A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces

Thiago Rocha Silva, Jean-Luc Hak, Marco Winckler

A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces

Thiago Rocha Silva* and Jean Luc Hak†
ICS-IRIT, Université Paul Sabatier
Toulouse, France
**rocha@irit.fr*
†jean-luc.hak@irit.fr

Marco Winckler
SPARKS Team, I3S Université Nice Sophia Antipolis, France
winckler@irit.fr

Nowadays many software development frameworks implement Behavior-Driven Development (BDD) as a mean of automating the test of interactive systems under construction. Automated testing helps to simulate user's actions on the User Interface and therefore check if the system behaves properly and in accordance to scenarios that describe functional requirements. However, tools supporting BDD run tests on implemented User Interfaces and are a suitable alternative for assessing functional requirements in later phases of the development process. However, even when BDD tests can be written in early phases of the development process they can hardly be used with specifications of User Interfaces such as prototypes. To address this problem, this paper proposes to raise the abstraction level of both system interactive behaviors and User Interfaces by means of a formal ontology that is aimed at supporting test automation using BDD. The paper presents an ontology and an ontology-based approach for automating the test of functional requirements of interactive systems. We demonstrate the feasibility of this ontology-based approach to assess functional requirements in prototypes and full-fledge applications through an illustrative case study of e-commerce applications for buying flight tickets.

Keywords: Behavior-Driven Development (BDD); automated requirements assessment; ontological modeling; user interfaces; prototyping; testing of interactive systems.

1. Introduction

Assessing interactive systems is an activity that requires a considerable amount of efforts from development teams because it implies to assess systems features with respect to the many possible data and system outputs that might occur when a user is interacting with the system. Conducting this activity manually is a very time consuming and error prone task due to the diversity of user scenarios and the many ways of testing data. Moreover, the system behavior should pass acceptance testing,

which is aimed to determine if the user's point of view about a feature is in accor dance with the requirements previously specified. Thus, the automation of tests for assessing the system behaviors becomes a convenient choice, requiring the use of frameworks to simulate the user's actions when interacting with the system.

In recent years, there is an increasing interest both from academic and industrial communities in Behavior Driven Development (BDD) [1–3] for supporting auto mated acceptance testing of functional requirements. One of the strengths of BDD is to support the specification of requirements in a comprehensive natural language format specification, the so called User Stories [4] that encompass testing Scenarios. With the help of external frameworks, it is possible to automate the test of Scenarios directly on the User Interface (UI). The execution of such executable requirements works as a "live documentation" informing developers about the status of the system with respect to clients' requests set in the acceptance tests.

During the last seven years, we have been involved in the development of web applications where we have observed certain patterns of low level behaviors that are recurrent when writing BDD Scenarios for testing functional requirements with the User Interfaces (UI). Besides that, we could also observe that User Stories specified in natural language often contain semantic inconsistencies. For example, it is not rare to find Scenarios that specify an action such as a selection to be made in semantically inconsistent widgets such as a Text Field. These observations motivated us to in vestigate the use of a formal ontology for describing pre defined behaviors that could be used to specify Scenarios. On one hand, the ontology should act as a taxonomy for terms removing ambiguities in the description. On the other hand, the ontology would operate as a common language that could be used to write tests that can be run on many artefacts used along the development process of interactive systems.

In this paper, we introduce our ontological model for describing interactive behaviors on UIs. The ontology aims to support testing automation of interactive systems specified using a scenario based approach, covering UI concepts in both presentation and dialog aspects. For the presentation layer, we have modeled the semantics of several web and mobile UI elements. For the dialog layer, we have modeled the semantics of User Stories as a State Machine. Such models have allowed us to provide a semantically consistent catalog of interactive behaviors that can be used for automating the test of UIs in different levels of abstraction.

Results of our ontology validation are also presented by demonstration of its correctness through a consistency checking. In addition, we describe an exploratory case study that has been conducted for the flight tickets e commerce domain. In this study, we have used our ontology based tools to support the assessment of evolu tionary prototypes and final UIs. In the following sections, we discuss the founda tions for this work, how we have built the ontological model to support the automated assessment of interactive systems, followed by its validation. We con clude with a discussion and future works.

2. Foundations

2.1. *Computational ontologies and related works*

Computational ontologies [5] come to play as a means to formalize the vocabulary and the concepts used in User Stories, Scenarios and user's behaviors. Without a common agreement on the concepts and terms used it would be difficult to support the assessment of user requirements. Some approaches have tried to define languages or at least a common vocabulary for specifying UIs for interactive systems. Useful abstractions for describing interactive systems include the components that compose the presentation of a User Interface and the dialog parts that describe the system behavior.

The Camaleon Framework [6] treats the presentation and the dialog in three levels of abstractions: Abstract, Concrete and Final User Interfaces. The idea is that as abstract user interface component (such as a Container) could be refined to a more concrete representation (such as a Window) that will ultimately feature a final im plementation in a target platform (e.g. MacOS or Windows). User Interface (UI) specifications include more or less details according to the level of abstraction as shown in Fig. 1. The UsiXML (USer Interface eXtensible Markup Language) [7] implements the principles of the Cameleon framework in a XML compliant markup language featuring many dialects for treating Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. UsiXML is a declarative language that captures the essence of User Interface components. At the highest level of abstraction, UsiXML describes concepts of widgets, controls, containers, modalities and interaction techniques. UsiXML contain a few basic ele ments for describing the dialog part such as the concept of events, conditions and actions. For that, some authors have proposed to use a notation based on statecharts called SWC (StateWebCharts) [9] to specify the UsiXML dialog. The same authors [8] have demonstrated that when using SWC, it is possible to describe the system behavior at different levels of abstraction using UsiXML.

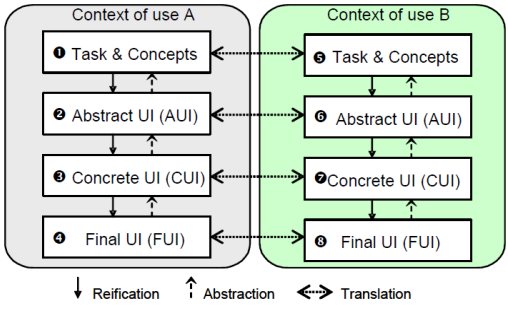

Fig. 1. The Cameleon reference framework (from [7]).

As far as a common vocabulary is concerned, the W3C published a glossary of recurrent terms for presentation components called MBUI (Model based User

Interface) [10]. For the dialog component, SWC [9] and SXCML (State Chart XML: State Machine Notation for Control Abstraction) [11] offer a language based on the State Machine concepts.

2.2. *User Stories*

User Stories in Software Engineering was first proposed by Cohn [4] as a mean to formalize artifacts for describing system' features and their corresponding acceptance criteria. User Stories are formatted to fulfill two main goals: (i) assure testability and non ambiguous descriptions and (ii) provide reuse of business scenarios. User Stories express concrete examples of what should be tested to consider these features as "done". Below we present a template proposed by North [12] and Cohn [4]:

```
Title (one line describing the story)
Narrative:
As a [role]
I want [feature]
So that [benefit]
Acceptance Criteria: (presented as Scenarios)
Scenario 1: Title
Given [context]
 And [some more context]...
When  [event]
Then  [outcome]
 And [another outcome]...
Scenario 2: ...
```

A User Story contains a Title, a Narrative and a set of Scenarios representing the Acceptance Criteria. The Title provides a general description of the story, referring to a feature that this story represents. The Narrative describes the role (played by a user), the feature itself, and the benefits it will bring to the business and/or to the role. The Acceptance Criteria are defined through a set of Scenarios defined with a Title and three main clauses: "Given" provides the context, "When" describe events that trigger the Scenario and "Then" shows the expected outcomes (that should be checked). Each clause can include an "And" statement. Each statement in this representation is called Step.

In Behavior Driven Development (BDD) [1], the user's point of view about the system is captured by User Stories. The BDD approach assumes that clients and teams can communicate using this semi structured natural language description, in a non ambiguous way. Following this assumption, we have defined a conceptual model to represent users' functional requirements. A functional requirement defines state ments of services that the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. Requirements should be expressed in a way they can be reused to assess the system's behavior.

Figure 2 presents the conceptual model of our approach. Requirements are expressed as a set of User Stories (US) encompassing a Narrative and Acceptance

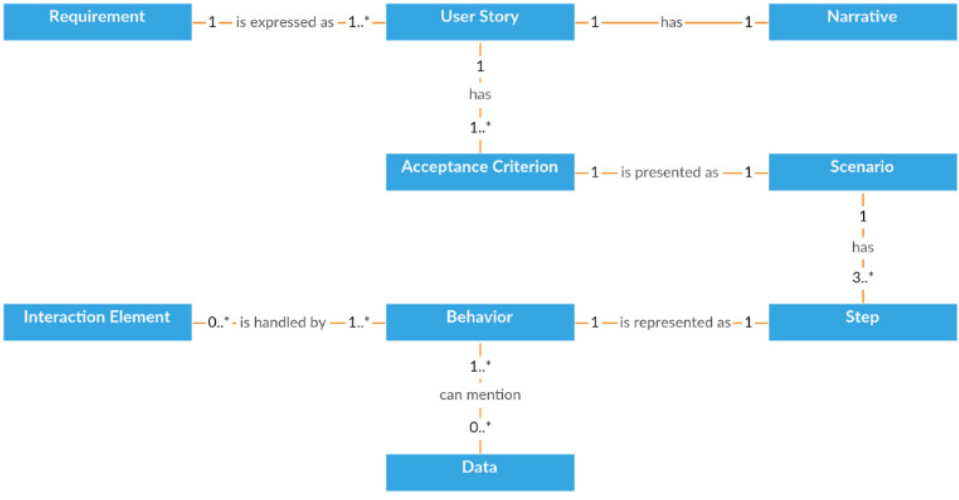

Fig. 2. Conceptual model of user requirements.

Criteria. Acceptance Criteria are presented as Scenarios composed by at least three main Steps ("Given", "When" and "Then") that represent the expected system' Behaviors. Behaviors handle actions on Interaction Elements in the User Interface (UI) and include data using in the test. These concepts and rules are defined as classes and axioms in the proposed ontology presented hereafter.

3. Ontology Modeling

Our ontology for describing interactive systems is based on concepts borrowed from different languages found in the literature. From Camaleon [6] and UsiXML [7] we borrow concepts of abstract and concrete UIs. Presentation and definition of graphical components come from W3C MBUI [10]. From W3C Web Ontology Language we get concepts for graphical components (behavior and presentation aspects) commonly used to build web and mobile applications, and also the textual representations used to describe how users interact with those graphical components. SWC [8] inspire concepts used in the dialog.

The ontology has been modeled in Protégé 5.0. Figure 3 presents the classes of the ontology and their properties divided in 4 wide groups: Platform Concepts, UI Concepts, State Machine Concepts and Scenario based Concepts. The first group defines the web and mobile platforms covered by the ontology. The second one encompasses concepts allowing modeling the UI. The classes Dialog, Presentation and Platform model the concept of a Prototype. A Prototype is built for at least one Platform and is specified by no more than one Dialog and one Presentation. The third group specifies the State Machine concepts. A Dialog is described as a

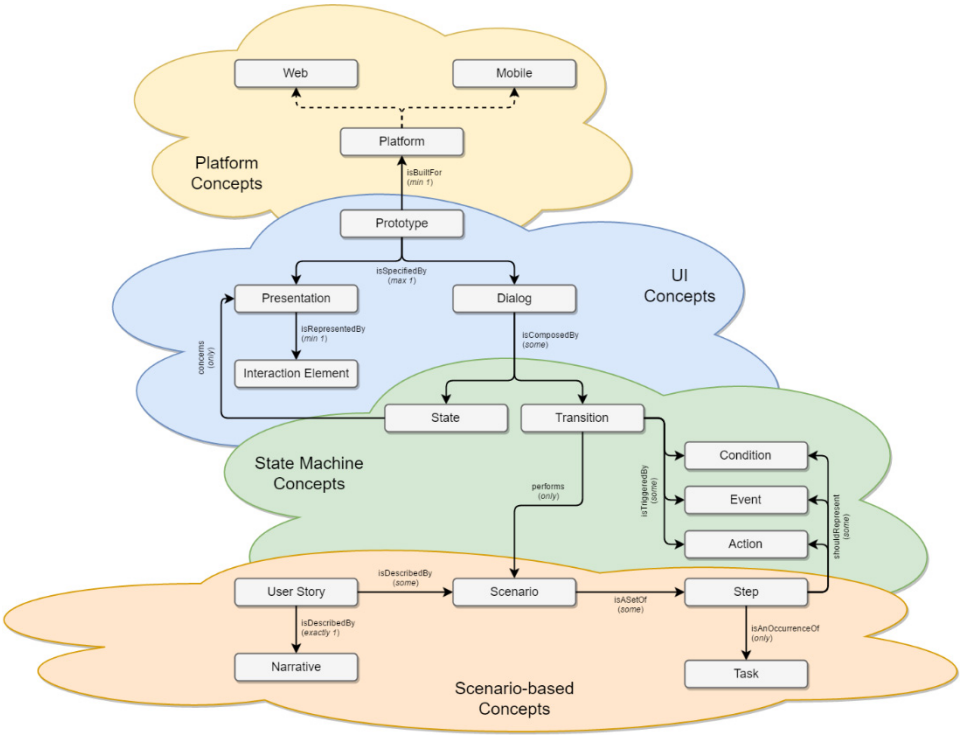

Fig. 3. Main classes and their properties in the ontology.

State Machine while a Presentation is composed by Interaction Elements. Likewise, in the fourth group, the classes Narrative, Scenario, Step and Task model the concept of a User Story. A User Story is described by exactly one Narrative and some Sce narios. A Scenario is an occurrence of only one Task and is a set of Steps. A Step shall represent some Event, Condition and/or Action that are Transition elements from the State Machine, performing the Dialog component of a Prototype.

Concepts have been modeled as Classes. Relationships between concepts have been modeled as Object Properties (subtype "relations"). Classes that handle data have such descriptions modeled as Data Properties. As core elements in the ontology, UI Elements and the interactive behaviors are respectively as Classes and Object Properties (subtype "behaviors").

In the following subsections, we detail the basic concepts of Object (Sec. 3.1) and Data Properties (Sec. 3.2), as well as the four main group of concepts described above: Platform (Sec. 3.3), UI (Sec. 3.4), State Machine (Sec. 3.5), and finally Scenario based concepts (Sec. 3.6). The current version of the ontology bears an amount of 422 axioms (being 277 logical axioms), 56 classes, 33 object properties, 17 data properties and 3 individuals. A visual representation of all the concepts

can be found at https://goo.gl/IZqSJ0 and its complete specification in OWL can be found at https://goo.gl/1pUMqp.

3.1. *Object Properties*

Relationships between individuals in classes are represented as Object Properties. We have classified those properties in "Relations" and "Behaviors". "Relations" groups conceptual relationships between objects from internal classes, i.e. objects that do not directly address interactive behaviors. "Behaviors" on the other hand groups conceptual relationships between interactive behaviors and UI Elements on the UI. The "Relations" group is detailed hereafter and the "Behaviors" groups will be detailed in the Sec. 3.6.

3.1.1. *Relations*

The sub property "relations" defines the semantic correspondence between internal classes. Table 1 presents the whole set of relationships between objects of internal classes defined in the ontology. The class that drives the property is called Domain Class and the class affected by the property is called Range Class. The Restriction Type adds constraints to the modeled property. Figure 4 illustrates the relations between elements in the State Machine. As a sub property of Relations, objects from the Dialog class are composed by some States and Transitions. This relationship is described by the property *isComposedBy* (left side of Fig. 4). Accordingly, objects from the Transition class are triggered by a sequence of some Conditions, Events and Actions. This relationship is described by the property *isTriggeredBy* (right side of Fig. 4).

Table 1. "Relations" as object properties in the ontology.

Domain class	Object property	Restriction type	Range class
State	*concerns*	only	Presentation
Step	*isAnOccurrenceOf*	only	Task
Scenario	*isASetOf*	only	Step
Prototype	*isBuiltFor*	min 1	Platform
Dialog	*isComposedBy*	some	State
	isComposedBy	some	Transition
User Story	*isDescribedBy*	exactly 1	Narrative
	isDescribedBy	some	Scenario
Presentation	*isRepresentedBy*	min 1	Interaction Element
Prototype	*isSpecifiedBy*	max 1	Dialog
	isSpecifiedBy	max 1	Presentation
Transition	*isTriggeredBy*	some	Event
	isTriggeredBy	some	Condition
	isTriggeredBy	some	Action
Transition	*performs*	only	Scenario
Step	*shoudRepresent*	some	Event
	shoudRepresent	some	Condition
	shoudRepresent	some	Action

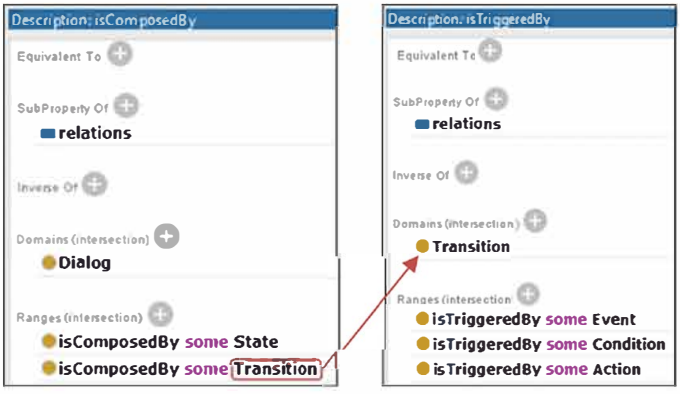

Fig. 4. Object properties isComposedBy (*left*) and isTriggeredBy (*right*).

3.2. *Data Properties*

Data Properties are used to describe semantically data domains used by each class that handles data. The root tree shown in Fig. 5(a) gives an overview of the properties created, while Fig. 5(b) expands the Data Property "message", showing that this kind of data is used by the UI Elements "Message Box", "Notification", "Tool Tip" and "Modal Window". "Message" has also been defined to range the primitive data String. Table 2 shows the whole set of Data Properties created, their respective Domain Classes as well as their Datatypes. As some UI Elements can handle another UI Elements or even different Datatypes, we have defined the generic type "element" for modeling this property. For example, Menus present options for users, but these options can be of any type, i.e. images, text, or even another UI Element such as a

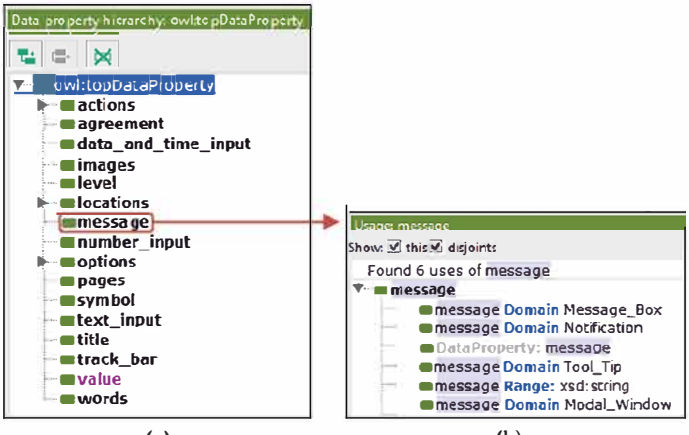

Fig. 5. (a) *Left*: Data properties; (b) *Right*: Data property "message".

Table 2. Data properties in the ontology.

Data property	Domain classes	Datatype
Actions	Menu Item, Link, Message Box, Button, Modal Window	element
State		xsd:boolean
Agreement	Notification	xsd:string
Data and Time Input	Calendar	xsd:dataTime
Images	Image Carousel	xsd:hexBinary
Level	Prototype	
Locations	Breadcrumb	xsd:string
State		xsd:boolean
Message	Message Box, Notification, Text, Tool Tip, Modal Window	xsd:string
Number Input	Numeric Stepper	xsd:double
Options	Tabs Bar, Checkbox, Dropdown List, Toggle, List Box, Radio Button, Accordion, Menu, Progress Bar, Dropdown Button	element
State		xsd:boolean
Pages	Pagination	xsd:integer
Symbol	Icon	xsd:hexBinary
Text Input	Search Field, Text Field, Autocomplete	xsd:string
Title	Button, Field Set, Link, Label, Menu Item	xsd:string
Value	Slider	xsd:double xsd:string
Words	Tag	xsd:string

Menu Item. The other Datatypes come from the standard XSD specification. Finally, notice that the only Data Property that does not use a Datatype is the property "Level", which refers to the level of a Prototype.

3.3. *Platform concepts*

Concepts of the platform are modeled in the ontology to determine which kind of UI is supported by the model. So far, the ontology supports only interactive behaviors for web and mobile UIs. As a consequence of such choice, only UI Elements that are supported by web and mobile environments have been described in the superclass Interaction Elements. The set of UI Elements that suits each platform is presented as Object Properties in Sec. 3.4. Finally, the classes Web and Mobile have been modeled as specializations of the class Platform, which allows us to eventually cover other platforms in the future.

3.4. *UI elements concepts*

UI Elements in the ontology represent an abstraction of GUI components in web and mobile platforms. Figure 6 illustrates a hierarchy of UI Elements.

As we shall see in Fig. 6, the four main superclasses are Container, Information Component, Input Control and Navigational Component. The first one contains ele ments that group other elements in a User Interface, such as Windows and Field Sets.

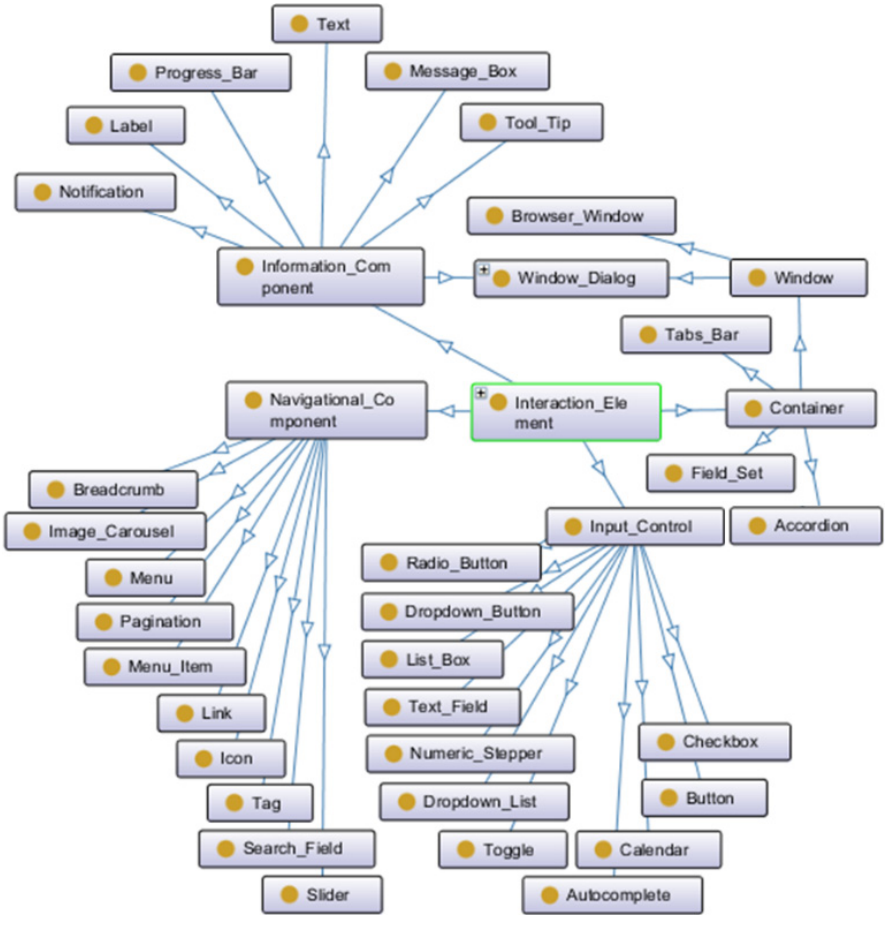

Fig. 6. Graph describing the hierarchy of user interface (UI) elements.

The second one contains elements in charge of displaying information to the users such as Labels and Message Boxes. The third one represents elements that accept users inputs such as Buttons and Text Fields. Finally, the last one contains elements useful to navigate through the system such as Links and Menus. Some elements like Dialog Windows, for example, are inherited by more than one superclass, once they keep semantic characteristics of Containers and Information Components as well.

The complete list of UI Elements modeled in the ontology is presented in Table 3, specifying for each one the correspondent superclass, a brief description and both Data and Object Properties associated. In Data Properties (DP) is identified the type of data handled by the UI Element as well as the Object Properties (OP) describing, for Interaction Elements, whether they are supported by web and/or mobile platforms.

Table 3. UI elements in the ontology.

	Int. element	Description	Properties
Container	Accordion	An Accordion is a vertically stacked list of items that utilizes show/hide functionality. When a label is clicked, it expands the section showing the content within. There can have one or more items showing at a time and may have default states that reveal one or more sections without the user clicking.	**DP**: options **OP**: Web, Mobile
	Field Set	A Field Set element represents a set of form controls optionally grouped under a common name.	**DP**: title **OP**: Web, Mobile
	Tabs Bar	A Tab Bar is a container widget that has typically multiple Tab Bar Buttons, which controls visibility of views. It can be used as a tab container.	**DP**: options **OP**: Web, Mobile
	Window	A Window is an area on the screen that displays information, with its contents being displayed independently from the rest of the screen.	—
Window	Browser Window	The top of a typical Web browser window contains a title bar that displays the title of the current page. Below the title is a toolbar with back and forward buttons, an address field, bookmarks, and other navigation buttons. Below the toolbar is the content of the current Web page. The bottom of the window may contain a status bar that displays the page loading status.	**OP**: Web
	Window Dialog	A Window or Dialog Box is a small window that communicates information to the user and prompts them for a response.	**OP**: Web
Window Dialog	Modal Window	A Modal Window requires users to interact with it in some way before they can return to the system.	**DP**: actions, message **OP**: Web
Information Component	Label	A Label displays content classification.	**DP**: title **OP**: Web, Mobile
	Message Box	A Message Box is a small window that provides information to users and requires them to take an action before they can move forward.	**DP**: actions, message **OP**: Web, Mobile
	Notification	A Notification is an update message that announces something new for the user to see. Notifications are typically used to indicate items such as, the successful completion of a task, or an error or warning message.	**DP**: agreement, message **OP**: Web, Mobile

Table 3. (*Continued*)

	Int. element	Description	Properties
	Progress Bar	A Progress Bar indicates where a user is as they advance through a series of steps in a process. Typically, progress bars are not clickable.	**DP**: options **OP**: Web, Mobile
	Text	Informative content in a page.	**DP**: message **OP**: Web, Mobile
	Tool Tip	A Tooltip allows a user to see hints when they hover over an item indicating the name or purpose of the item.	**DP**: message **OP**: Web, Mobile
	Window Dialog	—	—
Input Control	Autocomplete	The Autocomplete widgets provides suggestions while you type into the field.	**DP**: text_input **OP**: Web
	Button	A Button indicates an action upon touch and is typically labeled using text, an icon, or both.	**DP**: actions, title **OP**: Web, Mobile
	Calendar	A Calendar (date picker) allows users to select a date and/or time. By using the picker, the information is consistently formatted and input into the system.	**DP**: data_and_time_input **OP**: Web, Mobile
	Checkbox	Checkboxes allow the user to select one or more options from a set. It is usually best to present checkboxes in a vertical list. More than one column is acceptable as well if the list is long enough that it might require scrolling or if comparison of terms might be necessary.	**DP**: options **OP**: Web, Mobile, allowsMultiple
	Dropdown Button	The Dropdown Button consists of a button that when clicked displays a drop-down list of mutually exclusive items.	**DP**: options **OP**: Web, Mobile, allowsUnique
	Dropdown List	Dropdown Lists allow users to select one item at a time, similarly to radio buttons, but are more compact allowing you to save space. Consider adding text to the field, such as 'Select one' to help the user recognize the necessary action.	**DP**: options **OP**: Web, Mobile, allowsUnique
	List Box	List Boxes, like Checkboxes, allow users to select a multiple items at a time, but are more compact and can support a longer list of options if needed.	**DP**: options **OP**: Web, Mobile, allowsMultiple

Table 3. (*Continued*)

	Int. element	Description	Properties
	Numeric Stepper	A Numeric Stepper serves the same function as a Numeric Input Object. It is a method of entering numeric data in which the numbers can be typed directly into the input object. However, numeric values can also be adjusted by using up and down arrows next to the numeric input. Clicking the up and down arrows normally causes the value to increment by one.	**DP**: number_input **OP**: Web, Mobile
	Radio Button	Radio Buttons are used to allow users to select one item at a time.	**DP**: options **OP**: Web, Mobile, allowsUnique
	Text Field	Text Fields allow users to enter text. It can allow either a single line or multiple lines of text.	**DP**: text_input **OP**: Web, Mobile
	Toggle	A Toggle button allows the user to change a setting between two states. They are most effective when the on/off states are visually distinct.	**DP**: options **OP**: Web, Mobile, allowsUnique
	Grid	A Grid or a Datagrid is a graphical control element that presents a tabular view of data.	**DP**: text_input **OP**: Web, Mobile
Navigational Component	Breadcrumb	Breadcrumbs allow users to identify their current location within the system by providing a clickable trail of proceeding pages to navigate.	**DP**: locations **OP**: Web
	Icon	An Icon is a simplified image serving as an intuitive symbol that is used to help users to navigate the system. Typically, icons are hyperlinked.	**DP**: symbol **OP**: Web, Mobile
	Image Carousel	Image Carousels allow users to browse through a set of items and make a selection of one if they so choose. Typically, the images are hyperlinked.	**DP**: images **OP**: Web
	Link	A Link is a reference to data that can be directly follow by clicking. It points to a whole document or to a specific element within a document.	**DP**: actions, title **OP**: Web
	Menu	Menu is a list of options or commands presented to an operator.	**DP**: options **OP**: Web, Mobile

Table 3. (*Continued*)

Int. element	Description	Properties
Menu Item	A Menu Item is a resultant item in a list of options or commands presented to an operator by clicking in a menu.	**DP**: actions, title **OP**: Web, Mobile
Pagination	Pagination divides content up between pages, and allows users to skip between pages or go in order through the content.	**DP**: pages **OP**: Web
Search Field	A search box allows users to enter a keyword or phrase (query) and submit it to search the index with the intention of getting back the most relevant results. Typically, search fields are single-line text boxes and are often accompanied by a search button.	**DP**: text_input **OP**: Web, Mobile
Slider	A slider, also known as a track bar, allows users to set or adjust a value. When the user changes the value, it does not change the format of the interface or other info on the screen.	**DP**: value **OP**: Web, Mobile
Tag	Tags allow users to find content in the same category. Some tagging systems also allow users to apply their own tags to content by entering them into the system.	**DP**: words **OP**: Web
Tree	With a Tree, we can display hierarchical data. Each row displayed by the Tree contains exactly one item of data, which is called a node. Every Tree has a root node from which all nodes descend. By default, the Tree displays the root node. A node can either have children or not. We refer to nodes that can have children — whether or not they currently have children — as branch nodes. Nodes that cannot have children are leaf nodes.	**DP**: actions **OP**: Web

3.5. *State machine concepts*

The dialog part of a User Interface, as illustrated by Fig. 7, is described in the ontology using concepts borrowed from abstract State Machines. A Scenario meant to be run in a given UI is represented as a Transition, illustrated by Fig. 8. States are used to represent the original and resulting UIs after a transition occur (States A and B in Fig. 8). Scenarios in the Transition state always have at least one or more Conditions (represented in Scenarios by the "Given" clause), one or more Events (represented in Scenarios by the "When" clause), and one or more Actions (repre sented in Scenarios by the "Then" clause). The clauses "Given", "When" and "Then" have been modeled as Individuals of each respective class.

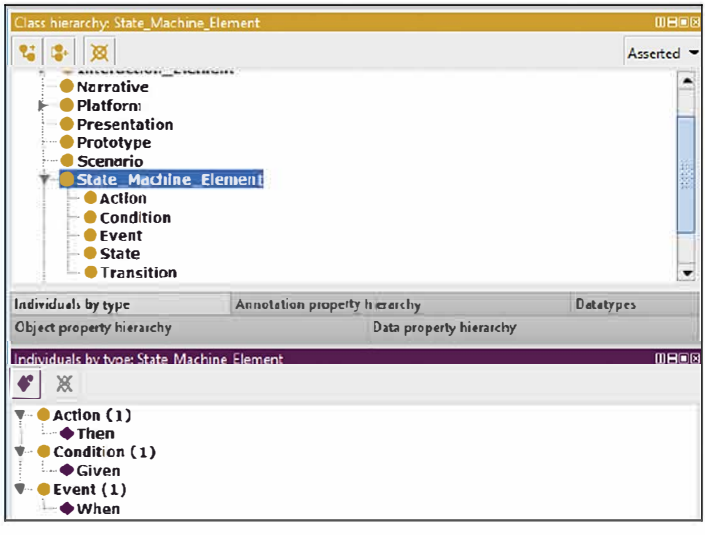

Fig. 7. State machine elements and their individuals.

Fig. 8. A transition being represented in the state machine.

3.6. *Scenario-based concepts*

Scenario based concepts allow us to model behaviors that describe how users are supposed to interact with graphical elements of the User Interface. An example of behavior specification is illustrated by Fig. 9.

Fig. 9. Behavior "*chooseRefferingTo*".

Behaviors are structured and described in natural language so that they can also be read by humans. The specification of behaviors encompasses when the interaction can be per formed (using Given, When and/or Then clauses) and graphical elements (i.e. Radio Button, CheckBox, Calendar, Link, etc.). Altogether, behaviors and graphical elements are used to implement the test of expected system behavior. In the example below, the behavior receives two parameters: a "$elementName" and a "$locatorParameters". The first parameter is associated to data, the second parameter refers to the Interaction Element supported by this behavior: "Radio Button", "CheckBox", "Calendar" and "Link". To comply with semantic rules, the behavior "*I chose* \ "£*elementName* \ "*referring to* \ "£*locatorParameters* \ "" shown in Fig. 9 can be modelled into a predefined behavior "*chooseReferringTo*" as shown in Fig. 10.

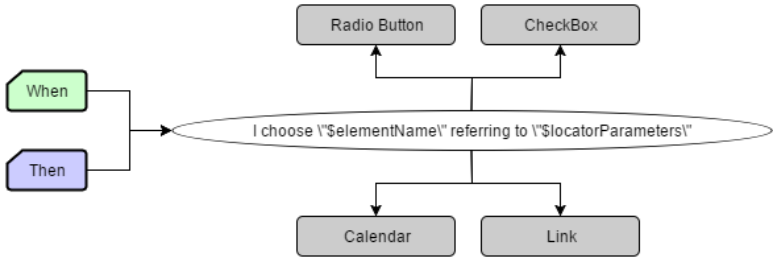

Fig. 10. Components on the ontology used to specify a behavior.

The ontology includes a large set of predefined behaviors grouped by context of use, as shown in Table 4. Notice that each Behavior is associated to diverse transition components (**C**ontext, **E**vent and/or **A**ction) that compose a Transition. The column UI Elements enlists the set of Interaction Elements that can fit to trigger a particular behavior.

Table 4. Predefined behaviors described in the ontology.

Checkbox and Radio Button Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
theFieldIsUnchecked	■	■	■	Checkbox Radio Button
theFieldIsChecked	■	■	■	Checkbox Radio Button
assureTheFieldIsUnchecked		■		Checkbox
assureTheFieldIsChecked		■		Checkbox
Common Behaviors				
Behavior	**Transition**			**UI Elements**
	C	**E**	**A**	
choose		■	■	Calendar Checkbox Radio Button Link
chooseByIndexInTheField		■	■	Dropdown List
chooseReferringTo		■	■	Calendar Checkbox Radio Button Link
chooseTheOptionOfValueInTheField		■	■	Dropdown List
clickOn		■	■	Menu
		■	■	Menu Item Button Link
clickOnReferringTo		■	■	Menu Menu Item Button Link Grid
doNotTypeAnyValueToTheField ≡ *resetTheValueOfTheField*		■		Text Field
goTo	■	■	■	Browser Window
goToWithTheParameters	■	■	■	Browser Window
isDisplayed	■	■	■	Window
setInTheField ≡ *tryToSetInTheField*		■	■	Dropdown List Text Field Autocomplete Calendar
setInTheFieldReferringTo		■	■	Dropdown List Text Field
typeAndChooseInTheField		■	■	Autocomplete
willBeDisplayed			■	Text
willNotBeDisplayed			■	Text
willBeDisplayedInTheFieldTheValue			■	Element
willNotBeDisplayedInTheFieldTheValue			■	Element
willBeDisplayedTheValueInTheFieldReferringTo			■	Element
willNotBeDisplayedTheValueInTheFieldReferringTo			■	Element
isNotVisible	■	■	■	Element
valueReferringToIsNotVisible	■	■	■	Element
waitTheFieldBeVisibleClickableAndEnable	■	■	■	Element
waitTheFieldReferringToBeVisibleClickableAndEnable	■	■	■	Element
theElementIsVisibleAndDisable	■	■	■	Element
theElementReferringToIsVisibleAndDisable	■	■	■	Element
setInTheFieldAndTriggerTheEvent		■	■	Text Field
clickInTheRowOfTheTree	■	■	■	Tree

Table 4. (*Continued*)

Data Generation Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
informARandomNumberWithPrefixInTheField	■	■	■	Text Field
informARandomNumberInTheField		■		Text Field
Data Provider Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
inform	■	■		Grid
informTheField ≡ informTheFields		■		Grid
selectFromDataSet	■	■		-
informTheValueOfTheField		■	■	Element
informKeyWithTheValue ≡ defineTheVariableWithTheValue	■	■	■	-
obtainTheValueFromTheField	■	■	■	Element
Debug Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
printOnTheConsoleTheValueOfTheVariable		■	■	-
Dialog Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
confirmTheDialogBox	■	■	■	Window Dialog
cancelTheDialogBox	■	■	■	Window Dialog
informTheValueInTheDialogBox	■	■	■	Window Dialog
willBeDisplayedInTheDialogBox		■	■	Window Dialog
Mouse Control Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
moveTheMouseOver		■		Menu Menu Item Button Link
Table Behaviors				
Behavior	**Transition**			**UI Elements**
	C	E	A	
clickOnTheRowOfTheTableReferringTo		■	■	Grid
storeTheCellOfTheTableIn		■	■	Grid
storeTheColumnOfTheTableIn		■	■	Grid
compareTheTextOfTheTableCellWith		■	■	Grid
compareTheTextOfTheTableColumnWith		■	■	Grid
clickOnTheCellOfTheTable		■	■	Grid
clickOnTheColumnOfTheTable		■	■	Grid
chooseTheOptionInTheCellOfTheTable		■	■	Grid
chooseTheOptionInTheColumnOfTheTable		■	■	Grid
typeTheTextInTheCellOfTheTable		■	■	Grid
typeTheTextInTheColumnOfTheTable		■	■	Grid

The vocabulary chosen to express each behavior emerged from Scenarios specified in our past projects. It outlines only one of the several possible vocabularies to represent the same user's behaviors, and could be extended in the future by more representative phrases or expressions. Some synonyms concerning the user's goal have been also identified in order to increase the expressivity of the ontology. For example, the behavior *doNotTypeAnyValueToTheField* is considered equivalent

to the behavior *resetTheValueOfTheField* as they perform or assert exactly the same action on the affected UI element, looking for the same output. Likewise, the behavior *setInTheField* is equivalent to the behavior *tryToSetInTheField* as they refer to the same action. However, *tryToSetInTheField* better expresses violation attempts in the business rules.

4. Validation

The ontology has been validated in two steps: at first, consistency has been con tinuously checked through the use of reasoners. Then, using a tool support, we applied the approach to a case study in the flight tickets e commerce domain using a set of tools we have developed for dealing with tests over Prototypes and for testing the implementation.

4.1. *Consistency checking*

Consistency checking was done using the reasoners FaCT++, ELK, HermiT and Pellet. FaCT++ started identifying no support for the datatypes xsd:base64Binary and xsd:hexBinary used to range images and symbols in the Data Properties. Those properties have been used to define domains for objects in the classes Image Carousel and Icon, respectively. ELK has failed by no support to Data Property Domains as well as Data and Object Property Ranges. HermiT and Pellet have succeeded pro cessing the ontology respectively in 4926 and 64 milliseconds, as presented in Fig. 11.

```
INFO  13:00:09  ------------------------------ Running Reasoner ------------------------------
INFO  13:00:09  Pre-computing inferences:
INFO  13:00:09      - class hierarchy
INFO  13:00:09      - object property hierarchy
INFO  13:00:09      - data property hierarchy
INFO  13:00:09      - class assertions
INFO  13:00:09      - object property assertions
INFO  13:00:09      - data property assertions
INFO  13:00:09      - same individuals
INFO  13:00:14  Ontologies processed in 4926 ms by null
INFO  13:00:14
INFO  13:01:01  ------------------------------ Running Reasoner ------------------------------
INFO  13:01:01  Pre-computing inferences:
INFO  13:01:01      - class hierarchy
INFO  13:01:01      - object property hierarchy
INFO  13:01:01      - data property hierarchy
INFO  13:01:01      - class assertions
INFO  13:01:01      - object property assertions
INFO  13:01:01      - data property assertions
INFO  13:01:01      - same individuals
INFO  13:01:01  Ontologies processed in 64 ms by Pellet
INFO  13:01:01
```

Fig. 11. Results of ontology processing: HermiT (top) and Pellet (bottom).

4.2. *Validation by a case study*

To illustrate how the ontology can be used to support the specification of require ments and the testing automation for interactive systems, we have chosen a flight tickets e commerce application. Below we describe one of the User Stories from this case study with a Scenario for searching flights. Therein, the user should provide at least: the type of sought ticket (one way or round trip), the departure and the arrival airports, the number of passengers, and finally the dates. In the Scenario "One Way Tickets Search", a typical search of tickets is presented concerning a one way trip from Paris to Dallas for 2 passengers on 12/15/2016. According to the business rule, the expected result for this search is a new screen presenting the title "Choose Flights", in which the user might select the desired flight from a list of flights matching his search.

```
User Story: Flight Tickets Search

Narrative:
As a frequent traveler
I want to be able to search tickets, providing locations and dates
So that I can obtain information about rates and times of the flights.

Scenario: One-Way Tickets Search
Given I go to "Find flights"
When I choose "One way"
And I type "Paris" and choose "CDG - Paris Ch De Gaulle, France" in the field "From"
And I type "Dallas" and choose "DFW - Dallas Fort Worth International, TX" in the field
"To"
And I choose the option of value "2" in the field "Number of passengers"
And I choose "12/15/2016" referring to "Depart"
And I click on "Search"
Then will be displayed "Choose Flights"
```

4.2.1. *Ontology support for testing prototypes using PANDA*

PANDA (Prototyping using Annotation and Decision Analysis) [13] is a tool support specifically created to support the development of UI prototypes built upon an UI ontology. Using our ontology, PANDA can also support the test of BDD Scenarios. For that, PANDA starts by reading an OWL file describing our ontology. Using the inner organization of ontological classes, PANDA dynamically instantiates a palette of widgets (see Fig. 12) that can be used to build a Prototype. From an interaction point of view, the construction of Prototypes is done by performing drag and drop operations. From a storage point of view, a Prototype is an XML file that describes a composition of widgets whose description is semantically annotated by elements of our ontology.

For the construction of the palette, PANDA uses a description of a widget we called "OntologicalClass" which feature its name, list of subclasses and set of properties. This ontological class has been defined as a generic class that is cus tomized through its properties. Indeed, those classes represent each component of a

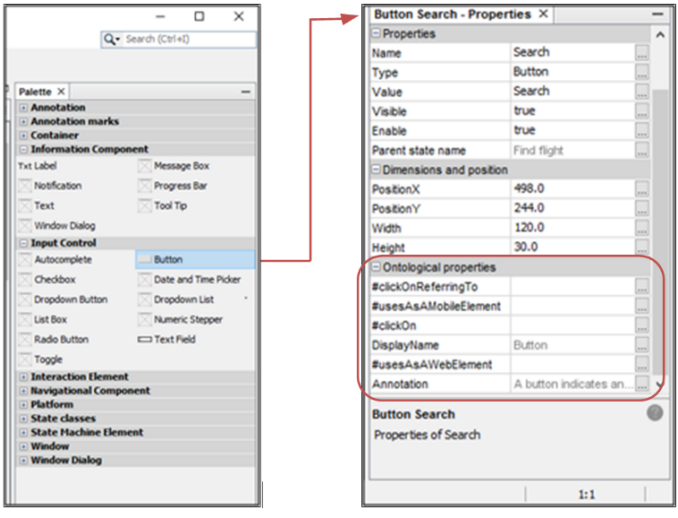

Fig. 12. Pallets with the widget button and its properties extracted from the ontology.

Prototype in PANDA and its behaviors regarding their usage in the prototyping tool: they are placed in an edition area in which the user can edit the instance of a property. Thus, for the Presentation component, PANDA uses a flexible structure that allows to dynamically instantiate the set of widgets that will be used to build Prototypes.

PANDA creates a category for each superclass including: Container, Information Component, Input Control, Interaction Element, Navigational Component, Plat form, State Machine Element, Window and Window Dialog. Each category contains a set of widgets defined by the classes inheriting the superclass. As for the properties, ontological classes are displayed in the property window in the category "Ontological properties". Each property identified in the ontology is therefore inserted in the list of properties of the class with a name and a value.

For the Dialog component, our ontology encompasses behavioral properties to describe the interaction supported by a class. For example, a Button must feature a behavioral property "clickOn" which indicates that buttons support an event click. Click events allow the designer to specify interactions on widgets. If a button has a behavioral property "clickOn", PANDA adds an event handler to handle click events when users interact with the Prototype. Figure 13 shows how Scenarios are tested in PANDA. For each Step of Scenarios, PANDA assesses actions with respect to widget properties defined in the ontology. For example, in the Step "And I click on "Search"", PANDA looks for any widget named "Search" in the initial State, and check if the description of the widget in the ontology support the behavior "clickOn". The results of the tests are displayed by a colored symbol next to each Step, a red "X" representing failure, a green "V" representing success, and a black "?" representing an untested Step.

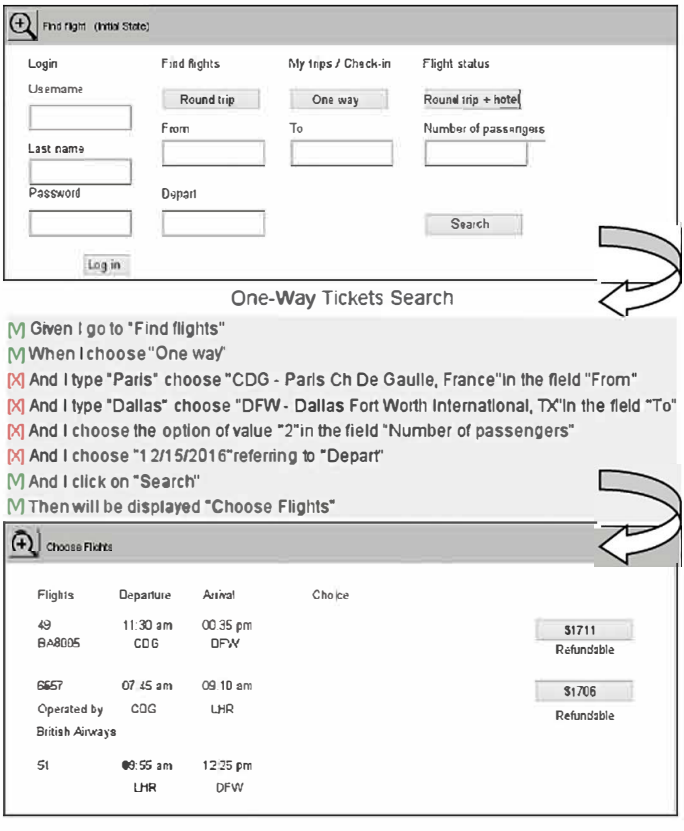

Fig. 13. A state machine transition between sketches of a PANDA prototype for the user story "Flight Tickets Search". From top to bottom: The initial State "Find Flights", a Transition represented by the Scenario "One-Way Tickets Search", and finally the resultant State "Choose Flights".

4.2.2. *Ontology support for testing web final UIs*

To test the Scenarios over Web Final UIs, we have employed a set of frameworks to provide automated simulation of user's interaction. More specifically, we have used Selenium WebDriver to run navigational behavior as well as JBehave and Demoiselle Behave to parse Scenario scripts. The ontology is charged as a CommonSteps Java Class, pre defining behaviors that can be used when writing Scenarios, and where each action and/or assert for each behavior is defined. This class implements the dialog component and contains all the knowledge about how performing the men tioned behaviors on the UI elements, thus when using them to write Scenarios, tests are delivered without any additional effort of implementation. Hence, methods in this class have been writ ten for every Step addressed on the ontology. As illustrated in Fig. 14, behaviors "*When/Then I choose* "..." *referring to* "..."" are addressed to the Selenium method click(), with the appropriate sequence of actions to perform this task on the Final UI. As this behavior can be performed only in Radio Buttons,

```
@When(value = "I choose \"$elementName\" referring to \"$locatorParameters\"", priority = 10)
@Then(value = "I choose \"$elementName\" referring to \"$locatorParameters\"", priority = 10)
public void informWithParameters(String elementName, List<String> locatorParameters) {
    locatorParameters = DataProviderUtil.replaceDataProvider(locatorParameters);
    Element element = runner.getElement(currentPageName, elementName);
    element.setLocatorParameters(locatorParameters);
    if (element instanceof Radio) {
        ((Radio) element).click();
    } else if (element instanceof CheckBox) {
        ((CheckBox) element).click();
    } else if (element instanceof Link) {
        ((Link) element).click();
    } else if (element instanceof Calendar) {
        ((Calendar) element).click();
    } else {
        throw new BehaveException(message.getString("exception-invalid-type", element.getClass().getName()));
    }
}
```

Fig. 14. Behavior "chooseRefferingTo" being structured as a Java method.

Check Boxes, Links or Calendars, the concrete in stance of any of these elements are searched on the Presentation layer.

The Presentation component includes the MyPages Java Class that makes the mapping between abstract UI elements of the ontology and the concrete/final UI components instantiated on the interface being tested. For that purpose, we make use of annotations in Java code following the Page Objects pattern [14] as illustrated in Fig. 15. UI components are identified through their XPath references or some other unique ID eventually used for some frameworks to implement the interface. This link is essential to allow the framework to automatically run the Steps on the right components on the Final UI.

```
public class MyPages {
@ScreenMap(name = "Find Flights", location = "..")
public class MainPage {
        @ElementMap(name = "Search", locatorType = ElementLocatorType.XPath, locator
= "...") // concrete UI component
        private Button Search; // abstract UI element
        ...
}
```

Fig. 15. Concrete and abstract UI elements being associated in a Java class.

For behaviors not addressed by the ontology, the MySteps Java Class allows developers and testers to set their own business behaviors and implement as well how they should be attended by the Selenium methods on the UI components. For both classes the main incomes are behaviors extracted from the User Stories that can be represented in simple packages of text files.

In short, once the ontology is charged, it is enough to identify on the Final UI under testing the concrete UI elements that were instantiated to represent abstract UI elements. Afterwards, when Scenarios are triggered, the application runs and Selenium performs Step by Step the specified behaviors, reporting testing results either by the JUnit green/red bar or by JBehave reports with the context and attached print screens of each identified failure.

4.3. *Mapping ontological concepts*

The ontology based approach we have proposed for testing UIs allows us to establish a direct mapping of abstract concepts in the ontology and concrete instances in scenarios, prototypes and final UIs. Table 5 provides an example of how these con cepts are mapped for the Scenario "One Way Tickets Search".

Table 5. Mapping ontological concepts for scenarios, prototypes and final UIs.

Ontological concepts	Scenario	Prototype and final UI
Condition: Given Behavior: *goTo*	*Given I go to "Find flights"*	Browser Window: "Find flights"
Event: When Behavior: *choose*	*When I choose "One way"*	Link: "One way"
Event: When Behavior: *typeAndChooseInTheField*	*And I type "Paris" and choose "CDG - Paris Ch De Gaulle, France" in the field "From"*	Autocomplete: "From"
Event: When Behavior: *typeAndChooseInTheField*	*And I type "Dallas" and choose "DFW - Dallas Fort Worth International, TX" in the field "To"*	Autocomplete: "To"
Event: When Behavior: *chooseTheOptionOfValueInTheField*	*And I choose the option of value "2" in the field "Number of passengers"*	Dropdown List: "Number of passengers"
Event: When Behavior: *chooseReferringTo*	*And I choose "12/15/2016" referring to "Depart"*	Calendar: "Depart"
Event: When Behavior: *clickOn*	*And I click on "Search"*	Button: "Search"
Action: Then Behavior: *willBeDisplayed*	*Then will be displayed "Choose Flights"*	Text: "Choose Flights"

4.4. *Discussion*

The ontology presented in this paper describes behaviors that report Steps of Scenarios performing actions directly on the User Interface through Interaction Elements. Thus, the ontological model is domain free, which means that it is not dependent of business characteristics that are described in the User Stories. Specific business behaviors shall be specified only for the systems to which they make ref erence, not affecting the whole ontology. Therefore, it is possible to reuse Steps in multiple testing Scenarios. For example, the ontological behaviors *goTo*, *choose*, *chooseReferringTo*, *typeAndChooseInTheField*, *chooseTheOptionOfValueInThe Field*, *clickOn*, and *willBeDisplayed* presented in the case study can be reused for Scenarios of other system requiring those kind of user's actions.

However, Scenarios should be specified in the user interaction level, writing Steps for each click, selection, typing, etc. A possible solution to avoid this level of detail would be to work with higher level behaviors that are described by user's tasks. Nonetheless, user's tasks often contain information from specific application

domains. For example, high level Steps like "*When I search for flights to "Destination"*" encapsulate all low level behaviors referring to individual clicks, selections, etc.; however, it also contains information that refers to the airline domain (i.e. behavior "search for flights"). Therefore, that Step would only makes sense on that particular application domain. For further researches, it could be interesting to investigate domain ontologies to be used in parallel with our ontology, defining a higher level business vocabulary database in which business behaviors could be mapped to a set of interaction behaviors, covering recurrent Scenarios for a specific domain, and avoiding them to be written every time a new interaction may be tested.

Another aspect to be discussed is that even having mapped synonyms for some specific behaviors, our approach does not provide any kind of semantic interpreta tion, i.e. the Steps might be specified exactly as they were defined on the ontology. The JBehave plugin for Eclipse shows (through different colors) if the Step being written exists or not on the ontology. This resource reduces the workload to remember as exactly some behavior has been described on the ontology.

On one hand, the restricted vocabulary seems to bring less flexibility to designers, testers and requirements engineers. Nonetheless, on the other hand, it establishes a common vocabulary, avoiding typical problems of ambiguity and incompleteness in requirements and testing specifications. Further studies on Natural Language Pro cessing (NLP) techniques might help to improve the process of specification adding more flexibility to write Scenarios that could be semantically interpreted to meet the behaviors described on the ontology. This issue is certainly a worthwhile topic for further research.

It is also worthy of mention that the concepts and definitions in the ontology presented herein are only one of the possible solutions for addressing and describing behaviors and their relations with UIs. Despite the fact that our ontology covers concepts available in well known languages such as MBUI, UsiXML and SCXML, we do not assume that the coverage is exhaustive. For that, we suggest that other behaviors, concepts and relationships might be included in the future to express idiosyncrasies of specific interaction techniques (ex. multimodal interaction techni ques) and/or specific platforms (ex. ambient systems). If so, new elements can be added by direct imports into the ontology or simply adding new more expressive behaviors to the Object Property "behaviors" and linking them to the appropriate set of Interactive Elements.

Finally, when representing the various Interaction Elements that can attend a given behavior, the ontology also allows extending multiple design solutions for the UI, representing exactly the same requirement in different perspectives. Thus even if a Dropdown List has been chosen to attend for example a behavior *setInTheField* in a Prototype, an Auto Complete field could be chosen to attend this behavior on the Final UI, once both UI elements share the same ontological property for this behavior under testing. This kind of flexibility makes tests pass, leaving the designer free to choose the best solutions in a given time of the project, without modifying the behavior specified for the system.

5. Conclusion

In this paper, we have presented a behavior based ontology aimed at test automation that can help to validate functional requirements when building interactive systems. The proposed ontology acts as a base for a common vocabulary that is articulated to map interactive behaviors to UI Elements, allowing automation of acceptance test of functional requirements in Prototypes and/or in full fledge User Interfaces. The ontology also supports the design of User Interfaces by providing a consistent set of UI Elements that meet particular behaviors.

In addition, behaviors described in the ontology are already implemented for automating tests on UIs, which means we can freely reuse them to write new Scenarios in natural language, providing test automation with little effort from development teams. It allows specifying tests in a generic way, which benefits reuse along the development process. For that reason, we are also investigating the use of the ontology for testing model based artifacts such as low fidelity Prototypes and Task Models. Testing in this kind of artifacts could be conducted through a static verification of their source codes and would help to integrate testing in a wider spectrum of artifacts commonly used to build interactive systems.

We have also presented tools that demonstrate how this ontology can support testing of interactive systems. So far, only interactive Prototypes built in PANDA can be tested by the ontology once it requires that tools are able to read and support the set of described behaviors. On the other hand, tests in Web Final UIs can run independently of the frame works used to build these UIs. It is possible because tests provided by our tool assess the concrete UI elements found on the interface in the final HTML page.

5.1. *Future works*

Although the results presented in this paper are still preliminary, the current version of the ontology opens door for many interesting research questions that motivate our future work. First of all, we are planning to investigate the acceptability of the approach with users. The idea is to assess through empirical evaluation whether (or not) people involved in the development process of interactive applications are able to employ our approach to specify their functional requirements using the proposed template and the concepts present in our ontology. We are planning to conduct these empirical studies with developers, requirement engineers, clients and end users, in order to determine the potential in the context of multidisciplinary and complex development teams.

Currently we are also investigating more complex behaviors in real cases of software development. We suggest that it would be useful to collect data about the effectiveness and the workload when specifying tests using the ontology. Other case studies including mobile platforms are planned as well. In a longer run, we also want to explore idiosyncrasies of interaction techniques and/or platforms to check

hypothesis related to the coverage of concepts in the current ontology. These studies might also help to improve the ontology.

Future work should also consider ontologies as knowledge bases, keeping specific behaviors for specific groups of business models in domain ontologies. It would allow us to also reuse entire business scenarios in systems sharing similar business models.

Last but not least, we also want to investigate the reuse of User Stories created using our approach to assess other types of artifacts used during the development process. In this paper, we have shown how to test model based prototypes build with PANDA and full fledge implementation of an interactive system. However, we suggest that stories created with our approach can be potentially reused with other kind of artifacts that also describe some behavioral aspect of interactive systems. We are particularly interested in artifacts such as tasks models and business models. So far, we do not know how much our approach is applicable to these artifacts as they only partially describe the system behavior. For that, further studies are necessary.

References

[1] D. Chelimsky, D. Astels, B. Helmkamp, D. North, Z. Dennis and A. Hellesoy, The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends, *Pragmatic Bookshelf*, 2010.

[2] K. Pugh, *Lean Agile Acceptance Test Driven Development* (Pearson Education, 2010).

[3] G. Adzic, *Specification by Example: How Successful Teams Deliver the Right Software* (Manning Publications, 2011).

[4] M. Cohn, *User Stories Applied: For Agile Software Development* (Addison Wesley Professional, 2004).

[5] N. Guarino, D. Oberle and S. Staab, What is an ontology? in *Handbook on Ontologies* (Springer, 2009), pp. 1 17.

[6] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon and J. Vanderdonckt, A unifying reference framework for multi target user interfaces, *Interacting with Computers* **15**(3) (2003) 289 308.

[7] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon and V. López Jaquero, USIXML: A language supporting multi path development of user interfaces, *EHCI/DS VIS*, 2004.

[8] M. Winckler, J. Vanderdonckt, A. Stanciulescu and F. Trindade, Cascading dialog modeling with UsiXML, in *International Workshop on Design, Specification, and Ver ification of Interactive Systems*, 2008, pp. 121 135.

[9] M. Winckler and P. Palanque, StateWebCharts: A formal description technique dedi cated to navigation modelling of web applications, in *Design Specification and Verifi cation of Interactive Systems*, 2003, pp. 61 67.

[10] J. Pullmann, MBUI Glossary W3C, Fraunhofer FIT, 2016. [Online]. https:// www.w3.org/TR/mbui glossary/.

[11] J. Barnett *et al.*, State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C, 2016. [Online]. http://www.w3.org/TR/scxml/.

[12] D. North, What's in a Story? 2016. [Online]. http://dannorth.net/whats in a story/.

[13] J. L. Hak, M. Winckler and D. Navarre, PANDA: Prototyping using annotation and decision analysis, in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2016, pp. 171 176.

[14] M. Fowler, PageObject, 2016. [Online]. http://martinfowler.com/bliki/PageObject.html.