Journal of Universal Computer Science, vol. 19, no. 1 (2013), 110-131
submitted: 16/8/11, accepted: 28/12/12, appeared: 1/1/13 © J.UCS

An Aspect-Oriented Approach for Spatial Concerns in Web Applications

Matias Urbieta, **Gustavo Rossi**, **Silvia E. Gordillo**
(LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{murbieta,gustavo,gordillo}@lifia.info.unlp.edu.ar)

Armanda Rodrigues, **Joao Araujo**, **Ana Moreira**
(CITI, Departamento de Informática, FCT/Universidade Nova de Lisboa, Caparica, Portugal
{ar, ja, amm}@di.fct.unl.pt)

Abstract: The growing availability of on-line geographical information, since the advent of open map servers in the 2000s, originated a new generation of Web applications, those which combine "conventional" Web functionality with typical features of traditional Geographic Application System (GIS). The rapid growth in number and complexity of Web applications with geo-referenced data together with the need to support fast requirements change, demands for increased modularity. The volatility of some of these changing requirements, both in the scope of their geographic nature or in the period of time in which they are valid, stresses the importance of the applications' modularity. A solution is to take into consideration the crosscutting nature of these requirements and decouple their realization from "conventional" requirements in separate software modules.
This paper proposes an end-to-end Aspect-Oriented approach to deal with spatial requirements from the early stages of applications development throughout to implementation. A significant contribution of this approach is the characterization of the most common spatial requirements in Web-GIS applications. The result is the improvement of the overall application's modularity, thus facilitating its evolution.

Keywords: spatial concerns, aspect-oriented software development, Web application
Categories: D.2.1, D.2.2, D.2.10, D.2.13

1 Introduction

The development of map APIs to add geospatial components to Web indexing services (e.g., Google Maps, Open Street Maps), along with the widespread availability of Global Positioning devices, has led to an increase in the availability of applications with geographical features [Chow, 08]. These applications belong to diverse domains, including environmental applications[1,2], local government[3], territorial planning [Rodrigues, 09] and citizens' government[4]. Social networks and applications that share contents in the Web have also began to exhibit these features,

[1] Drought Monitor, http://drought.unl.edu/DM/monitor.html
[2] USGS Portal, www.usgs.gov
[3] New York Finantial Digital Tax Map. http://gis.nyc.gov/dof/dtm/mapviewer.jsf
[4] Google Flu Trens, http://www.google.org/flutrends/

e.g., Flickr[5], Youtube[6]. This increasing use of maps has enriched community knowledge.

In some cases, the introduction of geographical features to a Web application can be achieved in a transparent way, by decoupling and isolating the geographic features, for example using mashups [Yee, 08]. Mashups are black-box service-based solutions that help implementing most of the application logic describing the interaction of components, particularly the geographical one. Unfortunately, these solutions many times do not work. This happens when we need to add geographical functionalities not included in the black-box services (e.g. as provided by Google maps) and therefore we cannot consider the spatial features as completely monolithic.

As we show in Section 2, including geographical features in Web applications pose two challenging problems to designers. The first one is related with the very nature of geographic functionality which involve algorithms and data that should be designed to evolve according to the requirements without compromising maintenance. The second problem lies in the impact of these features in design and/or implementation layers of the Web application, such as the user interface or the underlying business objects. The difficult in modularizing geographical functionality leads to introduce behaviour related to those features in several layers

Summarizing, we face two different though related research problems:

- Geographical functionalities and data not only evolve with requirements but they need to be designed to make this evolution seamless.
- These functionalities affect different aspects of Web applications and as a consequence cannot be just considered as black-box services which can be transparently added in this software.

We exemplify the previous problems in Section 2 with a motivating example and in Section 3 with a more analytical discussion.

These phenomena has been already studied in the broader field of software design The existence of spread code describing a certain concern or behaviour among several modules is known as *crosscutting* and two of its major issues are known as *scattering* and *tangling* [Filman, 04] in aspect-orientation. It has been shown [Filman, 04] that scattering and tangling compromise maintenance and evolution. In the context of our research we have found that several of the added geographic requirements to Web applications are crosscutting in nature.

Unfortunately aspect-oriented techniques have not been widely used so far in our target fields (Web applications and Geographical Information Systems) and have been completely ignored as a way to solve the problems resulting in the integration of the two fields. While a further discussion is left to Section 6 on related work we can state here that most approaches to use aspects to deal with geographical functionality are focused on implementation rather than design or requirements issues. In the Web arena meanwhile there are some aspect-oriented development approaches which work well with "conventional" multimedia data but fall short to capture the richness of the integration of geographical features in Web applications.

This paper has two outstanding contributions to solve the previously exposed problems:

---

[5] Flickr, http://www.flickr.com/map/

[6] YouTube, http://www.youtube.com

- We first propose a characterization of geographic requirements in Web applications according to their impact on the application's architecture.
- We propose an aspect-oriented approach to deal with geographic requirements uniformly throughout the development cycle.

To make the presentation concrete and practical, the characterization is exemplified with real cases and the aspect-oriented approach is illustrated with Web applications that use open cartographies (such as Open Street Maps [Open, 11]), where developers can introduce new features, such as custom path-finding algorithms, without the limitations (e.g., a fixed API, restricted information access, and reduced extensions mechanisms) found in proprietary formats (Google Maps).

The remaining of this paper is structured as follows. Section 2 motivates the research problem with a real example. Section 3 presents our characterization of spatial concerns. Section 4 starts with a background section on aspect-oriented software development and follows by describing the major steps of the aspect-oriented approach to deal with geographic concerns. Section 5 illustrates the proposed approach and concerns characterization through a case study. Section 6 presents related work in engineering complex GIS applications and highlights our contribution with respect to those works. Finally, Section 7 concludes this work discussing the lessons learned, our main conclusions and some future work.

2 Motivating Example

This section discusses how new and unexpected requirements impact an application's architecture at different levels, from basic algorithms to user interface features.

Suppose that we need to develop a Web-based delivery service application for mail and package distribution providing some kind of package tracking. To improve the system functionality, we decided to add geographical features to support path routing report. A path routing report is a plan, used by a messenger, which provides the shortest paths from the company to the package target or from one package delivery address to the next. Once a delivery request is registered, customers can login on the company web application to track package delivering in real-time. Since we need to have open access to cartography and algorithms, we choose to implement some of the geographical features and use open services instead of relying on proprietary applications (which in fact provide some similar support) like Google maps.

To make the example more concrete, let us suppose that we are in Montpellier, France, and we want to deliver a package from Place de la Comédie to L'Antigone. For planning a package delivery, the system must resolve the shortest route between both points. Fig. 1 shows the map of the area using the outdoor representation and two possible paths to reach the target. While the slashed path uses the streets and, therefore, is calculated using the road network, the dotted path shows a shortcut that crosses a shopping centre, shortening the distance to the target. However, the latter implies a combination of outdoor and indoor representations since it requires passing inside a building. For simplicity, in this example, we do not consider path properties which may lead to the preference of one path over others, such as how fast is the path or which kind of packages can be delivered over it.

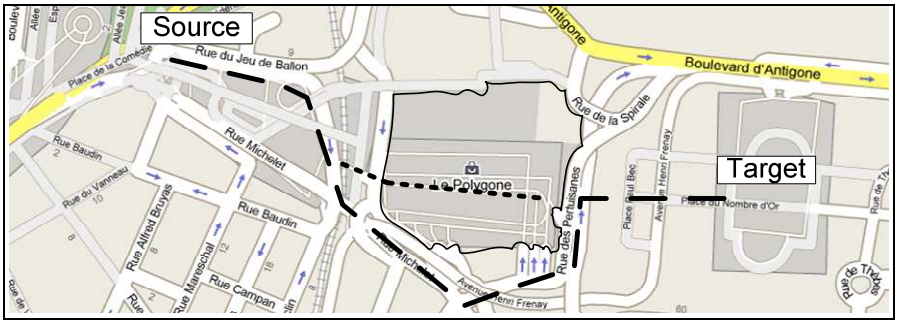

Figure 1: Available paths from Place de la Comédie to L' Antigone

If indoor paths were not planned when the application was built, we need to improve it by establishing a new criterion for the path search that takes into account indoor paths (corridors, lifts, stairs, etc.). In this way, the "best" path from the current point of sales to the target (using the shorter distance criteria of the well-known path-finding A* algorithm [Schiller, 04]) may be a mixed solution, involving the road network, as well as indoor paths going through buildings. Eventually, it might happen that the messenger must pose additional requirements considering time, vehicle, etc.. These requirements stress one specific aspect of path finding algorithms; more specifically, when layers are modelled as a graph with *nodes* linked by *edges*, the introduction of this new functionality affects the way in which a node resolves its available edges. The scenario can be more complex when the system evolves, requiring some kind of integration with shops' timetable services to determine in real-time whether some paths are available. The impact is very precise: there is a specific part of a geographical object (a graph) behaviour that must be frequently edited to realize these new requirements.

Indoor paths-finding features will be tangled with other path resolution features such as public transportation, city roads, etc., and scattered on each place a path is resolved such as graph nodes which calculate adjacencies.

In some of these examples, it can be argued that the required functionality may be obtained by building new data layers as in GIS applications [Longley, 05]. The new layer hosts the map "irregularity" (e.g., blocked streets), and the graph will be obtained by a complex process of layers composition. To implement this solution, the data composition logic and the corresponding user interface features must be introduced; without a suitable methodology, these changes will be detrimental to the software maintenance. It is often impossible to frame these "irregularities" in static layers, as when applications use third-party Web services that provide this information dynamically, i.e., when the graph structure is computed on the fly. This is the case when we want to consider problems such as accidents (e.g., in the public transportation system), which are informed in a service invocation basis.

3 Spatial concerns in Web applications

In software engineering a concern represents a matter of interest that groups a coherent set of requirements [Sutton, 02]. In this context, we consider a spatial concern as a special kind of concern that refers to the set of requirements that deal with geographical features of an application, such as locations captured by a Global Positioning System (GPS), maps (indoor and outdoor) and routing algorithms. For this analysis, we assume the availability of typical and basic geographical resources [Longley, 05] such as *Maps*, *Graphs* and *Layers*, provided and managed by an open source geographical server such as OpenStreetMap [Open, 11].

3.1 Typical Spatial Concerns

This section characterizes the five most common types of spatial concerns, classified based on the impact they have on the underlying application. These types of concerns are: spatial business object, rich spatial data, spatially-constrained behaviour, map adjustments, and geographic interfaces. For each type we provide guidelines to facilitate their identification and simplify their realization in Web software. These guidelines are presented in a template composed of three fields: Name (as title), Description & Example, and Impact. The examples will focus on the type of applications described in Section 2, although most of the discussion can be applied to a broader range of applications.

As shown later in Section 5, we propose that these concerns be used as patterns, for achieving modularized design solutions through the reuse of their specification and implementation recommendations.

3.1.1 Spatial Business Object

Description & Example. To enhance applications adding some sense of location to business objects, as well as the corresponding spatial functionality to manipulate this location. For instance, a bus service management system can be improved by providing real-time bus locations, offering more precise and timely information to managers and passengers. This requires adding location and estimated arrival time to the original bus stop map.

Impact. This problem involves the introduction of spatial operations (to compute distances) and location information to describe the object's geographical features. The latter may consist in enriching the object with latitude and longitude variables or with a variable pointing to a full-fledged location entity. In the example of the Bus object that becomes location-aware, we might have tangling code because new presentation logic is demanded for adding a map to the Bus information view (and the same happens in every similar example) and scattered code. Additionally, to the operation that returns the current bus's position, new business logic may be appended to compute the arrival time to a given bus stop. Moreover, if we have several location-aware objects, such as ferries or taxies, with the same code for supporting this functionality becomes scattered. The introduction of location-aware requirements also demands new user interface widgets for presenting the location information in a map.

3.1.2 Rich Spatial Data

Description & Example. To enrich a geographical object with information describing non-spatial characteristics and to provide ways to manipulate this information. A typical example arises when adding videos to a specific location on a map.

Impact. Already defined data structures must be modified or new ones may need to be created to support this new kind of functionality. Additionally, the application's linking structure may be modified. Finally, presentation issues must be adjusted (e.g., by introducing rich interaction support). We might also experience tangling when including the logic that enables adding or removing movies to/from specific locations.

3.1.3 Spatially-Constrained Behaviour

Description & Example. To modify the behaviour of an object according to its actual location. For example, in a real estate agency, different houses are shown in a city map (the houses are business objects enriched with spatial information). As customers get interested in the taxes to be paid when owning a house, the agency must add a new behaviour that computes the tax according to the geographical area where the house is located. The situation might get more complex if some tax is only applied during a certain period of time (e.g., as a consequence of an unusual meteorological event).

Impact. The introduction of new (geographically constrained) business logic may either involve a complete set of methods or enhance already defined ones. These changes may generate tangled code, when combined with existing algorithms, such as other taxes, and may generate scattered code, when the same logic is introduced within different objects types. For example, when a zone suffers a flood, the government may promote a tax reduction policy for those fields and buildings affected by the catastrophe. The behaviour for computing the tax may be scattered in those objects responsible of tax computation. In an object-oriented model, the Field and Building objects could be responsible of their own tax computation. Alternatively we could delegate the computation to strategy objects but we would still have a conditional statement on location variables to configure the strategy.

3.1.4 Map Adjustments

Description & Example. To extend or restrict the available spatial information according to the application's constraints. The spatial data available in a Web application may be restricted or extended according to a specific concern, which may imply that certain parts of a map are unavailable or are useless for specific operations or services. These types of extensions or restrictions may be temporal or permanent. When geographic objects un/availability is temporal, status calculation can be done in real-time and may require collaboration with other artefacts, components, or external systems. This type of concern may arise in (at least) two different ways: augmentation, and restriction of the available geographical data.

Impact. A concrete and very direct impact arises in algorithms that manipulate spatial data; specifically, the respective requirements lead to changes in path-finding features. For example, in the case of adding new transportation services, computing node adjacencies (segments in a graph) should change, since new edges, such as

public transportation and highways, must be taken into consideration. This introduces scattered and tangling code at each place where a path is resolved. In graph-based representation of streets, the feature introduces scattered code each place where a node is asked for its adjacencies, because each search algorithm defined in the application is affected by the requirement. In the case of blocked or unavailable streets, the graph is modified, by removing the corresponding edges. Applying graph changes directly to the corresponding database may be unmanageable when the nature of changes is volatile. For example, the responsibility for determining whether a given street segment must be pruned or not can be delegated to a third-party service provided by the highway administration. In this case, it is unreasonable to modify the database because the information is volatile and only available on demand.

3.1.5 Geographic Interfaces

Description & Example. To modify or upgrade the user interface of geographic objects. Though not strictly a spatial concern, it is clear that most of the previously cited examples might introduce changes in the application's user interface, specifically in the geographic objects (e.g., maps). Adjusting the spatial dataset availability ("Map Adjustments") are changes related to spatial objects' behaviour. However, as previously mentioned, a lack of a suitable presentation may produce a misleading perception of the application functionality. For example, if blocked streets are taken into account in a path search, they must be appropriately visualized in the map (i.e. using a standard red colour).

Impact. The presentation layer needs to be improved in order to provide a proper presentation for spatial concerns such as: map widgets when enabling spatial behaviour to business objects ("Spatial Business Object"); and labelling availability/unavailability reason of spatial objects ("Map Adjustments").

3.2 Discussion

Although the classification presented does not intend to be comprehensive, it covers an exhaustive set of common types of concerns related with geographic information in the context of Web software. The following conclusions can be drawn: (*i*) most concerns, if not all, introduce code tangling or scattering in other concerns. Indeed, some specific concerns are more affected than others (e.g., path finding and manipulation). Consequently, if we want to avoid the well-known problems that arise when dealing with tangled or scattered code [Filman, 04], a good alternative is to use an improved modularization mechanism to allow us to keep these concerns separated, and treat them independently during the development process; (*ii*) some concerns are volatile, as it was shown in the Map Adjustments examples. Volatile concerns should be kept separated from core concerns, to facilitate evolution. Aspect-oriented mechanisms can also be used to support their modularization [Moreira, 06]; (*iii*) the classification presented aims at providing reusable solutions, once the concern is identified and matched against this concern characterization.

4 Our Approach in a nutshell

4.1 Background

We next outline some background concepts helpful to understand our approach. For conciseness, we reference the reader to the most important literature in each field.

4.1.1 Aspect-Oriented Software Development

Abstraction, modularization and encapsulation are crucial principles to achieve separation of concerns [Dijkstra, 76] in any type of systems. Geographic Information Systems are no exception. Traditional Software Engineering approaches (e.g., object-oriented) cannot modularize broadly scoped properties, such as response time, visualization and persistence. Typically the specification and implementation of such concerns that do not align well with the main decomposition criteria end up spread across many modules and thus tangled with modules that address other concerns. This *crosscutting* phenomenon is not limited to non-functional requirements; functional requirements can also often cut across parts of a system [Rashid, 06].

Aspect-Oriented Software Development (AOSD) [Kiczales, 97] provides mechanisms to identify, modularize, represent and compose crosscutting concerns [Rashid, 03]. Crosscutting concerns are encapsulated in separate modules, known as *aspects*, and composition mechanisms are later used to weave them back with other core modules, at loading time, compilation time, or run-time [Baniassad, 04]. The result is a reduction of software development complexity and improvement of understandability, minimizing the impact of change through encapsulation of different concerns in separate modules [Kiczales, 97]. The main concepts included in AOSD are *aspects*, *joinpoints*, *pointcuts* and *advices*. A *joinpoint* specifies a well-defined point in the execution of the base program that will be affected by an aspect, such as a method call, an access to a variable, etc.. A *pointcut* specifies a set of joinpoints and data associated to them. An *advice* defines code that can be executed when a joinpoint is reached in the program execution.

From the various existing aspect-oriented modelling approaches, we have chosen MATA [Whittle, 07] to support modelling and composition in our approach.

4.1.2 MATA

MATA (Modelling Aspects Using a Transformation Approach) is a representative of UML structural and behavioural modelling development with aspects, considers aspect composition as a special case of model transformation, and uses some of the mechanisms provided by Pattern Specifications(PS) [France, 04], like roles, described next. It supports a very good set of expressive composition types, when compared to other approaches (e.g., Jacobson's aspectual use cases [Jacobson, 04]). For example, an aspect sequence diagram can be composed with a base sequence diagram, using parallel, alternative and loop fragments as part of the composition rule. Most of other approaches have often been limited to the AspectJ advices (i.e., before, after, around).

The composition mechanism of MATA is based on graph transformations. A graph transformation is a graph rule r: L → R from a left-hand side (LHS) graph L to a right-hand side (RHS) graph R. In MATA the composition of a base model, Mb,

with an aspect model, Ma, which crosscuts the base, is specified by a graph rule, r: LHS →RHS:

- A pattern is defined on the left-hand side (LHS), capturing the set of points in Mb where new model elements should be added;
- The right-hand side (RHS) defines those new elements and specifies how they should be added to Mb.

MATA supports composition for several UML diagrams (e.g., class, sequence, activity and state diagrams). Its graph rules are specified in the UML's concrete syntax, with some extensions to allow for more expressivity in the compositions. The following stereotypes are used in MATA rules given in a diagram:

- «create»: applied to any model element, specifying the creation of an element.
- «delete»: applied to any model element, specifying the deletion of an element.
- «context»: used with container elements that are created; it avoids creating an element inside a created element, forcing it to match an element in the base.

Examples of MATA usage will be found in section 5 when describing approach's instantiation.

4.1.3 Pattern Specifications

Pattern Specifications (PSs) [France, 04] are a way of formalizing the reuse of models. The notation for PSs is based on the Unified Modelling Language (UML). A pattern specification describes a pattern of structure or behaviour defined over the roles which participants of the pattern play. Role names are preceded by a vertical bar ("|"). A PS can be instantiated by assigning concrete modelling elements to play these roles. A role is a specialization of a UML metaclass restricted by additional properties that any element fulfilling the role must possess. Hence, a role specifies a subset of the instances of the UML metaclass. A model conforms to a PS if its model elements that play the roles of the PS satisfy the properties defined by the roles. Thus, a conforming diagram must instantiate each of the roles with UML model elements, multiplicity and other constraints. Note that any number of additional model elements may be present in a conforming diagram as long as the role constraints are maintained. As in [Whittle, 04], we extend the notion of pattern specification from that of [France, 04] by allowing both role elements and concrete modelling elements in a PS. This provides greater flexibility in reuse as often one may wish to reuse a partially instantiated model rather than a model only containing role elements.

4.2 An Overview of the Approach

This sections presents end-to-end approach outlined in Fig. 3, covering activities from requirements analysis to implementation. This approach particularly focuses on requirements evolution, where classification, composition and instantiation are the most important tasks to develop a highly evolvable system. During the design process, crosscutting spatial concerns will be modelled using MATA**.** First, a conventional requirement gathering is done by means of meetings with stakeholders producing a set of use cases. Next, the relationship among core and spatial requirement is realized in a crosscutting matrix that, later on, will help designers to get a better modularization in designs. After specifying requirements, for example with use cases of both kinds, base and aspectual, these must be modelled using class

and sequence UML diagrams with MATA. These diagrams establish a connection between elements of core and (crosscutting) spatial use cases.
Finally, these models are translated to the underlying aspect-oriented programming language (In this paper, and in our case studies, we used aspectual extensions of existing languages such as Java and Python).

This approach consists of height major steps, each one briefly introduced next. As Step 3 plays a fundamental role in the process and deals mostly with spatial issues, we describe it in detail in Section 5.

Step 1: **Identify and specify core concerns.** To identify core concerns of the problem domain we use traditional requirement gathering techniques for identifying stakeholders' needs and relate these in coherent groups, so called concerns. To identify the requirements, we can use, for example, interviews, elicitation requirements workshops, or use any information available about the system. Each concern can be then described using several different techniques, instantiating a particular template such as those in [Moreira, 06]. We can also use viewpoints or use cases, for example, if we look for a more standard technique from the very beginning.

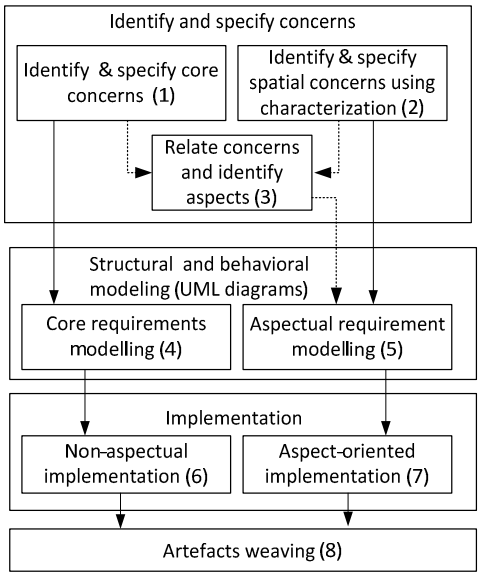

Figure 2: Overall process to develop web applications with spatial concerns

Step 2: **Identify and specify spatial concerns using characterization.** As in step 1, a spatial concern is elicited using traditional requirement gathering techniques, and later documented. The characterization process aims at classifying spatial concern depending on the impact they introduce in the application. The spatial concern is checked against each spatial concern characterization (Section 3.1) through a pattern matching. There are cases where the studied concern can be mapped to more than one concern

characterization (e.g., due to its complexity or the granularity of its description). In this case, the concern must be decomposed.

Step 3: **Relate concerns and identify aspects.** Detect possible crosscutting by studying occurrences of spatial concerns in each use case of steps 1 and 2, and build a relation matrix describing crosscutting relationships [Conejero, 09]. For each use case, analysts must identify and identify the presence of concern requirements that cause tangled behaviour assigning a value (0 or 1) to the jointpoint cell in the Table 1. The jointpoint cell v_{ji} in the table specifies that $Concern_i$ affects $UseCase_j$.

Tangling behaviour is detected when the sum of weights is greater than one. On the other hand, when a concern is present in several use cases, there is likely a scattered behaviour. This information is marked with an S (for scattered), and C (for Core), as illustrated in last column of Table 1.

The number of concerns that crosscut a given use case is the metric we use to help detecting aspects that can be modularized in an aspect hierarchy and thus be reusable by extending the hierarchy. In this hierarchy, an abstract aspect defines common elements such as pointcut definition and behaviour shared by all crosscutting concerns, and each aspect specialization encapsulates specific behaviour in an advice. For example, suppose two new different concerns such as "Public demonstration" and "Ferry line" (see Section 3.1.4); as they reduce and augment available spatial data, they crosscut the path-finding use case (used to query paths) producing enhancements in Nodes adjacencies computation for filtering or appending adjacent Nodes. When detecting this scenario earlier, designers will be able to model an abstract aspect that defines a common pointcut (the specific place in Node object behaviour where the advice must be introduced) and routines, and extend the abstract aspect with specifications containing custom behaviour for augmenting spatial data in the case of "Ferry line" concern and reduce the available data for "Public demonstration" concern.

		Use cases			
		$UseCase_1$	$UseCase_2$	....	
Concerns	$Concern_1$	v_{11}	v_{21}	v_{i1}	[C, S]
	$Concern_2$	v_{12}	v_{22}	v_{i2}	[C, S]
	$Concern_3$	v_{13}	v_{23}	v_{i3}	[C, S]
	…	v_{1i}	v_{2i}	v_{ji}	[C, S]
Concern count per UC		$\sum_{k=0}^{i} v_{1k}$	$\sum_{k=0}^{i} v_{2k}$	$\sum_{k=0}^{i} v_{jk}$	

Table 1: Matrix describing crosscutting

Step 4: **Core requirements modelling.** Model concern structure and behaviour using UML structural diagrams (e.g., classes, components) and UML behavioural diagrams (e.g., sequence diagrams) using the heuristics of

[Moreira, 06].
Domain objects, such as Cars, Building, and Orders, as well as their relationships are modelled using UML class diagrams, constituting the base models. Application architecture is specified using component diagrams that package application concerns, and complex scenarios are configured using deployment diagrams.

Step 5: **Aspectual requirements modelling.** Use the MATA approach, introduced in Section 4.1.2, to describe flexible aspect's advices to eliminate tangling or scattering behaviour introduced by spatial concerns. Detected crosscutting behaviour in step 3 is modelled using MATA extensions for class and sequence UML diagrams. Tangled and scattered behaviour is modelled in such a way duplicate behaviour is encapsulated reducing application complexity. New class variables, methods and relationships are reflected in a UML class enhanced with MATA rules. After the weaving process, the new elements are merged with the base class models. The new behaviour is also modelled with MATA extensions producing a set of rules that configures the way in which the behaviour is merged in the base sequence diagrams.

Step 6: **Non-aspectual implementation.** Implement the modelled components obtained in step 4 using standard programming languages (such as Java, JavaScript, Python, etc.).

Step 7: **Aspect-oriented implementation.** Implement aspect advices using aspect-oriented languages (e.g., AspectJ [AspectJ, 11] for Java, Aspyct [Aspyct, 11] for Python). Pointcuts are defined from the diagrams obtained in step 5 and later refined with each relationship between use cases and spatial concerns available in the matrix built in step 3.
Following models produced in step 5, tangled and scattered geographic behaviour is coded in aspect artifacts.

Step 8: **Artefacts weaving.** Finally a weaving activity, which can be realized either at compilation or run time, will merge core and spatial concerns to obtain a complete application. This is usually performed by the underlying implementation technology such as AspectJ compiler or Python virtual machine.

All these activities will be further illustrated in Section 5, after we explain how to identify and specify spatial concerns.

5 Illustrating the approach with a case study

This section describes the application of our approach to the delivery system example introduced in Section 2. For the sake of conciseness we specifically focus on spatial concerns.

5.1 Identify and specify concerns (steps 1 and 2)

This activity starts by eliciting requirements using traditional techniques, such as interviews and workshops, and then follows by grouping related requirements in concerns. Table 2 presents a set of core requirements that the application must meet and lists requirements grouped in coherent sets, which basically relate to concerns. Some of them define the Delivery, Office, Route planning and Transportation concerns.

Let us now introduce some new realistic geospatial concerns. As a consequence, the system's designers will have to introduce these new unexpected and possibly volatile concerns that increase the system's complexity. Table 3 lists some new functional requirements extending the system's functionality, where column "Type Ref." (described in Section 3.1) characterizes the requirement that provides additional information to designers, for the next steps.

Concern	Req. ID	Description
Delivery	R1	An employee records a delivery service request.
	R2	For each package, a label with a unique id is assigned.
Office	R3	The user can check delivery office points in a map.
	R4	A new office point can be added.
	R5	Additional information can be added to office points.
	R6	Office points can be printed as a report.
Route planning	R7	A route plan is elaborated for the truck driver or biker, which describes the sequence of street segments s/he must take into account for delivery.
	R8	The truck driver can visualize the route plan as a sequence of street segments and the streets on a map.
Transportation	R9	The system manages transportation vehicles (trucks and bicycles) for package delivery.

Table 2: Basic requirements of the system

Concern	Req. ID	Type Ref.	Description
Delivery tracking	R10	3.1.1	A customer, using issued id, can query the package location in real-time.
	R11	3.1.5	The package location is shown in a company dashboard.
Blocked streets	R12	3.1.4	For express delivery service, the route plan computation must take into account blocked streets to avoid unsuitable plans.
	R13	3.1.5	The user interfaces and the report must describe the blocked streets in detail.

Table 3: Geospatial concerns set

For simplicity, we have not included the Map concern with its requirements (which enables users to manage and visualize Maps).

5.2 Relate concerns and identify aspects (step 3)

For each use case, we now detect concern requirements causing tangled behaviour. In Table 4 we can see that Delivery requirements are present just in the use case "Request delivery route", meaning that it is a core concern. On the other hand, Tracking is present at "Query package status" and at "View transportation information", meaning that the latter two are altered with new tangled logic.

When a concern is present in several use cases, we have scattering. Core concerns are marked with a C and scattered ones are marked with S (for scattered) in Table 4. The last row in this table shows the total number of requirements that crosscuts a use case, detecting candidate crosscutting behaviour. This may result in an aspect hierarchy, where an abstract aspect defines functionality shared by all crosscutting concerns, and each aspect specialization specifies specific behaviour, as explained in Section 4. Localizing abstract aspects helps designers to elaborate a model with less coupling and defining abstract aspect advices. A design arising from this concern analysis will be described later on.

		Use cases					
		Request delivery route plan	Delivery Request	Query package status	Fill delivery papers	View transportation information	
Concerns	Delivery	1	0	0	0	0	C
	Route planning	0	1	0	0	0	C
	Delivery tracking	0	0	1	0	1	S
	Blocked streets	1	0	1	0	0	S
	Concern count per UC	*2*	*1*	*2*	*0*	*1*	

Table 4: Concerns analysis by use case

5.3 Structural and behavioural modelling

In this step, models that detail concerns will be designed using the MATA approach [Whittle, 07]. For illustration purposes, we focus here on the blocked streets concern, specifically on two of its requirements: *taking into account blocked streets in route planning task* (Req. 12) and *showing blocked streets in a suitable way* (Req. 13). (Section 5.4 shows how we implement these requirements.)

Blocked streets are, for example, those streets in which traffic is not allowed in one or more segments, during a period of time.

5.3.1 Core requirements modelling (step 4)

In this step, base requirements are modelled using UML class diagrams and sequence diagrams. Fig. 3 shows a class diagram corresponding to the main business entities (Client, Package and Employee) and *Route planning* concern classes. The latter comprise Path finding resolvers that encapsulate path finding logic (in a Strategy pattern style [Gamma, 95]) and *StreetSegmentNode* that defines information for a

road segment (Street name, id, city, etc.) and address range covered (for example, address in range 100–150).

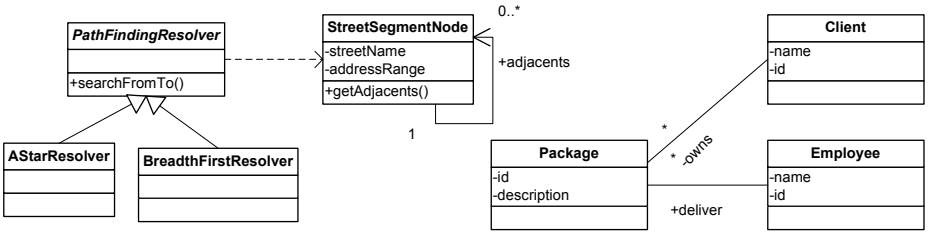

Figure 3: Application class diagram

Path finding functionality is depicted in the sequence diagram shown in Fig. 4. A path request goes through the Web interface layers up to reach a *PathFindingResolver* that collaborates with a set of *StreetSegmentNodes* for obtaining the expected path.

For simplicity, these diagrams specify only relevant classes and behaviour, which will be further discussed in the following sections.

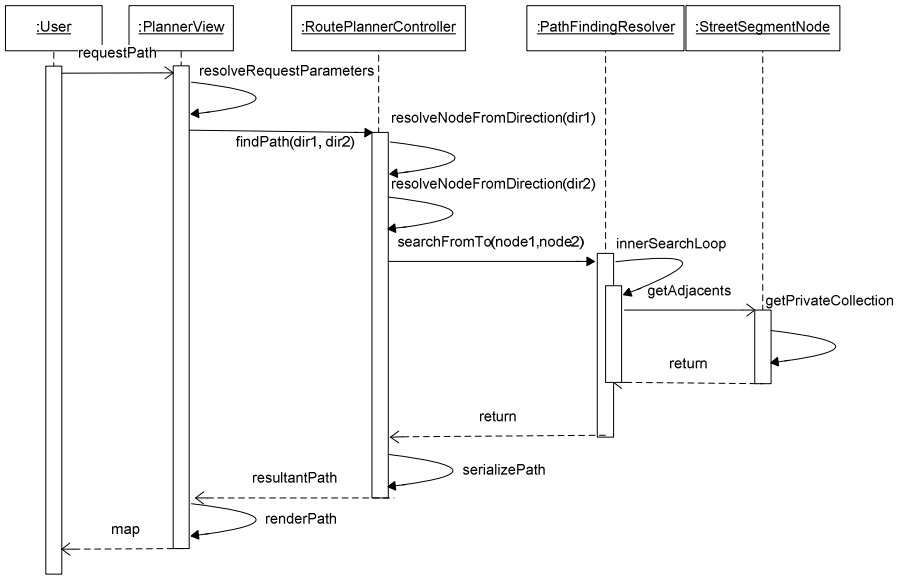

Figure 4: Sequence diagram for the Path finding requirement

5.3.2 Aspectual requirement modelling (step 5)

The main requirement of the *Blocked street* concern ("the route plan computation must take into account blocked streets to avoid unsuitable plans") aligns with the concern type "Map Adjustments" presented in Section 3.1.4, where a dataset can be augmented or reduced to include or exclude spatial data. The route plan computation

is based on a path-finding algorithm [Schiller, 04], which is supported by the use of a graph, where nodes represent the street network. The algorithm visits each node in the graph asking for its adjacent nodes, which are used for selecting the best option (in this case shortest path).

The augmentation or reduction of a dataset can be achieved by externalizing this special behaviour with aspect-oriented techniques. In this particular case, an aspect is defined to modularize the augmentation or reduction of the set of objects that are processed, as shown in Fig. 5.a. A caller object requests a list of adjacent nodes, which is annotated with a MATA context ("<<context>>") stereotype, serving as a pointcut. In the advice of the aspect, the adjacent nodes are computed by triggering the original *StreetSegmentNode's* getAdjacents method (returning all adjacent nodes) pointed with an "Any" operator, which allows that any sequence of messages can happen in the base. Next, blocked streets are filtered using the *filterBlockedStreets* method introduced by the aspect. This method uses a blocked street service (*BlockedStreetService* class), which determines if a given street segment is blocked using the *isBlocked* method. Finally, the filtered list is returned to the caller object, which does not know the aspect's behaviour.

To improve the user experience, requirement R13 demands a suitable presentation of blocked streets in the user interface; this is also accomplished using an aspect that enriches the map rendering process. Fig. 5.b shows an aspectual design for the view controller *RouteController* which decorates the renderPath method with the needed logic for rendering blocked Streets. The *renderBlockedStreet* method is invoked after the *renderPath* logic is executed, drawing a light blue line in the map for blocked streets, and adding pop-ups for describing the situation.

As Table 4 depicts, several concerns may crosscut the same use case generating tangled code. For example, the *Request delivery route plan* use case is affected by some requirements of the *Blocked Street* concern. Additionally, a new *City transportation policies* concern (not present in the table) crosscuts the use case. This allows restricting paths depending on transportation used for delivering; for example, bikes can be used for downtown deliveries since it is easy to drive in narrowed streets while trucks can be used for long distances. *City transportation policies* concern will have a solution pattern quite similar to that presented for *Blocked Street* and presented at Fig. 6.a. Notice that designers can profit from this fact to produce better and more reusable aspect designs. The pointcut definition for filtering node's adjacencies, and eventually the advice, can be generalized in an abstract aspect. Additionally, the abstract aspect may also define and introduce methods shared by each concrete aspect. A more advanced design can be attained by modelling the aspect's advice as a template method pattern [Gamma, 95] for describing shared filtering logic. In Fig. 6.a, *AbstractFilteringAspect* is modeled as a template method, defining a common behaviour for node filtering. Behaviour specialization is achieved by implementing the *processItem* method. In Fig. 6.b, we show how different aspects reuse logic defined by the abstract aspect.

For sake of space, we are not showing the composed model.

5.4 Implementation (steps 6 and 7)

This section presents a running example that implements the *Blocked Streets* concern modelled in Section 5.3. For the sake of conciseness, we will not cover the

implementation of core concerns; instead we will base the discussion on the design presented at Section 5.3.1.

The two main concern requirements (R12 and R13 in Table 3) have been implemented using aspects. The delivery component uses the A* algorithm, which is a best-first graph search algorithm that finds the lowest-cost path from a given initial node to a target node, using a heuristic function to determine the order in which the search visits nodes in the tree.

Our prototype system is a Web Application that allows users to search paths presenting the result in a map. The process is based on algorithms from the PyRoute Library [PyRL, 11] project, which consumes information from an OpenStreetMap [Open, 11] server. Following the rationale of our approach, the previously described aspectual models must be implemented using an aspect-oriented language. Our solution is divided into two parts: the aspect that introduces the notion of blocked segments within the search, and the controller that manages the introduction of the suitable presentation of blocked segments.

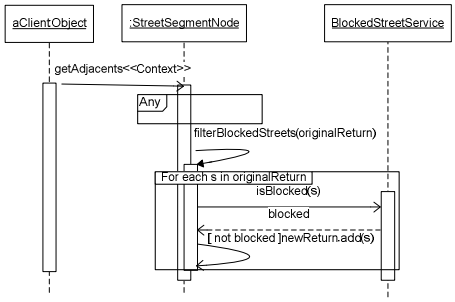

Figure 5.a: Blocked Street filtering aspect design

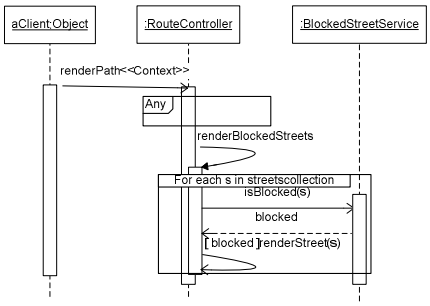

Figure 5.b: Aspect scenario for a suitable blocked street presentation

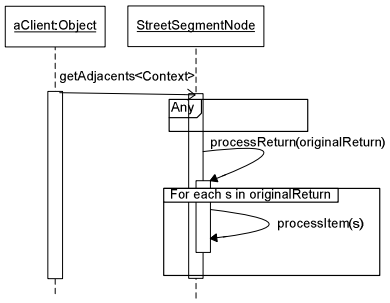

Figure 6.a: Abstract advice for street adjacency filtering

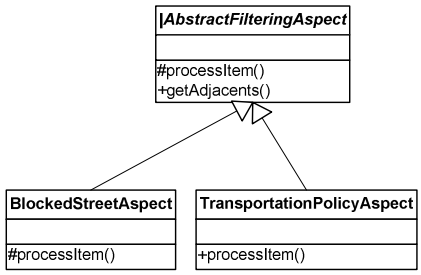

Figure 6.b: Aspect hierarchy for Street adjacency filtering

5.4.1 Processing blocked street segments

The algorithm works over a directed graph where each node is visited to reach the endpoint. When it is not reached, the algorithm traverses adjacent nodes repeating the

process once and again until the end point is reached. In this context, the aspect, which encapsulates the *Blocked street* concern, adds the needed behaviour after the original algorithm performs the search of the potential adjacents. This behaviour is achieved by wrapping the ***#getAdjacents*** (returns all the adjacent nodes of the current one**)** method in the *StreetSegmentNode* class. Thus, when the algorithm stops in a particular segment, it checks if the next segment corresponds to a blocked street. To do this, it builds (using the #getAdjacents method) a collection including all the possible segments that have not yet been traversed to reach the aimed destination. When this collection is returned, the aspect filters the blocked segments, using the blocked street service. Then, the aspect returns this new collection and the original algorithm continues normally. It worth to note that this geographical crosscutting feature is encapsulated in an aspect following its design taking advantage of AOP benefits. This aspect allowed reducing duplicated code present each point in the application where blocked streets should be computed.

5.4.2 Presenting blocked street segments

Blocked streets must be clearly presented to the user, allowing him/her to understand the result at a glance. The application uses the OpenLayers [OpenLayers, 11] framework, which provides facilities for rendering maps by overlapping different graphic layers. In our case, the application overlaps the basic map (in one layer) with the resolved path (in another layer). In order to implement the design presented in Section 5.3.2, we must introduce the logic for rendering street segments.

The solution for rendering blocked streets was applied as a new map layer (from now on the *blocked streets layer*) which is overlapped over both the basic map and the calculated path layers.

Furthermore, taking into account that presentation layers are usually defined using XML derived documents such as HTML, the solution section of the concern type proposes to use interface transformations for applying the changes. In [Ginzburg, 07] we presented an oblivious approach for introducing aspects in the interface Web tier, by means of user interface transformation. Using this approach, user interface structures and behaviour related to the blocked street concern are included in a map view, through EXtensible Stylesheet Language Transformation (XSLT).

Therefore, new widgets, labels, event handlers, etc. are appended to the map view without being aware of the changes produced by the *blocked street* concern.

Additionally, the single map controller, on the server-side, was enriched with the logic for handling requests generated by the *blocked street layer,* on the client side. The enrichment was applied using an intercepting aspect, which captures each request coming from the browser; it performs custom processing to map tiles requests coming from the *blocked street layer*, for generating the corresponding tile.

As a result of combining these two solutions, it is possible to introduce visual aspects in the application, keeping user interface and server side components free of tangling and scattered code.

6 Related work

Separation of concerns [Parnas, 72], as a research area, has provided an invaluable contribution to software engineering, improving modularity and, therefore, maintenance, evolution and reuse of applications. Our work proposes a refinement of some techniques of Aspect-Oriented Software Development (AOSD) [Filman, 04] to address the specificities of Web-GIS applications (applications that combines both Web and GIS features). We aim at providing better ways to isolate these kinds of application concerns and provide different types of support for the late weaving of software components, which realize these concerns.

In [Zipf, 03] the authors discuss the use of aspects in GIS applications. However, they only focus on the programming level, which aligns with our work. Our approach goes further by providing evaluation tools for analyzing spatial concerns in the early stage of the development process.

Web Modeling Language (WebML) has provided a language extension [Di, 07] to its visual formalism for modelling navigation, which comprises GIS operations. By arguing that Web-GIS is a particular class of data-intensive Web applications, they have provided different types of units [Ceri, 00] that allow presenting and dealing with geographic information. The extension demands a declarative specification of GIS features, without means for describing GIS crosscutting concerns in a seamless way.

AMACONT [Niederhausen, 09] is a Web design framework with orthogonal facilities for extending functionality in a transparent way by implementing some AOSD concepts. The tool can be used for developing applications that use the approach described in our research, since MATA designs can be translated to AMACONT aspects in a straightforward way.

In [Gordillo, 99], design patterns [Gamma, 95] have being used for providing solutions to recurrent problems appearing in the GIS domain. The proposed approach allows designers to decouple the conceptual definition of application objects, from their spatial representations, through the application of different patterns. Our research uses these ideas by first identifying recurrent problems (see Section 3) and then complementing them with a software engineering approach, which allows detecting GIS crosscutting concerns earlier, during the requirement gathering step.

In [Digital, 08], the authors propose a catalogue of common functionalities that can be used for defining a basic Web-GIS application. Nevertheless, the set of requirements fail to address the last generation of Web-GIS applications, which are saturated of tangled concerns. Additionally, it provides an implementation solution for merging third-party GIS engines with proprietary business models, but lacks a comprehensive approach to address the whole development process.

7 Concluding Remarks and Further Work

We have presented a novel approach to develop complex Web GIS applications involving spatial behaviours, such as those dealing with maps. This approach uses concepts of aspect-oriented software development to clearly separate the spatial concerns from other core application's concerns.

By carefully studying recurrent cases, we have defined a characterization that describes a "catalogue" of the most usual crosscutting spatial concerns defining their intent, scope and a solution according to their impact on the application. In order to support these crosscutting spatial concerns, our approach first identifies these concerns on the early steps of the software engineering process, and provides conceptual tools to isolate them and describe them separately from other application components. Spatial components are later weaved onto the core components with the typical aspect-oriented approach, therefore supporting seamless evolution and reuse of these isolated components. We have applied this approach on an open map servers, and in this paper we illustrated the use of our approach with a realistic case study, showing the whole process from design to implementation, even describing how we adapted a path search algorithm by extending it with aspects.

We are now working on several research lines related with this approach. Firstly, we are extending the characterization of spatial concerns to use it as a pattern-like catalogue that can help developers to reuse the solutions in the catalogue when similar cases arise. This work involves providing different "implementations" for the same pattern; for example, we are also describing heuristics to map our aspect-oriented solutions to "conventional" GIS software. These heuristics show how an aspect can be mapped for example to a Layer according to the type of problem being solved. A further research line in this direction is to include a more detailed analysis of the consequences of this kind of solutions, e.g. evaluating performance issues, which were not relevant in all our case studies but might arise in some critical Web-GIS software projects.

Acknowledgements

This work has been partially funded by CITI grant PEst-OE/EEI/UI0527/2011 and the project PO-07-09 between Mincyt (Argentina) and FCT MCTES (Portugal). The authors also want to thank Mr. Lucas Hahn and Mrs. Alejandra Lliteras who worked on implementation issues.

References

[AspectJ, 11] AspectJ http://www.eclipse.org/aspectj/, Accessed 4 June 2011.

[Aspyct, 11] Aspyct, http://www.aspyct.org/ , Accessed 4 June 2011.

[Baniassad, 04] Baniassad, E., Clarke, S.: Theme: An approach for aspect-oriented analysis and design, 26th International Conference on Software Engineering (ICSE), Scotland, 2004.

[Ceri, 00]Ceri, P., Fraternali, P., Bongio, A.: Web Modeling Language (WebML), A Modeling Language for Designing Web Sites, Computer Networks and ISDN Systems, 33(1-6), 2000.

[Chow, 08] Chow, T. E.: The Potential of Maps APIs for Internet GIS Applications, T. GIS 12(2): 179-191, 2008.

[Conejero, 09] Conejero, J. M., Hernández, J., Moreira, A., Araújo, J.:Adapting Software by Identifying Volatile and Aspectual Requirements, JISBD 2009:103-114, 2009.

[Di, 07] Di Martino, S., Ferrucci, F., Paolino, L., Sebillo, M., Vitiello, G., Avagliano G.: A WebML-Based Approach for the Development of Web GIS Applications. Web Information Systems Engineering (WISE),2007.

[Digital, 08] Digital Earth Summit on Geoinformatics: Tools for Global Change Research. International Journal of Digital Earth, 1: 1, 171 — 173.

[Dijkstra, 76] Dijkstra, E.: A Discipline of Programming. 0-13-215871-X. Prentice-Hall, 1976.

[Filman, 04] Filman, R., Elrad, T., Clarke, S., Aksit, M.: Aspect Oriented Software Development. Addison Wesley, 2004.

[France, 04] France, R., Kim, D., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique, IEEE Transactions on Software Engineering, Volume 30(3), 2004.

[Gamma, 95] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of reusable object-oriented software. Addison Wesley,1995.

[Ginzburg, 07] Ginzburg, J., Rossi, G., Urbieta, M., Distante, D.: Transparent Interface Composition in Web Applications, 7th International Conference on Web Engineering (ICWE2007), July, Italy, pp. 152-166,2007.

[Gordillo, 99] Gordillo, S. E., Balaguer, F., Mostaccio, C., Das Neves, F.: Developing GIS Applications with Objects: A Design Patterns Approach. GeoInformatica 3(1): 7-32, 1999.

[Jacobson , 04] Jacobson, I., Ng, P.W.: Aspect Oriented Software Development with Use Cases. Addison-Wesley Professional, 2004.

[Kiczales, 97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-Oriented Programming, 11th European Conference Object-Oriented Programming (ECOOP'97), Finland, LNCS 1241, Springer, pp.220-242,1997.

[Longley, 05] Longley, P.A., Goodchild, M.F., Maguire, D.J. and Rhind, D.W.: Geographic Information Systems and Science. Chichester: Wiley. 2nd edition, 2005.

[Moreira, 06] Moreira, A., Araújo, J., Whittle, J. Modeling Volatile Concerns as Aspects, CAiSE, pp. 544-558, 2006.

[Niederhausen, 09] Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., Meißner, K.: Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for AMACONT, ICWE 2009 Conference, San Sebastián, Springer, pp. 106-120, 2009.

[Open, 11] OpenStreetMap, http://www.openstreetmap.org/, Accessed 4 June 2011.

[OpenLayers, 11] OpenLayers, http://openlayers.org/, Accessed 4 June 2011.

[Parnas, 72] Parnas, D. L.: On the Criteria To Be Used in Decomposing Systems into Modules, Commun, ACM (CACM) 15(12):1053-1058, 1972.

[PyRL, 11] PyRouteLib, http://wiki.openstreetmap.org/wiki/PyrouteLib, Accessed 4 June 2011.

[Rashid, 06] Rashid, A., Moreira, A.: Domain Models Are NOT Aspect Free, Proc. MoDELS, ACM/IEEE, pp. 155-169, 2006.

[Rashid, 03] Rashid, A., Moreira, A., Araújo, J.: Modularisation and Composition of Aspectual Requirements, ACM International Conference on AOSD, pp. 11-20, 2003.

[Rodrigues, 09] Rodrigues, R., Rodrigues, A.: Spatial Operators for Collaborative Map Handling, Groupware: Design, Implementation, and Use, Proc. of the 15th International Workshop, CRIWG 2009, Peso da Régua, Douro, Portugal, LNCS, Springer-Verlag, 2009.

[Schiller, 04] Schiller, J., Voisard, A.: Location-Based Services. Morgan Kaufmann, Elsevier. USA. ISBN: 1-55860-929-6., pp. 67, 2004.

[Sutton, 02] Sutton, S., Rouvellou, I.: Modeling of Software Concerns in Cosmos. Proc. of ACM Conf. AOSD 2002, pp. 127-133, ACM Press 2002,

[Whittle, 04] Whittle, J., Araújo, J.: Scenario Modeling with Aspects, in IEE Proceedings Software, Vol. 151, no. 04, 2004, pp. 157-172,2004.

[Whittle, 07] Whittle, J., Moreira, A., Araújo, J., Rabbi, R., Jayaraman, P., Elkhodary, A.: An Expressive Aspect Composition Language for UML State Diagrams, Int. Conference on Model Driven Engineering, Languages and Systems (MODELS), Springer, 2007.

[Yee, 08] Yee, R.: Pro Web 2.0 Mashups: Remixing Data and Web Services, Apress, Berkeley, California, 1-59059-858-X, 978-1-59059-858-0, February, 2008.

[Zipf, 03] Zipf, A., Merdes, M.: Is aspect-orientation a new paradigm for GIS development?, 6th Agile Conference on Geographic Information Science, Lyon, 2003.