This technical report is an extended version of the homonymous paper published in
**Computer Communications,
special issue on Advanced Location-Based Services,
Elsevier, Spring 2008**.

The PoSIM Middleware for Translucent and Context-aware Integrated Management of Heterogeneous Positioning Systems

Paolo Bellavista, Antonio Corradi, Carlo Giannelli
Dip. Elettronica, Informatica e Sistemistica - Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna - ITALY
Phone: +39-051-2093001; Fax: +39-051-2093073
{pbellavista, acorradi, cgiannelli}@deis.unibo.it

Abstract

The widespread availability of devices with multiple wireless interfaces and the abundance of heterogeneous positioning techniques open new market potentials for Location Based Services (LBSs), even if complicating their development. The paper claims the need for novel middleware supports capable of managing dynamically retrieved client-side positioning systems in a synergic way and depending on context, i.e., LBS requirements, user preferences, device characteristics, and overall system state. To pursue this objective, we have designed and implemented PoSIM, a context-aware middleware for the synergic exploitation and control of heterogeneous positioning systems that facilitates the development and portability of LBSs. PoSIM is translucent, i.e., it can provide LBS developers with differentiated visibility of data characteristics and control possibilities of available positioning solutions, thus dynamically adapting to application-specific deployment requirements and enabling cross-layer management decisions. The paper describes the translucent PoSIM architecture, some primary implementation insights about our PoSIM prototype, and how to practically use our middleware to simplify LBS development via either the exploitation of pre-defined or the ad-hoc instantiation of events, filters, and policies.

1. Introduction

The growing presence of powerful mobile nodes with relatively high wireless bandwidth, e.g., via UMTS, IEEE 802.11, and Bluetooth 2.0 connectivity, is going to leverage the widespread availability of Location Based Services (LBSs). LBSs can provide service contents depending on the current position of served users, on the mutual location of clients and accessed server resources, and on the mutual position of users in a group [1]. To enable LBSs, the availability of low-cost and effective positioning systems is crucial. Several research activities have deeply worked on evaluating positioning mechanisms, techniques and systems: some solutions have been specifically designed for determining location, e.g., the well known Global Positioning System (GPS) [2]; other proposals try to estimate localization by monitoring characteristics of general-purpose communication channels, such as the IEEE 802.11-based Ekahau [3]. Detailed surveys about positioning solutions and systems can be found in [4, 5].

The point motivating our research activity is that the relevant work recently accomplished on positioning techniques has produced a wide set of currently available solutions that greatly differ on capabilities and provided facilities. For instance, they exhibit differences on:

- model used to represent location information. The representation model could be either physical (location information is provided as a longitude, latitude, and altitude triple), or symbolic (e.g., room X in building Y), or both;
- deployment environment. For instance, GPS can properly work outdoor, while another positioning system, such as Ekahau, may be more suitable for indoor environments;
- accuracy and precision of the positioning information. Accuracy is defined as the location data error range (10 meters for GPS), while precision is the error range confidence (95% for GPS);
- power consumption. The energy required for positioning typically depends on location update fre-

quency;

- user privacy, e.g., high for GPS because it determines the localization information in a completely client-side way with no explicit server-side visibility [2], low and deployment-dependent for Ekahau because a centralized Ekahau server positions clients by monitoring the received signal strength of their wireless interfaces [3];
- additional system-specific attributes, such as, for instance, the possibility to provide positioning data as a probability distribution function.

That heterogeneity of current positioning solutions, while being evidence of the relevant academic/industrial interest in the field, significantly complicates the development and deployment of LBSs. LBS developers currently have to know the details of the positioning system that will be available when deploying their services; LBS implementation is typically not portable and depends on the characteristics of the target positioning system (sometimes on the specific implementation of that positioning solution). Therefore, also due to the fact that current wireless clients tend to simultaneously host several wireless technologies useful for positioning (e.g., terminals with Wi-Fi and/or Bluetooth connectivity and/or equipped with GPS), there is a recent and emerging research trend in support infrastructures for uniformly integrating heterogeneous positioning techniques. The ultimate goal is easy LBS portability over different positioning solutions dynamically retrieved at LBS provisioning time.

The paper claims that also the above kind of support infrastructures is insufficient and that there is the need for novel context-aware middleware solutions capable of propagating differentiated levels of visibility up to the application level and of synergically managing heterogeneous positioning systems depending on LBS requirements, user preferences, device characteristics, and overall system state. To make practical examples of usage scenarios, such a middleware should seamlessly and transparently switch an LBS on its top from a positioning system to another depending on their availability, e.g., GPS outdoor and Ekahau indoor. The middleware should also associate, at any time, any LBS with the positioning technique that best fits the execution context, possibly by leaving that choice even to the LBS application logic, e.g., the positioning system with lower power consumption or the one with greater precision/update frequency. In addition, when several positioning systems can concurrently work, the middleware should either perform positioning data merging/fusion, e.g., according to context-aware requirements about robustness and confidence, or propagate a suitable view of all the location data produced by simultaneously working positioning systems to enable application-level choices on which positioning information to exploit. Let us note that proper management decisions could depend on synergic considerations deriving from the whole set of both running LBSs and positioning systems available at a client. For instance, if a positioning system is switched on because of LBS_1 requirements, it makes sense to exploit that positioning technique also for LBS_2, even if LBS_2 accuracy requirements are satisfied also by other positioning systems with lower energy consumption.

In other words, to take informed context-aware management decisions, the middleware should have easy access to low-level characteristics and control features of positioning systems. That visibility should sometimes be opened, in a highly portable and extensible way, also to advanced LBSs, which could take application-level service management choices depending on the awareness of low-level positioning details. We call *translucent* the original approach of middlewares that can support LBSs with both transparent and visible access to dynamically available positioning solutions in an integrated way.

By taking into account the above original guidelines of cross-layering, translucent, integrated control depending on execution context, we have designed and implemented the Positioning System Integration and Management (PoSIM) middleware. First of all, PoSIM is translucent in the sense that it enables differentiated visibility levels to flexibly answer all possible application requirements stemming from different LBS deployment scenarios. On the one hand, PoSIM enables LBSs to access and control positioning information at a high level of abstraction via the usage of pre-defined management policies and/or their refinement. On the other hand, PoSIM also allows LBSs to have full visibility of the characteristics of the underlying positioning systems via a PoSIM-mediated uniform access to them. In addition, application-level and system-level data are exploited in a cross-layer way to perform the most suitable management decisions on both LBSs and positioning systems.

Secondly, PoSIM enables the dynamic and integrated control of positioning systems by flexibly exposing their heterogeneous control/configuration features and location information at runtime, with no need of static knowledge about positioning-specific characteristics. In particular, PoSIM adopts declarative representations based on rules, policies, and ontologies, thus permitting to self-describe positioning system information, related metadata, and more generally execution context. In this way, PoSIM allows the access to heterogeneous positioning systems in a uniform and simplified manner, for both choosing (possibly merged) location information and configuring positioning system characteristics depending on applicable context. The PoSIM prototype is freely available for download at the PoSIM Web site [6] for educational and research purposes.

The rest of the paper is structured as follows. First, we present how the PoSIM proposal originally positions with regards to the main literature in the field. Then, Section 3 points out the primary design guidelines of the translucent and context-aware PoSIM middleware, while Section 4 details the PoSIM architecture and its primary components. Section 5 shows how we have integrated four heterogeneous positioning systems via our PoSIM prototype, also by practically describing a testbed deployment scenario where an example of advertising LBS is implemented. Conclusive remarks and directions of on-going research end the paper.

2. Related Work

Several research activities have recently started to address the emerging field of the integration of heterogeneous positioning systems. The main goal is to put together and suitably merge location information from different sources, by providing a uniform interface that LBSs can easily exploit independently of the positioning solutions available at runtime in their deployment environments. However, most researches only concentrate on the issues of uniform access and location fusion, without giving possibilities neither for synergic management of heterogeneous positioning systems nor for exploitation of context awareness to guide management decisions, which we claim as crucial aspects for advanced LBSs.

In this section, we first present research efforts on integration middlewares organized depending on the level of visibility propagated to LBSs, from transparent solutions that hide any low-level positioning system detail, to contributions with partial visibility and control for LBSs built on their top. The second part of the section, instead, focuses on JSR-179 that represents the most notable standardization effort for Java-based LBSs on mobile phones [7]. JSR-179 provides a standardized API to perform coarse-grained integration and some limited forms of control of underlying positioning systems.

2.1. Positioning Integration Middlewares

As already stated, a main property to differentiate positioning integration solutions in the literature is the degree of visibility propagated up to the LBS application level. Contributions in the field span from completely transparent approaches hiding LBSs from the complexity of direct interaction with positioning systems but not providing any control capability, to integration solutions allowing limited controllability but complicating the development of LBSs, which have to statically embed details about the exploited positioning techniques directly in their application logic.

In order of increasing level of visibility, the Alipes architecture focuses on the integration of heterogeneous positioning systems through appropriate wrappers to provide LBSs with a uniform API [8]. The goal is to force the exploitation of the available positioning system that best fits LBS accuracy requirements, by possibly performing location data fusion in order to achieve the required robustness of positioning data. Moreover, Alipes provides user-controlled privacy, by requesting explicit user permission before disclosing location information. The integration system proposed in [9] has the primary goal of seamless navigation via uniform map-based interfaces, regardless the actually exploited positioning system. Its main solution guideline is to exploit middleware components, called mediators-wrappers, to abstract from specific peculiarities of used positioning systems and maps. In addition, [9] permits to dynamically switch exploited positioning system in a completely transparent way. The integrated Platform for Location-based services (PoLoS) offers an API to facilitate the development of new LBSs [10]. It also supports the introduction of new positioning systems through a plug-in architecture; the middleware interacts with positioning systems in a standardized way via OSA/Parlay. Similarly, the Framework for Location Aware ModElling (FLAME) is a transparent integration middleware: it bases its positioning abstractions on a multi-step architecture for location data fusion, generation of geometric relationships, and event-based location data disclosure [11]. Finally, the Location Operating REference model (LORE) originally proposes different abstracting steps to provide high-level location data, independently from low-level details: positioning, modeling, fusion, query tracking, and intelligent notification [12]; in addition, it ensures privacy and security management, by controlling information disclosure, similarly to Alipes. Positioning system integration in LORE is achieved by the Common Adapter Framework that provides a standard API to fetch location information.

The above middlewares integrate positioning systems with the primary goal to facilitate LBS development. They tend to propose transparent approaches that hide LBSs from positioning complexity, but do not support any application-specific form of configuration, control, and management of positioning techniques. The main contribution of those proposals is to offer a framework to quickly prototype and deploy LBSs. However, they relevantly limit the capabilities of advanced LBSs, often interested in performing context-aware, cross-layer, and portable positioning management operations.

Only a few proposals have recently started to provide some forms of visibility of low-level features/characteristics, by introducing the partial possibility of cross-layer approaches and limited control. This demonstrates that it starts to be recognized the need for mediated visibility of underlying positioning systems, in order to achieve effective, application-specific, and context-aware management decisions, even if risking to complicate and slow down the realization of LBSs.

In particular, MiddleWhere provides LBSs with some low-level positioning details, such as location resolution, confidence, and freshness [13]. Adapter components act as device drivers, thus permitting to MiddleWhere to communicate with positioning system implementations: each adapter makes location descriptions uniform by hiding positioning system implementation peculiarities. The Location Services Module (LSM) supports not only positioning data merging but also some forms of control of heterogeneous positioning systems [14]. However, it performs merging and control in a hard-coded and not flexible manner: to achieve visibility of data/control features for a specific positioning system, LSM-based LBSs should have full static knowledge of positioning characteristics, e.g., should know name and syntax of positioning-specific control functions. Location Stack represents a state-of-the-art model of solution for location (and also context in general) data fusion [15]. It identifies several middleware components, deployed in layers, which can sequentially (as stages of a pipeline) provide increasing levels of abstraction: Sensors, Measurements, Fusion, Arrangements, Contextual Fusion, Activities, and Intentions. However, the first implementation of it, namely the Unified Location Framework (ULF), has shown that such a highly-layered system is unsuitable for properly propagating the visibility of low-level data such as accuracy and precision, often useful for application-level LBS decisions [16]. In other words, the ULF implementation experience points out the need for cross-layering to expose low-level details to LBSs and to activate direct control of positioning features from application logic.

In conclusion, most proposals in the literature only address the positioning integration issue while hiding low-level details depending on positioning technique and system implementation. MiddleWhere, LSM, and ULF are the only ones that offer partial visibility of positioning data characteristics and control features, but in a statically pre-determined way.

2.2. The JSR-179 Location API for J2ME

In the last years, the industrial research activity has primarily focused on the development of standards to address the wide heterogeneity of available positioning systems. The JSR-179 API, also known as Location API for J2ME, represents the most notable result of that standardization effort for Java-based LBSs on mobile phones [7]. JSR-179, inspired by the widespread interface of the GPS positioning solution, provides a standardized API to perform coarse-grained integration and control of positioning systems (*location providers* according to the JSR-179 terminology). To better understand how JSR-179 works and why it may be lacking for advanced context-aware management of simultaneously available positioning systems, here we rapidly report its main characteristics and offered functions.

To access an available positioning system, LBSs built on top of the JSR-179 API have to explicitly request for the instantiation of a location provider. In this request, an LBS may optionally specify selection criteria, i.e., constraints that drive the JSR-179 choice of the available positioning system to return. For instance, selection criteria may indicate that the location provider must supply speed and altitude data, and/or that the provided horizontal/vertical coordinates have to respect a minimum accuracy threshold. In addition, an LBS may specify the desired power consumption (low, medium, or high). Let us notice that these selection criteria are exploited only at location provider instantiation time, when the LBS requests the binding to a currently available positioning system for the first time; variations in positioning system availability and in selection criteria at provisioning time cannot induce any automatic modification in location provider selection. Figure 1 depicts the case of an LBS requesting a location provider: the result is the selection (and possibly activation if that implementation was switched off) of the available positioning system best fitting the criteria according to the pre-defined and not modifiable strategy embedded in JSR-179. *Location Provider 2* is associated with the LBS permanently until a new explicit location provider selection request is made.

The selected location provider returns location data to its associated LBS either on demand or via event notification. In both cases, there are several limitations on expressive power: in on-demand requests, if several LBSs are using the same location provider, each of them has to separately ask for location data; about notification of events depending on positioning data, there is the possibility only to specify proximity conditions, i.e., events are triggered only when the distance between the positioned client and fixed coordinates goes under a threshold. The provided positioning information may be a physical location (qualified coordinates), a symbolic one (address info), or both. Moreover, it may include additional data such as estimated speed and timestamp.

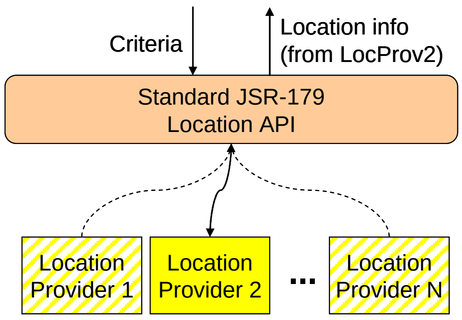

Figure 1. The JSR-179 API for criteria-based location provider selection.

Recognizing the limitations of JSR-179, a few academic research activities are working on the extension of JSR-179 capabilities to achieve greater flexibility and dynamicity. This extension work is still at its very beginning, also due to the novelty of the JSR-179 standardization effort. [17] proposes the integration and management of multiple positioning systems via a JSR-179 compliant API by increasing dynamicity with automatic transparent switch between location providers: it supports the simple case of automatically passing from GPS to Bluetooth-based positioning depending on outdoor/indoor location. However, [17] supports neither the dynamic change of positioning selection criteria (positioning system availability is the only criteria) nor the integration with statically unforeseen positioning solutions. Most important, the proposal does not include any function to control positioning systems from the application layer in an integrated and synergic way.

As better detailed in the following, PoSIM originally extends those approaches by relevantly increasing the flexibility and dynamicity of middlewares for integrated management of heterogeneous positioning systems. To the best of our knowledge, PoSIM is the only solution addressing the challenge of providing LBSs with translucent visibility of positioning data/control, by enabling context-aware management decisions depending on cross-layer considerations at service provisioning time. For an extensive and detailed comparison of the differences between JSR-179 and PoSIM API, the interested readers can also refer to [18].

3. PoSIM Objectives and Guidelines of Solution

PoSIM has the ultimate goal of providing a highly dynamic, flexible, and reconfigurable LBS support capable of mediating visibility of positioning system characteristics/data and of managing heterogeneous positioning systems in a context-dependent way. In fact, on the one hand, positioning systems should propagate via PoSIM any capability they are able to offer, dynamically retrieved by PoSIM components and made accessible to the application level in a properly simplified way. On the other hand, LBSs should be able to command the reconfiguration of positioning system behaviors in relation to their current requirements, e.g., by keeping switched-on only the positioning system with minimum energy consumption and satisfying accuracy when the client battery lifetime is under a specified threshold. However, it is crucial that the visibility of low-level details and the synergic control of positioning systems do not increase too much the complexity of LBS development.

PoSIM pursues these objectives by following three primary design guidelines: i) the provisioning of a translucent API favoring cross-layer interactions, ii) the exploitation of context to openly describe executing conditions and system/service/user requirements at the proper level of abstraction, and iii) the possibility to actively control not only location data merging/selection but also positioning system behavior.

First of all, PoSIM provides integrated management of heterogeneous positioning systems by adopting a *translucent* approach, intended as the simultaneous provisioning to the application layer of both high- and low-level API. Thanks to PoSIM, LBSs aiming to interact with positioning systems in a simplified manner, namely *simple LBSs*, can get a transparent access via high-level PoSIM API, thus perceiving the underlying available positioning systems as a unique multi-behavior positioning facility. On the contrary, LBSs willing to have direct visibility and to manage peculiar information/features of positioning systems, namely *smart LBSs*, can interact in a middleware-mediated but fully aware fashion, via low-level PoSIM API.

The translucent approach also allows LBSs built on top of PoSIM to get a uniform and aggregated access to all the characteristics of integrated positioning systems. On the one hand, PoSIM provides LBSs with a uniform API independently of the specific positioning solution, e.g., to reduce overhead it is possible to limit the accuracy of Ekahau-based and BTProximity-based positioning in the same way. On the other hand, PoSIM permits to access/configure all the available positioning systems aggregately, e.g., to gather all the data about current accuracy from all activated positioning system, with no need to interact with each positioning system separately. In this manner, LBSs can achieve a uniform aggregated (and thus simplified) access to lower layers. Let us note that other different research fields use “translucency” to name the flexible combination of both visibility and transparency: for instance, [19] and [20] adopt the translucent term to indicate similar hybrid visibility in the area of optical networks.

About the second solution guideline, PoSIM takes full advantage of information provided by LBSs and positioning systems both to provide location information at the application layer and to manage lower-layer characteristics effectively, by adopting a context-aware approach. PoSIM provides LBSs with a fully context-aware but simplified access to location information and positioning system control via pre-defined triggering events, to specify when information must be delivered, filtering rules, to specify which information must be selected, merged, and delivered, and declarative policies, to specify how positioning systems should behave. For instance, LBSs can specify their interest in receiving location information only when particular events occur, e.g., only when the user is close to her office, and only the positioning data with specified characteristics, e.g., discarding location information whose accuracy is below a threshold. Moreover, PoSIM-based LBSs can manage positioning system behavior in relation to dynamically changing lower-layer characteristics, e.g., turning off a positioning system only when its accuracy goes under a given threshold and it is not currently used by any other LBS. In other words, LBSs exploit context-awareness by simply de/activating pre-defined events, filters, and policies which, in their turn, depend on context information about underlying positioning characteristics and LBS requirements.

Delegating to PoSIM any support procedure to gather and manage context information greatly simplifies context-aware LBS development, but at the same time requires effectively dealing with heterogeneous data originated by several different components. For this reason, to allow an extremely open execution environment, we claim the need for representation formats that combine flexibility, openness, and interoperability. PoSIM achieves this goal by enabling the integration of ontologies to describe the semantic of positioning data and control facilities at runtime. PoSIM does not assume the mandatory adoption of a particular common ontology: it dynamically retrieves the ontology to be exploited in the deployment environment, thus not limiting at all the set of information and capabilities a positioning system is able to provide. The only requirement is that the integrated positioning systems must be wrapped to offer a generic flexible API, as better detailed in Section 4.4. Let us rapidly observe that, even if PoSIM has been specifically designed to integrate positioning systems, its architecture is also suitable to manage any context sources in general: PoSIM provides a uniform access, in terms of both visibility and information syntax, to context data and context source control.

By focusing on the third solution guideline, PoSIM not only works to expose the location data uniformly to the LBS application level, but originally permits to control positioning systems behavior with different levels of opportunities. While most state-of-the-art integration middlewares limit their efforts in merging heterogeneous systems to provide a uniform static interface for location gathering, PoSIM actually puts together positioning systems to enable the integrated synergic control of their behavior by considering them aggregately. For instance, via PoSIM an LBS could command to simultaneously lower the power consumption of every positioning system just specifying to set the *PowerConsumption* PoSIM control feature to low. In addition, an LBS built on top of PoSIM can take into account the available positioning systems in a relative way. For example, it is possible to turn on the two best positioning systems with regards to power consumption, while switching off the other ones. About translucency in positioning system control, simple LBSs only have the burden of specifying desired behaviors, by delegating PoSIM for any required action. In fact, PoSIM provides a set of pre-defined declarative policies that simple LBSs can only decide to de/activate. Note that the opportunity to control positioning systems via declarative policies greatly facilitates LBS development because LBSs leave the burden of any required monitoring/control action to the PoSIM middleware. Smart LBS, instead, can directly control each positioning system features and capabilities in a fully-aware manner, via uniform middleware-mediated API. In this case, LBSs can access low-level PoSIM API to interact with and control each positioning system separately, e.g., for the purpose of switching on/off and configuring a specific component.

4. PoSIM Architecture and Primary Components

We have followed the above guidelines of translucent, context-aware, and cross-layer control to design and develop our PoSIM middleware for the integrated management of heterogeneous positioning systems. PoSIM has the twofold goal of enabling both the mediated visibility of all the information provided by underlying positioning systems and the mediated control of their configurable characteristics for a synergic context-dependent management. In particular, PoSIM provides the application layer with mediated and facilitated access to either low- and high-level API: LBSs can interact with underlying positioning systems in either a visible or transparent manner, respectively. Nevertheless, PoSIM can appear to the application level as a middleware offering a single, multi-faceted, and flexible API, thus simplifying its usage and potentially leveraging its adoption.

Figure 2 depicts our PoSIM middleware architecture. To interact transparently with positioning systems, simple LBSs can exploit the Policy Manager (PM) and Data Manager (DM) high-level API to respectively control positioning systems behavior and get their location information. To interact in a more visible and flexible way, smart LBSs can exploit the Positioning System Access Facility (PSAF) low-level API to directly access the Positioning System Wrappers (PSWs) for the currently available and integrated positioning systems.

About control (top-down colored arrows) and data (bottom-up white arrows) flows in Figure 2, let us anticipate that i) PM is the middleware component devoted to control and enables application-level management capabilities based on context information gathered from PSAF, and ii) DM, instead, plays the role of exposing location information according to dynamically configurable differentiated modes. PSAF, instead, can provide LBS with both control capabilities and positioning data. Let us stress PM and DM exploits low-level PoSIM API, i.e., PSAF methods, to offer an encapsulated high-level API with more articulated and easy-to-use services at a higher level of abstraction.

PoSIM does not rely on any particular statically predefined ontology and on syntactic/semantic conventions on how to represent control capabilities and positioning data. It only defines a simple model distinguishing between positioning system *features* and *infos*. Features describe positioning system characteristics and capabilities, possibly with settable values useful for control/configuration, e.g., power consumption or ensured privacy level. Infos are not configurable location-related data, e.g., positioning information and its accuracy. Infos are the only data provided to simple LBSs while smart LBSs have visibility of both features and infos.

In the following, this section presents the main design and implementation guidelines of the PoSIM middleware. For each PoSIM component, it provides an overview of its functions and offered API, some practical usage examples to show how to take full advantage of its capabilities, and design/implementation insights.

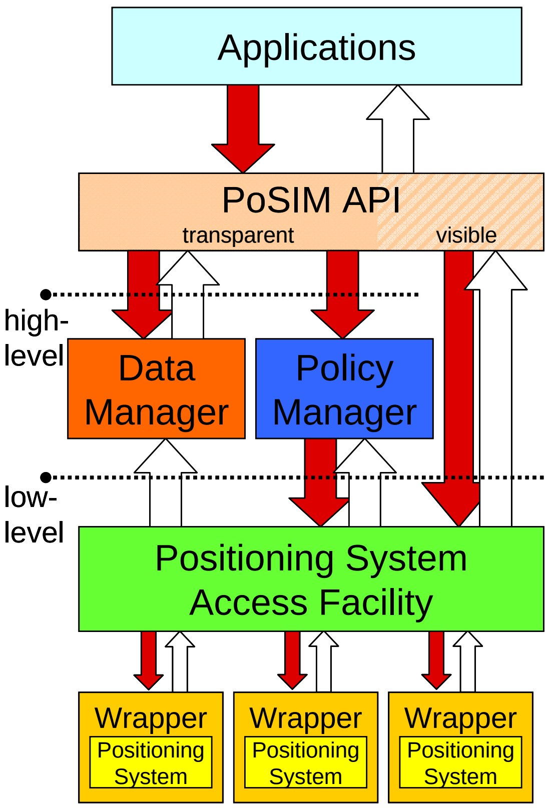

Figure 2. The PoSIM architecture (white arrows represent data flows, colored arrows are control flows).

4.1. Policy Manager

The Policy Manager (PM) is the PoSIM component responsible for enforcing the policies for dynamic control and management of heterogeneous positioning systems. In particular, the PM API allows simple LBSs to ask for pre-defined behaviors specified via default policies. It is PM to be in charge of autonomously and dynamically interacting with positioning systems to transparently satisfy LBS requirements. Let us point out that PM provides a context-aware control of positioning systems: it can take into account both application-level requirements, e.g., minimum power consumption, and current system state, e.g., by avoiding to turn off a positioning system in the case it is the only one switched on and there is at least one LBS calling for positioning data.

Via the high-level and transparent PM API, LBSs can actively control positioning systems by simply specifying the desired behavior with no visibility of any low-level positioning detail. In particular, the PM provided methods are:

- `insert(newBehavior)/delete(aBehavior)`, to add/remove a new/existing PoSIM behavior;
- `activate(aBehavior)/deactivate(aBehavior)`, to effectively require the activation of a behavior among the already defined ones.

Behaviors are implemented as declarative policies, i.e., set of actions that PM must perform whenever conditions specified in the policy apply. Conditions are relational expressions related to positioning system infos/features; actions are management operations that PM performs over positioning system features. Let us observe that PoSIM allows not only to enable/disable a given behavior at service provisioning time by de/activating declarative policies, but also to introduce novel behaviors by adding new policies. In addition, any activity related to behavior definition and de/activation is independent from the actual implementation of both PoSIM and positioning system components below the PM level. In this manner, on the one hand, changes in integrated positioning systems cannot affect behaviors; on the other hand, LBSs can actively specify the desired control behavior transparently, thus facilitating and leveraging their development.

Table 1. PoSIM policy representation.

```
policy ::= [salience] name policy_type
policy_type ::= isolated | ordering

isolated ::= conditions actions
ordering ::= ord_data bestN bestAct worstAct

bestAct ::= actions
worstAct ::= actions

conditions ::= cond | cond conditions
actions ::= action | action actions
cond ::= data value operator
action ::= Feature value
ord_data ::= numeric data

data ::= Info | Feature
bestN ::= non negative integer
salience ::= integer
name ::= string
value ::= string | integer | double
operator ::= =|!=|<|>|<=|>=|eq|neq
```

As Table 1 shows, PoSIM supports the specification and activation of two types of policies: *isolated* and *ordering* policies. Isolated policies separately apply the same condition-action rules to each positioning system retrieved at runtime in the execution environment. `Conditions` are a set of relational expressions, each one described with a data name/value and a relational operator. Supported relational operators include =, !=, <, >, <=, >=, and ‘eq'/'neq' (i.e., =/!= among strings). `Actions` are a set of operations on modifiable features, each one with an associated name and value. Given a positioning system, if all conditions are satisfied, the policy is triggered, namely *fired*, and all the features in `actions` are set to the values indicated in the policy, i.e., the policy actions are *enforced*. For instance, a PoSIM isolated policy could turn off the positioning systems with higher energy consumption if that does not endanger application-specific requirements about positioning precision and accuracy.

Table 2. The `lowPowerConsumption` isolated policy.

```
name:lowPowerConsumption
conditions:
        Feature(name:Power, value:8) op:>
        Info(name:Accuracy, value:5) op:<
actions:
        Feature(name:State, value:off)
```

Table 2 reports the `lowPowerConsumption` policy that switches off a currently available positioning system if its power consumption is greater than 8 and its accuracy below 5 (rapid notes about the mapping between power/accuracy values in the policy and their actual, possibly proprietary, counterparts in the integrated positioning systems are in Section 4.4).

Table 3. The `onBestAccuracy` ordering policy.

```
name: onBestAccuracy
ord_data:
        Info(name:Accuracy)
bestN:
        1
best actions:
        Feature(name:State, value:on)
worst actions:
        none
```

Ordering policies, instead, can compare available positioning systems in order to sort them according to a desired indicator, e.g., listing positioning systems from the best to the worst one in terms of accuracy. In other words, in a sense the scope of ordering policies is wider than that of isolated ones, since ordering policies tend to intrinsically manage positioning systems aggregately. Ordering policy actions consist of two sets of features, best and worst: PM enforces best actions for the best *bestN* positioning systems, while it executes worst actions for the remaining ones. For instance, an ordering policy could request to always turn on the positioning system with best accuracy. Table 3 depicts the `onBestAccuracy` policy that sorts positioning systems in relation to provided accuracy, and turns on the one with maximum accuracy.

In addition to the above examples, we have specified default policies in PoSIM, ready to be activated by simple LBSs. PoSIM already includes the following isolated policies of common usage:

- `onlyPhysical/onlySymbolic`, which activates only the positioning systems that offer physical/symbolic location information;

- `highAccuracy(threshold)`, which switches off all positioning systems whose accuracy is below `threshold`;
- `highPrivacy(threshold)`, which sets the privacy level of available positioning systems (at least) to the `threshold` value.

In addition, PoSIM defines the following ordering policies of common usage:

- `onlyBestAccuracy(bestN)`, which activates the *bestN* positioning systems (in terms of accuracy) by switching off all the others;
- `onlyBestConsumption(bestN)`, which keeps active only the *bestN* positioning systems in terms of lower consumption.

Let us notice that isolated policies compare info/feature values gathered at runtime with thresholds: a given isolated policy can be concurrently fired on different positioning systems (its triggering condition could be verified for several positioning systems at the same time); that should be carefully considered when specifying policies to avoid undesired behaviors. For instance, the above described `lowPowerConsumption` isolated policy is badly defined for most deployment environments because it could turn off all available positioning systems, thus making impossible to obtain any updated positioning information. This is one of the motivations why PoSIM also integrates ordering policies that provide the additional capability to manage positioning systems comparatively. In fact, ordering policies can enforce different actions depending on positioning system order and do not require specifying threshold values, which may be a hard task in many real-world deployment scenarios.

When different policies are simultaneously fired, in general there is also the possibility of conflicting actions. For instance, in the case of `lowPowerConsumption` and `onBestAccuracy` firing in the same time interval, the former may request switching off every positioning system, while the latter would turn on the positioning solution with highest accuracy. To help avoiding possible conflicts, any PoSIM policy is associated with a priority, either provided at development time (namely salience) or depending on policy activation order, e.g., recently activated policies are favored. Fired policies are enforced from the most prioritized to the least one, as further detailed in the following.

Let us also note that the definition of conflicting rules may not always be erroneous. For instance, consider again the simultaneous firing of `lowPowerConsumption` and `onBestAccuracy`, the former with less priority than the latter. If any integrated positioning system provides limited accuracy and imposes too high power consumption, `lowPowerConsumption` would turn off every positioning system. On the contrary, since `onBestAccuracy` has higher priority, certainly at least one positioning system will be maintained on, i.e., the one with best accuracy. In fact, as better detailed in the following, PoSIM recognizes conflicting actions (sets of operations working on the same positioning system features) and, in the case, only executes actions with higher priority.

Figure 3 depicts the PM architecture. The Policy Controller (PC) i) provides the capability to insert/delete and de/activate policies, ii) interacts with PSAF to get up-to-date info/feature values needed to evaluate the conditions of activated policies, iii) requests the Policy Engine (PE) to check for policy condition satisfaction and to execute the actions specified in fired policies.

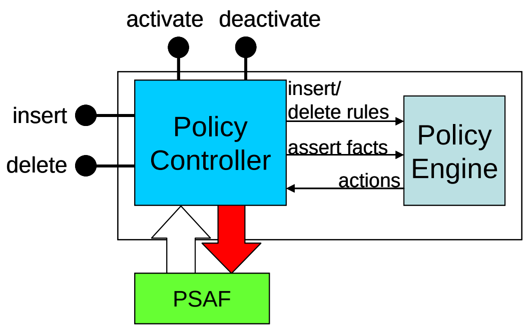

Figure 3. The architecture of the Policy Manager.

Delving into finer implementation details, the PoSIM PE exploits Jess [21], a rule engine based on the Rete algorithm [22]. PC automatically transforms new policies, described as Java classes, in Jess rules and, at their activation, provides PE with them. The Jess knowledge base includes only the infos and features that appear in at least one active policy condition, i.e., only infos and features relevant for currently activated policies. In that way, PC only retrieves the needed monitoring indicators from the underlying positioning systems, thus limiting the PoSIM middleware overhead.

By default, PE enforces policies by following the standard Jess "depth" (age-based) strategy, i.e., if several policies are simultaneously fired, PE performs the enforcement of the most recently activated one first and then fires the remaining ones in activation order. In addition, PoSIM administrators can add new policies by explicitly specifying a salience integer value, thus possibly affecting the order of policy enforcement. In particular, when specified, PoSIM policies are fired in relation to their salience, from the highest to the lowest: policies with the same salience value are fired with the Jess standard strategy (depth first); policy salience is set to 0 by default.

Let us observe that the PoSIM goal is not to specifically provide an original, powerful, and general-purpose policy management support. PoSIM simply exploits a subset of Jess existing capabilities, with the purpose of providing LBSs with the capability of dynamically adding and/or removing policies, even at service provisioning time, in an easy but conveniently flexible way. In fact, while in principle it could be possible to add in PoSIM whatever policy written in the Jess native language, we decided to limit the range of valid policies by imposing the mandatory structure reported in Table 1. On the one hand, that simplifies the work of PoSIM administrators by providing a rigid but sufficiently expressive discipline for policy specification. On the other hand, that limitation reduces the risks of erroneous policy specification, by also paving the way to effective automatic tools for conflict identification and analysis. Moreover, Jess policies do not apply actions directly; in other words, Jess has no direct access to the PSAF component. When policies are fired, requested actions are not performed immediately but first ordered according to policy priorities. Then, if PoSIM recognizes conflicting actions, it only executes the actions related to the policy with highest priority, by inhibiting remaining actions. For instance, if both *policy1* and *policy2* are fired and the higher-priority *policy1* requires to set power consumption to 3 while *policy2* would set consumption to 5, then PoSIM sets power consumption to 3 by not considering *policy2* at all. Finally, PM does not allow Jess loop rule activation, i.e., action enforcement does not produce the immediate re-evaluation of the conditions of all activated policies in a cyclic way, in order to simplify policy management and to limit enforcement costs.

4.2. Data Manager

The Data Manager (DM) is the PoSIM component responsible for offering an aggregated view of positioning information to the application level, thus providing differentiated context-dependent views of location data. In particular, DM aggregately provides PoSIM-based LBSs (specifying when and which positioning information they are interested in via the DM API) with the location info produced by the different integrated positioning systems and collected together in a single XML document. Let us stress that DM provides context-aware location information: PoSIM returns positioning data by taking into consideration both LBS requirements and positioning system information, e.g., by comparing the minimum accuracy required by an LBS with the accuracy level offered by each positioning system available.

In particular, LBSs can ask to be provided with the XML location data document in three different ways:

- *on demand*, exploiting either `onDemand()` or `onDemand(listener)` methods, which immediately provide the already estimated positioning data (last performed estimate). The latter method additionally applies LBS-specific filters, as better detailed in the following;
- *at regular time intervals*, exploiting the `periodical(interval, listener)` method, which commands a periodical delivery process to notify the listener every `interval` milliseconds;
- *in an event-driven fashion*, exploiting the `addEvent(event, listener)` method, which permits to specify a specific event to trigger future delivery of the location document.

LBSs can simply exploit easy-to-use pre-defined conditions to trigger location data delivery. For instance, the pre-defined `atLocation` condition triggers location notification only when the current symbolic location coincides with what specified as the invocation parameter. In addition, the `addFilter(filter, listener)` method provides a simple way to filter positioning data: for instance, the pre-defined `highAccuracy` filter automatically discards location information whose accuracy is below a given threshold. In addition, the proper exploitation of filtering rules permits to relevantly reduce the middleware overhead by avoiding useless notifications of non-relevant location changes. In summary, by exploiting the above methods, LBSs can specify both which information they are interested in and when they are interested in getting it without specific knowledge about the implementation details of the positioning systems they are using.

Let us notice that declarative policies and filter rules have very different roles in PoSIM and behave much differently. Policies actively control positioning system behaviors: they can modify positioning system features, which may impact on other features and on positioning info performance. For instance, the `lowPowerConsumption` policy deactivates positioning systems with a too high power consumption level, by possibly affecting positioning accuracy since some systems could be switched off by the policy enforcement actions. Moreover, a policy activation impacts on any LBS on top of PoSIM. For instance, the `highAccuracy` policy forces LBSs not to exploit positioning systems with low accuracy. On the contrary, a filter rule just avoids to deliver positioning information considered useless by a specific LBS, without any impact on positioning system working. Each LBS can declare its filtering rules, without any possible interference with other simultaneously working LBSs.

As rapidly mentioned, DM offers the access to any information generated by the integrated positioning systems, possibly added with context data from other sources, as an XML document. In that way, smart LBSs can have access to the wide set of location data and feature-related information available, in order to flexibly decide which information to exploit at the application level. In particular, the provided XML document consists of:

- a `timestamp` describing when DM created the XML document;
- a `source` for each exploited positioning system (embedded in a common `sources` parent tag);
- an `info` tag for each information provided by a `source`.

Table 4. The structure of the PoSIM document with positioning data and their characteristics.

```
<Data>
    <timestamp time=docTS/>
    <sources>
        <source name="GPS">
            <info Location="xyz" />
            <info Accuracy="high" />
            <info Timestamp =locTS />
        </source>
        ...
    </sources>
</Data>
```

PoSIM describes a delivery triggering event as a triple including an info name, a value, and an `evaluate(...)` method, which returns true if the positioning info must be delivered, false elsewhere, usually depending on the evaluation of the positioning info itself. In particular, we have decided to implement two main triggering event categories of common usage: *isolated* and *comparing*. An isolated event exploits an `evaluate(Info threshold)` method that compares the current info value with a fixed `threshold`. A comparing event, instead, uses an `evaluate(Info currentValue, Info previousValue)` method to compare the current info value with the data provided in the previous delivery.

To clarify how these two event types can cover most common usage scenarios, let us rapidly present the following simple examples of pre-defined PoSIM events:

- `atLocation(loc)` is an isolated event that triggers data delivery only if the current symbolic location is equal to *loc*;
- `distance(dist)` is a comparing event that triggers data delivery only when the current location differs from the previously delivered positioning info for more than *dist* meters.

Let us note that the `distance(...)` triggering event has the result of providing positioning information by exploiting spatial variation as the triggering period ("space-periodical" positioning update). For instance, that could be useful for an advertising LBS interested in providing new information to the user whenever she moves more than 50 meters from the previous update location. In addition, PoSIM allows to specify *and*/*or-aggregated* events, i.e., sets of isolated/comparing events that trigger information delivery whenever all events occur (*and* modality) or at least one event occurs (*or* modality) during a specified time interval.

Filtering rules, instead, consist of an info name, a value, and an `evaluate(...)` method which, given the gathered info value, returns true if the info should be discarded, false otherwise. In other words, whenever an integrated positioning system (or any context source more generally) has the specified info and that info does not satisfy the given `evaluate(...)` method, DM discards the entire `source` (see Table 4). For instance, one of the PoSIM pre-defined filter rule is called `onlyHighAccuracy(acc)` and discards every source whose accuracy is below `acc` (in a scale from 0 to 9, see Section 5.2).

Finally, let us note that expert users, such as PoSIM administrators, can develop and deploy new policies, new triggering events, and new filtering rules in a relatively easy way. In fact, all of them are implemented as Java classes that can be simply sub-classed to specify new specialized policies, events, and filters. In that way, the PoSIM behavior can be easily extended with impact on neither its implementation nor the application logic code. At the same time, simple LBSs and novice LBS developers can also work by only selecting their policies, events, and filters of interest among the set of pre-defined and most common ones already provided by default in the PoSIM distribution.

Figure 4 depicts the DM architecture. Data Builder (DB) collects infos from the currently exploited positioning systems and possibly aggregates them with context information of interest. DB periodically (every configurable polling period, 2 seconds is the default value) gets information from PSAF and provides gathered data as an XML document. Data Disclosure (DD) is the component that actually exhibits DM API, by exposing appropriate methods to specify how interested LBSs can get data. In other words, fed by DB monitoring information, DD delivers the XML document with positioning data to every registered LBS listener when either the polling period expires or an associated event occurs.

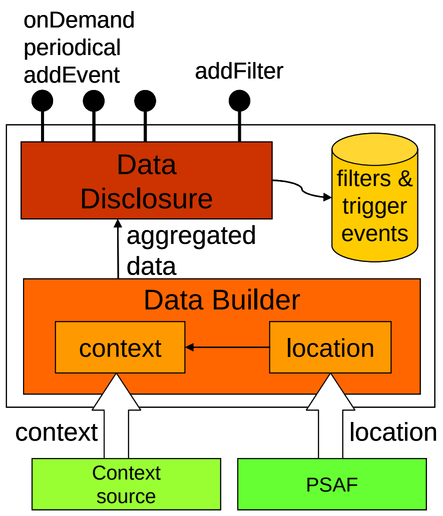

Figure 4. The Data Manager architecture.

The delivered XML document is the result of filtering the raw positioning data produced by the activated positioning systems with the filters specified by the interested listeners. Let us observe that each method of the DM API allows to specify a listener, apart from `onDemand()`; that increases the flexibility of our middleware solution if compared with other recently emerging proposals for positioning integration [17]. In fact, LBSs not only are able to simply gather location information with a one-shot interaction with PoSIM (`onDemand()` method), but also can ask for a more personalized delivery based on LBS-specific requirements implemented via the `listener` parameter. PoSIM can perform several articulated positioning data management actions, such as continuous location monitoring to verify if the available data are really of interest (`addFilter(...)` method) or if relevant events occur (`addEvent(...)` method). In that way, LBS development and deployment are greatly simplified; the only burden for LBS providers is to decide the triggering events, filtering rules, and time intervals for each of their listeners.

4.3. Positioning System Access Facility

Smart LBSs and PM/DM can directly control the integrated positioning systems by exploiting the lower level API of the Positioning System Access Facility (PSAF). PSAF supports APIs to dynamically handle the registration/cancellation and to retrieve/control infos/features of all the positioning systems locally available at the controlled client node. In particular, the PSAF API allows:

1. to dynamically un/register a positioning system implementation in the set of locally available positioning solutions (the only constraint is that the registered positioning implementation offers a PSW-compliant interface, see the following);
2. to interact with registered positioning systems via the Query/Control interface.

The PSAF Query/Control interface enables the interaction with registered positioning systems in an aggregated and synergic way, by taking decisions depending on the whole set of available systems. In particular, the Query interface includes the following methods:

- `getInfos(posSysSet)/getFeatures (posSysSet)`, which returns the set of info/features for the specified set of positioning systems;
- `getInfo(posSysSet,name)/getFeature (posSysSet, name)`, which returns the value of a specific info/feature for the specified set of positioning systems;
- `getAvailable()`, which returns the list of the currently available positioning systems.

The Control interface, instead, offers the method:

- `setFeature(posSysSet, name, value)`, which changes the value of the `name` feature for the specified set of positioning systems.

For instance, in response to the invocation of `getInfos(null)`, PSAF provides all the info of every registered positioning system, while the invocation of `setFeature(GPS, State, off)` commands PSAF to change to *off* the value of the feature *State* of the positioning system named *GPS*.

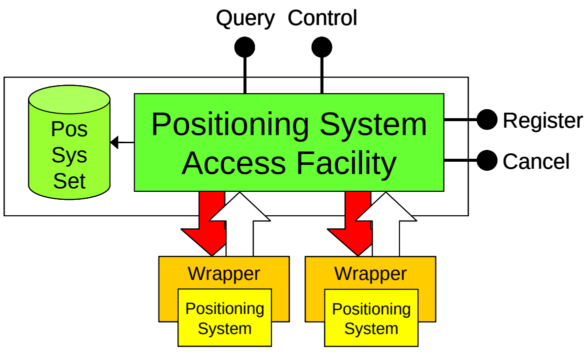

Figure 5. The architecture of the Positioning System Access Facility.

Smart LBSs and PoSIM middleware components can invoke the Query/Control methods; only PoSIM administrators, instead, can access the Register/Cancel interface. Let us stress that PSAF is the only way for higher middleware and application layers to access integrated positioning systems, thus guaranteeing controlled and system-safe accesses to low-layer positioning components, independently of their specific technique and implementation peculiarities. The only requirement is that positioning systems provide their infos/features via a specified interface; that interface is practically obtained by wrapping the implementations of positioning systems with PoSIM Positioning System Wrappers (PSWs). PSAF exploits Java introspection to dynamically determine and access the set of infos/features exposed by the wrappers and actually implemented by the underlying positioning systems that are currently available in its deployment environment.

4.4. Positioning System Wrapper

As already pointed out, the Positioning System Wrapper (PSW) is the crucial middleware component that hides positioning system heterogeneity. It exposes to the upper middleware layers a common API, independent of the wrapped positioning system and of its implementation details, by providing infos/features compliant to the exploited ontology for representing positioning-related data. For instance, if the ontology in use specifies that accuracy values are integers in the [0, 9] range, the PSW `getAccuracy()` method will provide location accuracy as an integer value. Any PSW component will interact with its wrapped positioning system, retrieve the associated accuracy value by exploiting positioning-specific awareness and syntax, and transform it accordingly to the adopted ontology, e.g., transforming a "high accuracy" string return value in the correspondent integer. That ontology is the only knowledge to be shared among the PoSIM components, which allows policies, triggers, and filters exploiting that ontology to be specified independently of the positioning implementation details.

Delving into finer details, PSW offers:

- a `getX()` method for each feature provided by the wrapped positioning system, where *X* is the name of the feature;
- a `setX(value)` method for each available modifiable feature, where `value` is the new value to be set for that feature;
- an `infoX()` method to read each location-related information provided by the wrapped positioning system, where *X* is the info name.

PSAF exploits Java reflection to correctly map its `getX()/setX()/infoX()` methods to the corresponding (sets of) lower-level invocations in the wrapped implementations of currently available positioning systems. For instance, given the wrapper of a particular positioning system, to get the current value of the *Location* info, PSAF invokes its `infoLocation()` method, while, to change the value of the *PowerConsumption* feature, PSAF invokes its `setPowerConsumption(newStrategy)` method, which changes the value of that feature to *newStrategy*.

As already pointed out, the distinction between infos and features is the only assumption PoSIM performs on provided information. In fact, thanks to the adoption of Java introspection, PoSIM components are independent from the details of information representation. The integration of a new and unexpected type of positioning system into PoSIM only requires encapsulating it in a PSW that provides its infos and features through the above PSW interface.

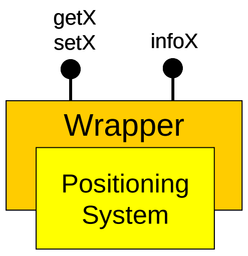

Figure 6. The Positioning System Wrapper API.

We have already stated the flexibility stemming from not relying upon any statically specified ontology. In this manner it is possible to adapt PoSIM to any legacy component/application, even if not known at middleware development time. In the current PoSIM prototype implementation, we propose and adopt a simple ontology that should be taken into account when defining declarative policies, filter rules, and triggering events. The definition of such an ontology is not the specific scope of our research work and the currently exploited ontology can be easily modified/replaced without affecting the implementation of PoSIM components. In particular, the adopted ontology defines three main feature/info categories: mandatory, common, and peculiar. According to the ontology, any integrated positioning system must offer mandatory features/infos. We consider as mandatory:

- *Location* info, the last location information provided by the wrapped positioning system,
- *Timestamp* info, the time in which the provided location info has been estimated,
- *PSState* info, either *on* or *off* to indicate whether the positioning data has been obtained either correctly or not,
- *Name* feature, to get positioning system name,
- *State* feature, a modifiable feature to switch on/off a positioning system,
- *ExploitedComm* feature, e.g., IEEE 802.11 for Ekahau and Bluetooth for the GPS solutions using Bluetooth connectivity towards their clients, and,
- *LocationType* feature, whose value can be *physical*, *symbolic* or *both*.

The infos/features classified as common are optional (some positioning systems may decide not to implement them) but pre-defined as they are of common and frequent usage. For instance:

- *Accuracy* info, related to the provided location information,
- *PrivacyLevel* feature, to indicate if the positioned client can hide its location information,
- *PowerConsumption* feature.

Finally, we consider also the possibility to include other a-priori unknown infos/features, peculiar to a specific positioning system and thus not usually shared between PSWs. For instance:

- GPS *FixType* info, which can be *2D*, *3D* or *no fix* and that makes sense only when considering the GPS positioning system,
- the Ekahau *Status* feature, providing detailed Ekahau-specific information about the working status of the Ekahau Positioning Engine.

Let us stress again the differences between common and peculiar features/infos. For instance, PoSIM models the power consumption feature as common (not mandatory) because it is not always possible to evaluate power consumption for every positioning system but the feature is of common usage and LBS developers could be aware of its possible availability. When the feature is available, there is the need to agree on the measurement unit of its returned value, e.g., in mwatt or in a scale from 0 to 9, and that is specified in the ontology. On the contrary, peculiar features/infos can be added freely by PSW implementers, with no impact on the adopted ontology and without any requirement on returned value semantic.

Table 5 reports mandatory, common, and some examples of peculiar features/infos. In the implemented ontology, the physical location information is modeled in terms of latitude, longitude, and altitude, while symbolic location information is represented as a layered (hierarchical) location, e.g. [Italy, Bologna, EngSchool, Lab2] [23]. Accuracy is represented by an integer value between 0 (minimum) and 9 (maximum). The privacy level has a value between 0, uncontrolled location information delivery, and 9, stealth mode, i.e., only the positioned client has access to its own location. Power consumption is modeled with a value in the [0, 9] range, usually measured in a static way (see the PoSIM implementation insights in the following section).

Among the above listed infos/features, let us rapidly focus on two of them, *State* and *PSState*, to better explain their semantic. The *State* feature returns *on/off* depending on the fact that the positioning system is switched on/off, thus being exploitable or not to obtain positioning data. Even if a currently switched off positioning system cannot provide localization info, the correspondent PSW can continue to offer old positioning data based on previous values, implicitly specifying they are history-based estimations via the timestamp info. Also *PSState* is either *on* or *off*, representing if the positioning operations of a switched-on positioning system have been performed in a correct way in the last time interval. For instance, even if a GPS device is active (*State* is *on*), it could not be able to provide a correct location information (*PSState* is *off*) since there are not enough satellites in line of sight (no fix according to the GPS terminology).

Table 5. Mandatory, optional, and peculiar infos/features as defined in the default PoSIM ontology.

Category		**Name**	**Modifiable**
Mandatory	Info	Location	n.a.
		Timestamp	n.a.
		PSState	n.a.
	Feature	Name	no
		State	yes
		ExploitedComm	no/yes
		LocationType	no/yes
Common	Info	Accuracy	n.a.
	Feature	PrivacyLevel	no/yes
		PowerConsumption	no/yes
Peculiar	Info	FixType (GPS)	n.a.
	Feature	Status (Ekahau)	no

5. PoSIM Implementation Insights

In this section we present our actual test-bed to exemplify how it is possible to provide infos and set/get features of three off-the-shelf positioning solutions (GPS, an IEEE 802.11-based positioning system, and a Bluetooth-based one) and a generic positioning system compliant with the JSR-179 Location API for J2ME. In addition, we present an example of development and deployment of an LBS built on top of PoSIM. Additional information and the downloadable code of the PoSIM prototype, together with the PSWs for the presented positioning systems, are available at the PoSIM Web site [6].

5.1. Integrated Positioning Systems

Several heterogeneous positioning systems are currently widespread. Here we focus our attention on three of them, GPS, Ekahau and BTProximity, because they exemplify positioning system heterogeneity in terms of exploited positioning technique (e.g., triangulation, proximity), provided information (e.g., physical, symbolic location), and positioning delivery mode (e.g., on demand, event-driven). The sub-section provides the few needed implementation insights about these three positioning techniques to understand the implementation decisions described in the following.

GPS is currently the most spread positioning system, exploited in several commercial applications ranging from navigation aid to car tracking. GPS determines node location via triangulation by exploiting knowledge about satellite constellation position and node-satellite constellation distance [2].

Ekahau [3] is a positioning system for Wi-Fi-based nodes and is based on techniques of scenario analysis and on characteristics of IEEE 802.11 communications, similarly to RADAR [24]. Scene analysis techniques include two phases: a preliminary off-line phase and an operational one. In the former phase, the positioning system gets knowledge about AP RSSI in the monitored environment, i.e., it associates physical locations with neighbor AP MAC addresses and corresponding RSSIs. In the latter phase, nodes send RSSI data to the Ekahau Positioning Engine (EPE), the Ekahau component which actually calculates node localization. EPE compares historical and currently observed RSSI data, inferring node current location. In our past research work, we have developed an original Wi-Fi-based positioning solution, someway similar to Ekahau, but along the guideline of avoiding the long and expensive phase of scenario analysis at the expense of a generally lower accuracy in positioning estimation [25].

BTProximity [6] is our original positioning system with user privacy capabilities, based on proximity techniques and Bluetooth communication technology. In particular, BTProximity simply associates one node with the location of the closest reference point, i.e., Bluetooth device, whose distance is inferred by exploiting baseband connection RSSI. Other Bluetooth-based positioning systems are available in the literature [26, 27]. Differently from them, BTProximity specifically focuses on privacy management: user privacy is achieved by carefully hiding node presence to reference points, that is not revealing to infrastructure nodes where the node is notwithstanding the node exploits reference points to determine its location. In particular, BTProximity supports the provisioning of three privacy levels: *low*, *medium*, and *high*. Each privacy level corresponds to a different Bluetooth device configuration, as better detailed in the following. In particular, when BTProximity privacy level is

- Low, the Bluetooth node periodically broadcasts a message, as reference points do, by revealing its presence to anyone (the Bluetooth node is in *Page/Inquiry Scan* mode [28]);
- Medium, the Bluetooth node does not broadcast messages but only accept incoming connections (the Bluetooth node is in *Page Scan* mode). If an external device knows the MAC address of the Bluetooth node, it could try to connect to it by performing a sort of blind connect; if the connection attempt is successful, node location is revealed. Moreover, the Bluetooth node connects to visible reference points to determine RSSI values, by potentially revealing its presence (the Bluetooth protocol requires active baseband connections to determine RSSI);
- High, the Bluetooth node completely hides its presence (stealth mode – the node is in *No Scan* mode). It neither broadcasts messages nor accepts incoming connections; it can only listen to reference points broadcasting messages. To maximize user privacy, the Bluetooth node does not even connect to reference points. Since without connection RSSI data is not available in Bluetooth, the Bluetooth node cannot understand which is its closest reference point. In this case, BTProximity provides, as current location, the set of the locations of all reference points in radio communication range.

Let us rapidly observe that BTProximity accuracy relevantly depends on required privacy level: the high-privacy level is intrinsically associated with a significantly lower accuracy than low and medium BTProximity privacy levels.

5.2. PSW Implementation Insights and Supported Infos/Features

The current PoSIM prototype includes wrappers for all the positioning systems presented in the previous section plus an additional generic PSW suitable for any positioning solution exposing a JSR-179-compliant API. Table 6 reports infos and features offered by the implemented PSWs and describes how they transform/represent gathered information to comply with the proposed ontology.

Table 6. Features/infos for the 4 positioning systems integrated in the current PoSIM prototype.

Positioning System	Category	Capability	PSW Implementation	Modifi-able
GPS	Info	Location	no required actions	n.a.
		PSState	*off* if invalid fix, *on* if valid fix	n.a.
		Timestamp	time of the last location update	n.a.
		Accuracy	dependent on HDOP	n.a.
		FixType	*no fix, 2D fix, 3D fix*	n.a.
	Feature	Name	GPS	no
		State	*on*: reading and parsing NMEA sentences *off*: not reading	yes
		ExploitedComm	serial port name, e.g., *COM2* or *rfcomm*	yes
		LocationType	physical	no
		PrivacyLevel	9 (stealth mode)	no
Ekahau	Info	Location	actions required to transform Ekahau map dependent information in absolute information	n.a.
		PSState	*off*: location information are not available *on*: location information are available	n.a.
		Timestamp	time of the last location update	n.a.
		Accuracy	either 5 (LatestLocation) or 7 (AccurateLocation)	n.a.
	Feature	Name	Ekahau	no
		State	*on*: RSSI sending and location gathering *off*: neither RSSI nor location gathering	yes
		ExploitedComm	IEEE 802.11a/b/g	no
		LocationType	both	yes
		PowerConsumption	dependent to the underlying IEEE 802.11 network interface (7 if always on, 4 if in power saving)	yes
		PrivacyLevel	either 3 (remote EPE) or 6 (local EPE)	no
		Accuracy	either 5 (LatestLocation) or 7 (AccurateLocation)	yes
		Status	detailed state information provided by EPE	no
BTProximity	Info	Location	no required actions	n.a.
		PSState	*off*: positioning deactivated, *on*: elsewhere	n.a.
		Timestamp	time of the last location update	n.a.
		Accuracy	8 if only one location, 6 if more than a location	n.a.
	Feature	Name	BTProximity	no
		State	*off*: positioning deactivated, *on*: elsewhere	yes
		ExploitedComm	Bluetooth device name, e.g., hci0	yes
		LocationType	symbolic	no
		PowerConsumption	2 (Bluetooth imposes limited power consumption)	no
		PrivacyLevel	5 (low), 7 (medium), 9 (high)	yes
JSR-179 (Location API for J2ME)	Info	Location	no required actions	n.a.
		PSState	*on*: state is *AVAILABLE*, *off*: elsewhere	n.a.
		Timestamp	time of the last location update	n.a.
		Accuracy	horizontal accuracy dependent	n.a.
	Feature	Name	JSR179	no
		State	*on*: gather location every second *off*: location gathering deactivated	yes
		ExploitedComm	JSR179	no
		LocationType	both	yes

GPS provides physical location information in terms of latitude, longitude, and altitude: no additional transformation actions on determined positioning data are required to be compliant with our default PoSIM ontology. The GPS PSW gathers information from a GPS device communicating through a serial port (possibly via a Bluetooth-based virtual serial port) by exploiting the standard Java Communication API [29]. This permits to achieve full portability independently of the underlying operating system.

In particular, when the State feature is on, the GPS PSW reads and parses NMEA 0183 sentences to achieve location information from the wrapped GPS positioning system. When that info is valid, i.e., the GPS device has a 2D or 3D fix, the PSState info value is set to on. The privacy level is fixed at the maximum value because the node computes its location in a completely decentralized manner, without any support by neighbors or network servers. Finally, the GPS accuracy is dynamically inferred from the Horizontal Dilution Of Precision (HDOP), a GPS-specific value dependent on the current configuration of the satellite constellation. In particular, our experiments have pointed out a rather linear relationship between HDOP values and accuracy in meters (see Figure 7). Therefore, the GPS PSW sets accuracy to 9 when HDOP is close to 0, to 0 when HDOP is greater than 30, and to linearly determined intermediate values otherwise.

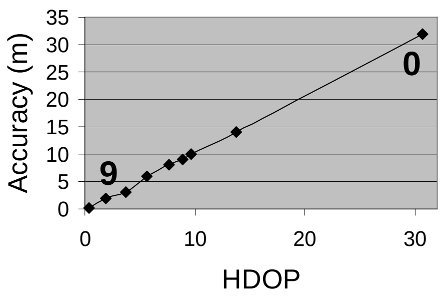

Figure 7. Experimental results about the relationship between HDOP and accuracy in GPS.

Ekahau can provide both physical and symbolic location data via event-driven API. However, positioning info is provided in relation to Ekahau internal maps; therefore, the Ekahau PSW must perform actions to transform the “proprietary” Ekahau location info accordingly to the exploited ontology. In particular, the Ekahau PSW is in charge of transforming physical coordinates and logical areas of Ekahau maps into latitude/longitude/altitude and layered location information, respectively, by exploiting additional context data related to maps. To this purpose, the only requirement is that Ekahau administrators orient their Ekahau maps to north and specify their top-left and bottom-right point coordinates, altitude, and possibly higher-layer symbolic location information, e.g., [Italy, Bologna], the country and the city where the map is located. When State is off, the Ekahau PSW stops locally gathering and sending RSSI data to EPE, with the benefit of relevantly limiting power consumption. To this purpose, we do not exploit the proprietary non-controllable Ekahau client but our own original and more flexible Ekahau client implementation with power consumption optimizations [6]. The Ekahau privacy level is limited: in fact, to gather RSSI values, the IEEE 802.11 node willing to be positioned must turn on its wireless card, thus providing the network infrastructure with a certain degree of knowledge of its location. Moreover, when the EPE server is not local (the location estimation is made by an EPE server not running on the client node itself), the location info is necessarily disclosed to the EPE node. Finally, let us note that Ekahau allows accuracy tuning: it is possible to request either an accurate or a latest computed location.

BTProximity only provides symbolic information via event-driven API, directly in the form required by the ontology. The BTProximity PSW accuracy depends on the number of locations provided; when only one location is provided, the accuracy is 8 (due to Bluetooth short range), when BTProximity provides a set of multiple locations corresponding to the visible Bluetooth reference points (e.g., because the privacy level is set to high), the accuracy level is set to 6.

Finally, as already stated, the JSR-179 PSW implements a generic wrapper to every JSR-179 compliant positioning system. The JSR-179 PSW provides both physical and symbolic information (when made available by the wrapped positioning solution). To test our implementation, we have developed a JSR-179 PSW encapsulating GPS devices and offering an interface that partially implements the JSR-179 API [6].

Figure 8 provides a global overview of our integrated positioning systems and related PSWs. An interesting aspect is that most middleware components are completely independent of the underlying operating system. On the contrary, each positioning system actively interacts, to some extent, with the operating system, often in a proprietary and non-portable way. Given that frequent dependency on operating system implementations, in order to maximize portability, we have designed and implemented a few interoperability middleware components (striped in the figure). Both GPS and Ekahau PSWs already have a good level of portability: on the one hand, the Comm API provides a portable access to serial port; on the other hand, the

`80211NetworkInterface` component allows to portably access IEEE 802.11 device features (we exploit the Comm API implementation provided by the rxtx project [30] since the Sun Comm API does not support Windows platform anymore). On the contrary, BTProximity can work only on Linux because its implementation is based on BlueZ drivers [31]; similar drivers are not yet available for Windows platforms (in particular, it is currently impossible to gather Bluetooth connection RSSI on Windows XP/Vista).

By delving into finer details, we have originally developed the `80211NetworkInterface` component to transparently gather IEEE 802.11 AP RSSIs. In fact, due to the relative novelty and high heterogeneity of wireless technologies, there is currently no standard specification, widely accepted by vendors and available in most common operating systems, of an application-level API to achieve full RSSI visibility. For this reason, we have implemented our own portable `80211NetworkInterface` component including different implementations of RSSI monitoring mechanisms, which are automatically selected depending on the characteristics of the client to be positioned. By exploiting `80211NetworkInterface`, our Ekahau PSW accesses an operating-system-transparent Java API to obtain the list of all APs in current visibility and their related RSSI data: for Linux clients, RSSI info is collected via the standardized Linux Wireless Extensions API [32]; for Windows CE/.NET clients, instead, `80211NetworkInterface` transparently binds the same Java API to the monitoring functionality provided by the Microsoft Network Driver Interface Specification User-mode I/O (NDISUIO), which is platform-dependent but portable among different wireless interface implementations [33]. For instance, it exploits the NDISUIO function `DeviceIOControl()` to query the `OID_802_11_ BSSID_LIST_SCAN` object to retrieve the complete list of currently reachable IEEE 802.11 APs. Finally, for rapid prototyping, testing, and evaluation purposes, the `80211NetworkInterface` component can also interwork with our original IEEE 802.11 simulator, thus making possible to exploit Ekahau positioning also if a real Wi-Fi deployment testbed is not available. For further details about our IEEE 802.11 simulator for positioning systems, please refer to `http://lia.deis.unibo.it/Research/SOMA/MobilityPrediction/`

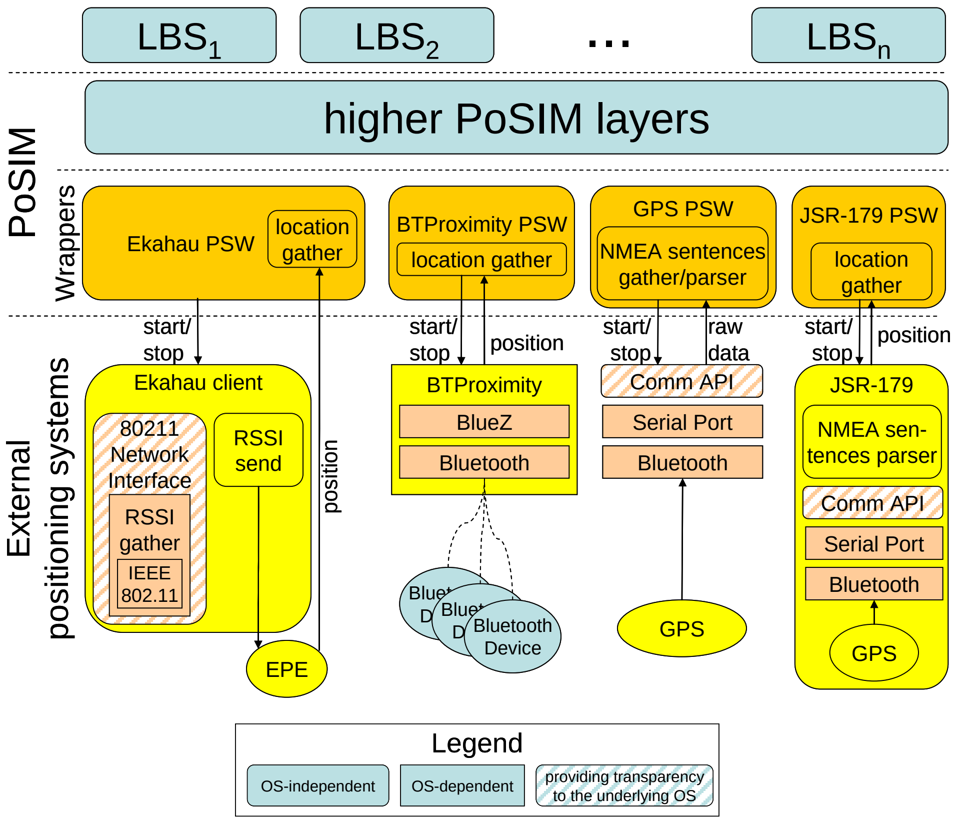

Figure 8. The detailed architecture of the implemented PSWs and integrated positioning systems.

Another interesting implementation insight is that PSW complexity greatly depends on the characteristics of wrapped positioning systems, in particular the positioning data format and the adopted access method. For instance, the BTProximity PSW can access the underlying positioning system via event-driven API, thus reducing possible overhead due to unnecessary polling; in addition, in this case, gathered information are already compliant with the adopted ontology. On the contrary, the GPS PSW has to command the reading of NMEA 0183 sentences provided by GPS via serial port and to parse these sentences to extract location information, while the Ekahau PSW has to transform map-related location information in absolute coordinates, thus increasing PSW implementation complexity. Further implementation details, out of the scope of this paper, and the PoSIM prototype source code are available at the project Web site [6].

5.3 An Example of PoSIM-based LBS

To practically show how PoSIM integrates heterogeneous positioning systems and supports rapid LBS prototyping and deployment, this section presents an example of LBS that takes advantage of PoSIM capabilities. In particular, we report about the development and testing of an Advertising service, deployed in a wide shopping mall consisting of several distributed buildings. The Advertising LBS aims to offer commercial information whenever a user is in the proximity of pre-defined locations, such as previously registered shops. Moreover, if the user accepts to disclose her location data, the LBS wants to record user paths for user movement pattern analysis, both inside buildings and in the paths between buildings. To gather the maximum amount of positioning-related data, the LBS needs to simultaneously exploit all the available positioning systems: GPS for outdoor localization, Ekahau for indoor physical and symbolic localization, and BTProximity for indoor symbolic localization.

Delving into finer details, it is possible to summarize the Advertising LBS requirements as follows: i) to simplify LBS working, the collected location information must be represented in a uniform way, ii) to correctly perform user tracking, physical information must be delivered at least every 10 meters, despite the actually exploited positioning system, and iii) a secondary requirement is to improve the robustness of LBS results by exploiting the location information from the most accurate source whenever multiple sources are simultaneously available.

PoSIM dramatically facilitates the design and implementation of such an Advertising LBS. First of all, PoSIM provides uniform location information in compliance with the representation syntax and semantic described in the adopted ontology. For instance, Table 7 reports the information provided by PoSIM inside a building: note that GPS accuracy is minimum because GPS is unsuitable for indoor localization (PSState is off), Ekahau provides both physical and symbolic location data, and BTProximity accuracy is only 6 because in this case it can only provide multiple locations of Bluetooth reference points in visibility.

Since Ekahau provides physical information in terms of latitude, longitude, and altitude, the `distance(...)` triggering event presented in Section 4.2 could be exploited even when the user moves from inside to outside a building, i.e., even when Ekahau becomes unavailable and GPS starts to be the positioning system actually providing the location data (and vice versa). In that way, the second requirement is easily fulfilled.

Table 7. The positioning information document provided by LBS in the Advertising LBS example.

```
<Data>
  <timestamp time="1173974696718" />
  <sources>
    <source name="GPS">
      <info Location = "physical: latitude =
          00.00 N, longitude = 00.00 E, al-
          titude = 50.0" />
      <info PSState="off" />
      <info Accuracy="0" />
      <info FixType="No fix" />
      <info Timestamp="..." />
    </source>
    <source name="Ekahau">
      <info Location = "physical: latitude =
          44.48 N, longitude = 11.32 E, al-
          titude = 103; symbolic: [(Italy,
          Bologna, ShopCentre, TravelAgen-
          cy)]"/>
      <info PSState="on" />
      <info Accuracy="7" />
      <info Timestamp="..." />
    </source>
    <source name="BTProximity">
      <info Location= "symbolic: [(Italy,
          Bologna, ShopCentre, TravelAgen-
          cy), (Italy, Bologna, ShopCenter,
          CoffeShop)]" />
      <info PSState="on" />
      <info Accuracy="6" />
      <info Timestamp="..." />
    </source>
  </sources>
</Data>
```

Finally, it is possible to answer to the third requirement by simply activating the `highAccuracy(8)` isolated policy and the `onBestAccuracy(1)` ordering policy. The former automatically deactivates positioning systems with accuracy lower than 8; the latter, with higher priority, always maintain the positioning system with highest priority switched on. Therefore, consequently to the enforcement of these two policies, the only positioning system available outdoor is GPS, and both Ekahau and BTProximity are deactivated there. When the node to be positioned moves indoor, the GPS accuracy rapidly decreases while Ekahau and BTProximity become available (BTProximity with high privacy level and thus with limited accuracy). When GPS accuracy goes lower than 8, the `highAccuracy` policy would try to deactivate it but the prioritized `onBestAccuracy` policy keeps GPS active because it is still the only positioning system available. Only when GPS accuracy goes lower than 7, i.e., below Ekahau accuracy, PoSIM deactivates GPS and activates Ekahau.

In place of `highAccuracy` and `onBestAccuracy` policies, the Advertising LBS could exploit the `onlyHighAccuracy` filter rule, thus gathering information only related to positioning systems with high accuracy. However, by adopting the two policies above, it is possible to achieve the additional goal of limiting power consumption because policies switch off positioning systems instead of just discarding unnecessary positioning information.

6. Conclusions

The widespread diffusion of several heterogeneous positioning systems pushes towards their integration to dynamically take maximum advantage of their capabilities also in a synergic way. Current middleware solutions for the integration of positioning systems lacks in dynamicity and flexibility: for instance, they typically do not allow to integrate positioning systems retrieved at service provisioning time and statically unknown. Moreover, they tend to hide underlying implementation characteristics, which are crucial for smart LBSs to have the needed fine-grained control of all the positioning solutions available on their client nodes.

The original translucent approach of the PoSIM middleware permits to access integrated positioning systems in both a transparent and middleware-mediated way, respectively fitting simple and smart LBS requirements. PoSIM not only supplies low-level information, but also permits an active control of integrated positioning systems, via the proper exploitation and/or definition of policies, events, and filters. In addition to providing a useful integration tool freely available for download and further refinement to the LBS community, the PoSIM project has also demonstrated, via practical examples, how the adoption of our middleware can leverage LBS development by relevantly facilitating both the synergic exploitation of positioning systems and the rapid LBS prototyping/deployment.

The encouraging results already obtained are stimulating further PoSIM-related research activities. We are extending the middleware openness by including even context sources not strictly related to positioning systems. For instance, we are currently testing an extended PoSIM prototype that can predict mobile node movements by monitoring RSSI sequences from IEEE 802.11 APs and Bluetooth peer nodes in visibility. The inference on future client movements may be exploited for different purposes, e.g., to dynamically change the interface exploited for wireless communications depending on previsions about user location and characteristics of nearby mobile nodes in next time intervals, thus realizing the needed first step towards a prompt and proactive always-best-connected communication system.

ACKNOWLEDGMENTS

Work supported by the MiUR FIRB TOCAI, the MiUR PRIN MOMA, and the CNR IS-MANET Italian projects.

References

[1] G. Chen, D. Kotz, "A survey of context-aware mobile computing research", Dartmouth College Technical Report TR2000-381, http://www.cs.dartmouth.edu/reports/

[2] J.G. McNeff, "The global positioning system", IEEE Transactions on Microwave Theory and Techniques, Vol.50, Iss.3, Mar 2002, pp. 645-652.

[3] Ekahau Positioning Engine, www.ekahau.com

[4] J. Hightower, G. Borriello, "Location systems for ubiquitous computing", Computer, Vol. 34, No. 8, Aug. 2001.

[5] J. Hightower, G. Borriello, "Location sensing techniques", UW CSE Technical Report, 2001.

[6] PoSIM Web site, http://lia.deis.unibo.it/Research/PoSIM

[7] JSR-179 Location API for J2ME, http://www.jcp.org/aboutJava/communityprocess/final/jsr179/index.html

[8] J. Nord, K. Synnes, P. Parnes, "An Architecture for Location Aware Applications", 35th Hawaii Int. Conf. on System Sciences, Hawaii, USA, Jan. 2002.

[9] Y. Hosokawa, N. Takahashi, H. Taga, "A System Architecture for Seamless Navigation", Int. Conf. on Distributed Computing Systems Workshops (MDC), Tokyo, Japan, Mar. 2004.

[10] M. Spanoudakis, A. Batistakis, I. Priggouris, A. Ioannidis, S. Hadjiefthymiades, L. Merakos, "Extensible Platform for Location Based Services Provisioning", Int. Conf. Web Information Systems Engineering Workshops, Rome, Italy, Dec. 2003.

[11] G. Coulouris, H. Naguib, K. Samugalingam, "FLAME: An Open Framework for Location-Aware Systems", Int. Conf. on Ubiquitous Computing, Goteborg, Sweden, Sept. Oct. 2002.

[12] Y. Chen, X.Y. Chen, F.Y. Rao, X.L. Yu, Y. Li, D. Liu, "LORE: An infrastructure to support location-aware services", IBM Journal of Research & Development, Vol. 48, No 5/6, Sept./Nov. 2004.

[13] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, M. D. Mickunas, "MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications",

ACM/IFIP/USENIX Int. Conf. on Middleware, Oct. 2004, Toronto, Ontario, Canada.

[14] J. Agre, D. Akenyemi, L. Ji, R. Masuoka, P. Thakkar, "A Layered Architecture for Location-based Services in Wireless Ad Hoc Networks", IEEE Aerospace Conf., Big Sky, Montana, USA, Mar. 2002.

[15] J. Hightower, B. Brumitt, G. Borriello, “The Location Stack: A Layered Model for Location in Ubiquitous Computing”, IEEE Work. on Mobile Computing Systems and Applications, Callicoon, NY, USA, Jun. 2002.

[16] D. Graumann, W. Lara, J. Hightower, G. Borriello, “Real-world Implementation of the Location Stack: The Universal Location Framework”, IEEE Work. on Mobile Computing Systems and Applications, Monterey, CA, USA, Oct. 2003.

[17] C. di Flora, M. Ficco, S. Russo, V. Vecchio, “Indoor and outdoor location based services for portable wireless devices”, Int. Conf. on Distributed Computing Systems Workshops (SIUMI), Columbus, Ohio, USA, Jun. 2005.

[18] P. Bellavista, A. Corradi, C. Giannelli, “Enhancing JSR-179 for Positioning System Integration and Management”, 1st Work. on Distributed Agent-based Retrieval Tools (DART'06), Pula-Cagliari, Sardinia, Italy, June 2006.

[19] B. Ramamurthy, H. Feng, D. Datta, J.P. Heritage, B. Murkjerjee, “Transparent vs. Opaque vs. Translucent, Wavelength-Routed Optical Networks,” OFC 1999 Tech. Dig., Washington, DC, Mar. 1999, vol. 1, pp. 59–61.

[20] G. Shen, R.S. Tucker, “Translucent optical networks: the way forward”, IEEE Communications Magazine, Vol.45, Iss.2, Feb. 2007, pp. 48-54.

[21] Jess, http://herzberg.ca.sandia.gov/jess/

[22] C.L. Forgy, “Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Artificial Intelligence, Vol. 19, No. 1, Sept. 1982, pp. 17-37.

[23] P. Bellavista, A. Corradi, C. Giannelli, "Efficiently Managing Location Information with Privacy Requirements in Wi-Fi Networks: a Middleware Approach", Second International Symposium of Wireless Communication Systems 2005 (ISWCS 2005), Siena, Italy, Sept 2005.

[24] P. Bahl, V.N. Padmanabhan, “RADAR: an in-building RF-based user location and tracking system”, 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), Vol. 2, pp. 775-784, Mar. 2000.

[25] P. Bellavista, A. Corradi, C. Giannelli, "Adaptive Buffering based on Handoff Prediction for Wireless Internet Continuous Services", The 2005 International Conference on High Performance Computing and Communications (HPCC-05), Sorrento, Italy, September 2005.

[26] A. Genco, “Three Step Bluetooth Positioning”, International Workshop on Location- and Context-Awareness (LoCA 2005), Munich, Germany, May 2005.

[27] G. Anastasi, R. Bandelloni, M. Conti, F. Delmastro, E. Gregori, G. Mainetto, “Experimenting an indoor bluetooth-based positioning service”, Int. Conf. on Distributed Computing Systems Workshops (ICDCSW’03), Providence, Rhode Island, USA, May 2003.

[28] Bluetooth Special Interest Group, "Bluetooth Specification Version 2.0 + EDR", www.bluetooth.org, Nov. 2004.

[29] Java Communications 3.0 API, http://java.sun.com/products/ javacomm/

[30] “RXTX : serial and parallel I/O libraries supporting Sun's CommAPI”, http://www.rxtx.org/

[31] M. Holtmann, "BlueZ, Official Linux Bluetooth protocol stack", http://www.bluez.org.

[32] J. Tourrilhes, "Wireless Extensions for Linux", http://www .hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Ex tensions.html

[33] "NDIS - Network Driver Interface Specification", http://www. microsoft.com/whdc/device/network/ndis/default.mspx