# POSTER
# Adaptive OSGi-Based Context Modeling for Android

Darren Carlson and Andreas Schrader

Institute of Telematics (Ambient Computing Group), University of Lübeck,
Ratzeburger Allee 160, 23538 Lübeck, Germany
{carlson,schrader}@itm.uni-luebeck.de

**Abstract.** Although contextual information is recognized as a foundation of self-adapting software, context modeling middleware is often prohibitively complex and limited to small-scale deployments. To mitigate this complexity, we are developing Dynamix, a wide-area context modeling approach for Android. Dynamix simplifies context-aware application development through an extensible, OSGi-based framework that runs as a background service on a user's Android-based device, modeling context information from the environment using the device itself as a sensing, processing and communications platform. Context modeling is performed by a tailored set of plug-ins, which are dynamically provisioned to the device over-the-air during runtime. User privacy is maintained by a user-configurable context firewall. This poster introduces Dynamix's OSGI-based context modeling approach.

**Keywords:** Ubiquitous computing, Context-awareness, Middleware, OSGi, Android, Component integration, Reusability, Over-the-air provisioning.

## 1 Introduction

The rapid adoption of smart-phones, tablet computers and net-books has paralleled a dramatic increase in the usage of mobile data applications (or "apps"). As apps become unmoored from the preconceptions of conventional desktop computing, mobile users expect them to adapt intelligently and fluidly across a broad range of everyday situations, execution environments and device platforms. However, although contextual information is widely recognized as an essential foundation of self-adapting software, context modeling and management middleware is often prohibitively complex and limited to small-scale deployments [1, 2]. As a consequence, mobile app developers transitioning from enterprise and desktop scenarios face significant (and often prohibitive) complexity when creating context-aware apps.

Over the last decade, a variety of approaches have been devised to help insulate developers from the demands of context modeling and management [3]. Recently, interest in wide-area context-awareness has generated a variety of techniques that model *preexisting* sources of environmental information using the capabilities of the user's mobile device [4]. In such scenarios, adaptive middleware is often used to provide various functionality [5], including sensor abstraction; context modeling and representation; service discovery and binding; and others. Unfortunately, existing

A. Puiatti et al. (Eds.): MobiQuitous 2011, LNICST 104, pp. 209–213, 2012.

adaptive middleware technologies (e.g., OSGi [6]) have been difficult to deploy on most mobile platforms.

The explosive rise of Google's Android platform is providing a foundation for domain-specific mobile middleware based on OSGi. Briefly, OSGi defines a comprehensive dynamic module system for Java, whereby software functionality is encapsulated within distinct logical units – called *Bundles* – that contain both executable software and metadata. In an OSGi-based application, Bundles are dynamically woven together at runtime using an OSGi Framework implementation (OSGi container). In terms of OSGi deployment, Android overcomes previous mobile platform limitations though a software stack that supports multitasking (including long-lived background processes), a comprehensive inter-process communication model and broad Java compatibility. Several recent projects [7-9] have demonstrated that Android can be effectively used as the foundation of context-aware applications. These projects benefit from Android's broad device support and extensive user base; however, Android developers still lack comprehensive, wide-area support for sensing, modeling, representing and provisioning context information.

Towards this end, we're developing a wide-area context modeling approach for Android, called Dynamix. The foundation of Dynamix is an extensible middleware framework, which runs as a background service on a user's device, modeling context information from the environment using the device itself as a sensing, processing and communications platform. Context modeling is performed by a tailored set of plug-ins, which are packaged as OSGi Bundles and provisioned to the device over-the-air (OTA) during runtime. Context plug-ins are used to insulate app developers from the complexities of context modeling, which often involve specialized domain knowledge. Figure 1 provides a high level overview of the Dynamix infrastructure.

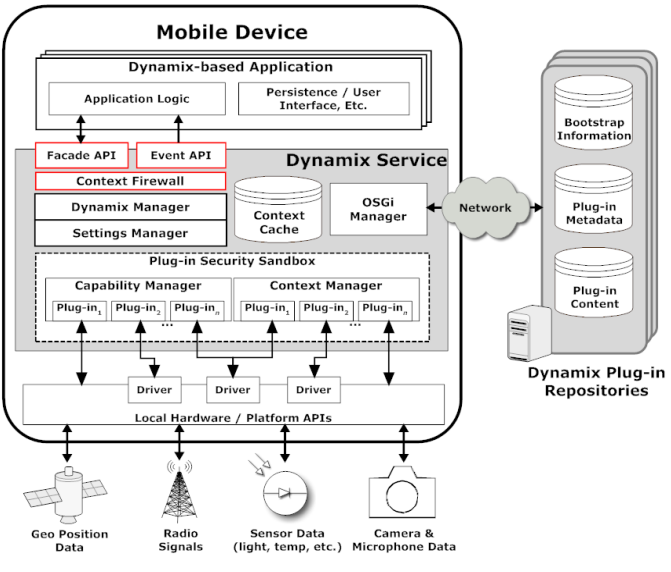


**Fig. 1.** Overview of the Dynamix Infrastructure

As shown in Figure 1, a Dynamix Framework instance (Dynamix Service) is situated between the host device's low-level capabilities (local hardware and platform APIs) and a set of Dynamix-based applications, which execute in their own runtime processes. Context modeling is performed on behalf of requesting apps using a tailored set of context plug-ins, which execute within an embedded OSGi container hosted by the Dynamix Service. Context plug-ins interact directly with a device's underlying hardware, platform APIs and communications facilities within a Plug-in Security Sandbox, which provides secured access to system resources, user data and Android services.

Within the Dynamix Service, an embedded OSGi container (Apache Felix) is managed by a custom OSGi Manger, which provides Dynamix-specific features, such as plug-in installation and updates; event services (e.g., plug-in installed or updated notifications); and integrated class-path management (to support plug-in access to bundled resources, such as images). The OSGi Manager works in tandem with the Capability Manager to actively prevent the loading of incompatible plug-ins (e.g., plug-ins that require unavailable hardware). A multithreaded Bundle installer was also developed to support parallel plug-in installations, progress notifications and Bundle verification. Plug-in Bundles can be provisioned from a customizable set of plug-in repositories. Plug-ins can be manually installed into a Dynamix Service, or deployed automatically in the background in response to app requests. Figure 2 shows the Dynamix Service's Home tab (left) and the Plug-ins tab (right) during a parallel plug-in installation process.

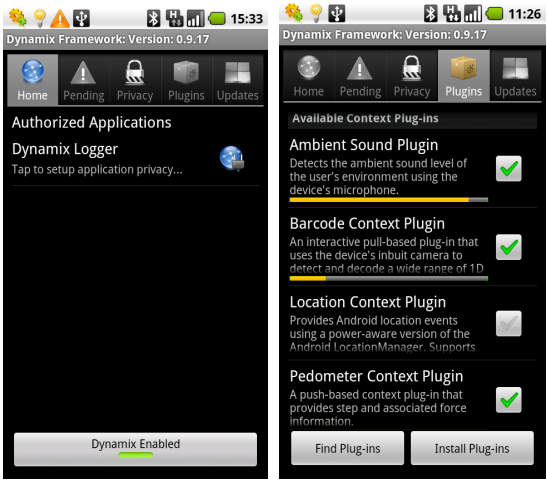


**Fig. 2.** The Dynamix Service's Home and Plug-ins Tabs

Dynamix apps are defined as standard Android applications that incorporate extra context modeling functionality provided by a local Dynamix Service. Dynamix apps communicate with the Dynamix Service using an Android Service Connection, which is a core component of the Android IPC model. Communication is facilitated through two Dynamix interfaces, which provide a simple way for Android apps to interact

with the Dynamix Service. The Facade API enables apps to request context modeling support. The Event API enables apps to receive system notifications and context events from the Dynamix Service.

Dynamix mediates the flow of context events (from plug-ins to applications) using a configurable Context Firewall, which enables users to precisely manage the privacy risk level of the contextual information available to each app. Users define a Context Firewall policy for each app using preconfigured Privacy Policies and custom settings. A Privacy Policy assigns an overall trust level to an application, ranging from blocked (no trust) to highest trust. A trust level determines the privacy risk level of the context information that it is allowed to flow from plug-ins to an application. Users can tap on each context plug-in in the Dynamix Service's user interface to view a detailed description of the supported privacy risk levels.

After an app has been granted security authorization by the Context Firewall, it requests context support by creating *context subscriptions* using the Facade API. A context subscription is a registration made by a specific Dynamix listener indicating that it wishes to receive context events of a particular type. During a context subscription registration, the requested context type is checked against the plug-ins installed in a Dynamix Service. If the Dynamix Service is able to support the context subscription type (i.e., it has a compatible context plug-in installed), it sets up the subscription, initiates context modeling, and informs the app using the Event API. If context support is not available locally, the Dynamix Service will attempt to dynamically discover and install compatible context plug-ins using its configured set of plug-in repositories. Once the context subscription is initiated, the Dynamix Service sends discovered context information to its registered listeners using an extensible Context Event object, which contains both event data (i.e., context information) and event metadata (e.g., context type, event source, string representation, expiration information).

As a proof of concept, we created and evaluated a prototype implementation of the Dynamix infrastructure shown in Figure 1. The prototype includes an embedded OSGi container (Apache Felix 3.2.2) and was tested on several Motorola Milestone, Samsung Galaxy Tab, and HTC Hero devices. We evaluated the framework's abstractions by creating eight example context plug-ins, which ranged in complexity from simple extensions of existing Android sensor services, to relatively complex integrations of externally developed proprietary and open-source projects. Prototype context plug-in functionality includes physiological monitoring; radio signal detection; geo-location detection; social network profile extraction; orientation primitives; ambient sound level detection; step detection and step force calculation; and bar-code scanning. We evaluated the Dynamix Application APIs by developing and evaluating several sample applications. Overall, five sample Dynamix apps were developed by several participants. The apps included a range of functionality, including context event logging; product pricing and reviews; virtual information space management; health monitoring; social network interaction; and third party framework integration. Integration of the Dynamix Framework was not observed to be overly complex, even for developers with little or no Android experience.

Moreover, the diversity of these initial apps provides encouraging indications that Dynamix can be used in a variety of wide-area scenarios.

Dynamix (formerly Aladdin) is developed within an Ambient Assisted Living project supported by the Federal Ministry of Education and Research (BMBF), Germany. ID: 16KT0942. For more information, please see http://www.smartassist.de [10] and http://dynamixframework.org.

## References

1. Want, R., Pering, T.: System challenges for ubiquitous & pervasive computing. In: Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA (2005)
2. Davies, N., Gellersen, H.-W.: Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. Pervasive Computing 1(1), 26–35 (2002)
3. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2(4), 263–277 (2007)
4. Carlson, D., Schrader, A., Busch, D.: Modular framework support for context-aware mobile cinema. Personal and Ubiquitous Computing 12(4), 299–306 (2008)
5. Endres, C., Butz, A., MacWilliams, A.: A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. Mobile Information Systems 1(1), 41–80 (2005)
6. OSGi Service Platform Release 4, `http://www.osgi.org/Specifications/HomePage`
7. van Wissen, B., Palmer, N., Kemp, R., Kielmann, T., Bal, H.: ContextDroid: an expression-based context framework for Android. In: Proceedings of PhoneSense (2010)
8. IST-MUSIC Consortium. IST-MUSIC: Context-aware self-adaptive platform for mobile applications, `http://ist-music.berlios.de`
9. Appeltauer, M., Hirschfeld, R., Rho, T.: Dedicated Programming Support for Context-Aware Ubiquitous Applications. In: Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Valencia, Spain (2008)
10. Schrader, A., Carlson, D., Rothenpieler, P.: SmartAssist - Wireless Sensor Networks for Unobtrusive Health Monitoring. In: Proceedings of the 5th Behaviour Monitoring and Interpretation Workshop at the 33rd German Conference on Artificial Intelligence (KI 2010), Karlsruhe, Germany (2010)