A Pattern for Context-Aware Navigation

MIHAELA CARDEI, BRANDON JONES, DANIEL RAVIV, Florida Atlantic University

Navigation systems nowadays have become ubiquitous, with many users relaying on them to find on-the-fly directions to new locations. With current advances in technology, modern navigation systems augment navigation directions with new information ranging from traffic, tolls, accidents and any other issue that could lead to delays in a given route, to social network aspects such as schedule, friends, to weather alerts. This paper presents an architectural pattern for context-based navigation, that describes an abstract way to integrate context information with the navigation process. This pattern applies to user navigation in any environments where context is relevant, not only to driving navigation. Someone could apply this pattern for driving navigation, indoor navigation, navigation using public transportation, robotic navigation, and so on. An example with contextual navigation on a university campus illustrates how the pattern can be used.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**] Software Architectures–Patterns; D.5.1 [**Programming Techniques**] Object-oriented Programming; D.4.4 [**Communications Management**] Network Communication; D.4.6 [**Security and Protection**] Access Controls

General Terms: Design

Additional Key Words and Phrases: Contextual Navigation, Pattern, Architecture, Policy.

1. INTRODUCTION

The recent boom in wireless technology has led to an even greater demand for always available real time data. One such demand is easy and quick access to accurate navigation data. These data are expected not only to lead one to the intended destination, but to also consider factors such as traffic, tolls, and accidents into the route computation. Furthermore, with the age of social media in full swing, the demand for more personalized information has increased [Tscheligi and Sefelin 2006]. The level of assistance expected for navigation has spread from mere directions to much more. As the success of the Android [Android Operating System] application *Waze* [Waze Navigator] has demonstrated, the user now expects information such as a friend's location. Real time user generated content, such as speed traps and accidents are also expected.

There are many applications and commercial devices that provide driving directions and navigation such as Waze [Waze Navigator], Google Navigation [Google Maps], in-car navigation, Magellan navigation devices [Magellan Smart Gps], and Garmin navigation devices [Garmin Navigation]. The trend is for these to integrate more contextual information with the navigation process. For instance, Google Navigation integrates current traffic information, while Waze also offers a social networking aspect allowing users to report traffic events, exchange short messages, and see each other on a map.

This paper presents a solution for the common problem of integrating contextual information for mobile user navigation applications. The pattern consists of an architecture for context-based navigation system, and specifications for the map, policy, and context-based information, which abstracts the way in which various contextual factors can be added in the system, based on the designer's specifications. It is important to note that this pattern

This work was supported in part by the NSF grant IIP 0934339.
Author's address: 777 Glades Road, Boca Raton,FL. 33486.
email: mcardei@fau.edu, bjones2013@fau.edu, ravivd@fau.edu.

applies to user navigation in any environments where context is relevant, not only to driving navigation. Someone can apply this pattern also for indoor navigation, navigation using public transportation, robotic navigation, multi-player immersive computer games, and so on.

An example with contextual navigation on a university campus illustrates how the pattern can be used.

2. THE CONTEXT-AWARE NAVIGATION PATTERN

Intent

The intent of this pattern is to integrate contextual information in applications assisting mobile user navigation in physical environments. We define *context* to be information about the user and the environment, beyond the map.

Example

Consider the typical college or university campus. Every semester many new students begin their academic career in a new and unfamiliar location. Obviously, an application providing campus navigation would be useful for the students. Computing navigation directions that would account for campus events, such as a graduation ceremony or a football game, would reduce trip delay and user frustration. However, there is much more to the campus life. The student would benefit from extra information surrounding the campus. Examples would include personalized class information, event information and any campus police announcements. Social media aspects, such as a friend's location, would likely be appreciated by most users. The issue is how to integrate such contextual information into the campus navigation process.

Context

Consider a user equipped with a terminal with a wireless network connection and a position sensor. The user needs navigation directions and navigation assistance to get from its current location to some destination location. Finding directions to the desired destination and keeping the user on the trajectory must depend on user context, map context (e.g. traffic signs), and external event context (e.g. accident reports).

Problem

There are many factors that affect navigation. How can we integrate context into navigation? The problem addressed is to structure the information necessary to provide context-aware navigation in a generic way and to provide an architecture for context-based navigation.

The possible solution is constrained by the following forces:

—Adaptability: Events can happen at any time. Although some events will be known in advance, the system must be able to accommodate an influx of spontaneous events.

—Functionality: The system must provide a means for generating navigation data, depending on the surrounding context.

—Security and privacy: The user's personal data must be secured. Access to data and event updates must be restricted to only those authorized.

—Scalability: The system must be scalable with respect to the number of users, map size, and context size.

—Extensibility: It must be possible to add new contextual information without affecting the existing architecture.

—Usability: The system must be user friendly and usable in various scenarios.

—Efficiency: The system responds to contextual changes in a timely manner.

—Cost: The cost of the system must be relatively small, otherwise, the system would be infeasible to implement and operate.

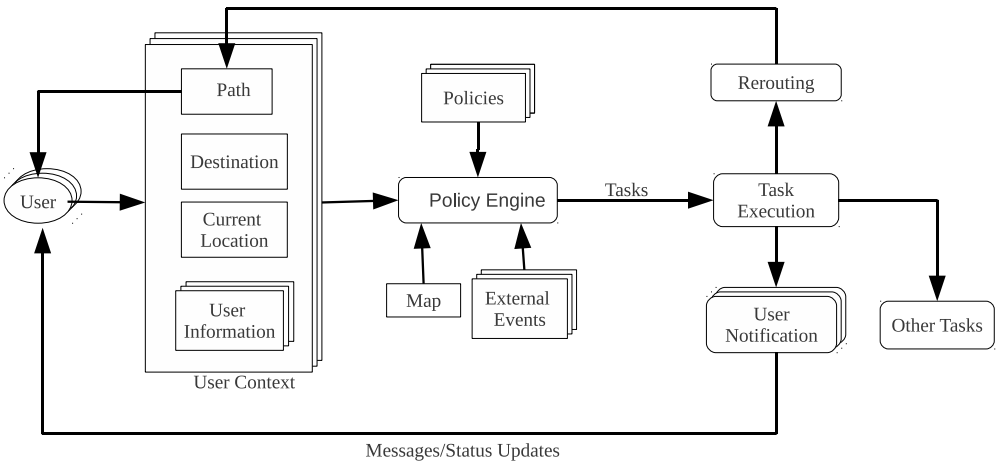


Fig. 1. Block diagram of the software architecture for context-aware navigation

Solution

Figure 1 contains a block diagram of the software architecture for the context-aware navigation, consisting of several core components:

—Mobile *users* use a wireless terminal and a client navigation application to get directions to some destination location.

—*User context* contains user information, current location, destination, and the path. User information contains authorization level, preferences of the user, and social media context (e.g. friends' location). The path represents the current navigation path provided to the user. Current location is provided by a sensor (e.g. GPS for outside location) or a localization service (e.g. for indoor localization).

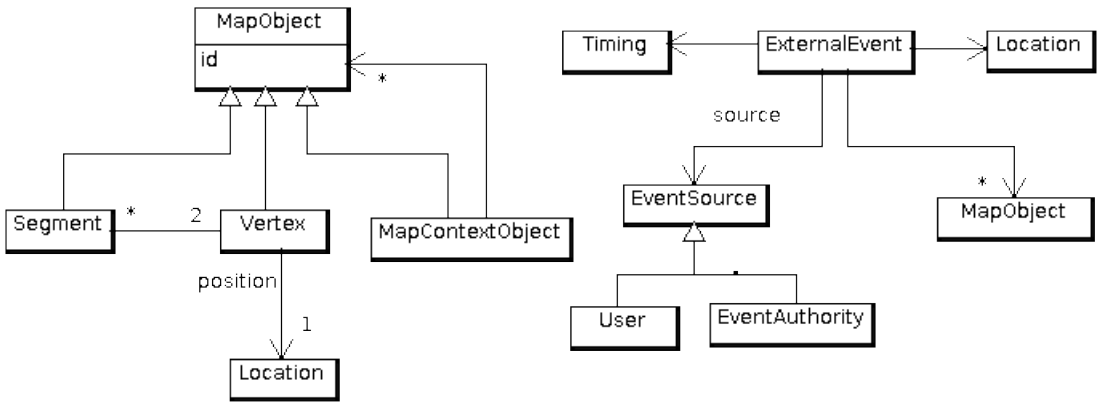
—The *map* consists of one or more graphical layers of the navigation environment (e.g. satellite picture) and a graph data structure (e.g. vertices and segments) that define the navigable trajectories for the user. The map graph imposes constrains on where the user can move (e.g. roads for driving or corridors for walking inside a building). The map also includes the map context information related to specific contextual information such as traffic signs.

—A *policy* is a set of rules describing a set of tasks to be executed when a set of conditions are satisfied. Policies basically specify how to react in certain cases, e.g. if a certain event is triggered.

—*External events* represent contextual information that have timing and location, and that are relevant to the navigation process, e.g. traffic events, accidents, parades, road work, etc.

—The *policy engine* takes input from several sources: the path, the user's current location, external events, the map, and any user context. These data are used for decision making against each policy and tasks are executed based on the currently defined policies and rules.

—*Tasks* are executable commands set out by the policy engine. Some examples of tasks are user notification (e.g. a nearby road accident, a friend location in the proximity) and rerouting (e.g. request to recalculate the path due to a recent event on the current trajectory). Tasks are executed on a task engine. For example, the client displays notifications, annotates path with relevant information, and on the server side, it can notify users of certain events. Other Tasks include updating user account information(e.g. recording the route) and crowdsourcing traffic information.



a. Map objects schema

b. Concepts describing external events

Fig. 2. Class diagrams for map concepts and external events

In general, map and context data are stored in an XML format or in a database. For this pattern we describe the map data and context information ontology using the UML class diagrams in Figure 2. The map contains information needed to build navigation paths and to represent contextual information. The root of the class hierarchy is the `MapObject` class. This class is used to store attributes common to all map objects, such as an object ID. The `Vertex` class represents map objects associated with a location, such as a building, or a parking lot.

The `Vertex` class is associated to the `Location` class, which for driving navigation contains latitude and longitude coordinates.

A `Segment` object represents a section of a street, an alley, or a section of a navigation path. A navigation path is computed as an ordered list of `Segment` instances. A segment is defined by a pair of vertex endpoints.

The class `MapContextObject` identifies objects that have contextual information and that belong to the map, such as traffic signs, but are not vertices or segments. Hence, `MapContextObject` is also a subclass of class `ContextElement`, described in Figure 3. A `MapContextObject` may refer to zero or more `MapObject`s that are relevant for it. For example, a traffic sign (stop sign) map context object refers to one or more segments.

A slightly different approach to organize the map data is to rely on `nodes` for geographic points with coordinates, and on `ways` to define open or closed sequences of nodes, for streets and areas. This method is used by the OpenStreetMap project [Open Street Map Project] and Waze, with an XML data format. Contextual information is attached to map objects with tags and attributes.

The class diagram for external events is presented in Figure 2b. An `ExternalEvent` is associated to a `Timing` object that specifies when the event takes place. The event can be for example a scheduled event (e.g. a parade) or a random event that occurred at a time and has a duration, e.g. an accident event, or road work. An `ExternalEvent` object is associated to a `Location` and to zero or more `MapObject`s. For example, an accident event may be associated to a road segment object and may require closing traffic for several hours.

An `ExternalEvent` is issued by an `EventSource`, that can be a `User` or an `EventAuthority`. For example, an accident on a highway may be reported by a `User`, while a road closure due to construction may be issued by a police department `EventAuthority`.

The concepts related to policy rules, context elements, and the relationships between them are shown in Figure 3. The diagram includes classes with behavior pertaining to policy processing (policies, rules, tasks) and also subclasses of `ContextElement` that are instantiated in memory at runtime from an XML or database store using a query engine.

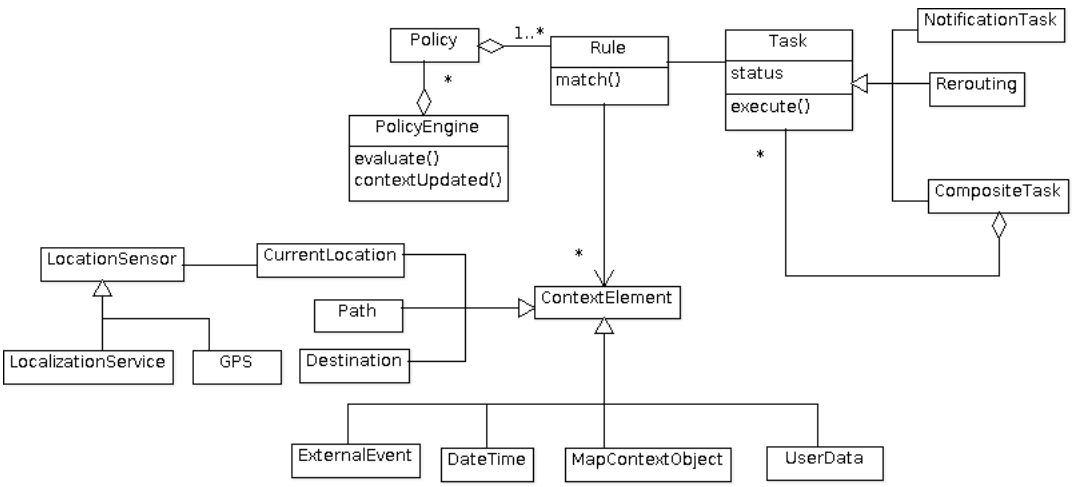


Fig. 3. Context and policy concepts class diagram

The context is described by the `ContextElement` class that can be specialized to satisfy a variety of contextual information, such as the the current time (`DateTime` instance), external events, map context objects, path, and destination. The current location is obtained from location sensors that can be either a GPS device or other `LocalizationService` for indoor navigation.

The `UserData` class is a base class for describing application-specific contextual information that relates directly to the user. This can be refined, for instance, to indicate user type (faculty/student), activities and scheduling, and a user's social network.

The `PolicyEngine` object evaluates a set of policies, where each `Policy` has one or more rules (instances of class `Rule`) that are matched against the current state of the user's context. If the context matches the rule conditions, the `Task`s associated to that rule are enacted using the *execute()* method. This is the typical mechanism for forward-chained rule-based systems (such as implemented with CLIPS or Jess).

Notification tasks involve sending data to the clients, such as messages with status updates or commands. Rerouting task involves computing a new route to the destination. A `CompositeTask` implements complex sequences of tasks involving sub-tasks using the composite design pattern.

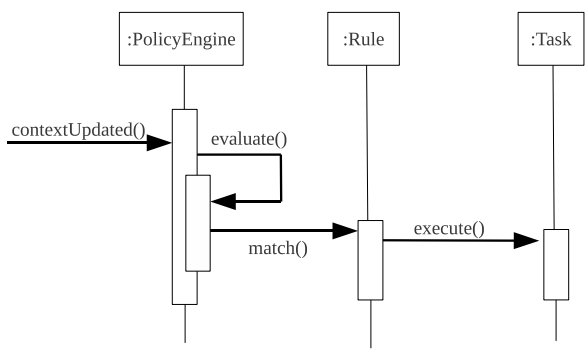


Fig. 4. Context Evaluation and Task Execution Sequence Diagram

When the context is affected by an update to a relevant `ContextElement` object, the `PolicyEngine` is activated and will evaluate applicable policies, as shown in the sequence diagram in Figure 4. For each rule that matches the current context, its task will be executed. Task execution could be distributed between the client and any service infrastructure.

Fig. 5. Overall contextual navigation flow

The generalized contextual-navigation process happens in a series of five steps, see Figure 5. First, the current location is determined using a position sensor. Based on the intended destination, the map context, user context, and external events are matched with their corresponding policies, conflicts are resolved, and tasks are generated. Conflicts can be resolved for example using priorities. Policies may support reification to specify how to resolve conflicts or inconsistencies. Possible tasks include path rerouting and user notifications.

Intended Applications

Google Maps and *Google Navigation* [Google Maps] display navigation information. As the technology evolves it has begun to incorporate various types of navigation, such as bus routes, driving directions, walking directions an biking directions. Both technologies are beginning to incorporate contextual data, such as current traffic conditions.

The Android [Android Operating System] application *Waze* [Waze Navigator] offers real-time driving navigation. It also allows user contextual input through a confirmation system. Users can report traffic accidents, traffic jams, and other events. Other users can then confirm on deny the reported event using a reputation-based system.

Implementation

Context-aware navigation can be achieved with commercially available tools and equipment. Position sensors (e.g. GPS [GPS System]) are virtually standard in all modern mobile devices, easily giving the user the ability to transmit their current location.

Tools like Google Maps [Google Maps API] are freely available and would provide a map for user reference. It allows to translate points on the map into GPS coordinates. This allows a means to link a GPS coordinate to a given event.

The navigation system needs a mechanisms to compute the trajectory to the desired destination. This mechanism depends on the map graph, contextual information and policies. Map graph provides information about the map objects, such as *Segment* and *Vertex*. Contextual information is given by user context, map context, and external events. The policies determine the impact of each contextual information.

One way to implement the shortest delay path (also called the shortest path) is to use Dijkstra's algorithm on the map graph, where policies applied to user context generate a certain weight to edges besides the Euclidean distance. For example, an accident on the highway or a street parade would increase the weight of the segments associated to that street. This would ensure the shortest path algorithm would view the area as undesirable to traverse and traffic would be detoured around the accident. Once the accident was cleared, the weights could be reduced to normal levels.

As long as the user stays on course and there is no new contextual information, there is no need to recalculate the shortest path each time. User tracking can be implemented using a small set of policies on the client side, to reduce the overhead associated to the communication with the server.

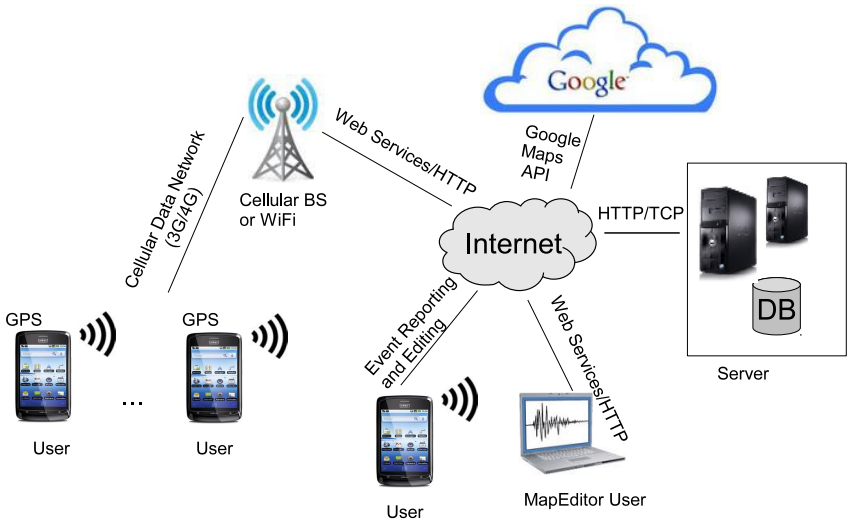


Fig. 6. Context-based Campus Navigation System

The storage of data and communication with the users could be achieved with a simple server and database combination. The map objects can be stored in an XML file on the server side. The file can be easily parsed in order to get the map graph and map data structure.

The *policy engine* could be implemented as follows:

—custom based code that implements if-then-else matching of statements. In this case, policies are binary data objects containing properties and relationships.

—Prologue or a reasoning engine (e.g. Jess [Jess Engine]). Policies are rules and the context form the facts in the knowledge base.

—XML base reasoning or query engine (e.g. RDF triples [Resource Description Framework] with SPARQL query language [Sparql Query Language], OWL language [Owl Language], or SWRL [Swrl Semantic Web Rule Language]). The policies are XML statements. For example, policies can be documented using the eXtensible Access Control Markup Language [XACML Language]. This provides both a means for setting actions, when the events would occur, and user authorization. By using XACML to note policies an open source XACML policy engine, such as Xengine [Liu et al. 2008] could be used for the processing of events and user actions. Since XACML is based on XML, any specialized elements needed for a rule could be easily created.

Example Resolved

Our research group at Florida Atlantic University has designed and implemented a prototype for the campus navigation system [Cardei 2013]. The campus navigation system is presented in Figure 6. Since Google Maps does not provide information on campus location, we have designed a Campus Map Editor tool that allows authorized users to augment the map from Google Maps Api v3 with location information of the map objects, such as buildings, parking lots, traffic signs, and so on. The campus map objects are then stored in an XML file on the server. In Figure 7, we can see the campus map graph containing vertices (e.g. red square icons for buildings, and blue square icons for parking lots) and segments (e.g. green edges for walking segments and blue edges for driving segments). The editor allows both creating a new map and managing current versions (e.g. add a new building).

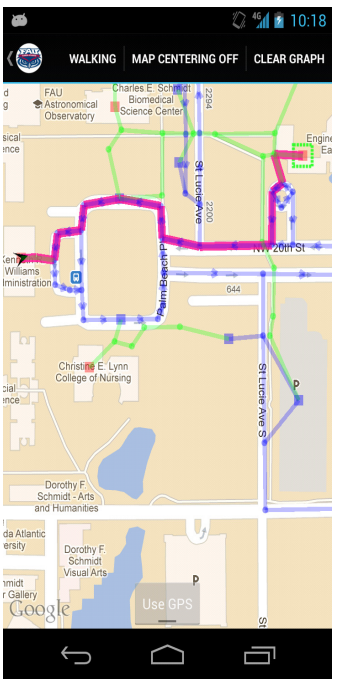


Fig. 7. Campus Navigation App

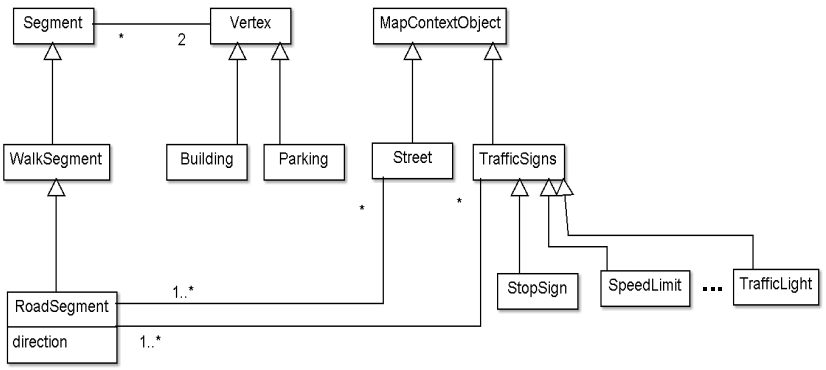


Fig. 8. Campus map class diagram

The campus map class diagram is presented in Figure 8. A group of segments form a street context object. For example if there is a football game, then the street near the stadium has high traffic. Traffic signs provide map contextual information. For example, stop signs and speed limits trigger delays on the navigation path.

The Event Reporting and Editing users add campus contextual information that is stored on the server. Besides the *map context*, *user context* and *external events* provides relevant information for the navigation path.

User context contains information such as access control list for user data (current position, path, schedule), social network data (friends, colleagues) and interests. For example, notifications are displayed on the terminal if a friend is in close proximity or if a seminar on a topic of interest is currently ongoing in a nearby building.

As specified in the Figure 2b., the *ExternalEvent* class is associated to *Location* and *Timing* classes, which contain information about where and when the event takes place. Some of the events may be random (e.g. road accident) while others may be scheduled (e.g. graduation ceremony).

For the campus navigation system, some of the subclasses of the *ExternalEvent* class are: Accident, RoadConstruction, RoadClosure, RoadObstruction, TrafficJam, SportEvent, Graduation, CulturalEvent, AcademicEvent, NaturalDisaster, and so on. The scheduled events can be edited and reported in advance by authorized users, while random events such as accident or road closure are reported by the police department. Users can also report events as they travel down the navigated path. The user reports can represent both emergency or non-emergency situations.

An issue to tackle with user reported events is a mechanism to confirm validity of the reports. Although one solution would be to merely take the reports at face valid, it could easily lead to issues. False reporting could not only lead to an unpleasant experience but possible dangerous situations if emergency events are falsely reported. An alternate approach, is a reputation-based mechanisms used by the Android [Android Operating System] application *Waze* [Waze Navigator], where a report can be confirmed or invalidated by other users, affecting the credibility of a user. Another approach is a reward-based mechanism.

The navigation application on the user terminal uses the position sensor to determine the current location and sends a request to the server to compute the navigation path. The policy engine on the server processes the request using the applicable context information (e.g. map context, user context, and external events). The server returns the navigation path and any contextual information to the user, which is then layered on top of the map from a provider such as the Google Maps service.

New context may result in rerouting or notifications which are displayed on the user terminal. Using standard networking protocols, users or authorized personnel can report events to the sever. Users can be notified of the event changes on their next path update or immediately if the event is of heightened importance.

Also using standard networking protocol, authorized MapEditor Users can access the server to add, edit or delete map objects (e.g. adding a new building).

Consequences

The context-based navigation pattern has the following advantages:

—Adaptability: The separation of policies and the policy engine allows for the addition of new policies (new features) without having to change and rebuild the system.

—Functionality: The policy engine allows integration of user's current location, intended destination, and contextual information to generate a trajectory path. New events can be reported and accounted for when the trajectory path is updated.

—Security and privacy: Policies can inform granular access to user information based on access control lists. Authentication and encryption are implemented using standard web protocols.

—Scalability: Policy matching and task execution may be distributed between the clients and a variety of distributed services running on multiple servers. The client must operate even when disconnected from the network. A reduced set of the user context could be stored (cached) on the client side. The policy matching algorithm is parallellizable and can be distributed on a cluster of servers. Task execution can also be distributed between multiple services and the client.

—Extensibility: Since the policy engine and the policy specifications are generic, new policies can be added at any time without altering the system.

—Usability: The system is user friendly and is able to respond to changes affecting the user. The system sends user notifications about the traffic conditions and other events.

—Efficiency: The system is able to respond to changes affecting the user in near real-time.

—Cost: The cost is minimal. The system only requires purchasing a server and it relies on open source software.

The context-based navigation pattern has the following disadvantages:

—In a rich and dynamic contextual environment, frequent notifications from the server may drain the battery.
—The risk of faulty user notifications must be mitigated with a reputation or reward system.

3. RELATED PATTERNS

A pattern for web-based social navigation is presented in [Riedl 2001] and further explained in [Riedl and St. Amant 2003].

4. CONCLUSIONS

This paper presents a Context-aware Navigation Pattern. The policy-based architecture presented in the pattern allows users to not only be provided with navigation, but with contextual information about their current surroundings. We presented the various components of the architecture, and show an example of using this architecture for a context-aware campus navigation system.

ACKNOWLEDGMENT

We thank our shepherd Alejandra Garrido, for her careful comments that helped improve the paper.

REFERENCES

Android operating system, http://www.android.com/, last accessed May 2013.

CARDEI, M., ZANKINA, I., CARDEI, I., RAVIV, D. 2013. Campus Assistant Application on an Android Platform In *IEEE SoutheastCon 2013*.

Garmin Navigation, http://www.garmin.com/, last accessed May 2013.

Google Maps API, https://developers.google.com/maps/, last accessed May 2013.

Google Maps, http://www.google.com/mobile/maps/, last accessed May 2013.

Global Positioning System, http://www8.garmin.com/aboutGPS/, last accessed May 2013.

Jess, the Rule Engine for Java Platform, http://herzberg.ca.sandia.gov/, last accessed May 2013.

KARIMI, H. 2011. Me-friends-web (mfw): A model for navigation assistance through social navigation networks. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*. 430–435.

KARIMI, H., BENNER, J., AND ANWAR, M. 2011. A model for navigation experience sharing through social navigation networks (sonavnets). In *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*. 557–560.

LIU, A. X., CHEN, F., HWANG, J., AND XIE, T. 2008. Xengine: a fast and scalable xacml policy evaluation engine. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. SIGMETRICS '08. ACM, New York, NY, USA, 265–276.

Magellan SmartGps, http://www.magellangps.com/, last accessed May 2013.

Open Street Map Project, http://www.openstreetmap.org/, last accessed June 2013.

Oasis, http://www.oasis-open.org, last accessed May 2013.

Owl Language, http://www.w3schools.com/rdf/rdf_owl.asp, last accessed May 2013.

Resource Description Framework, http://www.w3.org/TR/rdf-concepts/, last accessed May 2013.

RIEDL, M. O. 2001. A computational model and classification framework for social navigation. In *Proceedings of the 6th international conference on Intelligent user interfaces*. IUI '01. ACM, New York, NY, USA, 137–144.

RIEDL, M. O. AND ST. AMANT, R. 2003. Social navigation: modeling, simulation, and experimentation. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. AAMAS '03. ACM, New York, NY, USA, 361–368.

SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, last accessed May 2013.

SWRL, A Semantic Web Rule Language, http://www.w3.org/Submission/SWRL/, last accessed May 2013.

TSCHELIGI, M. AND SEFELIN, R. 2006. Mobile navigation support for pedestrians: can it work and does it pay off? *interactions 13,* 4, 31–33.

Waze, http://www.waze.com, last accessed May 2013.