

# WIND: an Interaction Lightweight Programming Model for Geographical Web Applications

The Nhan Luong<sup>1</sup>, Patrick Etcheverry<sup>1</sup>, Thierry Nodenot<sup>1</sup>  
and Christophe Marquesuzaa<sup>1</sup>

<sup>1</sup> IUT de Bayonne Pays Basque, LIUPPA-T2I, 2 Allée du Parc Montaury  
64600 Anglet, FRANCE  
{thenhan.luong, patrick.etccheverry, thierry.nodenot,  
christophe.marquesuzaa}@iutbayonne.univ-pau.fr

**Abstract.** Recent research has attested the implementation and the effectiveness of Geographic Information Systems in geographic teaching [1, 2]. Different works and experiments have shown that current Web Mapping Services and frameworks are partially unfitted for the design and easy programming of web applications dedicated to the teaching of geographic information. Our research problem is enabling to teachers to design by themselves an Active Reading Learning Scenario [3] making use of geographical information avoiding any programmer intervention. In this article, we report on Web Interaction Design (WIND), a web interaction lightweight programming model that we designed to help users to describe interactions between textual, map and calendar components. We present the core concepts of WIND (event-reaction-interaction processing), its API and we exemplify the WIND model from different examples. The main characteristics of WIND are then presented and discussed: WIND is integrative (it combines textual, map and calendar components; it also combines Web Mapping Services, etc.). WIND is fully executable (thanks to the WIND JavaScript API). WIND promotes lightweight programming. WIND is object-oriented: users can consistently describe interactions whatever the source and target components are. WIND is a declarative model enabling users to design web interactions between textual, map and calendar components. We present different examples and source codes that pinpoint the added-value of the WIND model and its API.

**Keywords:** geographic information, interaction programming, UML, JavaScript, Web Mapping Services and providers.

## 1 Introduction

Today, there is a tremendous amount of web-based cartographic applications that are available to users thanks to proprietary Web Mapping Services (*e.g.* Google Maps, Microsoft Bing Maps, Yahoo! Maps, France's IGN Geoportail) but also thanks to Free and Open Source Software (FOSS) from the Open Source Geospatial (OSGeo)

Foundation: OpenLayers, MapServer, etc. These mapping applications are handled more and more easily thanks to their interaction usage. The main advantage is that the map is updated immediately when users pan it around, zoom in or out on a specific area. Moreover, this allows users to search location, to get direction as well as to add their own information about a place. The term “*interactive map*”, therefore, is coined to describe the map with such intuitive interaction techniques put at the users disposal. Most Web Mapping Services provide an Application Programming Interface (API) to support the addition of users-defined overlays (textual or graphical supplements, hyperlinks). However, users require advanced computer science skills to exploit such an API.

More and more educational applications are built around geographic information. Teachers use such applications to teach their learners the concepts of their geographical area, to check understanding of reading texts relating a trip within their region [4]. Some teachers use Web 2.0 tools that have interactions to promote the interest of their learners. The communication via such tools also prevents any installation on computers in the classroom.

Our goal is to create applications by describing how the interactions work. The source code of applications should be automatically generated from this description. We are interested in the interaction models allowing to describe the interaction design. Several types of model focus on different aspects of interaction design:

1. task models describe the activities relevant to the needs of design: GOMS (Goal, Operator, Method, and Selection rules) [5], UAN (User Action Notation) [6], CTT (ConcurTaskTrees) [7];
2. dialogue models describe the structure of communication between the users and the application: finite-state machine [8], Petri net [9];
3. architectural models provide a generic structure of the application from which it is possible to create a particular interactive application: Arch [10], MVC [11], PAC [12].

However, most of interaction models are on high-level abstraction to envisage an automatic code generation.

The research work presented in this paper reconsiders such Web Mapping Services and interaction design models from the viewpoint of educators (*e.g.* teachers of primary school or secondary school) trying to produce dedicated web applications. They focus on specific localized textual documents called “travel stories” that embed a lot of geographic information about the movements of an actor within a territory. Such geographic information is composed of three complementary features: the spatial feature, the temporal feature and the phenomenon feature [13]. In 2007, we developed a first prototype with the Google Maps API coupled with PostGIS (*cf.* <http://erozate.iutbayonne.univ-pau.fr/forbes2007/exp/>). It is a prototype dedicated to assist a learner for the active reading and better following a travel story on an interactive application. This educational added-value was evaluated in a primary school with two 5<sup>th</sup> grade classes (9-10 year old children). We set up two specific features of these educational applications compared with currently available web-based cartographic applications:

1. The focus is on interaction between textual, map, and calendar components and not on data visualization;
2. The textual component may display normalized textual documents with the defined CSS styles on tagged words. The map component may display geographic information on a Web-based map with multi-layer mapping services (mainly Google Maps layers, Microsoft Bing Maps layers, and France's IGN layers). The calendar component may display temporal information on an interactive calendar like Google Calendar. The map is no longer the central component, neither is the text, nor the calendar: they play an equivalent role and the user (learner) needs to interact from any of these components and the system should react on any of these components.

Furthermore, this work demonstrated that current Web Mapping Services provide APIs which are partly inadequate to answer 1/ and 2/ above requirements. This paper will present our results in designing an interaction model and in implementing an API that developers could use to produce web-based mapping services providing end-users with rich interaction abilities.

In the next section, we present the WIND interaction model. In section 3, we focus on this API to demonstrate its usability and its advanced potentiality. In the last section, we discuss the advantages and limits of the approach as well as future works.

## 2 The WIND Model

This section presents the WIND model features allowing interactions to be specified and implemented within web applications. We want to highlight that our work focuses on interaction designing rather than interface designing [14].

We distinguish the types of user who are able to use a web framework to create an interactive application: *developers* who are people that have deep knowledge of Web technology, as well as programming skills; *power users* who do not have significant programming skills but understand the technological difficulties of the problem being treated; and *end-users* who only acknowledge the problem at the design level and do not have any programming skills [15]. According to these three types of user, our approach targets currently *developers* and *power users*, but our final goal is *end-users* (teachers and learners).

The web cartographic applications to be implemented should allow users to handle the interactions between its different components such as a map, a text and a calendar. An interaction may be simply defined as a triplet  $\langle \textit{area}, \textit{event}, \textit{reaction} \rangle$ . The *reaction* allows the user to perceive the result of his/her action (*event*) on an *area* of the interface. The WIND model has been designed according to this principle.

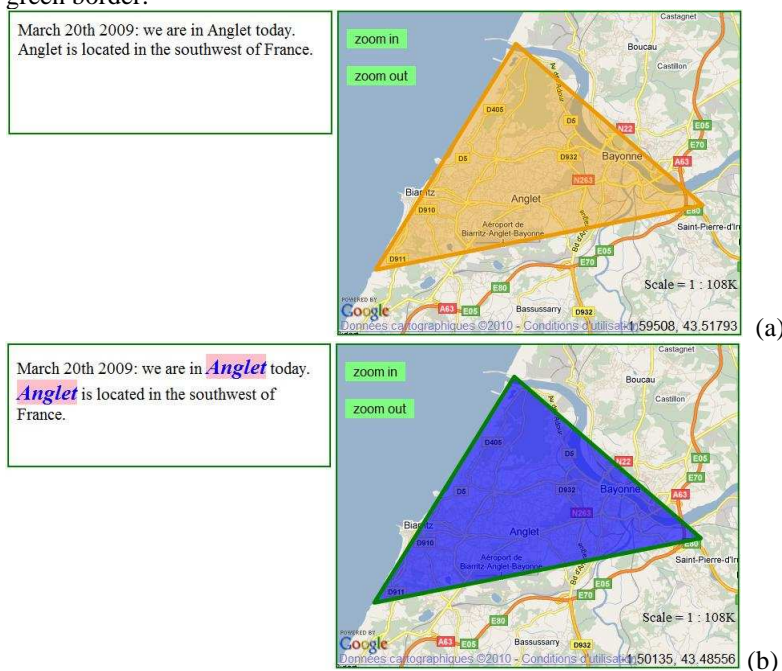
### 2.1 Example of an Interaction

The application presented on Fig 1 considers a very simple interaction enabling the user to click on the *Anglet* region (in France) of the map (Fig 1 - a). The system then

reacts highlighting all occurrences of the word “*Anglet*” in the text and changing the background colour of the *Anglet* region on the map (Fig 1 - b).

This interaction can be described in the following way: there is a map area which is sensitive to a user click. This reactive area is decorated with an orange background colour. On a left-click event on this map area, the following changes occur:

- in the text area, the instances of the word “*Anglet*” become decorated with bold, italic, a pink background colour and a blue foreground colour;
- in the map area, the *Anglet* region is decorated with blue background colour and green border.



**Fig 1:** a simple interaction example

## 2.2 WIND: Core Concepts

WIND (**Fig 2**) is an operational model that allows both describing and implementing interactions in web applications mixing texts (*Text*), maps (*Map*) and calendars (*Calendar*).

The WIND model relies on the fact that an interactive application presents an interface composed of reactive areas that may trigger system reactions. Thus, an interaction is defined by an area (*SensiblePart*), which, under a specific user action (*event*) will launch system reactions (*Reaction*). A reaction always results (partly) in the modification of an area (*SensiblePart*) of the interface, in order to allow the user to immediately perceive the result of his/her action. In some cases, modified areas can

become new reactive area (*SensiblePart*) such as menu opening or button appearance. It is, therefore, able to trigger new system reactions (*Reaction*).

Since WIND describes interactions within applications containing maps, texts and calendars, the reactive areas (*SensiblePart*) on which the user will be able to act could also be text areas (*TextPart*), map areas (*MapPart*) or calendar areas (*CalendarPart*). On a map, for example, a reactive area (*MapPart*) will concern a geographical area defined by a point (a location), a line (a river, a route) or a polygon (a city, a region area). On a text, a reactive area (*TextPart*) could be defined by a particular word or a set of words (an expression, a sentence, a paragraph) while on a calendar, a reactive area (*CalendarPart*) will take the form of a specific date or period.

A reactive area (*SensiblePart*) can be sensitive to various user actions (*event*): a click on the area, a mouse-over or a double-click for example. The system reaction (*Reaction*) following a user action (*event*) on a specific area (*SensiblePart*), may result in a visual effect on one or several areas (*SensiblePart*) of the screen. A visual effect may be defined with different styles: boldfacing for a text area (*TextPart*), background colour of a town on a map (*MapPart*) and so on.

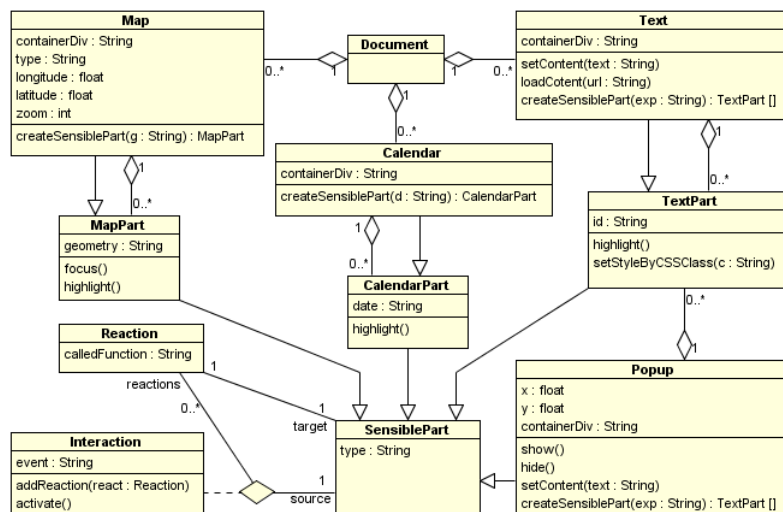


Fig 2: The WIND UML model

### 2.3 Code of a WIND Interaction

The programming languages used for mapping tools play an important role. JavaScript is a programming language for the interaction of HTML and CSS objects in web page; it allows programmers to create new HTML elements, to modify them, to remove them, to associate them with CSS attributes and to deal with events without refreshing the whole web page. Taking advantages of JavaScript, WIND is supported by an API allowing programmers to implement each interaction described at a conceptual level. This API proposes a homogeneous layer built on lower level APIs,

specialized in the handling of textual, cartographic and calendar elements. Thus, the WIND functions allow programmers to handle cartographic elements and their associated interactions using open source API such as OpenLayers or proprietary API like IGN Geoportail API. Programmers, however, only handle the functions of the WIND API<sup>1</sup>, they do not need to know the technical features of the lower level APIs called by WIND.

The WIND API integrates the whole model concepts and allows programmers to write their code handling the concepts of interactions, reactive areas, events and system reactions. The WIND API is implemented according to an object-oriented approach allowing programmers to handle each interaction as an object made up of reactive zones, events and system reactions. Thus, coding an interaction deals with building an interaction object composed of a reactive area object (which becomes sensitive to a user event) and one or more system reaction objects.

As an example, the implementation of the interaction described in the section 2.1 has been coded following four main steps:

*Step 1: Creating the application components.*

```
// Building the text component
var t = new WIND.Text('mytext');
t.setContent("March 20th 2009: we are in Anglet today.
Anglet is located in the southwest of France.");

// Building the map component
var m = new WIND.Map('mymap', {'type': "Google_Street",
'longitude': -1.51, 'latitude': 43.49, 'zoom': 11});
```

*Step 2: Creating the reactive areas.*

```
// Building the text sensible parts
var tp = t.createSensiblePart('Anglet');

// Building the map sensible part
var mp = m.createSensiblePart("POLYGON((-1.52 43.53,-
1.58 43.46,-1.44 43.48,-1.52 43.53))");
```

*Step 3: Creating the system reactions.*

```
<style>
.town {
  color : blue; font-weight : bold; font-size : 15pt;
  font-style : italic; background-color: pink; }
</style>
var r1 = new WIND.Reaction(mp, 'highlight');
var r2 = new WIND.Reaction(tp, 'setStyleByClass.town');
```

*Step 4: Creating the interaction.*

```
var il = new WIND.Interaction(mp, 'click', null);
il.addReaction(r1); il.addReaction(r2);
il.activate();
```

---

<sup>1</sup> The WIND API was completely coded from scratch by ourselves. We are currently studying to release it under a BSD license.

### 3 Added-value of the WIND Model

WIND has five main characteristics:

- WIND is **integrative**.

By design (*cf.* requirements of web educational interactions taking advantage of route narratives), WIND provides developers with different interaction components. The map is not the central component, neither is the text nor the calendar component. These components play an equivalent role. The user can include within a web application as many instances of each type of component, and we think that WIND could benefit other types of applications in addition to educational ones.

WIND is also integrative because it offers an overlayer to be compatible with several mapping providers (*e.g.* Google Maps, Bing Maps, IGN Geoportail) without requiring more programming skills from the users.

- WIND is **object-oriented**.

WIND has been designed to take advantages (especially, encapsulation and inheritance) of object-oriented programming language (hence its UML model). The WIND objects are simply created by their class constructor. Methods are disposed to implement relationships between classes. For example, the *createSensiblePart* method of each class (*Map*, *Text/Popup* or *Calendar*) implements the aggregation relationship between the *Map* class and the *MapPart* class or the *Text/Popup* class and the *TextPart* class or the *Calendar* class and the *CalendarPart* class; the *addReaction* method implements the association relationship between the *Interaction* class and the *Reaction* class.

The highlight of the WIND model is interaction programming. For this purpose, the *SensiblePart* class plays an important role. It is not only a source reactive area of interaction but also a target reactive area of reaction. Thanks to polymorphism of the *SensiblePart* class (*e.g.* the *MapPart*, *TextPart*, *CalendarPart* and *Popup* classes inherit from the *SensiblePart* class), the WIND model enables programming interactions for whatever reactive areas. Moreover, interaction programming always has the same structure.

The WIND model allows programmers to add classes inherited from the *SensiblePart* class. These subclasses allow diversifying the reactive areas for interaction. Moreover, the WIND model allows enriching methods of the *Reaction* classes that return visualization as well as improving *event* attribute of the *Interaction* class.

- WIND is **executable**.

WIND is not a contemplative design model [16, 17]. Thanks to its API, it is an executable web interaction model that enables fast prototyping techniques: developers can rapidly design an interaction and immediately assess it. All instances of the model can be transformed to executable codes. It is very simple to execute because the WIND API has hidden the complex methods. Each reaction is applied to each reactive area thanks to the *calledFunction* attribute when calling the *Reaction* constructor. To simply implement it, the *calledFunction* attribute has a *String* type. For example, if the *calledFunction*

attribute is “*highlight*”, then the reactive area will be applied by the *highlight* method. For this first prototype, we have only supported a few methods for the *SensiblePart* classes.

The interaction is executable when it is applied by the *activate* method. The reactions defined for the interaction were registered to system and will be executed when an event happens on the reactive area defined for this interaction.

- WIND is **declarative**.

Google Maps and OpenLayers programmers will notice that a WIND application has a very simple structure: the JavaScript code of a web interaction includes no conditional or loop statement; there is no event-manager to program. All these elements are devoted to the WIND JavaScript API, and the developer/designer only has to declare sensible parts, events, and reactions.

However, at the present time, using the API to encode interactions still requires some programming skills. Our aim is to enable end-users (particularly teachers) to design such interactions. To reach such an objective, we need to provide them with a dedicated simple authoring environment. The solution that we advocate consists of designing editors that can produce an XML file describing the interaction to be programmed. Currently, we have implemented a first prototype of our toolset that can parse any WIND-compliant XML file in order to generate interactions that the WIND API can execute (*cf.* for example the XML file at <http://erozate.iutbayonne.univ-pau.fr/wind/model.xml> and its use by an XML parser to produce a web application that the user can exploit: <http://erozate.iutbayonne.univ-pau.fr/wind/generator.php?file=model.xml>). Such examples show that WIND is declarative.

- WIND promotes **lightweight programming**.

The WIND API was programmed with JavaScript language. WIND is executed on the client-side (it is compatible with any up-to-date web browser). This characteristic is quite important because we want teachers to be able to produce WIND applications by themselves and we know that most teachers do not have the skills to install and to maintain server-side frameworks like GeoDjango [18].

Such WIND capabilities are highlighted by the demonstration available at <http://erozate.iutbayonne.univ-pau.fr/wind/>. This figure shows the user-interface provided to the end-user: it is an extension of the interaction explained in the section 2.1 that exhibits three interacting components: a text, a calendar and a map.

The implementation of these interactions remains simple (*cf.* source code below) owing to the characteristics of WIND. This application was implemented following four main steps:

1. defining the application components (a text, a calendar and a map);
2. defining the reactive areas of each component;
3. defining the possible system reactions;
4. building the interactions upon previously defined reactive areas and system reactions.

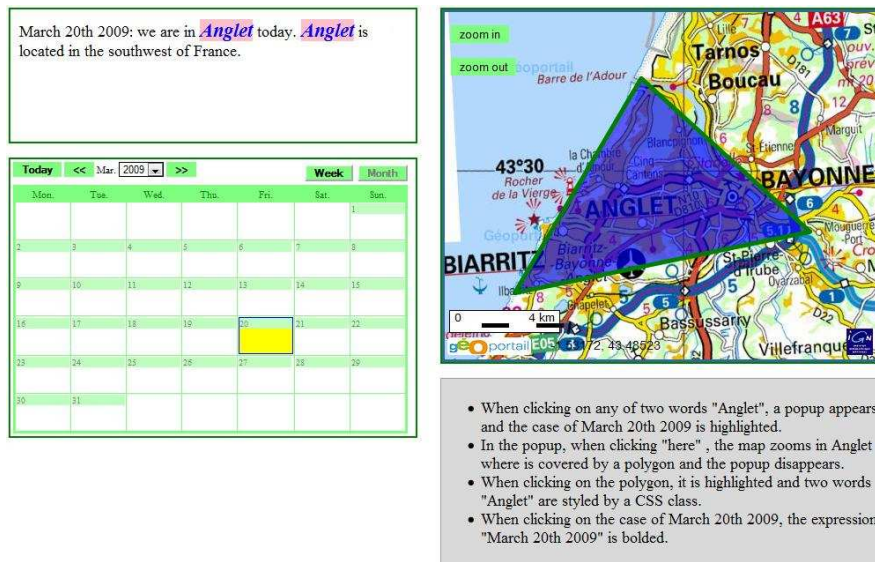


Fig 3: The user interface of a web-based application designed with WIND

```
// DEFINING THE APPLICATION COMPONENTS - Step 1
// Text component
var t = new WIND.Text('mytext');
t.setContent("March 20th 2009; we are in Anglet today. Anglet
is located in the southwest of France.");
// Map component
var m = new WIND.Map('mymap', {'type': "IGN_Street",
'apiKey': '3*4*****5*4*', 'longitude': -0.24,
'latitude': 44.4, 'zoom': 6});
// Calendar component
var c = new WIND.Calendar('mycalendar');

// DEFINING THE REACTIVE AREAS - Step 2
// Text reactive areas
var tp = t.createSensiblePart('Anglet');
var tp1 = t.createSensiblePart('March 20th 2009');
var p = new WIND.Popup(650, 240, 'mypopup');
p.setContent("Click here to zoom in Anglet.");
var tp2 = p.createSensiblePart('here');
// Map reactive area
var mp = m.createSensiblePart("POLYGON((-1.52 43.53,-1.58
43.46,-1.44 43.48,-1.52 43.53))");
// Calendar reactive area
var cp = c.createSensiblePart("2009-03-20");

// DEFINING THE SYSTEM REACTIONS - Step 3
var r1 = new WIND.Reaction(mp, 'focus');
var r2 = new WIND.Reaction(mp, 'highlight');
var r3 = new WIND.Reaction(tp, 'setStyleByClass.town');
var r4 = new WIND.Reaction(p, 'show');
var r5 = new WIND.Reaction(p, 'hide');
```

```

var r6 = new WIND.Reaction(cp, 'highlight');
var r7 = new WIND.Reaction(tp1, 'bold');

// BUILDING THE INTERACTIONS - Step 4
var i1 = new WIND.Interaction(tp, 'click', null);
i1.addReaction(r4); i1.addReaction(r6);
i1.activate();
var i2 = new WIND.Interaction(mp, 'click', r2);
i2.addReaction(r3);
i2.activate();
var i3 = new WIND.Interaction(tp2, 'click', r1);
i3.addReaction(r5);
i3.activate();
var i4 = new WIND.Interaction(cp, 'click', r7);
i4.activate();

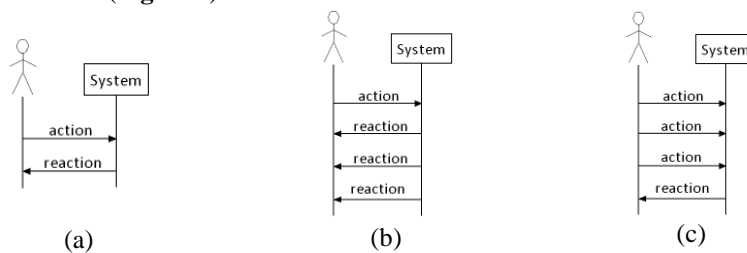
```

The *createSensiblePart* method of the *Map* class can take input a geometry string as WKT (well-known text) format and parse it automatically. It is important to underline that interactions can be implemented in any order. An interaction becomes active and executable after the call of the *activate* method.

## 4 Discussion and Future Directions

We presented WIND, a computational model for interaction lightweight programming of geographical web applications. We showed that the WIND model is fully executable thanks to the API that we provide to developers. This API enables developers to describe interactions between Map / Text / Calendar components without having to know anything about the underlying Web Mapping Services (IGN Geoportail API, Google Maps API, OpenLayers API, etc.).

We are currently working on the following improvements. In this current version of WIND, we only consider interactions in which the system reacts immediately to the user's action. A system reaction can be either simple (**Fig 4 - a**) or composed of several reactions (**Fig 4 - b**).



**Fig 4:** Different kinds of interactions

We have identified three main areas for improvement:

- The WIND model does not allow developers programming interactions where the system reacts after a sequence of several specific user actions (**Fig 4 - c**). To take into account this kind of interaction, we plan to improve the WIND model by

integrating the concept of an aggregated event. An aggregated event would be triggered when the user raises a set of events (ordered-set, unordered-set, etc.).

- We also plan to extend the reactive area concept because, currently, the WIND model only considers areas of low semantic level. Defining a reactive area on a map only consists in defining a point, a line or a polygon but we plan to extend the reactive area definition by introducing higher level concepts such as city or river for a map area. This work consists of specializing the *SensiblePart* class and improving the WIND API in order to integrate geographical methods allowing to localize higher level areas on a map.
- We also consider that the model could integrate some temporal dimension of both system reactions and user events. In the first case, this means proposing temporal operators allowing the programmer to define the chronology of system reactions resulting from a user event. In the second case, the goal consists in using these temporal operators to specify temporal sequences of user events that will define an aggregate event able to launch one or more system reactions.

Such improvements would be very useful to describe educational interactions.

Finally, one of our main aims is to enable teachers (i.e. *end-users*) to take into account a particular route narrative, from its first interpretation by the PIIR (*Prototype Interprétation Itinéraires dans des Récits*) toolset<sup>2</sup> (cf. <http://erozate.iutbayonne.univ-pau.fr/Nhan/piir/>) to the design of a learning scenario, taking advantage of the geographic semantics embedded in such a document. This will require further improvements of our toolset and will lead us to implement a visual programming environment using the WIND API.

## Acknowledgments

This research is supported by the French Aquitaine Region (project n°20071104037) and the Pyrénées-Atlantiques Department (“*Pyrénées : Itinéraires Educatifs*” project).

## References

1. Kerski, J. J.: The Implementation and Effectiveness of Geographic Information Systems Technology and Methods in Secondary Education. *Journal of Geography*, 102 (3), pp. 128-137 (2000)
  2. Demirci, A.: Evaluating the Implementation and Effectiveness of GIS-Based Application in Secondary School Geography Lessons. *American Journal of Applied Sciences*, 5 (3), pp. 169-178 (2008)
  3. Murray, T.: Hyperbook Features Supporting Active Reading Skills. Chapter 8 in *Web-based Intelligent e-Learning Systems: Technologies and Applications*, pp. 156-174 (2005)
  4. Nodenot, T., Loustau, P., Gaio, M., Sallaberry, C., and Lopisteguy, P.: From electronic documents to problem-based learning environments: an ongoing challenge for educational
- 
- 2 P. Loustau's thesis had already provided our research group with this toolset for the French language [19, 20].

- modeling languages. In 7th International Conference on Information Technology Based Higher Education and Training, pp. 75 – 86 (2006)
5. Card, S. K., Moran, T. P., Newell, A.: The psychology of human-computer interaction. Lawrence Erlbaum Associates, Hillsdale (1983)
  6. Hartson, H. R., Siochi, A. C., Hix, D.: The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems*, Vol. 8, N° 3, pp. 181–203 (1990)
  7. Paterno, F.: *Model Based Design and Evaluation of Interactive Application*. Springer (1999)
  8. Wagner, F.: *Modeling Software with Finite State Machines: A Practical Approach*, Auerbach Publications (2006)
  9. Petri, C. A.: *Fundamentals of a theory of asynchronous information flow*. Proceedings IFIP Congress (1962)
  10. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *ACM SIGCHI Bulletin*. Vol. 24, N° 1, pp. 32-37 (1992)
  11. Krasner, G. E., Pope, S. T.: A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*. Vol. 1 (3), pp. 26-49 (1988)
  12. Nigay, L.: *Conception et modélisation logicielles des systems interactifs: application aux interfaces multimodales*. Thèse de doctorat, Université Joseph Fourier, Grenoble 1 (1994)
  13. Gaio, M., Sallaberry, C., Etcheverry, P., Marquesuzaà, C., Lesbegueries, J.: A Global Process to Access Documents'Contents from a Geographical Point of View. In: *Journal of Visual Languages and Computing*, ISSN:1045-926X, pp. 3-23 (2007)
  14. Beaudouin-Lafon, M.: *Designing Interaction, not Interfaces*. In: *Proceedings of International Working Conference on Advanced Visual Interfaces*, pp. 15-22 (2004)
  15. Albinola, M., Baresi, L., Carcano, M., Guinea, S.: Mashlight: A lightweight mashup framework for everyone. In *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web* (2009)
  16. Bock, C.: UML without Pictures. In: *IEEE Computer Society Press*, Vol. 20 (5), pp. 33-35 (2003)
  17. Mellor, S., Balcer, M.: *Executable UML - A Foundation for Model-Driven Architecture*. Addison-Wesley (2002)
  18. Geodjango – A world-class geographic web framework. <http://geodjango.org/>
  19. Loustau, P., Nodenot, T., Gaio, M.: Spatial decision support in the pedagogical area: From travel stories to geocoded itineraries. In: *Proceedings of 6th International Conference on Human System Learning*. Toulouse, France, pp. 14-21. Europa IEEE. (2008)
  20. Loustau, P., Nodenot, T., Gaio, M.: Spatial decision support in the pedagogical area: Processing travel stories to discover itineraries hidden beneath the surface. In the 11th AGILE International Conference on Geographic Information Science, pp. 359-378 (2008)