Coupling Transparency and Visibility: a Translucent Middleware Approach for Positioning System Integration and Management (PoSIM)

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 diffusion of wireless terminals with multiple communication interfaces and the proliferation of heterogeneous positioning techniques opens new possibilities for Location Based Services (LBSs), even if potentially complicating their development. In particular, LBSs can significantly benefit from middleware supports to uniformly access the set of positioning systems available at their wireless clients and to dynamically choose the most suitable one depending on application/execution context, also by fusing concurrent positioning data from multiple sources. The paper presents our PoSIM middleware for the integrated and synergic access/control of heterogeneous positioning systems in a highly flexible way. PoSIM provides LBSs with two differentiated levels of visibility for positioning management: a transparent way based on high-level context-based policies, and a fully-aware access to advanced functions/configurations, mediated and uniformed by the middleware independently of the underlying positioning solution. In particular, the paper mainly concentrates on the description of the PoSIM architecture and of its API, by pointing out our primary design and implementation choices.

Keywords: Mobile Computing, Positioning, Middleware, System Management, System Integration.

I. INTRODUCTION

The growing availability of powerful mobile devices with relatively high wireless bandwidth, e.g., via UMTS, IEEE 802.11, and Bluetooth 2.0 connectivity, is going to leverage the widespread diffusion 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, positioning techniques are crucial. Several research activities have deeply worked on evaluating mechanisms and technologies for positioning: some solutions have been specifically designed for determining location, e.g., the well known Global Positioning System (GPS); other proposals try to estimate positioning information by monitoring characteristics of general-purpose communication channels, such as the IEEE 802.11-based Ekahau [2]. For a more exhaustive positioning system survey please refer to [3, 4].

Currently available positioning solutions greatly differ on capabilities and provided facilities. For instance, they diverge in:

- the representation model of the provided location information. That 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;
- the applicable deployment environment. For instance, GPS can work only outdoor, Ekahau primarily indoor;
- 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, which typically depends on location update frequency;
- user privacy, e.g., maximum for GPS, limited and dependent on the deployment environment for Ekahau. In fact, GPS determines location in a completely decentralized and autonomous manner. On the contrary, in Ekahau-based positioning, clients have to disclose their locations, to some extent and at a certain granularity, to be capable of communications (clients must associate to an AP for communication purposes);
- and additional supported features, which can be peculiar of specific positioning systems. For instance, some positioning solutions can provide location data as a probability distribution function.

That heterogeneity among the available positioning systems, together with 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), motivate the need for novel middleware solutions capable of integrating the available positioning techniques, of controlling them in a synergic way, and of dynamically selecting the most suitable positioning solution depending on execution context. First of all, that middleware should allow to seamlessly switch from a positioning system to another depending on

availability, e.g., GPS outdoor and Ekahau indoor. Then, it should suggest to exploit, at any time, the positioning technique which best fits user preferences, application requirements, and device resource constraints. For instance, the positioning system with lower power consumption in the case of priority given to battery preservation, or the one with greater accuracy and precision, or the one with most frequent updates, or the one providing either physical or symbolic location information. Moreover, when several positioning systems can concurrently work, the middleware could perform fusion operations on location data, e.g., to increase accuracy and/or confidence. For all these purposes, there is the need to make also low-level characteristics of positioning systems easily accessible to the upper layers (middleware level and/or application level), thus enabling application-specific control of positioning techniques, possibly by avoiding to complicate LBS development and deployment.

By taking into account the above motivations, we have designed and implemented the Positioning System Integration and Management (PoSIM) middleware. PoSIM primarily focuses on three aspects. First of all, it is capable of integrating positioning systems at service provisioning time in a plug-in fashion, by exploiting their possibly synergic capabilities and by actively controlling their features. Secondly, PoSIM allows positioning systems to flexibly expose their control/configuration features and location information at runtime, without requiring static knowledge of positioning-specific characteristics. Third, it can perform location data fusion depending on applicable context, e.g., application-specific requirements about accuracy or client requirements about device battery consumption.

In addition, PoSIM 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 integrated positioning in a transparent way at a high level of abstraction. In fact, LBSs may simply specify the behavior positioning systems must comply with via declarative policies; PoSIM is in charge of actually and transparently enforcing the selected policies. On the other hand, PoSIM allows LBSs to have full visibility of the characteristics of the underlying positioning systems via a PoSIM-mediated simplified access to them. In this case, PoSIM provides LBSs with a uniformed API, independently of the specific positioning solution, that permits to access/configure all the available localization systems homogeneously and aggregately. We call *translucent* the original PoSIM approach that supports LBSs with both transparent and visible integrated access to available positioning solutions.

After a brief positioning of the PoSIM proposal with regards to the main literature in the field, the paper presents the primary design guidelines behind the original translucent architecture of the PoSIM middleware. Then, the paper describes the main choices followed in the implementation and deployment of the PoSIM components. Conclusive remarks and directions of on-going research end the paper.

II. Related Work

Several research activities have recently addressed the area of dynamically fusing positioning information from different sources [5, 6]. Most solutions propose transparent approaches that hide applications from positioning complexity, but do not support any application-specific form of control on the positioning techniques currently available at a node.

Only a few proposals have just started to delineate cross-layer supports that provide application-level visibility of low-level details and control features of available positioning techniques [7, 8]. However, [7] only claims the need for cross-layer middleware solutions to smartly select the most suitable positioning system at runtime. [8], instead, supports the control of positioning systems in a hard-coded and not flexible manner. In addition, to achieve the visibility of data and control features of a specific positioning system, [8] requires its full static knowledge, thus significantly increasing the LBS development complexity.

The JSR-179 API [9], also known as Location API for J2ME, represents a notable result of standardization effort for Java-based LBSs on mobile phones. In particular, JSR-179 provides a standardized API to perform coarse-grained integration and control of positioning systems. For a more detailed related work and a thorough comparison of the functionality offered by the JSR-179 and the PoSIM APIs, please refer to [10].

PoSIM considers the aforementioned contributions and answers similar issues by greatly improving the dynamicity, flexibility, and extendibility of the support for positioning integration and management. To the best of our knowledge, no support solution in the literature addresses the challenge of cross-layer integrated control of available positioning systems by considering runtime application-level requirements in a flexible and extensible way and at dynamically differentiated levels of visibility.

III. The PoSIM Architecture

PoSIM provides LBS developers with integrated management of heterogeneous positioning systems by adopting a *translucent* approach. Thanks to PoSIM intermediation, simple LBSs can interact transparently with positioning systems perceived as a unique, multi-behavior service. In this case, LBSs can control positioning systems easily, by just specifying the required behaviors via declarative policies or by simply selecting the policies to enforce among the predefined ones, e.g., to give priority to low energy consumption. Smart LBSs, i.e., applications willing to have direct visibility and to manage peculiar data/features of positioning systems, can interact in a middleware-mediated but fully aware fashion: they can have a PoSIM-based uniform access to all the characteristics of a specific positioning system, e.g., to limit the accuracy of Ekahau-based positioning to reduce its network overhead.

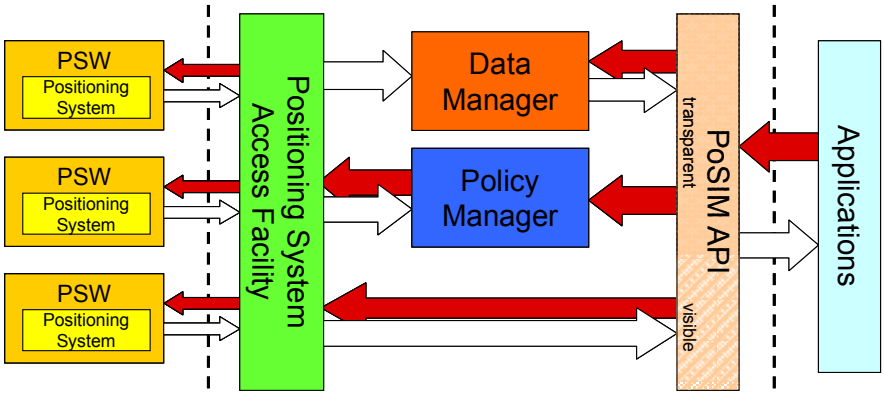

Figure 1. PoSIM components (white arrows represent data flows, coloured arrows control flows).

Figure 1 depicts our PoSIM middleware architecture. To interact with positioning systems in a transparent manner, simple LBSs can exploit the Policy Manager (PM) and Data Manager (DM) APIs to respectively control positioning system behavior and get their location information. To interact in a visible and more flexible manner, smart LBSs can exploit the Positioning System Access Facility (PSAF) to access the Positioning System Wrappers (PSWs) for the currently available and integrated positioning systems.

About the control and data arrows in Figure 1, let us anticipate that PM can control positioning systems but does not provide any location data, while DM exposes location information according to dynamically configurable differentiated modes. PSAF, instead, provides LBS components with both control capabilities and positioning data. Nevertheless, PoSIM appears to the application level as a middleware component offering a single and flexible API, thus simplifying its usage and potentially leveraging its adoption.

PoSIM does not assume the adoption of any particular or pre-defined common ontology to be used by all the integrated underlying positioning systems to uniformly represent their location data and control features. PoSIM distinguishes only between *features* and *info*. Features describe positioning system characteristics and capabilities, possibly with settable values and useful for the control/configuration of positioning systems, e.g., power consumption or ensured privacy level. Info is location-related data, e.g., actual positioning information and its accuracy, not modifiable from outside the positioning systems. Info is the only data provided to simple LBSs while smart LBSs have visibility of both features and infos.

IV. PoSIM Components: Design and Implementation Insights

As briefly stated in the previous section, the main PoSIM components are PM, DM, PSAF, and PSW. In the following, the paper presents the main design and implementation guidelines for each of these components, together with an overview of their offered APIs. For additional details about the implementation of PoSIM components and their API specification, please refer to [11].

A. Policy Manager

The Policy Manager (PM) is the PoSIM component responsible for transparently enforcing policies about positioning system integration and management. In particular, the PM API allows simple LBSs to ask for pre-defined behaviors implemented as declarative policies, without any knowledge of how the currently available positioning systems are actually exploited. For example, a PoSIM-enforced policy could turn off positioning systems with relatively high energy consumption if that does not endanger application-specific requirements about precision and accuracy. Figure 2 depicts an example of `lowPowerConsumption` policy that switches off a currently available positioning system if its power consumption is greater than 8 and its accuracy is below 5 (rapid notes about the mapping between power/accuracy values in the policy and their counterparts in the integrated positioning systems are in the following).

```
name:lowPowerConsumption
condition:
        Feature(name:Power, value: 8) op:greater
        Info(name:Accuracy, value:5) op:lower
action:
        Feature(name:State, value:off)
```

Figure 2. The `lowPowerConsumption` policy.

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 related to currently enforced policies, namely *relevant data*, iii) requests the Policy Engine (PE) to perform policy enforcement, i.e., to check for policy condition satisfaction and possibly to trigger policy operations.

Delving into finer design/implementation details, the PoSIM PE exploits Jess, a rule engine based on the Rete algorithm [12, 13]. PC 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 info and features that appear in at least one active policy condition, that is only info and features relevant for active policies. In that way, PC only retrieves the needed monitor-

ing indicators from the underlying positioning systems, thus limiting the middleware overhead.

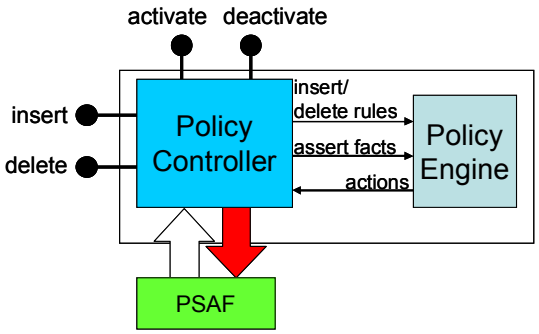

Figure 3. The PoSIM Policy Manager.

Currently, PE enforces policies by following the standard Jess "depth" (age-based) strategy, i.e., if more than one active policy is triggered, it performs the enforcement of the most recently activated one, and then enforces the other ones. We are working on extending PE to assign different priorities to policies and to consider those priorities when determining the order with which to enforce policies among the triggered ones.

B. Data Manager

The Data Manager (DM) works to offer an aggregated view of positioning information to the application level. In particular, DM provides simple PoSIM-based LBSs with the info produced by the different integrated positioning systems as an aggregately single XML document, whose tags are exploited to specify the semantic of the provided content. Simple LBSs can specify when and which positioning information they are interested in via the DM APIs: location retrieval is possible either on request, or specifying a time period, or via event notification. In particular, LBSs can simply define the conditions to trigger location data delivery. For instance, the pre-defined `atChanges` condition triggers location notification only when current and previous physical location differ of more than a specified threshold.

In addition, LBSs may request DM to filter positioning data. For instance, the pre-defined `highAccuracy` filter automatically discards location information with accuracy below a given threshold. Note that the proper exploitation of filtering rules permits to reduce the network overhead due to non-relevant changes of location data. Triggering events and filtering rules are implemented as Java classes, which can be easily sub-classed to specify specialized triggers and filters.

Delving into finer details, the Data Builder (DB) subcomponent collects info from the currently exploited positioning systems and possibly aggregates them with context information. Data Disclosure (DD) offers an appropriate API to specify how LBS is willing to get data. In particular, `onDemand(listener)` immediately provides the already available XML-based document with positioning data, `periodical(interval, listener)` installs a periodical delivery process every `interval` milliseconds, `addEvent(event,listener)` registers a specific event to trigger the XML-based document delivery. Moreover, DM also provides the `addFilter(filter,listener)` method to register new filters.

In other words, fed by DB monitoring information, DD delivers the XML document with positioning data to every registered listener whose delivery period is expired or triggered by an associated event applies. The delivered XML-based document is the result of filtering the raw positioning data with the filters provided by the interested listeners. Let us observe that each method above in the DM API allows to specify a listener; that increases the flexibility of our middleware solution if compared with other recently emerging proposals for positioning integration [10].

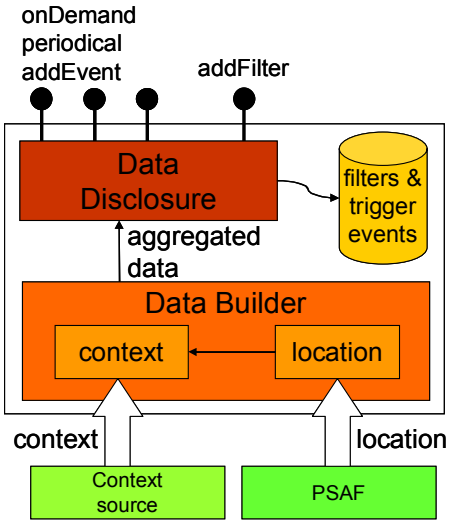

Figure 4. Data Manager.

Finally, expert users, such as PoSIM administrators, can develop and deploy new policies, new triggering events, and new filtering rules. In that way, the PoSIM behavior can be specialized and extended with impact on neither its implementation nor the application logic code. Anyway, simple LBSs and novel developers can simply work by selecting among the existing set of most common policies, events, and filters. For a detailed description of pre-defined policies, triggering events, and filters included in PoSIM, please refer to [11].

C. 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 info/features of all the currently available positioning systems. The only requirement is that positioning systems provide their info/features via a specified interface; that interface is obtained by wrapping the actual positioning system components with PoSIM Positioning System Wrappers (PSWs). PSAF exploits Java introspection to dynamically determine and access the set of info/features actually implemented by the underlying positioning systems that are currently avail-

able in its deployment environment.

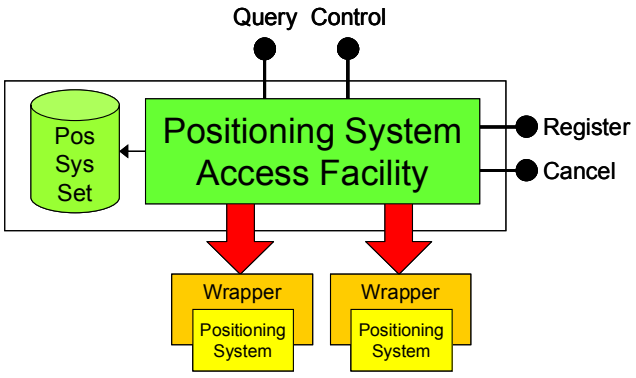

Figure 5. Positioning System Access Facility.

In particular, the PSAF API allows:

1. to dynamically register/cancel a (PSW-compliant) positioning system in order to insert/delete it from the available positioning system set;
2. to interact with registered positioning systems via the Query/Control interface.

The PSAF Query/Control interface permits to interact with registered positioning systems in an aggregated manner. In particular, the Query interface consists of the following methods:

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

The Control interface offers the method:

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

Both smart LBS components and PoSIM middleware components can invoke the Query/Control methods; the Register/Cancel interface can be accessed only by PoSIM administrators. Let us stress that higher layers are allowed to interact with integrated positioning systems only via PSAF, thus guaranteeing controlled and system-safe accesses to low-layer positioning components, independently of their specific technique and implementation peculiarities.

D. Positioning System Wrapper

As already pointed out, the Positioning System Wrapper (PSW) is the crucial middleware component to hide positioning heterogeneity. It offers a common API, independent of the specific positioning system and its implementation details, by providing info/features compliant to a common pre-defined ontology for positioning-related data. For example, the PSW `getAccuracy()` method could provide location accuracy specified as an integer value in the [0, 9] range. Any specific PSW component interacts with its wrapped positioning system, retrieves the associated accuracy value by exploiting positioning-specific awareness and syntax, and transforms it accordingly to the adopted ontology, e.g., transforming a "high accuracy" string return value in the correspondent integer. The ontology is shared among the PoSIM components and permits to specify policies, triggers, and filters independently of the positioning implementation details.

Delving into finer details, PSW offers:

- the `getX()` method for every feature provided by the wrapped positioning system, where *X* is the name of the feature;
- the `setX(value)` method for every modifiable feature, where `value` is the new value of the feature;
- the `infoX()` method for every 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.

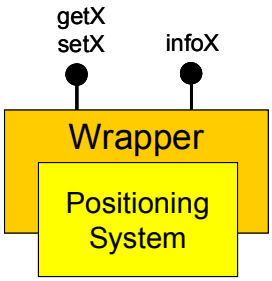

Figure 6. Positioning System Wrapper.

V. CONCLUSIONS

The widespread diffusion of several heterogeneous positioning systems pushes towards the adoption of their integration to dynamically take advantage of their peculiar 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 newly introduced positioning systems at service provisioning time. Moreover, they tend to hide specific characteristics and underlying implementation details, which are crucial for smart LBSs to have the needed fine-grained control of currently available positioning techniques. The paper proposes the original PoSIM middleware based on our translucent approach: the middleware permits to control integrated positioning systems both in a transparent and non-transparent way, respectively fitting simple and smart LBS requirements.

The encouraging results already obtained in the PoSIM project are stimulating further related research activities. We are extending the middleware openness by including an additional wrapper for our original Bluetooth-based positioning system (currently the PoSIM prototype includes wrappers for GPS and Ekahau). Moreover, we are extending the

set of PoSIM-supported criteria, filter rules, and triggering events, in order to fit all the personalization requirements of most common LBSs by simply requesting LBS developers to select which integration/control strategy should be applied among the pre-defined ones.

ACKNOWLEDGMENTS

Work supported by the FIRB WEB-MINDS Project of the Italian Ministry of University and Research and by the Strategic IS-MANET Project of the Italian Research Council.

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] http://www.ekahau.com

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

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

[5] 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. Middleware, 2004.

[6] M. Spanoudakis et al., “Extensible platform for location based services provisioning”, Int. Conf. Web Information Systems Engineering Workshops (WISEW), 2003.

[7] D. Graumann, W. Lara, J. Hightower, G. Borriello, “Real-world implementation of the location stack: the Universal Location Framework”, IEEE Workshop on Mobile Computing Systems & Applications (WMCSA), 2003.

[8] J. Agre, D. Akenyemi, L. Ji, R. Masuoka, P. Thakkar, "A layered architecture for location-based services in wireless ad hoc networks", IEEE Aerospace Conference, Mar. 2002.

[9] JSR-179, http://www.jcp.org/aboutJava/communityprocess/final/jsr179/index.html

[10] 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.

[11] PoSIM, http://lia.deis.unibo.it/Research/PoSIM

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

[13] Charles 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.