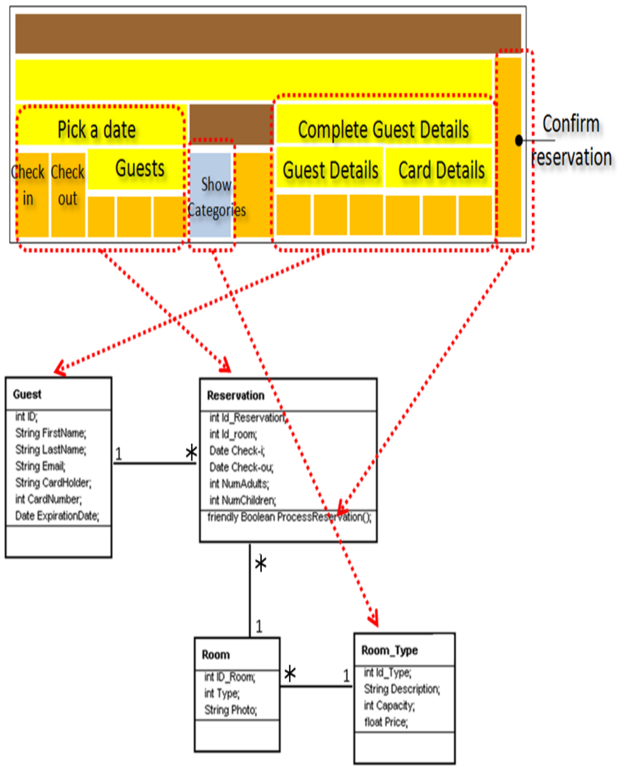
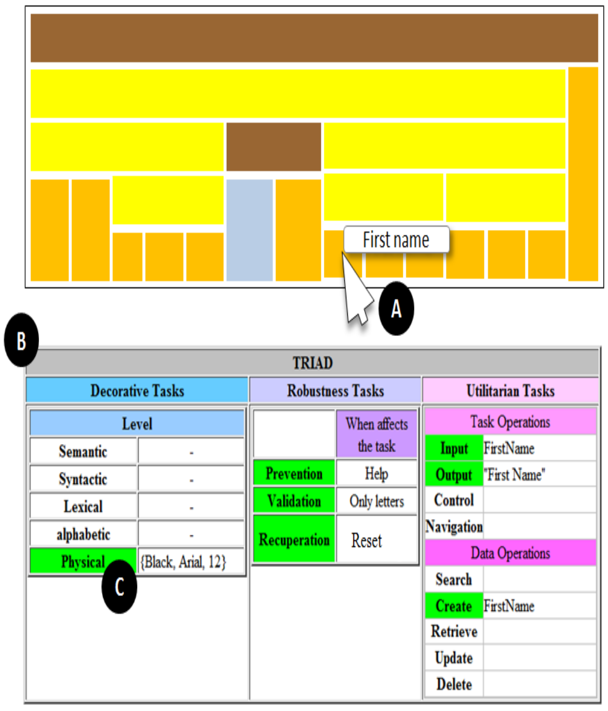


Figure 9. Mapping model.

Step 2: Building the Abstract User Interface

The next step is translating the TH into an AUI model. The process implies the introduction of more details and precisions. The AUI is improved with precisions about location that we introduce in the previous phase. According to an algorithm described in [7] and improved in [8], each inner node is interpreted as an Abstract Container (AC). In Figure 12a is depicted how this operation is done: rounded squares specify task grouping. The algorithm of [7] is straightforward: inner nodes are labeled and converted into Abstract Containers and leaf nodes in Abstract Interaction Components (Figure 13b).

A new visual notation is introduced for modeling AUI representation (see Figure 13). This one includes three main features: First, temporal operators are used in Figure13a in order to explain the abstract dialogue inherited from the previous model. Second, in Figure 13b pointed arrows are used to express abstract adjacency (i.e., the spatial disposition of components). And Third, the RIA patterns are applied and used in each AC and AIC to indicate their application, Figure 13c.



TRIAD

Decorative Tasks	
Level	
Semantic	-
Syntactic	-
Lexical	-
alphabetic	-
Physical	{Black, Arial, 12}

Robustness Tasks	
	When affects the task
Prevention	Help
Validation	Only letters
Recuperation	Reset

Utilitarian Tasks	
Task Operations	
Input	FirstName
Output	"First Name"
Control	
Navigation	
Data Operations	
Search	
Create	FirstName
Retrieve	
Update	
Delete	

Figure 10. ZUIT representation of RSC with instantiated attributes (b).

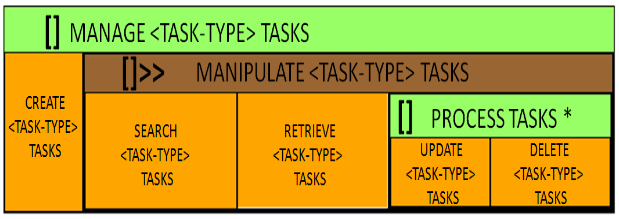


Figure 11. TRIAD instantiation pattern for selecting attributes of tasks.

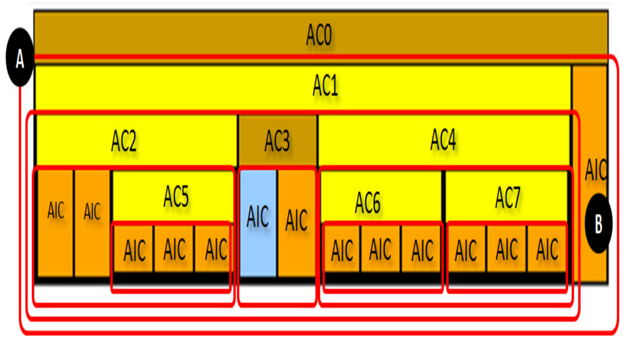


Figure 12. Visual explanation of recovery of Abstract components from ZUIT.

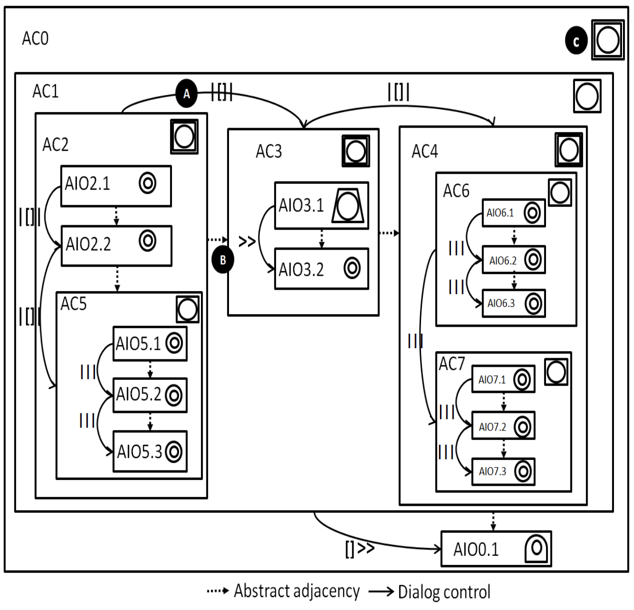


Figure 13. AUI representation of the RSC.

Step 3: Building the Concrete User Interface

A Concrete User Interface (CUI) model is composed of Concrete Interaction Objects (CIO) and concrete relationships. CUI model designates a specific modality (e.g., Graphic or speech). Therefore, behavior and UI elements are adjusted in order to fulfill requirements of this modality. The graphical modality could be described in terms of graphical input **GI** and graphical output **GO, GI** is defined by tuple (pointing device **P**, direct manipulation) where **P** usually is a mouse. **GO** is defined by tuple (Screen, Drawing language) where drawing language could be procedural, declarative, pixel or vector based [7].

At this level, compromise with any toolkit or platform is not yet done. Nevertheless, it is possible to define a CUI in term of valid elements of the modality. For instance, if the graphical modality is selected and the element to be reified is an **explicit container**. Then, the valid set of elements includes window, box, dialogue (defined in UsiXML).In the other hand, if a foreign interaction is associated to an AIC. Then, a RIA- CIO could be selected.

The process of choosing Concrete Interaction Components (CICs)

From the triad table (see Figure 10b) the developer has already chosen the different facets (input, output, control or navigation), data operation (CRUD), data types, and cardinalities among others. Nevertheless, the process has to be refined in order to select components related to the current modality. In Table 5, we can see possible derivations from AUI elements into CUI ones in RSC for **AC2**. The choice is driven by the selection of options (using a decision tree [8]) in the triad and also depending on developer preferences.

Abstract Interaction Component	Facet Specification or/and RIA label	Relevant Information to define which CIC will be selected	Possible Concrete Interaction Component to be used
Pick a date	Explicit container/ Output facet	The container is fixed	A box with a label (output) "Pick a Date"
Check in Date	Local input/ Output "Check in"	Data type, Domain value	A label "Check in" with a date picker
Check out Date	Local input/ Output "Check out"	Data type, Domain value	A label "Check in" with a date picker
Select Guest number and Type	Implicit container/ Output facet	The container is fixed	A box with a label "Select Guest number and Type"
Adult Number	Local input/ Output "Adult num"	Data type, Domain value	A label "Check in" with a dropdown list
Room Number	Local input/ Output "Room num"	Data type, Domain value	A label "Check in" with a dropdown list
Children Number	Local input/ Output "Children num"	Data type, Domain value	A label "Check in" with a dropdown list

Table 5. Translation between AIO types into CIO ones of AC2.

Modeling the behavior

The modeling of behavior has been treated until here in terms of rough granularity. That is, the dialogue in a sort of "Big Picture". Now, it is time to pay attention to details. And for this, we used the Abstract Data View notation or ADV-charts [12].

This is a notation for describing the behavior of interactive systems. Also it provides a way to define the flow control and the relationships between UI components and their events. ADV-charts are composed by ADVs, states, attributes and transitions. The representation of ADVs is a rectangle with the name of the ADV (in our case ADV are treated as equivalent to Concrete Containers and CICs). States are depicted as rounded rectangles which contains the name of the state. More than one state is possible per ADV. States could be alone or in a cluster. States are linked to other with transitions (arrows) which are indexed and explained.

In Figure 14a, it is an example of ADV Book a room, in Figure 14b a state, Figure 14c a transition. And finally, Figure 14d represents a concurrent execution of components. It is needed to define each transition after defining ADV-charts. First, we describe its preconditions, then the event that triggers the transition and finally post conditions. Note: For the sake of simplicity, only a sub set of the available transitions are shown in Table 6.

#	Triggered by	Comments
1	Display	Active the application
2	Focus (produced by key-board or mouse click)	-
3	Dynamic recovery of data view	Post conditions {screen = screen + ShowCatego-ries}

Table 6. Some of the transitions to model the behavior of RSC

Now, with modality selected and Behavior more detailed, possible CUI representations are shown in Figure 15 (HTML- based code to have a visual representation) and Figure 16 (with GrafiXML tool [1]) both represents UsiXML CUI code (see Figure 17).

Step 4: Building the Final User Interface

In this step are produced Final User Interfaces (FUI). That is, UIs described by a specific platform (.NET, LZX, SWF and GWT among others). The CUI is translated again with XSLT templates and finally, we have code in the target language. In this way, native code is produced in order to be treated by interpreters, compilers, generators or converters of platform.

We translate CUI specifications into native widget sets available in the chosen platform. One of the advantages of the method is its capability of redirection. That is, CUI models could be oriented to different final platforms. This section shows the CUI definition of RSC into AJAX code (in particular, JQuery [5]) a possible final result is shown in Figure 18.

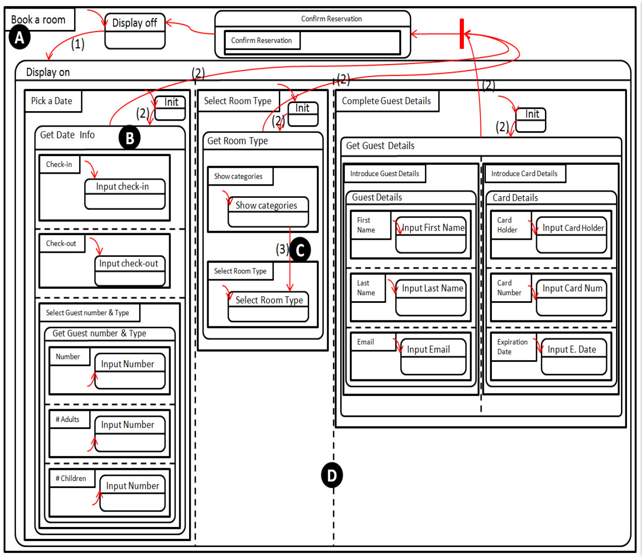


Figure 14. Updated Concrete dialogue of RSC

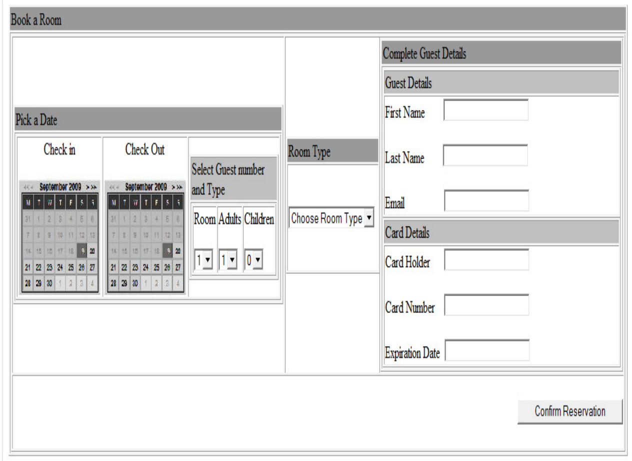


Figure 15. CUI representation of RSC.

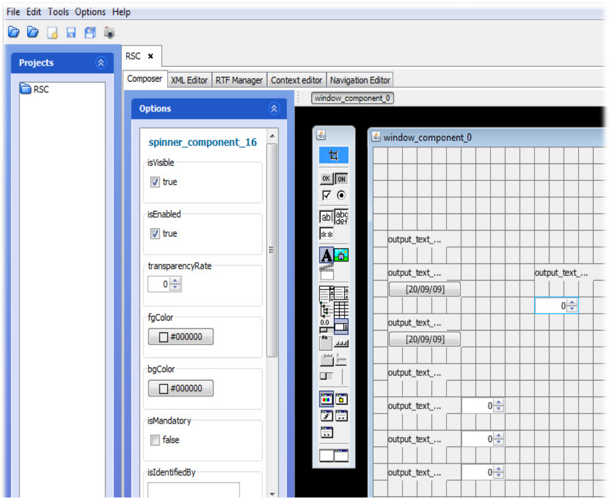
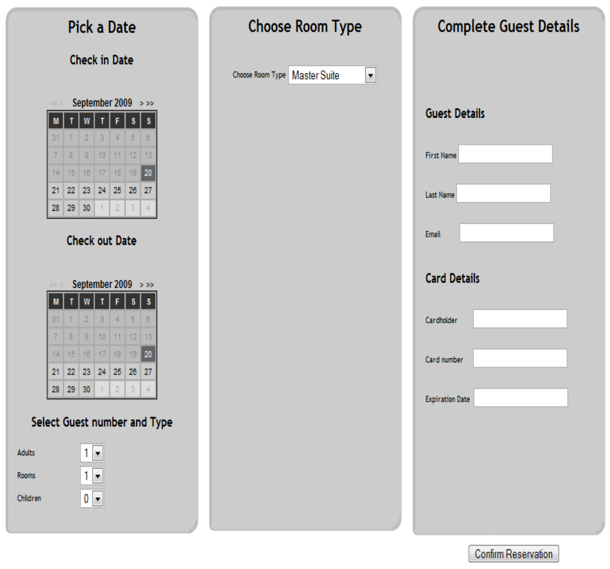


Figure 16. Screenshot of GrafiXML tool for CUI development.

```
<cuiModel id="RSC-cui_20" name="RSC-cui">
   <window id="window_component_0" name="window_component_0"
     width="626" height="498">
     <gridBagBox id="grid_bag_box_1" name="grid_bag_box_1"
       gridHeight="24" gridWidth="31">
       <constraint gridx="1" gridy="5" gridwidth="4"
         gridheight="1" weightx="1.0" weighty="1.0"
         fill="both" insets="0,0,0,0">
         <outputText id="output_text_component_2"
           name="output_text_component_2"
            tooltip="/uiModel/resourceModel/cioRef[@cioId='output_text_component_2']/resource/@tooltip"
            defaultTooltip="Pick a Date"
            content="/uiModel/resourceModel/cioRef[@cioId='output_text_component_2']/resource/@content"
            defaultContent="Pick a Date" isVisible="true"
           isEnabled="true" isBold="true" textColor="#000000"/>
       </constraint>

       ...
```

Figure 17. Fragment of CUI definition of RSC.



Choose Room Type: Master Suite
Adults: 1
Rooms: 1
Children: 0

Figure 18. UI example with JQuery embedded in HTML

CONCLUSION AND DIRECTIONS FOR FUTURE WORK

In this paper, we have presented TRIAD, a MDA method for developing UIs of RIAs, which provides an ensemble of models in order to treat the complexity of RIA design. The first step includes a Zoomable representation of the UI that have been proposed to avoid many of the disadvantages of typical task representation in UsiXML, such as tree node explosion and lack of structure [13]. Piccolo framework [4] is used to implement ZUITs. The design method helps developers to focus in their work instead of dealing prematurely with tasks related to operation and security. This is done by the introduction of the triad concept as integral part of development. Task model has received special attention. Particularly, its container structure[9]. AUI representation has been enriched with the RIA notation, which is proposed to introduce, early in the development, information about RIA-oriented tasks. We used a minimalistic case study to show these concepts and their applicability. Therefore, we just give a glance of the real potential of the method to deal with RIAs. Finally, this approach is not only applicable to RIAs and it could be used for others types of applications.

Our main objective is to establish TRIAD as a plausible alternative method for UI-oriented development of RIA applications. Obviously, this is a first version. TRIAD approach will continue evolving. This process is due to updating current features: Our Repository of triad tasks is being updated with news tasks; Better behavior modeling; Adaptation to mobile technologies; as well as, a better understanding of the quality in the overall process; Better metrics in order to measure the task weights and exploit more semantic of ZUITs. Finally a Development environment is being developed in order to avoid manual process between models as in the current state.

ACKNOWLEDGEMENTS

This work is supported by the Programme AlBan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. (E06D101371MX), The University of Zacatecas (UAZ) and the Belgian Computer Human Interaction Laboratory. The authors also acknowledge the support of the Région Wallonne for ITEA2 Call 3 UsiXML project, under reference 2008026.

REFERENCES

1. UsiXML. http://www.usixml.org/ (May 10th, 2009)
2. Martínez-Ruiz, F.J., Muñoz Arteaga, J., Vanderdonckt, J., González-Calleros, J.M., *A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications*, Proc. of 4th Latin American Web Congress LA-Web'2006 (Puebla Cholula, 25-27 October 2006), J.A. Sanchez (ed.), IEEE Computer Society Press, Los Alamitos, 2006, pp. 32-38.
3. OMG, http://www.omg.org , (May 10th, 2009)
4. http://www.piccolo2d.org (May 10th, 2009).
5. http://jquery.com (May 10th, 2009).
6. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J., *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction*, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), J. Jorge, N.J. Nunes, J. Cunha (eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203-217.
7. Limbourg, Q., Vanderdonckt, J., *UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence*, in Matera, M., Comai, S. (Eds.), "Engineering Advanced Web Applications", Rinton Press, Paramus, 2004, pp. 325-338.
8. Albin, S. T. (2003). The art of software architecture. John Wiley and Sons.
9. Martinez, F., Vanderdonckt, J., Muñoz-Arteaga, J. Web user interface generation for multiple platforms. In *Proc. of 7th Int. Workshop on Web-Oriented Software Technologies IWWOST'2008* (Yorktown Heights, July 14th, 2008), L. Olsina, O. Pastor, D. Schwabe, G. Rossi, M. Winckler (eds.), CEUR Workshop Proceedings, Vol. 445, 2008, pp. 63-68. Accessible at http://ceur-ws.org/Vol-445/02icwe2008ws-iwwost11-martinez-ruiz.pdf
10. Bartalos, Peter & Bieliková, Mária: (S)CRUD pattern support for semantic web applications. Safarik University, Kosice, Slovakia (2008).
11. Montero Simarro, F.; Lopez Jaquero, V., IDEALXML: An Interaction Design Tool. Computer-Aided Design of User Interfaces V. Publisher Springer, 2007, pp.245-252.
12. Carneiro, L. M., Cowan, D. D., and Lucena, C. J. 1994. ADVcharts: a visual formalism for interactive systems. SIGCHI Bull.26, 2 (Apr. 1994), 74-77.

13. Martínez-Ruiz, F., Vanderdonckt, J., and González-Calleros, J.M. Model Driven Engineering of Rich Internet Applications Equipped with Zoomable User Interfaces. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHC'2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 44-51.
14. Johnson, B. and Shneiderman, B. 1991. Tree-Maps: a space-filling approach to the visualization of hierarchical information structures. In Proceedings of the 2nd Conference on Visualization '91 (San Diego, California, October 22 - 25, 1991). G. M. Nielson and L. Rosenblum, Eds. IEEE Visualization. IEEE Computer Society Press.
15. Pederiva, I., Vanderdonckt, J., España, S., Panach, I., and Pastor, O. The Beautification Process in Model-Driven Engineering of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2007* (Rio de Janeiro, September 10-14, 2007), Lecture Notes in Computer Science, Vol. 4662, Springer-Verlag, Berlin, 2007, pp. 409-422.
16. Raskin, J. 2000. The Humane Interface: New Directions for Designing Interactive Systems.Addison-Wesley, Reading, Mass.
17. Lenorovitz, D.R.; Phillips, M.D.; Ardrey, R.S. & Kloster, G.V. "A taxonomic approach to characterizing human-computer interaction". In: G. Salvendy (Ed.), Human-Computer Interaction. Amsterdam: Elsevier Science Publishers, 1984, pp.111-116.
18. Linaje, M., Preciado, J.C., Sánchez-Figueroa, F., 2007. A Method for Model Based Design of Rich Internet Application Interactive User Interfaces. In The Seventh Int. Conf. on Web Engineering (ICWE'07).
19. Nora Koch, Matthias Pigerl, Gefei Zhang and Tatiana Morozova. Patterns for the Model-based Development of RIAs. In Proc. 9th Int. Conf. Web Engineering (ICWE'09), LNCS, volume 5648, pages 283-291. Springer, Berlin, June 2009.
20. Ceri, S., Fraternali, P., Matera, M., and Maurino, A.(2001). Designing Multie-Role, Collaborative Web Sites with WebML: a Conference Management System Case Study. IWWOST'01, Valencia, Spain, June 2001.
21. Matias Urbieta, Gustavo Rossi, Jeronimo Ginzburg, Daniel Schwabe. Designing the Interface of Rich Internet Applications. Proc. 5th Latin American Web Congress (LAWeb' 07), pp.144-153, IEEE, 2007.
22. Valverde F., Pastor, O., Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development. . In *Proc. of 7th Int. Workshop on Web-Oriented Software Technologies IWWOST'2008* (Yorktown Heights, July 14th, 2008), L. Olsina, O. Pastor, D. Schwabe, G. Rossi, M. Winckler (eds.), CEUR Workshop Proceedings, Vol. 445, 2008.