# FLEXIBLE FRAMEWORK FOR LOCAL WIRELESS SERVICES

Tero Hakkarainen
*VTT Information Technology, Finland*
*P.O.Box 1206 (Sinitaival 6), FIN-33101 Tampere*
*tero.j.hakkarainen@vtt.fi*

Ali Lattunen
*VTT Information Technology, Finland*
*P.O.Box 1206 (Sinitaival 6), FIN-33101 Tampere*
*ali.lattunen@vtt.fi*

Vespe Savikko
*VTT Information Technology, Finland*
*P.O.Box 1206 (Sinitaival 6), FIN-33101 Tampere*
*vespe.savikko@vtt.fi*

**ABSTRACT**

One of the main objectives in setting up wireless services is to provide feasible content for the mobile users wherever and whenever they might need it. The dream of having services with meaningful content available at the right place, at the right time and with lowest possible cost is coming true with local wireless services. Local service development has its unique set of problems, like the dynamic nature of the service environment and co-operation of different wireless communication systems. In this paper, we present a flexible framework that takes these aspects into consideration, models them in an intuitive way and provides an easy-to-use service development approach.

**KEYWORDS**

wireless, wireless web, local wireless services.

## 1. INTRODUCTION

In an increasingly tough race for customers, wireless service providers are searching for new ways of gaining added value for their services. Location-based services with individually selected content are regarded as one of the most promising service concepts for providing both innovative and meaningful wireless applications, especially when available through charge-free networks.

Local wireless services concept is an approach for providing pertinent wireless content only there, where it is relevant. Selected location-sensitive services are provided within application-specific areas. Elegant content selection can be performed not only by user's location but also by context, time and user preferences. Local services are normally accessed by handheld terminals like mobile phones or palmtop computers. Terminal requirements for service access are an appropriate content browser and sufficient communication capabilities.

In a technical sense, local wireless services can provide all the same services as cellular network based services do. In addition, their special characteristics, especially location awareness, enable a set of more attractive applications. Typical applications for local wireless services are information kiosks, niche marketing, guidance services, wireless gaming, m-payment, local chat and dating services. Examples of local wireless services are depicted in Figure 1.

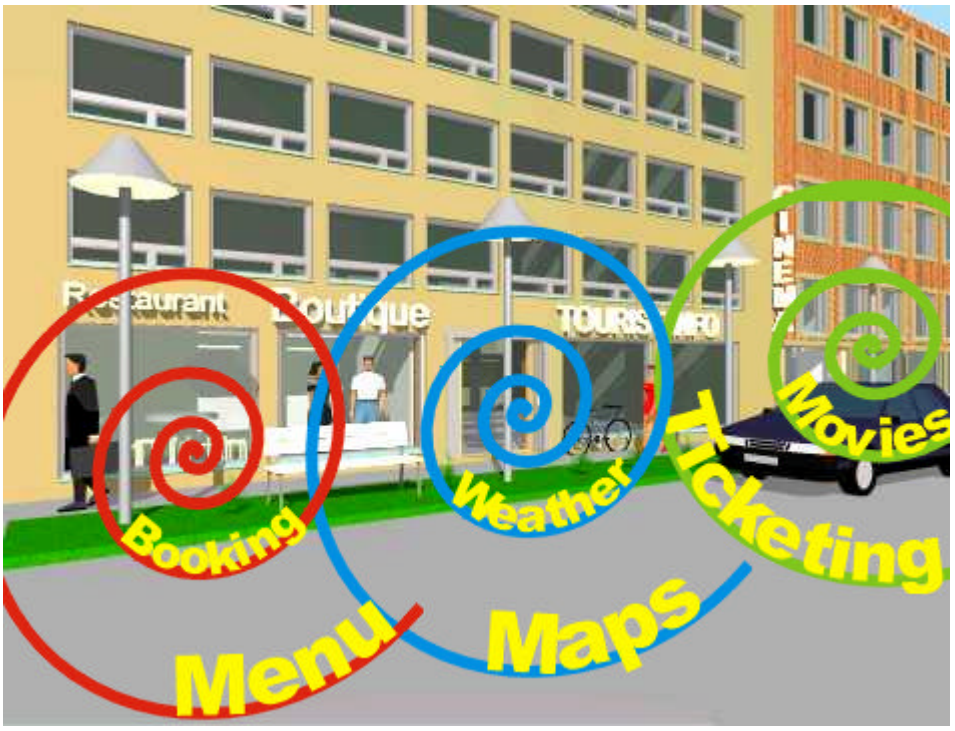


*Figure 1. Local wireless services in urban environment.*

There are certain aspects in development of local wireless services that are different from those of traditional wireless service development: the dynamic nature of local wireless environment and the co-operation of different wireless communication systems. Thus, a flexible development framework is needed for taking these special aspects into consideration and providing a familiar service development approach.

In this paper we present Flower, a development framework for local wireless services developed at VTT Information Technology. First, we look at related work in Section 2. Next, we further define the problem space by stating the design goals for the development framework in Section 3. Then, we present the overall architecture of Flower in Section 4. In Section 5, we describe the key aspects of application programming in Flower. After this groundwork, we go deeply into the issues of flexibility in Section 6, where we look at different local network configurations and show how Flower deals with them. Next, we further discuss flexibility related issues in Section 7. Finally, we wrap this paper up with conclusions and experience gained from a pilot project in Section 8.

## 2. RELATED WORK

To our knowledge, no general-purpose local service programming environments have been introduced so far. Instead, most work in the area has been focused on creating specific service applications, as seen in e.g., Impulse (Youll et al. 2000) and GUIDE (Cheverst et al. 2000).

Impulse introduces user agents that seek context-sensitive information based on users' interests and position data retrieved from GPS receivers. Information providers are Internet resources registered to a special server that binds the providers to specific geographical locations. Data is transferred wirelessly to the user terminal.

GUIDE is an intelligent electronic tourist guide. It is used in the city of Lancaster, where tourists may loan wireless tablet terminals and access context-sensitive information on city attractions. The data is transferred through a cell-based wireless communication network which also offers a positioning service.

In Flower framework, the intention is not to implement end-to-end local service applications, but to offer developers the necessary tools for creating them. Since the framework strives for generality, tight bindings to specific hardware and application-specific optimisations are not acceptable. This is by far the most important distinction between Flower and systems described above.

## 3. FRAMEWORK DESIGN GOALS

The communication capabilities of mobile terminals are increasing steadily. In the past few years, many new technologies have emerged for both local and wide area communications. For the benefit of wireless local services, the technologies offer a decent data rate and are able to transfer hypertext content.

So far, however, only few products have actually utilised the technological potential with local services. One reason for this is the incompleteness of the standardisation work, which has also contributed to the poor availability of local service enabled terminals. Another reason is the unsuitability of currently available service development platforms as such for local service development.

The life span of a service framework is strongly dependent on the framework characteristics, which must therefore be carefully laid out. We have found the following characteristics vital:

**Use of established web technologies.** Building the local service framework on established web technologies makes it possible to put most of the effort on the framework elements that are local service specific. Another advantage for using existing and widely used technologies is the enhanced learning curve for developers migrating from web programming.

**Modularity.** The incompleteness of the local service standardisation work creates an urgent need for carefully addressed modularity, which makes it possible to replace any framework module, if necessary. Explicitly defined module responsibilities also facilitate the addition of new features and support for new technologies.

**Scalability.** Scalability is an important characteristic for a service platform that may have to serve a single terminal or a large amount of terminals. The number of service access points in the framework must be freely adjustable by the service provider. This, in turn, creates high requirements on the stability and robustness of the framework.

**High performance and lightness.** The framework should be able to serve several terminals concurrently. On the other hand, fitting the whole framework into a laptop is an important goal to strive for. Many local service applications would benefit from a platform that could be easily moved from one place to another.

**Support for various communication technologies.** The local service framework should not be bound to any single communication technology. A versatile set of supported technologies has a strong influence on the attractiveness of the framework. Functional entities that are either technology or manufacturer dependent must be encapsulated into separate modules that communicate with the rest of the system through a standard interface.

**Support for a range of programming languages.** In web programming, Java is one of the most popular languages today. Local service framework should therefore have a Java application programming interface. C++ interface would be equally valuable since the language has a large existing code base. With C++ it is also easier to connect to physical peripherals in, e.g., remote control applications.

**Application programming interface adapted to local service environment.** The control over the dynamic nature of local service environment, most importantly the nomadic terminals, must be given special attention in the framework. For certain types of services, an accurate model of the physical service environment is advantageous. In general, dynamically updated information on changes in the framework state must be offered to local service applications.

## 4. ARCHITECTURE

The overall architecture of the Flower system is dictated by the need to attract the web developers. In other words, the locality aspects of the Flower applications should not hinder the use of the traditional web development techniques. At the same time, the Flower application programming interfaces should fully

integrate into the local service environment and its characteristics. Flower's solution is to model applications as servlets (Sun Microsystems 1999). Figure 2 shows the key components of the Flower system:

**Access Points.** Client terminals connect to the Flower system through access points using some wireless communication mechanism. The protocol for information exchange can be either HTTP (Hypertext Transfer Protocol) or WAP (Wireless Application Protocol). The role of access points is to act as transparent channels between terminals and Flower applications. Additionally, access points monitor their service range and try to detect the presence of client terminals. Gathered client information is synchronised with the Flower Manager. Framework-wide support for different communication techniques is implemented in access point level, which means that there are separate access point implementations for each supported technique. Implementations not using short-range communications benefit from entities called virtual access points, which are used for adapting to the local service environment. Virtual access points are presented in Section 6.

**WAP Gateway.** WAP content is routed through a WAP gateway. The gateway acts as a converter between WAP and HTTP, thus enabling the Web Server to serve both WAP and web terminals. In Flower framework, WAP Proxy (VTT Information Technology 2000) gateway supporting PAP (Push Access Protocol) (WAP Forum 2001c) is used. The push mechanism allows a Flower application to send content to clients without their initiative.

**Flower Layer.** The Flower Layer implements the Flower API, the application programming interface of the Flower system, on top of the Servlet API. In the Flower API terminology, applications are called flowerlets since they have both similarities and differences with servlets. The Flower API is discussed in more detail in Section 5.

**Flower Manager.** The Flower Manager is a server inside the Flower framework that monitors the status of the access points and terminals connected to them. Furthermore, the Manager is accessible to the flowerlets through the Flower API.

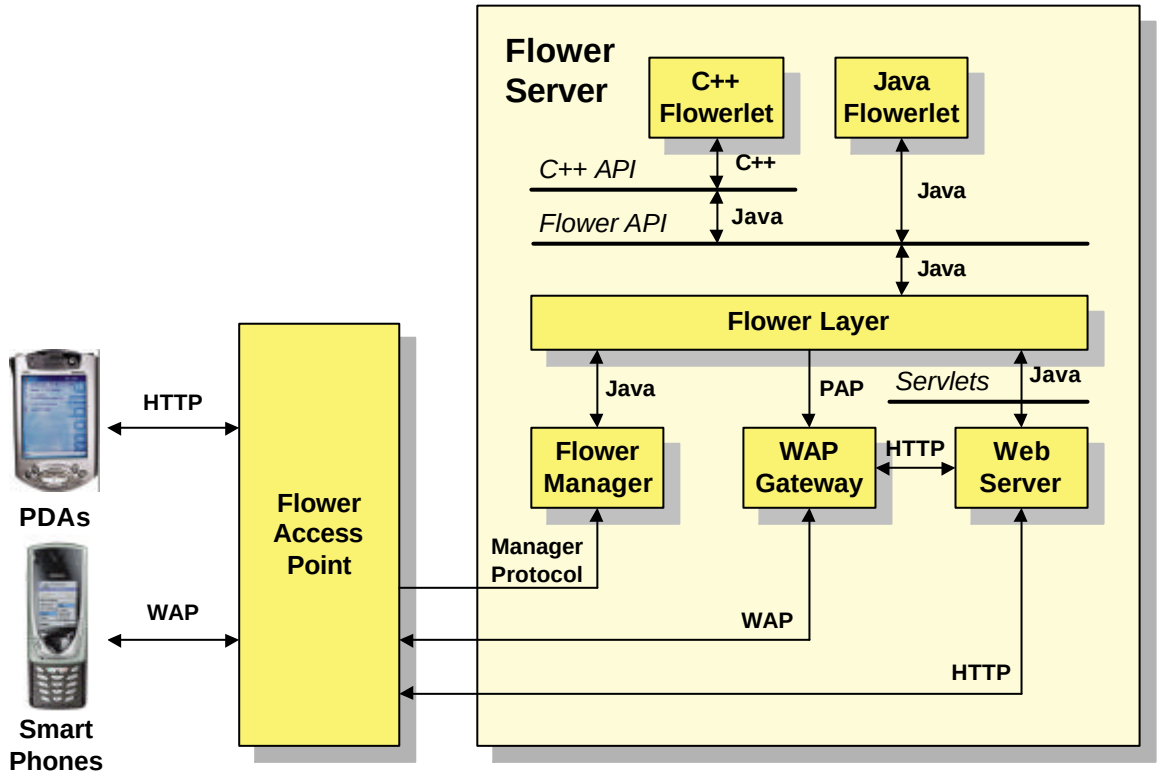


*Figure 2. Flower system architecture.*

## 5. FLOWER API

The Flower API is built as a layer on top of the Servlet API. The layer hides those aspects of servlets that are either not applicable to flowerlets or have been implemented differently by the Flower API (e.g., session mechanism). The structure of the Flower API is nearly identical to the Servlet API, thus making it intuitive to the experienced servlet developers. The basic mode of operation is naturally similar to servlets: flowerlets receive requests and create responses. The following services can be utilised through the Flower API:

- Convenient access to the Flower Manager data
- Event-based notification mechanism for changes in the Flower system state
- Automatic data routing to and from client terminals
- Acknowledgement and push mechanisms
- C++ interface

The Manager of the Flower system is modeled as a Singleton (Gamma et al. 1995) object and is thus conveniently accessible from flowerlet code. The Manager interface contains methods for querying the status of the system, which consists of active access points and connected terminals. Since access points and terminals have their Java class counterparts in the Flower API, they can be conveniently utilised in the flowerlet code. The Manager interface also provides an event mechanism that makes it possible for a flowerlet to get an instant notification about the changes in the system state (e.g., a terminal is discovered by an access point). When the Flower Layer creates a request for a flowerlet, it augments the request with corresponding Manager information, like the originating terminal. The augmented information is used, most importantly, in issuing a response. In addition to the Manager related details, the request also contains information about the protocol used. This means that a flowerlet can format its responses accordingly (e.g., WAP, HTTP).

In addition to basic transactions, the Flower API also supports acknowledgement and push mechanisms. The former simply means that, depending on the type of the response, the corresponding access point can notify the Manager whether the response could be sent to the target terminal. The Manager object of the Flower API further notifies the waiting flowerlet thread. This kind of mechanism is essential for local services since a client terminal is typically mobile and can leave the access point service area at any time.

The actual push mechanism is implemented by WAP Gateway utilising PAP messages. However, since formatting a correct multi-part MIME (Multipurpose Internet Mail Extensions) message can be an error-prone task, the Flower API has a direct support for Service Indication (WAP Forum 2001a) and Service Loading (WAP Forum 2001b) messages.

Although the Flower API is modelled according to the Servlet API, C++ developers have also been taken into consideration. A C++ version of the Flower API is provided as part of the Flower system. The API is implemented as a wrapper for the Java version using the Java Native Interface (JNI). On the other hand, this choice makes it possible to retain the same level of functionality between the two alternatives, but it also means that C++ developers have to be familiar with some JNI features. However, since the C++ implementation encapsulates most of the JNI code, the latter aspect is not very significant in practise.

The servlet engine, e.g., Apache Tomcat (Apache Software Foundation 2002), treats flowerlets as servlet instances and thus the configuration aspects of the servlets are applicable to flowerlets as well. In other words, flowerlets can be grouped into web applications and configured with standard XML (Extensible Markup Language) files as specified by the servlet specification (Sun Microsystems 1999). This also means that flowerlets can peacefully coexist with a more typical content of a web server, thus making it possible to create applications that cross application area boundaries. For example, a local service could offer a World Wide Web interface as well, thus making certain features globally accessible.

## 6. FLEXIBILITY

In the near future, local wireless services would be most beneficial for urban areas with a large concentration of users and most of the application areas. However, terminals will undoubtedly fall into a number of categories (PDAs, cellular phones, laptops, etc.) utilising various communication techniques. Although the diversity of framework access methods is a strength for Flower, it also causes problems in mapping the user locations into service areas, which is an important process in respect of the concept of

locality. With short-range communication techniques (e.g., Bluetooth), users can be explicitly located due to reasonably small service areas. The case is different for wider area communication techniques (e.g., GSM, WLAN) where the service area is considerably larger. More sophisticated positioning systems must therefore be employed. Thus, for a local service framework, flexibility and interoperability are the key characteristics.

The problem in locating users in large service areas can be tackled by defining an entity called virtual access point. Virtual access point is an abstract entity that is always related to exactly one physical access point. It covers a portion of the physical access point's service area. Virtual access points could typically be created for

- dividing large physical service areas into smaller, more maintainable fragments
- giving logical meanings (e.g., subway entrance) to fragments of physical services areas.

Figure 3 shows a sample diagram of two physical access points. The service area of the larger access point contains three virtual access points. The resulting configuration consists of five access points (two physical and three virtual) showing up as individual functional entities.

As stated previously, physical access points must implement the positioning of users in the service area to the corresponding virtual access points. From the framework's point of view, the details of the positioning implementation are left open, which means that the most applicable technique may be chosen for each access point implementation. It is important to note that virtual access points exist solely for service management purposes and that their virtual nature is invisible to users. By introducing them we are able to raise the level of both abstraction and interoperability. Exposing virtual access point management interfaces to developers would, however, be interesting since it would enable creation of services that are able to dynamically change their location. A delightful example of such a service would be a local service for a parade that runs inside a large service area.

The basic concept of access points makes it possible to separate the service data and location data routes. It would be entirely possible to, e.g., map a user to a specific location with short-range hotspots and transfer the location-dependent service data using wide area communication. A necessary prerequisite is, however, that a predefined mapping exists between the identities of user's terminal in the different network schemes.

Flower's support for virtual access points is implemented in a straightforward way. A physical access point is responsible for managing its existing virtual access points. As long as each virtual access point has its own socket connection to the Manager, it is considered to be a "real" access point by the rest of the system. This approach has an additional level of flexibility, since different physical access points can implement the virtuality independently. For example, while one access point could use WLAN positioning to monitor the users, another could utilise a network of SoapBox proximity sensors (Tuulari, Ylisaukko-oja 2002).

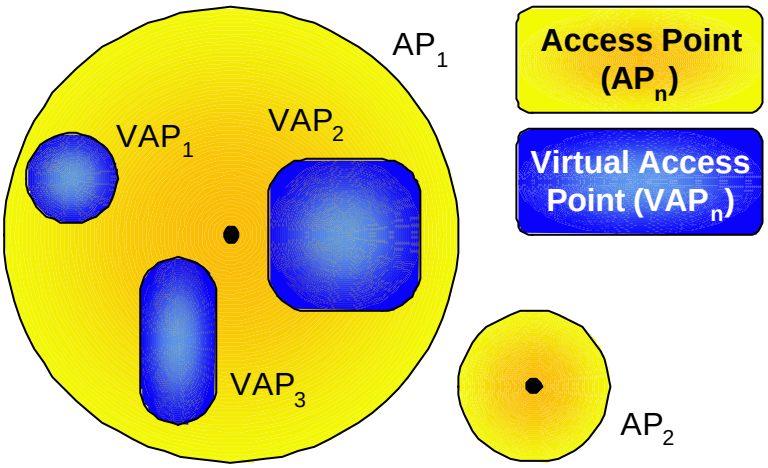


*Figure 3. Sample virtual access point configuration.*

## 7. DISCUSSION

In Flower systems, access points are autonomous. In other words, a physical access point has no knowledge of its peers and every access point is visible in the same way through the Flower API, regardless of its implementation (physical or virtual). While this is a sound design decision in terms of flexibility and robustness, it has its limitations: Flower does not handle overlapping service areas of different physical access points very well. Terminals are assumed to be accessible through only one access point at a time.

What if two physical access points are simply too close to each other? The solution is access point co-operation: the access points are managed by a single entity that represents them to the Flower Server. Thus, the former physical access points are implementation mechanisms for virtual access points and overlapping is resolved by the managing entity — the new physical access point.

In the future, support for overlapping service areas might be added to Flower. First, there must be enough practical experience of the problem, since adding the support would affect system semantics. For example, push mechanisms would have to be able to select from alternative routes, provided by access points, to the target terminal. We are also reluctant to add any unnecessary complexity to the Flower API.

## 8. CONCLUSION

In Flower framework, access points form the common interface that hides user terminal details from the rest of the system. Vice versa, from the user's point of view, access points represent the whole framework. This important distinction makes it possible to separate generic and communication technique specific details from each other. The mechanism has been further fine-tuned by introducing virtual access points that make fundamentally different techniques more alike. With the described architecture we are able to make the service application programming interface more abstract and uniform.

The experiences gained from using the Flower framework show that the web programming model can be fluently employed also in local service realisation. The servlet technology that forms the base of the application programming interface has turned out to be flexible enough for derivation to local service purposes. As a result, local service development with Flower does not significantly differ from traditional web programming. Since local environment specific operations exist in the same semantic level as standard servlet methods, their use should be encouraged.

Local wireless services concept is a novel and promising concept that makes it possible to provide totally new types of location dependent and regionally restricted mobile services. The presented Flower framework takes account of special characteristics and facilities of local service environment and gives service developers means to utilise them through Flower API. Flexibility of Flower permits various wireless communication techniques to be utilised side by side in the same framework.

Flower was on trial in a pilot project applying local wireless services to parking control and payment in an operational carpark environment (Lattunen, Hakkarainen 2003). The aim of the project was to introduce a new concept for wireless parking transactions, which would be free of connection charges. Hands-on experience from the pilot application allow us to conclude that the effort to build local wireless services has been substantially decreased by the Flower framework.

## REFERENCES


Apache Software Foundation. 2002. *The Apache Jakarta Project.* Retrieved: March 20, 2003, from http://jakarta.apache.org/tomcat/.

Cheverst, K. et al, 2000. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. *Proceedings of CHI'00*. Netherlands, pp. 17-24.

Gamma, E. et al, 1995. *Design Patterns ¾ Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, USA.

Lattunen, A. & Hakkarainen, T., 2003. Parking Control by Local Wireless Services. *EPA Congress 2003.* London, UK. In press.

Sun Microsystems, 1999. *Java Servlet Ô Specification.* Retrieved: March 20, 2003, from http://java.sun.com/.

Tuulari, E. & Ylisaukko-oja, A., 2002. SoapBox — A Platform for Ubiquitous Computing Research and Applications. *Proceedings of Pervasive 2002*. Zürich, Switzerland, pp. 125-138.

VTT Information Technology, 2000. *WAP Proxy*. Retrieved: March 20, 2003, from http://www.vtt.fi/tte/projects/WAP/wap_proxy/index.html.

WAP Forum, 2001a. *Service Indication, version 31-Jul-2001*. Retrieved: March 20, 2003, from http://www.wapforum.org/.

WAP Forum, 2001b. *Service Loading, version 31-Jul-2001*. Retrieved: March 20, 2003, from http://www.wapforum.org/.

WAP Forum, 2001c. *Push Access Protocol, version 29-Apr-2001*. Retrieved: March 20, 2003, from http://www.wapforum.org/.

Youll, J. et al, 2000. Impulse: Location-based Agent Assistance. *Proceedings of the 4th International Conference on Autonomous Agents*. Barcelona, Spain.