

# MIRA: A Model-Driven Framework for Semantic Interfaces for Web Applications

Ezequiel Bertti, Daniel Schwabe

Department of Informatics, PUC-Rio.  
Rua Marques de Sao Vicente, 225. Rio de Janeiro, RJ 22453-900, Brazil

{ebertti, dschwabe}@inf.puc-rio.br

**Abstract.** A currently recognized barrier for the wider adoption and dissemination of Semantic Web technologies is the absence of suitable interfaces and tools to allow suitable access by end-users. In a wider context, it has also been recognized that modern day interfaces must deal with a large number of heterogeneity factors, such as varying user profiles and runtime hardware and software platforms. This paper presents MIRA, a framework for defining and implementing Semantic Interfaces for Web applications, including those on the Semantic Web. A Semantic Interface is defined as being one capable of understanding and adapting to the data it presents and captures, and its schema, if present. Moreover, the interface must also be able to adapt to its context of use – the device being used, any available information about its user, network conditions, and so on. Using a model-driven approach, MIRA allows developers to define such interfaces, and generates code that can run on clients, servers or both. We have carried out a qualitative evaluation that shows that MIRA does indeed provide a better process for developers, without imposing any significant performance overhead.

**Keywords:** HCI, Interface, Adaptation, Semantic Web, Data-driven design, Model-Driven Interface Design.

## 1 Introduction

In his ESWC 2013 keynote talk<sup>1</sup>, David Karger defines a Semantic Web application as “*one whose schema is expected to change*”. Analogously, we define a “Semantic Interface” one that is able to cope in an effective manner with this variability. Along a different dimension, a Semantic Interface should also exploit the semantics of interactions themselves, allowing the interface to exhibit similar characteristics to the way humans interact among themselves – namely, a degree of context awareness of the interaction process itself.

It has long been observed that the design and implementation of the interface components of Web applications (and other as well) consumes over 50% of the

---

<sup>1</sup> See this blog post for a summary and slides - <http://goo.gl/vqXglr>.  
Video available at [http://videlectures.net/eswc2013\\_karger\\_semantic/](http://videlectures.net/eswc2013_karger_semantic/)

development effort, as reported by Myers and Rosson already in the nineties [12]. Since then, in spite of the lack of published similar measures, it is safe to assume their figures must have surely increased, due to the evolution of the computing platforms, the advent of the Internet and the Web, and the now popular gestural and vocal interface modalities. Sources of heterogeneity affecting application development include:

- Different computing platforms, affording a variety of interaction modalities and diverse input/output capabilities;
- Multiple, often dynamically varying contexts of use, be it at a desktop with a wired network or on the go using a smartphone, head-mounted display or a watch, wirelessly connected in a variety of underlying network infrastructures. Such environments may have high degree of noise, and sometimes restricted bandwidth;
- Multiple, ever evolving set of tasks that must be supported, derived from an increasing number of different workflows that users adopt and must be supported by the application. The example of Homebrew Databases cited by Karger [1] well illustrates this point;
- Highly diverse types and profiles of end users, ranging from very novice to experts, being from many different cultures and speaking a multitude of languages;
- The increasing need to integrate data with no schema, or whose schema changes frequently. This data typically comes from different sources, often not under the control or responsibility of the application designer. A prime example is the use of data in the Linked Data Cloud.

To further aggravate the situation, the context of use, i.e., each component of the triad <user, platform, environment> often changes dynamically while the application is being used, which calls for so-called Plastic UIs [5], capable of adapting while preserving the “user experience” while the user is engaged with the application.

A common approach to deal with such complexity is to use formal models to represent various aspects of the artifact being designed, breaking up the problem into smaller, more manageable tasks. Regarding interfaces, the Model-Based User Interface (MBUI) development approach has been used to address these challenges and maintain or decrease the level of effort necessary to design and implement application interfaces, through the introduction of suitable abstractions.

The Cameleon Reference Model is a current reference framework for User Interfaces gaining adoption [4], the item of several years of research of a major European research project, which proposes four abstraction levels for modeling UIs: Task and Domain, Abstract Interface, Concrete Interface, Final User Interface.

The Domain model describes the domains of the application, and the Task model describes the sequence of steps needed to perform the tasks (with respect to interactions with the User Interface).

The Abstract Interface model describes the composition of interface units in an implementation and modality independent way.

The Concrete Interface model describes the interface in terms of platform-dependent widgets, but still modality- and implementation language independent.

The Final User Interface is the actual running code that the end user accesses when interacting with the application.

When considering the Semantic Web, and particularly applications leveraging Linked Data (so-called Linked Data Applications, LDA's), there exist several proposals of development environments or frameworks for supporting their development, such as CubicWeb<sup>2</sup>, the LOD2 Stack<sup>3</sup>, and the Open Semantic Framework<sup>4</sup>. In addition, semantic wiki-based environment such as Ontowiki<sup>5</sup>, Kiwi<sup>6</sup>, and Semantic Media Wiki<sup>7</sup> have also been used as platforms for application development over Linked Data. There are also proposals of frameworks for building visualizations such as Exhibit [9], or Fresnel<sup>8</sup>, among others.

While useful, they do not present a set of integrated models that allow the specification of an LDA, and the synthesis of its running code from these models. Therefore, much of the application semantics, in its various aspects, remains represented only in the running implementation code.

We have been working in the past years on the Semantic Hypermedia Design Method (SHDM) [7] and its implementation environment Synth [3], which aim to allow Model-Based development of Web Applications, including Linked Data based ones..

Our experience with SHDM and Synth [13] has led us to observe that the abstractions used for designing the interface are also applicable in a more general context, beyond LDAs, and independently of the other models in SHDM.

In this paper we present MIRA<sup>9</sup>, a framework incorporating an updated version of the of User Interface models used in SHDM, its implementation architecture, which can be leveraged by any application that provides a REST interface to its “business logic”. This includes, for example, RDF-based applications.

We present our approach in this paper as follows. After describing the example we are going to use through the paper in Section 2, we present our approach for interface modeling in Section 3. We discuss the implementation in Section 4. Section 5 discusses the evaluation of MIRA, and Section 6 presents the related work, discusses future work and draws some conclusions.

## 2 A Running Example

To help make the concepts discussed in the paper more concrete, we first briefly show an example interface<sup>10</sup> built over the Europeana RDF database<sup>11</sup>. Suppose the user

---

<sup>2</sup> <http://www.cubicweb.org>

<sup>3</sup> <http://lod2.eu/WikiArticle/TechnologyStack.html>

<sup>4</sup> <http://openstructs.org/open-semantic-framework>

<sup>5</sup> <http://ontowiki.net/Projects/OntoWiki>

<sup>6</sup> <http://www.kiwi-project.eu>

<sup>7</sup> [http://www.semantic-mediawiki.org/wiki/Semantic\\_MediaWiki](http://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki)

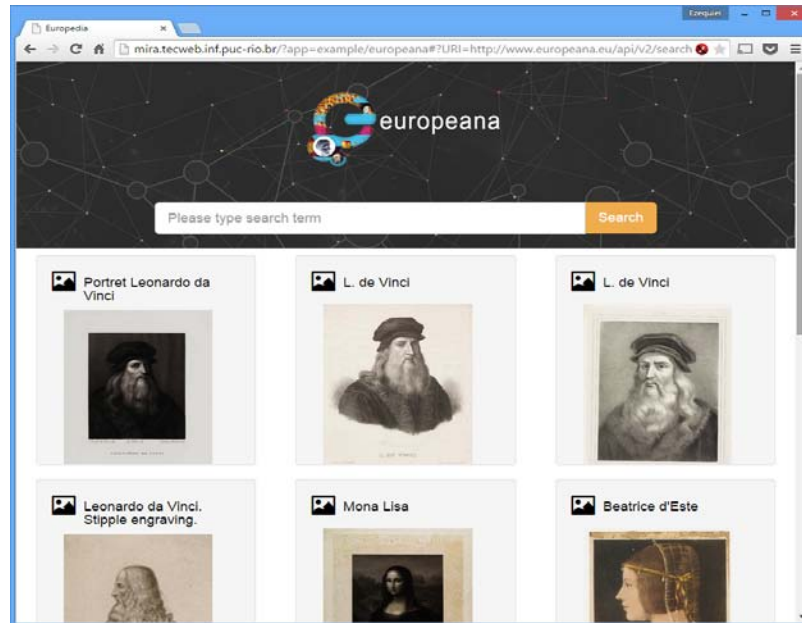
<sup>8</sup> <http://www.w3.org/2005/04/fresnel-info/>

<sup>9</sup> <http://mira.tecweb.inf.puc-rio.br>

<sup>10</sup> Examples and source code available at <http://mira.tecweb.inf.puc-rio.br/>

<sup>11</sup> <http://data.europeana.eu/>

starts with a query string “da Vinci”; **Fig. 1** shows an interface with the resulting items.

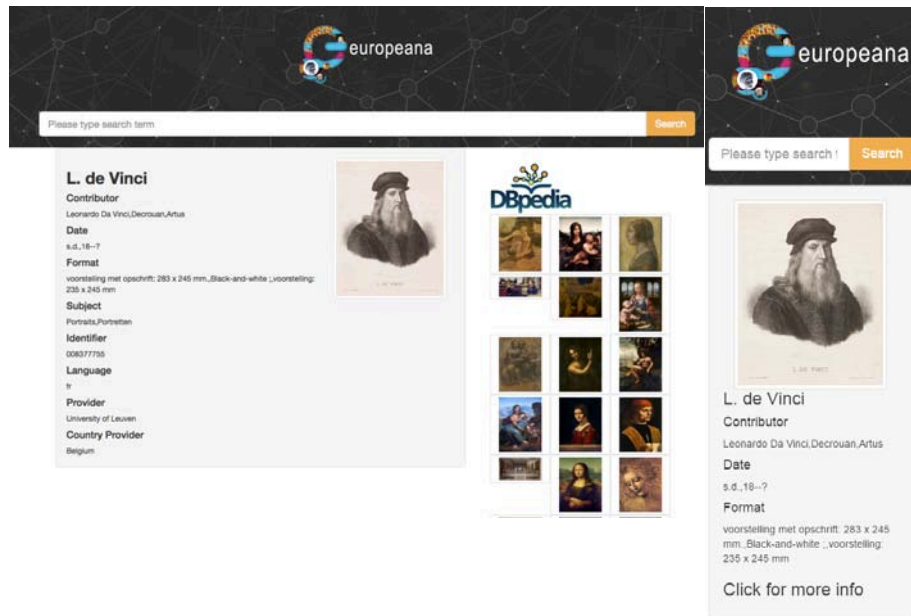


**Fig. 1** – Search items for the query “Da Vinci” in the Europeana collection

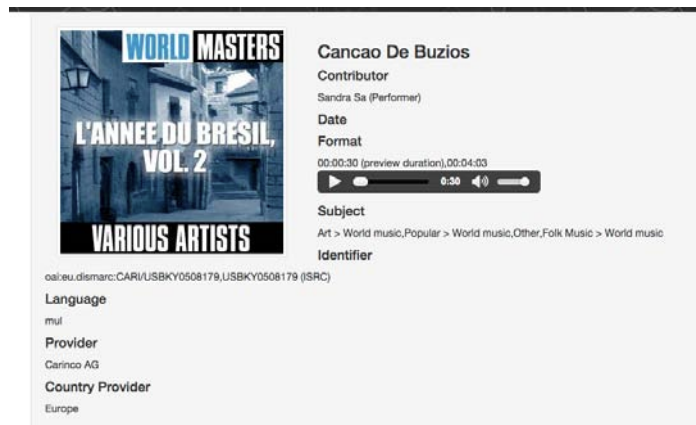
Clicking on the second entry leads to the interface shown in **Fig. 2**. Items in this collection have a variety of properties, including some with links to DBPedia. When present, this link is shown next to the other properties using DBPedia’s logo, but only in the desktop version. In addition, datatype properties can have different formats, and an appropriate interface widget is used to display each format. **Fig. 3** shows an item with an audio datatype property.

### 3 A Semantic Interface Model

The MBUI approach leverages abstractions to deal with complexity by following the principles of “separation of concerns”. Web (and interactive) applications exhibit many different behaviors, catering to different concerns. Some behaviors address the actual goal of the application, i.e., the so-called “business logic”, such as adding a product to a shopping cart, or proceeding to checkout. Other behaviors address the interaction between the user and the application, e.g., choosing one item from a large list of options - e.g., picking a book from a large list of known titles. The latter must be supported by the interface.



**Fig. 2** –Detailed item interface, accessed on a desktop (left) and on a mobile device (right).



**Fig. 3** – An item with an audio datatype property

Based on this separation of concerns, the Interface Model in SHDM [15] leverages the idea that it is possible to separate the “essence” of the interface from its look-and-feel. This is achieved by decomposing the interface specification into an Abstract Interface Model, and a Concrete Interface Model.

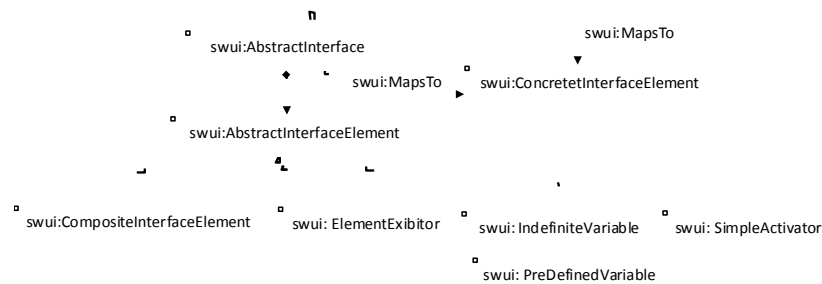
Briefly, the Abstract Interface focuses on the roles played by each interface widget in the information exchange between the application and the outside world, including the user. It is abstract in the sense that it does not capture the look and feel, or any information dependent on the runtime environment. The Concrete Interface model is responsible for the latter.

In its essence, the interface must be able to display information to the user, upon request from the application; capture information provided by the user; or signal to the application the occurrence of some event caused by the user or by the environment.

Accordingly, the Abstract Interface meta-model (see **Fig. 4**) defines an abstract interface as a composition of abstract interface elements (widgets). These in turn can be an ElementExhibitor, which is able to show values; and IndefiniteVariable, which

is able to capture an arbitrary input string; a DefinedVariable, which is able to capture input values (one or several) from a known set of alternatives; and a SimpleActivator, which is able to react to an external event and signal it to the application.

From an Abstract Interface, a mapping specification made by the designer determines how each abstract widget will be mapped onto one or more Concrete Interface elements, based on several possible factors, which we will discuss later.



**Fig. 4 - The Abstract Interface metamodel in SHDM**

SHDM follows the basic abstraction levels of the Cameleon Reference Model. The Domain Model, in SHDM is simply a set of RDF triples, which form a graph, and may include RDFS or OWL definitions. It is often the case that there does not exist any schema definitions in the Domain Model, only instances of resources representing information items.

A mapping specification made by the designer determines how each abstract widget will be mapped, on one side, onto one or more Concrete Interface elements, and, on the other side, onto which Domain Model elements, including the operations defining the application behavior to achieve the desired tasks.

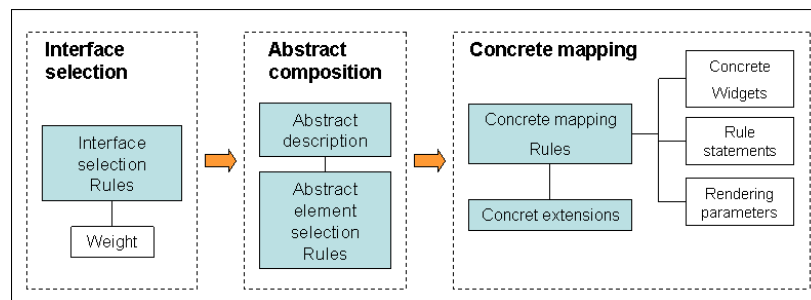
### 3.1 Information Flow in the Interface

In the traditional MVC model, each request to the model is accompanied by the view that will be used to display the items. In MIRA, the request does not include the view to be used. Instead, the following steps, summarized in **Fig. 5**, determine the actual interface that will be used to display the items,

1. The applications receives a request, in the form of a REST operation, and returns its items;
2. A set of Abstract Interface Selection rules is evaluated, determining an Abstract Interface instance, among those defined by the designer, that is able to display the items. These rules can take into account not only the data returned, but also any other context information, such as device being used, network bandwidth available, user information, etc...;
3. The selected Abstract Interface instance is assembled. Each Abstract Interface is a composition of widgets, each of which may have a condition associated with it. This condition is evaluated also based on the data and the context information; only those widgets with enabled conditions are included in the

actual assembled Abstract Interface instance. Some widgets may not have an associated condition and will be always included.

4. A second set of rules is evaluated to determine the mapping between each Abstract Interface widget in the select Abstract Interface instance and a corresponding Concrete Interface widget. The final Concrete Interface is also a composition of widgets. Each widget may encapsulate complex interface behaviors, and depends on the actual runtime environment under which the interface will be displayed.



**Fig. 5 – Interface generation flow.**

The first step is the selection of the abstract interface, determined by its own set of rules. The result of executing these rules is a ranked list of candidate Abstract Interfaces, based on a weighting function defined by the UI designer.

The highest-ranking Abstract Interface is then chosen. Its own composition is again determined by executing another set of rules, which may include or exclude widgets from the initial base Abstract Interface composition defined by the designer.

Next, a third set of rules is executed to determine how each Abstract Interface widget will be mapped onto concrete interface widgets.

This rule-driven approach has several advantages:

1. It allows taking into account actual runtime data and context information in determining which interface should be used. Since the rules can refer to actual input data to be exhibited through it, as well as to the Domain Model, it is fair to say that the interface definition is now Semantic, in the sense of being aware of the data types and values of the data it is exposing;
2. It allows adapting the interface to both the user and to the execution environment, allowing a user experience that is in tune with the user's device and environment capabilities. Once again, such rules may take into account the semantics of the user or context model to alter the concrete interface.
3. It becomes a design choice whether the adaptation process will be run only at design time, or also during runtime. Running them during the application execution provides maximum flexibility, as the interface can change dynamically in reaction to several context changes, such as change of device, reduced bandwidth, loss of modality due to either circumstantial reasons (e.g., no visual access during driving) or due to hardware failure (e.g., display failure).

### 3.2 Abstract Interface Definition

The Abstract Interface is a composition of elements, forming a hierarchy of widgets. Each element may have conditions associated to it, which determine if that element will be included in the final Abstract Interface composition or not. This conditional inclusion allows defining variations of the same basic interface that share common elements.

Below we show part of the Abstract Interface specification for the interfaces in **Fig. 2** and **Fig. 3**.

```
var abstract = [{
  name: 'topic', widgets : [
    {name: 'header', children:[ {name: 'logo'}, {name: 'search_form', children:[ {name:
'search_group', children:[ {name: 'search_field',
      {name:'search_button'} ]}]},
    {name:'content', children: [
      { name: "item", children: [
        {name: 'item_panel', children: [
          { name:"item-box", children:[
            {name: 'item-media-link', bind:'$dataObj.rdf_prop("edm:isShownAt")[0]["@id"]'
              children: [
                {name: "item-media", bind:'$dataObj.rdf_prop("edm:object")[0]["@id"]' } },
            { name:"item-title", bind:'$dataObj.rdf_prop("dc:title")[0]' },
            { name:"item-contributor"},
            { name:"item-contributor-value", bind:'$dataObj.rdf_prop("dc:contributor")' },
            { name:"item-date"},
            { name:"item-date-value", bind:'$dataObj.rdf_prop("dc:date")[0]' },
            { name:"item-format"},
            { name:"item-format-value", bind:'$dataObj.rdf_prop("dcterms:extent")[0]' },
            { name:"item-player", when:'isSound,hasPreview',
              bind:'$dataObj.rdf_prop("edm:isShownBy")[0]["@id"]' },
            { name:"item-extra-info", children:[
              { name:"item-subject"},
              { name:"item-subject-value", bind:'$dataObj.rdf_prop("dc:subject")[0]' },
              { name:"item-identifier"},
              { name:"item-identifier-value", bind:'$dataObj.rdf_prop("dc:identifier")[0]' },
              { name:"item-language"},
              { name:"item-language-value", bind:'$dataObj.rdf_prop("dc:language")[0]' },
              { name:"item-provider"},
              { name:"item-provider-value", bind:'$dataObj.rdf_prop("edm:dataProvider")[0]' },
              { name:"item-country" },
              { name:"item-country-value", bind:'$dataObj.rdf_prop("edm:country")[0]' } ]]}},
            { name: 'sidebar-dbpedia', when: 'hasDbpedia', datasource:
'url:<%= $env.methods.get_datasource_dbpedia_uri($dataObj.rdf_prop(
"dc:contributor")) %>', children:[ { name:"dbpedia-item"},
/* ... */
      ] } ] } ],
  ] } ] }
```



```
{name: 'footer'} ]}
```

Each widget has a “name” and may have a sub-tree of widgets, indicated by the array “children”. The element “bind” binds a value from the data (Domain Model) to an abstract widget. In the example, several widgets have values bound to RDF properties using the MIRA built-in auxiliary function “rdf\_prop”, which has been defined to ease handling Json-LD structures. The widget “sidebar-dbpedia” has a condition to be included named “hasDbpedia” (highlighted in the text), which is specified in the Conditions section detailed in section 3.3. This condition tests whether a DBpedia link is present in the triples having “dc:contributor” as property. The widget “item-player” also has a condition, which causes it to be included only when there is an audio file linked to the entry (as in the case of interface in **Fig. 3**).

### 3.3 Conditions and Interface Definition Parameters

MIRA uses condition-action rules for selecting the Abstract Interface, for including widgets (or not) in the selected Interface, and for deciding on the mapping to apply to generate the concrete widget.

To ease the specification, conditions are declared separately, and can thus be reused within different rules. These conditions can test

- Any expression involving the Domain Model. For instance, it can test the type and value of a data item (e.g., datatype property), or whether the element being exhibited is a hypertextual link (or an Object Property).
- Hypertextual parameters received in an http request;
- Browser header information, including browser, platform, operating system, etc.
- Environment variables, e.g., date and time of day, location.

Some of the conditions used in the example are shown below.

```
var conditions = [  
  {name: 'isItem', validate: '$data.action == "search.json"'},  
  {name: 'isJsonLD', validate: '$data["@context"] != null'},  
  {name: 'hasDbpedia', validate:  
    '$env.methods.get_datasource_dbpedia_uri($dataObj.rdf_prop(  
      "dc:contributor")) != null'},  
  {name: 'hasPreview', validate: '$dataObj.rdf_prop("edm:isShownBy").length >  
    0'},  
  {name: 'isSound', validate: '$dataObj.rdf_prop("edm:type")[0] == "SOUND"}];
```

Each condition specification has the general form {name: <name>, validate: <expression>}, where <expression> is the Boolean function corresponding to the condition itself.

In the example, the condition “isItem” tests if the action executed. was a search; the condition “isJsonLD” tests if the response from the request contains a “@context” element; “hasDbpedia” tests if there are DBpedia URLs in the “dc:Contributor” property; and “isSound” tests if the “edm:type” property has value “Sound”.

These condition expressions also illustrate some of the built-in functions and variables available in MIRA, such as “\$data.object”, used to access complex JSON-LD structures; “rdf-prop”, which extracts RDF properties from JSON-LD structures.

### 3.4 Abstract Interface Selection Rules

The first step in defining the Interface is establishing the selection rules for the Abstract Interface. The pre-condition in these rules define when each Abstract Interface is applicable, allowing, for instance, to

- Select the interface only if the user is logged in;
- Select the interface only if the application is being accessed from a mobile device;
- Select the interface only for certain types of data passed as input during runtime. Notice that this is often necessary if one wants to deal with “raw” data in RDF, which may not have any schema or vocabulary information associated with it.

In our example, the Abstract Interface selection rules are

```
var selection = [  
  { when: 'isItem', abstract: 'items' },  
  { when: 'isJsonLD', abstract: 'topic' }  
];
```

The first rule selects the abstract interface “items” if the condition “isItem” (defined in Section 3.3) is satisfied. Similarly, the abstract interface “topic”, illustrated in **Error! Reference source not found.** and specified in section 3.2 is selected if the “isJsonLD” condition (defined in Section 3.3) is true.

### 3.5 Concrete Interface Mapping Rules

For each Abstract Interface widget, there is a mapping rule that determines how it is mapped onto concrete widgets. As a consequence, the Concrete Interface’s nesting structure is defined by the Abstract Interface’s own structure, since it is assumed that this structure should be determined by the application’s semantics.

Abstract Interfaces may be mapped to more than one Concrete Interface, in which case they must include a “maps” element in its specification. By default, if no explicit mapping is given, MIRA matches both by having the same name.

Each Concrete Widget has a “name”, a “widget” type, and a series of optional “tags”, which depend on the type of widget. MIRA includes a SimpleHTML pre-defined type that maps to HTML, and a Bootstrap type, which maps to Bootstrap framework<sup>12</sup> primitives. A condition may be specified in the optional “when” element; when present, the mapping is applied only if the condition is true. Widgets are processed in the order of the specification; if there is more than one specification for the same widget, that last one evaluated is used.

---

<sup>12</sup> <http://getbootstrap.com/>

Below we show some of the rules that map the Abstract Interface “topic” shown in section 3.2 into the interfaces shown in **Fig. 2**, followed by some comments.

```

1. var concrete = [{
2.   name: 'topic', head: head, maps: [
3.     { name: 'header', widget: 'SimpleHtml', tag: 'div', class: 'container-fluid text-center
      fundo' },
4.     { name: 'logo', widget: 'SimpleHtml', tag: 'img', src: "imgs/europedia.png" },
5.     { name: 'search_form', widget: 'SimpleHtml', tag: 'form',
      onsubmit: 'do_search(event);' },
6.     { name: 'search_group', widget: 'SimpleHtml', tag: 'div', class: 'input-group
      form_center col-sm-8' },
7.     { name: 'search_field', widget: 'SimpleHtml', tag: 'input', class: 'form-control input-
      lg', placeholder: "Please type search term" },
8.     { name: 'search_button', widget: 'BootstrapFormGroupButton', class: 'btn-warning',
      value: "Search", events: { 'click': 'do_search' } },
9.     { name: 'content', widget: 'SimpleHtml', tag: 'div', class: 'container' },
10.    { name: 'item', widget: 'SimpleHtml', tag: 'div', class: 'row' },
11.    { name: 'item_panel', tag: 'div', md: '12' },
12.    { name: 'item_panel', when: 'hasDbpedia,isDesktop', tag: 'div', xs: '12', sm: 12, md: 8,
      lg: 8 },
13.    { name: 'item-box', tag: 'div', class: 'well' },
14.    { name: 'item-extra-info' },
15.    { name: 'item-extra-info', when: 'isMobile', widget: 'Collapsed', title: { value: 'Click for
      more info' } },
16.    { name: "item-title", tag: 'h2', value: $bind },
17.    { name: "item-media-link", tag: 'a', pull: 'right', href: $bind, xs: 12, sm: 12, md: 4,
      lg: 4 },
18.    { name: "item-media", tag: 'img', img: 'thumbnail', src: $bind },
19.    { name: "item-contributor", tag: 'h4', value: 'Contributor' },
20.    { name: "item-date", tag: 'h4', value: '@Date' },
21.    { name: "item-type", tag: 'h4', value: 'Type' },
22.    /*...*/
23.    { name: "item-contributor-value", value: '$dataObj.rdf_prop("dc:contributor")[0]' },
24.    { name: "item-contributor-value", value: '$dataObj.rdf_prop("dc:contributor")[1]' },
25.    /*...*/
26.    { name: "item-player", when: 'isSound,hasPreview', widget: 'AudioPlayer',
      source: $bind },
27.    { name: "sidebar-dbpedia", xs: 12, sm: 12, md: 4, lg: 4 },
28.    /*...*/
29.    { name: 'footer', widget: 'TecWebRodape' } ] ] }

```

Lines 3-9 show the specification for the header of the page and the search box, mapping to plain HTML tags. Lines 10-26 show the mappings for the details of each “item”. In line 12 a widget has a condition specifying that it will be mapped only if “hasDBPedia, isDesktop” (see section 3.3) is true. Since the same element is also mentioned in line 11, the item is that a link to DBPedia will be shown only when

present. Lines 14-15 achieve an analogous effect; the element “items-extra-info” will be included only if the application is running on a mobile device. Lines 23-24 illustrate the use of built-in function “rdf\_prop” to retrieve property values to be exhibited. Line 26 shows the conditional inclusion of an audio player widget if there is an “edm:isShownBy” property value of type “sound” (see **Fig. 3**).

These examples illustrate how MIRA can leverage schema and data information to adapt both the form and the content of the generated interface. Notice that it would be similarly easy to change the concrete widget used based also on meta-property of the data, for example, changing from a pull-down menu widget to a check-box list widget depending on the number of items returned by a request. Such a change cannot be achieved by simple changes in CSS.

As seen from the examples in the mapping rules, concrete widgets are treated as software components outside the model itself; different concrete widgets should be defined for different runtime platforms. In this sense, we diverge from the Cameleon model, as Concrete Widgets are rendered directly to the Final User Interface.

A Concrete Widget should be self-contained, and capable of self-rendering based only on their input parameters. In addition to the built-in concrete widget types, MIRA provides an interface that allows the developer to add new concrete widgets whenever necessary<sup>13</sup>.

### 3.6 Interface events

A common behavior found in rich interfaces is the inter-dependence among widgets, such that changes in the value assigned to one widget (either capture or exhibition) will trigger changes in other widgets. For example, when booking a flight, once the departure date has been entered, the widget showing the return date is updated for instance, disabling dates earlier than the departure date.

MIRA provides the “event” primitive as part of conditions, that allowing widgets to react to events and take action when needed, taking the semantics of the data into account. In the flight reservation dialog, for example, the action can query the Domain Model for the earliest return date given a departure date for a flight in a given fare code.

## 4 Implementation

MIRA has been implemented in Javascript, following the UMD standard<sup>14</sup>, allowing it to be executed in both client browser and servers using Node.js. This flexibility allows MIRA to access sources that block “cross-site” scripting on the client (as is the case in several RDF repositories) by running this part of the code in a server environment. In addition, running on a server may be desirable in cases where the clients are expected to be mobile devices with limited computing and power consumption capabilities.

---

<sup>13</sup> Several examples can be found in the Flickr application example at MIRA website

<sup>14</sup> <https://github.com/umdjs/umd>

We did several code complexity analysis of MIRA's source code (see [2] for details). For reasons of space, we don't include them here, but it we can say that MIRA is of similar complexity as popular Javascript Interface Frameworks such as Backbone.js, Angular.js and JQuery. MIRA is open source and publicly available at <https://github.com/TecWebLab/mira>.

## **5 Evaluation**

There are at least two aspects we considered important in assessing MIRA. The first is its "expressive power", in the sense of being able to allow implementation of really complex, sophisticated interfaces.

The second, and most important one, is whether it actually brings benefits to the development of application interfaces, including ones with characteristics discussed in Section 1. We next discuss each.

### **5.1 Expressive power**

We have used MIRA to implement several application, ranging from simple straightforward o more sophisticated ones. The most complex application mimics some of the interfaces found in the well-known Flickr website, which is very complex and sophisticated. We have found that MIRA allows implementing such interfaces with simpler code. The exaple application can be accessed at the MIRA website. A second example is the Europeana interface used as an example in this paper. It should also be noted that in all examples, rules were used to generate mobile-friendly versions without requiring changes in the Abstract Interface definition, supporting our hypothesis that such decompositions are indeed viable and useful.

### **5.2 Qualitative Evaluation**

Given the complexity of carrying out a complete evaluation of the benefits of MIRA in interface development, we did a detailed qualitative evaluation with a small number of individuals.

We defined two applications with essentially the same functionality, but in different domains. The first required the developer to present a list of ads for real estate properties, where the presentation depended on the type of real estate property. Clicking on a property presents a new interface with its details, again with different presentation depending on the type of the property.

The second required the developer to present a list of items for soccer matches, with different presentations depending on the previous winning records of each team, and whether it was a home or away game. Clicking on a team presents the detailed items of its latest games, with different presentations depending on the item and location (home or away).

Each individual was given an hour-long introduction to MIRA, which included a complete walkthrough of an example application, and access to the complete detailed online documentation, with examples. They were also asked to fill in a questionnaire

to allow us to assess their prior professional experience and background, particularly with respect to interface development.

Subjects were given the option to choose one of the applications using any interface development framework they were already familiar with, if any. Then they developed the other application using MIRA.

For each application, they were given the documentation of the REST interface, including the format of the data returned, an HTML wireframe, a natural language specification of the business rules they should implement.

We used the “think out loud” approach during the development of each application, asking the subject to say out loud their doubts, thought processes, decisions, etc..., which were videotaped.

The items of these tests are summarized in the table below.

	Background	Framework used	Conventional	MIRA
1	Database developer	Jquery	1h35 ①	1h55 ②
2	Junior Frontend Developer	Backbone	1h31 ①	1h41 ②
3	Systems Analyst, very little programming experience	N/A	N/A	1h57 ①
4	Senior Frontend Developer	Jquery + Underscore	1h57 ②	1h26 ①
5	Senior Frontend Developer	Angular	1h18 ②	1h07 ①

Key:

① Soccer game items first ② Soccer game items second

① Classifieds first ② Classifieds second

All users were able to complete both tasks, with the exception of user 3, who had little prior programming knowledge and had never implemented an interface. For this user, it is actually remarkable that he was able to implement the interfaces using MIRA.

All subjects were able to develop also a mobile version of each application.

Whereas the development times were comparable, it should be stressed that the time using MIRA included the “learning curve” time needed to familiarize themselves with the framework. It is expected that if they were to develop a third application using MIRA their development time would decrease significantly.

Post-test interviews confirmed that the users were satisfied with their items and would consider using it in their daily tasks. They also stated that it would be much simpler to change the implementation if the requirements changed.

Details of the whole evaluation process can be found in [2].

These preliminary items give us strong evidence that MIRA can indeed improve the development process for Web applications, including Semantic ones.

## 6 Discussion and Conclusions

We have described a data- and model-driven rule based model and runtime architecture to specify interfaces to applications, including semantic ones according to Karger’s definition. It is data-driven since the actual interface is self-assembled as a

item of the execution of the various rule-sets that use the instance data. It is model-driven in the sense that it can use schema information if it is available as data, as in the case of RDF, RDFS and OWL repositories, as illustrated in the example.

The work presented here is related to a very large number of models and approaches that have been proposed in the literature (see, for example, [11]); it would be beyond the scope of this paper to make a comparison with every one of them. Several of the Interface Models in SHDM, e.g., the Abstract Interface and the Concrete, have counterparts in the many proposed models, e.g., Maria [14], UsiXML [9], UIML [8], among many, as well as those in WebML [6], differing mostly in the level of abstraction and on the underlying formalism (e.g., XML vs RDF). Each has advantages and disadvantages, a discussion of which would require another paper altogether. A similar observation can be made regarding the use of rules, the difference still remaining in the underlying models.

For RDF-based applications, there are a few frameworks for application development, such as Graphity<sup>15</sup> and Callimachus<sup>16</sup>, but none have interface-specific models beyond HTML. There are several frameworks for exhibiting RDF data, such as Exhibit and Fresnel, but they do not allow developing full-fledged interfaces, focusing on presentation and visualization instead.

The main contribution of this work is the update of the original SHDM Interface Models to be applicable to any application providing a REST interface, and a standards-based implementation framework that leverages these models to generate running, adaptable interfaces. In addition, extensions were defined to smoothen the use of MIRA specifically with RDF data, using Json and Json-LD formats. Qualitative evaluations indicate that MIRA can effectively ease the development of adaptive interfaces of the kind required by Semantic Applications.

We are continuing this work in several directions. The first is to continue the evaluation of the approach, both in terms of performance, but also in terms of its expressivity and usability for developers. Second, we continue building tools and components to ease using MIRA. Third, we want to explore the design trade-offs for multi-platform applications, as well as for distributed, multi-device interfaces.

**Acknowledgments.** D. Schwabe was partially supported by CNPq (WebScience INCT). E. Bertti was partially supported by a grant from NIC.br and the W3C Office Brazil. This work also had partial support from the Microsoft Open Source Initiative.

## References

1. Volda, A., Harmon, E. & Al-Ani, B. (2011). Homebrew databases: Complexities of everyday information management in nonprofit organizations. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2011). Vancouver, BC, May 7-12. New York: ACM Press, pp. 915–924.
2. Bertti, E.; “MIRA – A Model-Driven Interface Framework for REST Applications”, MSc dissertation, Department of Informatics, PUC-Rio, March 2015. (in Portuguese).

---

<sup>15</sup> <https://github.com/Graphity>

<sup>16</sup> <http://callimachusproject.org/>

3. Bomfim, M. de S.; Schwabe, D., Design and Implementation of Linked Data Applications Using SHDM and Synth. Proceedings of the Int. Conf. on Web Engineering ICWE 2011, LNCS 6757, Heidelberg, 2011, pp. 121-136.
4. Calvary, G. et al.: The CAMELEON Reference Framework, CAMELEON Project, September 2002, available at <http://giove.isti.cnr.it/projects/comeleon/pdf/CAMELEON%20D1.1RefFramework.pdf>, 2002.
5. Coutaz, J., and Calvary, G. HCI and Software Engineering for User Interface Plasticity. In Human Computer Handbook: Fundamentals, Evolving technologies, and Emerging Applications, 3rd edition, Julie Jacko Ed., Taylor and Francis Group Ltd., May 2012.
6. Ceri, S.; Fraternali, P.; Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites. Proc. of the WWW9 Conf., Amsterdam, May 2000.
7. Lima, F.; Schwabe, D.: Application Modeling for the Semantic Web, Proceedings of LA-Web 2003, Santiago, Chile, Nov. 2003. IEEE Press, pp. 93-102,
8. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J., Human-Centered Engineering with the User Interface Markup Language, Human-Centered Software Engineering, Ch. 7, Springer Verlag, London, 2009 pp 141-173
9. Huynh, D.F., Karger, D.R., Miller, R.C., Exhibit: Lightweight structured data publishing. In: Proc. of the 16th International Conference on World Wide Web, Banff, Canada, 737-746, 2007.
10. Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., López-Jaquero V., USIXML: A Language Supporting Multi-path Development of User Interfaces. in Working Conference on Engineering for Human-Computer Interaction and International Workshop on Design Specification and Verification of Interactive Systems(EHCI/DS-VIS), Hamburg, Germany , pp. 200-220, 2004.
11. Meixner, G., Paternó, F., and Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development. i-com 10(3) (2011), pp. 2-11.
12. Myers, B.; Rosson, M. B.: Survey on User Interface Programming. Proc. 10th Annual ACM CHI Conference on Human Factors in Computing Systems, pp. 195-202, 2000.
13. Nascimento, V.; Schwabe, D; "Semantic Data Driven Interfaces for Web Applications.", Proc. of the 13th International Conference on Web Engineering (ICWE 2013), LNCS 7977, Springer 2013, ISBN 978-3-642-39199-6, pp 22-36.
14. Paterno F.; Santoro C.; Spano L. D., Maria: A Universal, Declarative, Multiple Abstraction Level Language for Service-Oriented Applications in Ubiquitous Environments, ACM Transactions on Computer-Human Interaction (TOCHI), 16(4), Nov 2009
15. Silva de Moura, S.; Schwabe, D. Interface development for hypermedia applications in the semantic web. Proc. WebMedia and LA-Web, 2004, Ribeirão Preto, Brazil, October 2004. IEEE Press, pp 106-113.