Dynamic Interaction Plugins Deployment in Ambient Spaces

Bashar Altakrouri[1,2] and Andreas Schrader[1]

[1] Ambient Computing Group, Institute of Telematics
University of Luebeck, Luebeck, Germany
{altakrouri,schrader}@itm.uni-luebeck.de
https://www.itm.uni-luebeck.de/
[2] Graduate School for Computing in Medicine and Life Sciences,
University of Luebeck, Luebeck, Germany

Abstract. A large-scale dynamic runtime deployment of existing and future interaction techniques remains an enduring challenge for engineering real-world pervasive computing ecosystems (ambient spaces). The need for innovative engineering solutions to tackle this issue increases, due to the ever expanding landscape of novel natural interaction techniques proposed every year to enrich interactive eco-systems with multitouch gestures, motion gestures, full body in motion, etc. In this paper, we discuss the implementation of *Interaction Plugins* as a possible solution to address this challenge. The discussed approach enables interaction techniques to be constructed as standalone dynamically deployable objects in ambient spaces during runtime.

Keywords: Ambient Assisted Living, Natural User Interfaces, Kinetic Interactions, Dynamic Interaction Deployment, Sharing Interactions.

1 Introduction

The human computer interaction (HCI) research continues to enrich users' interactions with real-world pervasive ecosystems (ambient spaces), with an increasing interest in designing for the whole body in motion as part of the Natural User Interface (NUI) paradigm [9][10]. Various definitions of NUI can be found in the literature. Nonetheless, those definitions mostly refer to the user's natural abilities, practices, and activities to control interactive systems [11] and can be simplified to voice-based and kinetic-based interactions [15]. The later defines those interactions, which are mostly caused and characterized by motion and movement activities, ranging from pointing, clicking, grasping, walking, etc. [2]. Herein, we focus on motion-based interactions as a dominant subset of Kinetic-based interactions. This type of interaction has been adopted widely in various commercial domains and became accessible to the end user, ranging from gaming (e.g., motion-controlled active play by Microsoft Kinect[1] or the Wii system[2]),

[1] http://www.microsoft.com/en-us/kinectforwindows/, latest access on 26.06.2014.
[2] http://www.nintendo.com/wii, latest access on 26.06.2014.

S. Sauer et al. (Eds.): HCSE 2014, LNCS 8742, pp. 73–89, 2014.

data browsing, navigation scenarios (e.g., tilting for scrolling photos as in iOS[3] and Android[4] devices) and many more.

Despite its success, the NUI paradigm poses a number of significant challenges to the design, engineering and deployment of NUI technologies, especially when considering a far more diverse and heterogenous user population, e.g., due to aging and demographic changes; unknown interaction context, due to increasing user mobility to unknown environmental settings at design time; and spontaneous construction of interactive environments in-situ at runtime. Thus the focus on isolated design of natural interface techniques will not be adequate in current and future ambient spaces, which rapidly experience increasing emergence of interconnected mobile devices, smart objects, and seamlessly integrated context-aware services.

In our previous work, we proposed a shift towards completely dynamic on-the-fly ensembles of interaction techniques at runtime. The Interaction Ensembles approach is defined as "Multiple interaction modalities (i.e., interaction plugins) from different devices are tailored at runtime to adapt the available interaction resources and possibilities to the user's physical abilities, needs, and context" [2]. This shift imposes new dissemination, deployment, and adaptation requirements for engineering interaction techniques and interactive systems for NUIs.

In this paper, we tackle some of the emerging deployment aspects of interaction in ambient spaces. We believe that the deployment of interaction in NUIs is a key player to enable a realistic adoption of NUIs in the design and engineering of interactive systems. The paper presents a detailed overview and a reference implementation for building and deploying interaction components for ambient spaces, called Interaction Plugins. We have previously defined an Interaction Plugin (IP) as *"an executable component in ambient interactive systems that encapsulates a single natural interaction technique with a set of interaction tasks as input and delivers higher level interaction primitives to applications based on specific interaction semantics"* [1][2]. While our previous work in [1][2] proposed the concept of IP on a high conceptual level, the work presented in this paper primarily describes a detailed description of this approach and the resulting implementation.

To the best of our knowledge, there is no research specifically targeted at community-based creation and sharing of encapsulated natural interaction techniques for ambient spaces. The Interaction Plugin approach is based on three main design characteristics, namely, matching users and natural physical context, precise and extensible natural interaction descriptions (human and machine readable), and flexible deployment of interactions at runtime [1].

Interaction Plugin approach fosters soft-wired (de-coupled) applications and devices in order to overcome the limitations of the static binding and to address one of the most challenging requirements in pervasive environments, namely the "come as you are" requirements. This approach matches calls from the HCI community to overcome various challenging issues for user interfaces in ambient

[3] `http://www.apple.com/ios/`, latest access on 26.06.2014.
[4] `http://www.android.com/`, latest access on 26.06.2014.

spaces. Pruvost et al. [12] noted that interaction environments are becoming increasingly heterogeneous and dynamic, hence they are no longer static and closed; the interaction context is becoming increasingly more complex; and, increasing adaptability is required for sustainable utility and usability.

We believe that investigating this approach is essential to understand some of the challenges for engineering interactive systems in ambient spaces and setting proper interaction dissemination guidelines, where interactions are becoming increasingly dynamic, adaptive and multi-modal. Our approach aims at avoiding mismatch problems between user's needs and device's offers by employing the best matching natural interaction techniques to the given context, hence the user independence (acceptability by permitting customizability) and usability qualities required by Wachs et al. [15] are inherently enhanced.

2 Background and Related Research

Reviewing HCI literature reveals an extensive effort in the area of traditional user interfaces adaptation in terms of context modeling, user modeling, automatic generation of interfaces, etc. Most of the well-established concepts mainly target the conventional Graphical User Interface (GUI) paradigm such as plasticity [3] and the WWHT framework [13]. Despite their strong relevance, most available adaptation approaches fail to satisfy four enduring challenges drawn from the natural characteristics of ambient environments, presented by Pruvost et al. [12]:

- Heterogeneity and Distributivity: The interaction eco-system contains a variety of interaction devices with various capabilities.
- Dynamic Media Mobility: Interaction capabilities are highly dynamic as interaction devices may join or leave the ambient space at anytime.
- User Mobility: The user mobility in ambient spaces challenges the interactive system attention to the user's interaction needs.

In addition, most adaptation approaches focus on interface issues such as information presentation but not the interaction per se. Pruvost et al. [12] clearly indicated that locking interaction devices in their own closed world is certainly an issue for interaction systems adaptability in ambient spaces. This closeness results into reducing the richness and unity of those interaction devices in various context scenarios. They also argued for highly adaptable user interfaces that preserve utility and usability across contexts. In their described adaptation vision, they have presented the concept of Off-the-shelf Interaction Objects, which are pre-implemented bundles of code, intended to be reused and composed at runtime. The objects aim to provide the necessary adaptation required for the interaction technique. While their vision is focused on the structural adaptation of user interfaces and the adaptation of a running dialogue, our work is more concerned with the sharing and deployment aspects of NUI, especially kinetic interactions. The Gestureworks Core[5], which is limited to multitouch interactions,

[5] `http://gestureworks.com`, accessed on 26.06.2014.

is one of the earliest multitouch gesture authoring solution for touch-enabled devices on a variety of platforms such as Flash; C++; and Java. Based on the GML (GML), the solution comes with a rich library of pre-built gestures and allows for new custom gestures and gesture sequences to be built by designers. For motion-based interactions, the OpenNI[6] is an open source SDK used for the development of 3D sensing applications and middleware libraries. The main targets of this framework include enhancing the natural interaction techniques development community; making it possible for developers to share ideas and problems; to share code with each other; and to address the complete development lifecycle by a standard 3D sensing framework.

Standalone deployable interaction components for NUIs are becoming essential needs for ambient spaces. To our best knowledge, the work presented in this paper is one of few HCI research efforts to tackle this problem. Interestingly, dynamic component integration has been a rich and yet challenging investigation aspect for the ubiquitous and pervasive computing research. The dynamic component integration approach enables software components to be discovered, downloaded, and integrated on-demand, as a means of adapting an application's behavior and enhancing its features [14]. This area of investigation has been recently successfully applied to mobile environments as in the Mobile USers In Ubiquitous Computing Environments (MUSIC) system that supports dynamic component integration on Android using OSGi (OSGi) containers [8]; the Context-Aware Machine learning Framework for Android (CAMF) that promotes plugin-based adaptation on Android [16]; and the Funf Open Sensing Framework[7] that promotes statically-linked context modeling plug-ins integration. More recently, the Dynamix framework [7] was introduced as an open plug-and-play context framework for Android.

3 STAGE Architecture and Implementation

Our current implementation utilizes the wide spread and adoption of mobile devices for rich personalization, customization, and context acquisition in ambient spaces. Hence, this work fosters the use of mobile devices as customized and personalized interaction hubs. In our implementation, called STAGE, we leverage Dynamix framework [7] as a mechanism for deploying natural interactions. Dynamix is used due to its unique capabilities and flexibility, especially related to dynamic discovery and deployment of suitable context plugins during runtime. Hence, it was feasible to adopt and extend context plugins for interactions. Although grounding the implementation on mobile devices may appear limiting from the first glance, the use of the mobile devices as a personalized interaction hub to facilitate interactions in ambient spaces provides various benefits in terms of personalization, decentralization, user control, and lightweight infrastructure. Figure 1 illustrates our underlying technical approach for implementing natural interaction techniques as deployable and shareable IPs, based

[6] `http://www.openni.org`, latest accessed on 26.06.2014.
[7] `http://funf.org`, latest access on 26.06.2014.

on the Dynamix framework on the Android platform. As intended, the Dynamix framework runs as a background service (Dynamix Service) and is situated between Dynamix enabled applications and the device's interaction resources (i.e., interaction devices).

The STAGE technical implementation appears mainly in two areas: first, the interaction application side; second, the context plugin side. Dynamix-enabled applications are standard Android applications with extra context modeling functionality provided by a local Dynamix Service. In STAGE implementation, two new components are introduced, namely the Interaction Manager and Ability Manager, to adjust the Dynamix framework to our needs. Additionally, Dynamix context plugins were adopted and extended in the IP implementation.

3.1 Context Modeling and Deployment with the Dynamix Framework

The Ambient Dynamix framework was first proposed as a rich framework for on-demand discovery and runtime integration of plugins for context acquisition (sensors) and modification (actuators) in wide-area mobile contexts. Interestingly, Dynamix features strong and flexible discovery and deployment of context plugins at runtime. The framework was successfully used to model and deploy context in different none-HCI related scenarios as in [5][6][4]. The two features are essential building blocks for the implementation of the Interaction Plugin concept. In fact, our work is the first to utilize and build on this framework for deploying interactions in ambient spaces. In this section, the most relevant internal components of the framework based on [7] will be described. The full specifications and more extensive discussion can be found in the Dynamix online portal[8].

Dynamix runs as a background service on Android-based devices, which allows multiple Dynamix-enabled applications to subscribe to Dynamix context events. Principally, a Dynamix-enabled application is a normal Android application that implements the necessary Dynamix API. Various context resources deliver raw sensor data to the framework, which are then modeled by Dynamix according to the available context plugins.

A context plugin is a standalone deployable OSGi container that allows for seamless runtime deployment (i.e., installation, uninstallation) according to the application needs. The flexibility of the Dynamix plugin's technical structure provided an adequate ground for our development of IP, as discussed lengthly in this section.

The OSGi Manager resides at the core of Dynamix implementation, which is build based on the Apache Felix OSGi. The manager is mainly dedicated for all plugins deployment actions, including the installation and un-installation of plugins from their OSGi containers. The OSGi framework allows for seamless integration of software units (called bundles) at runtime. Internally, the OSGi framework is composed of multiple layers responsible for bundle execution, bun-

[8] `http://ambientdynamix.org`, visited on 26.06.2014.

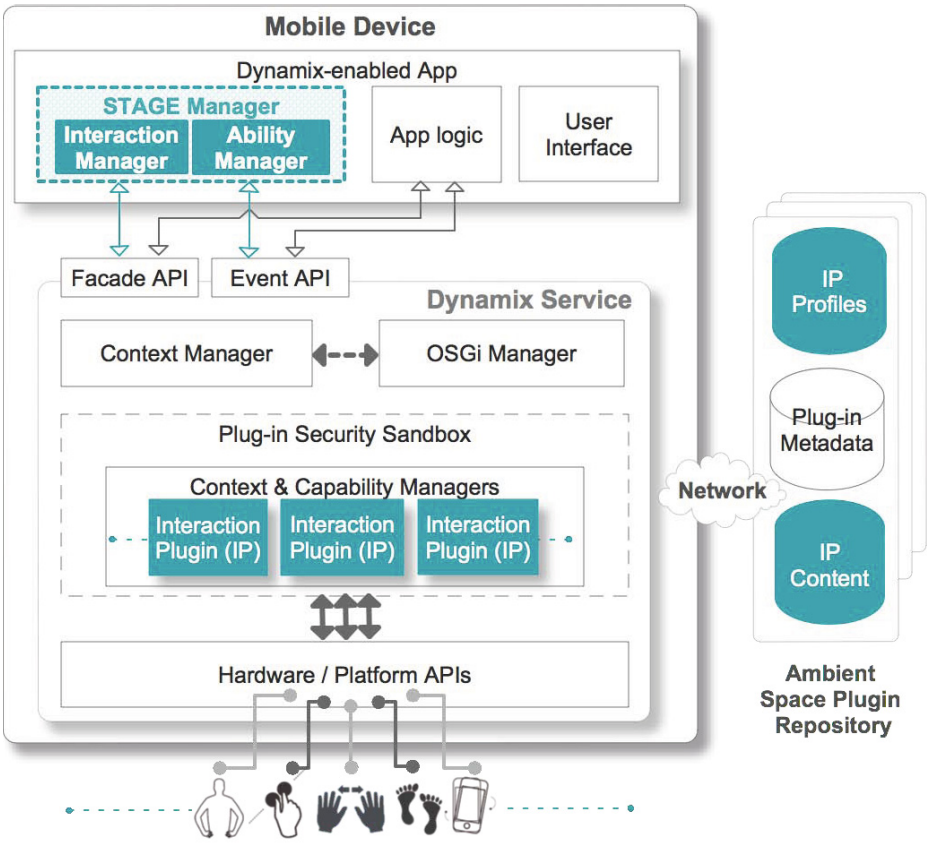

Fig. 1. A high-level overview - STAGE realization based on Dynamix (based on [1])

dle management, bundle life cycle, service and binding management, and security control. There are various communication aspects in Dynamix. We will focus, herein, only on the communication between plugins and applications. The communication between the Dynamix framework and Dynamix-enabled application is featured using the Facade and Event API. The Facade API controls all requests for context modeling support. Moreover, Dynamix currently uses POJO (Plain Old Java Objects) objects to encode the events shared between plugins and applications. The use of POJO objects allows both sides to work with Java objects, hence increasing the operability and reducing the programming and modeling load.

3.2 Implementing STAGE-Enabled Applications

Dynamix applications are designed to subscribe to dynamix and receive modeled context events. Those two features ease the implementation of context-based applications dramatically. Nonetheless, Dynamix-enabled applications are not fully adequate to the Interaction Plugin conceptual design. Hence, STAGE introduces

two important extensions to the architecture, in order to facilitate the use of this framework for IPs, on the application and plugin levels. The changes and extensions are distinctly tinted (i.e., fully or partially shaded) in Figure 1. The STAGE components on the application side are further illustrated in Figure 2. STAGE-enabled applications contain the STAGE Manager (which contains the Interaction and Profile Managers) and the Interaction Profile Manifest (which encapsulates all necessary information about the required and relevant interaction capabilities by the application).

STAGE Interaction Manager. This class controls the activation of the available IP, based on the ability (the physical abilities required for the interaction e.g., major life activities), movement (the exact movements required for the interaction e.g., body part involved, type, and degree of movements), and interaction (the interaction semantic) profiles. The communication between the Interaction Manager and Dynamix service is facilitated using the Dynamix Facade API. Following the dynamix context plugin model, Interaction Plugins are tailored OSGi-based Bundles, which are loaded into the Dynamix embedded OSGi container at runtime. Once loaded and activated, an Interaction Plugin sends interaction encoded events to subscribed applications (as interaction primitive events) using POJO. Principally, Interaction Plugins are hosted in an Internet-based plugin repository, which in addition to the IP Bundle files, hosts the IP profiles and (optionally) additional related plugins.

In addition to the usual Dynamix context sensing tasks, IPs can be queried by the Interaction Manager to access the information encoded in the IP profiles. Currently, the Dynamix Service provides all plugin discovery services, plugin filtering based on the interaction requirements (interaction primitives required), and plugin installation support. Filtering and activating the available interaction plugins are currently handled by the Interaction Manager. The STAGE Interaction Manager is composed from the following components, as shown in Figure 2:

- *Subscription Manager* is responsible for communicating directly with the Dynamix facade API, in order to control and mange IP subscriptions;
- *Event Handler* consumes all Dynamix context events and filters all relevant Interaction Events;
- *Interaction Profile Builder* is responsible to serialize the content of the Interaction Profile Manifest into a runtime object;
- *Plugin Manager* keeps track of all accessible IPs and handles the status, subscription, plugin information, and discovery requests of all plugins;
- *Ensemble Engine* is responsible for monitoring interaction resources (i.e., IP) and build possible Interaction Ensembles adequate for the given context based on the application's required interaction capabilities, the available IP, and the user's physical ability profile; and
- *STAGE Interface Controller*: STAGE-enabled applications require to control the application's GUI elements according to the fired interaction primitives from the IP (once an interaction primitive is fired, the corresponding action on the interface is executed, e.g., a selection interaction primitive may be interpreted as a button press).

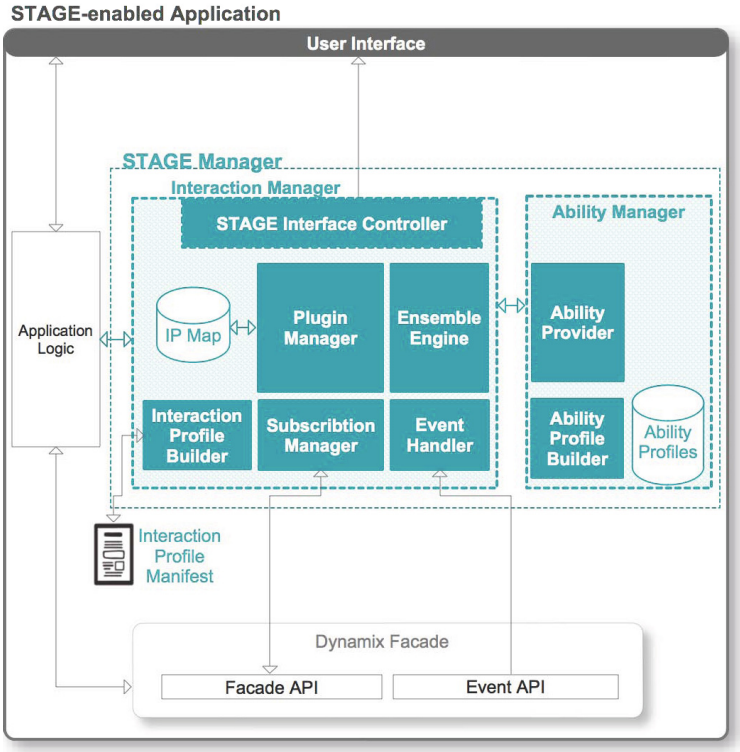

Fig. 2. STAGE-enabled application architecture

Ability Manager. This module is responsible for extracting the user's ability and disability qualities, based on the available Ability profile and Movement profile. Currently, both profiles are modeled and represented in tailored XML formats. This component is essentially split into the following modules:

- *Ability Profile Builder* is essentially an XML parser responsible to extract and serialize the content of the Ability Profile into a runtime object; and
- *Ability Provider* provides all necessary information regarding the required physical abilities for executing the interaction tasks adequately by the user.

Interaction Profile Manifest. This profile manifest captures the main interaction semantics including the interaction primitives such as pointing, selecting, dragging, etc. This is particularly important in ambient spaces because it provides indication on the main use of the interaction technique and its offerings. We are considering interaction profile as a prime key for interaction adaption and matching in ambient spaces. In ambient spaces, interaction profiles offer the needed manifestation for coupling interaction techniques with applications. Principally, it is used to in order to identify the required interaction capabilities for the application. The manifest file is constructed in XML format, and contains essentially the required interaction primitives. Figure 3 resembles a simple profile example (i.e., an application requires a selection interaction primitive).

```
<?xml version="1.0" encoding="UTF-8"?>
<appInteractionProfile>
    <primitives>
        <primitive>
            <type>de.itm.STAGE.nui.primitives.selection</type>
            <dynamixPluginDetails>
                <pluginId></pluginId>
                <dynamixPluginType>AUTONOMOUS</dynamixPluginType>
            </dynamixPluginDetails>
        </primitive>
    </primitives>
</appInteractionProfile>
```

Fig. 3. STAGE-enabled application architecture

STAGE-Enabled Application Interaction Sequence. The interaction sequence, shown in Figure 4, illustrates the communication steps between Dynamix and the STAGE Manager. The sequence illustrates the process of interaction requesting a certain IP, deploying IP in runtime, and delivering the event information to the requesting application. The Interaction Manager triggers and establishes the connection with the Dynamix framework. Upon connection, it receives all Dynamix related events as long as the Dynamix connection remains alive. For instance, the received events involve Dynamix activation events, deactivation events, and plugin installation events. Next, the Interaction Manager requests the ability profile from the Ability Manager. Moreover, it serializes the application's interaction profile, using its profile builder module.

The Interaction Manager requests Dynamix for all accessible plugins (i.e., those plugins available in its repository). Once the list is received, the Interaction Manager identifies all accessible IPs and ignores all other context plugins. Plugins that implement the required "InfoObject" datatype object proposed by STAGE are identified as IPs. Otherwise, the plugins are considered conventional Dynamix context plugins (i.e., not related to interactions). The "InfoObject" contains the essential Uniform Resource Identifiers (URIs) for the IP and its three profiles, which are used later for filtering adequate IP in a given context. Next, it configures the Interfaces Controller event listener, in order to allow the controller to utilize the detected events for controlling the interface controllers accordingly. Moreover, the Interaction Manager sends Dynamix subscription requests to the available IP. While getting accepted subscriptions, the manager requests the plugin's ability and interaction profiles in order to build a map of IP and physical abilities required for each. This information forms the core base for the Ensemble Manager's matching algorithm. Accordingly, the Interaction Manager requests the Interface Controller to activate those GUI elements that are possible to be controlled by the available interaction resources (i.e., IPs). Other GUI elements, not supported by the available interaction resources, are not activated for NUI interactions, but can be still used conventionally (using the conventional touch interface).

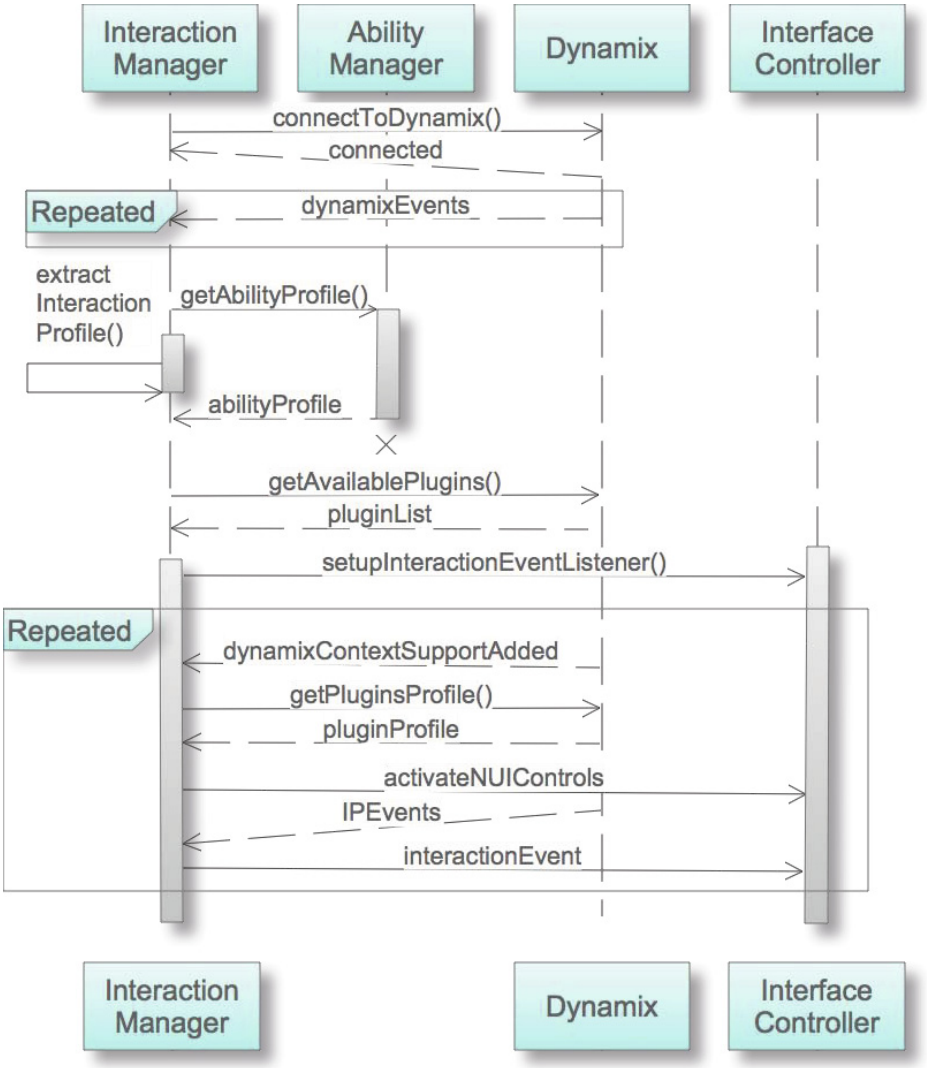

Fig. 4. Sequence Diagram for STAGE-enabled application

At the end of interaction sequence, the STAGE Manager is able to receive modeled interaction events from the various IPs. The triggered interaction primitives are then sent to the Interface Controller to perform the required interaction tasks. It is important to know that multiple plugins may deliver the same interaction primitive. In this case, the Ensemble Engine may decide on which plugin should be used according to the best match with the user's physical abilities (Ensemble Engine's matching algorithms are not covered in this paper due to the size limits).

3.3 Implementing Interaction Plugins

An IP is essentially an OSGi Bundle containing the plugin's logic and context acquisition code. In our case, Dynamix is responsible for handling the plugin's lifecycle, based on its embedded OSGi framework. Figure 5 illustrates the technical implementation of the IP. The distinctly tinted (i.e., fully or partially shaded) components in the diagram indicate the core STAGE components and extensions:

- *Context Acquisition & Modeling* module resembles the logic required to access the context provider and its raw context data. The raw context data resources may be local resources (i.e., running on the same device, for example a built-in orientation sensor) or remote sources (i.e., accessible ambient context providers available in an accessible ambient space, for example an external camera sensor);
- *STAGE Handler* is responsible to model the interaction events based on the received information from the previous module;
- *Plugin Factory* provides the required mechanisms for plugin instantiation (handled completely by Dynamix);
- *PluginRuntime* component contains the plugin's core lifecycle methods;
- *Profile Builder*, as part of the PluginRuntime component, is responsible for serializing and building the interaction, movement, and ability profiles associated with each IP. Those profiles are modeled as XML embedded in the plugin's file structure and accessible though the IPluginInfo object;

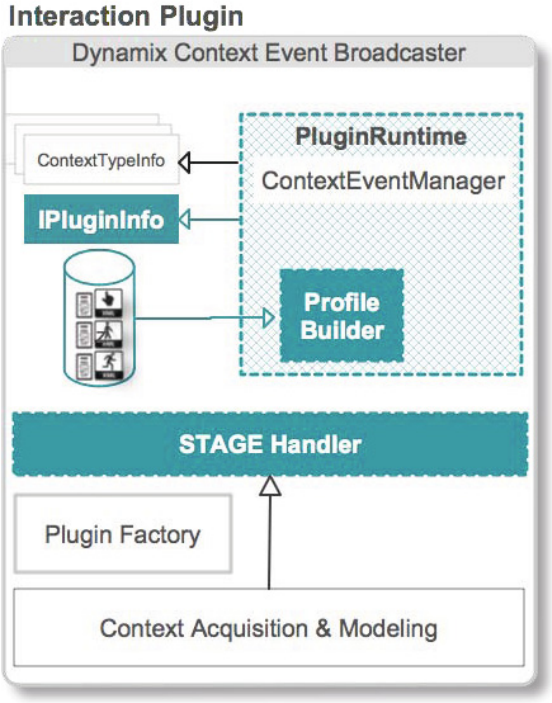

Fig. 5. IP internal architecture

- *ContextTypeInfo Objects* contains the plugin's supported datatypes. In our implementation, those context type objects are used as interaction primitive objects; and
- *IPluginInfo Object*, which is introduced as a new plugin information datatype and dedicated for the necessary movement, ability, and interaction profiles.

4 Examples and Scenarios

The practical benefits of IPs can be seen in an endless variety of showcases and scenarios. For instance, a person with limited arm rotation may take advantage of replacing the arm-based rotation IP exceeding a certain rotation degree with a foot-based rotation IP which utilizes an accelerometer sensor embedded in her smart sport shoes and wirelessly connected to her smartphone. Another example is allowing a person with total absence of three fingers due to an injury to interact with typical mobile operation systems, which often requires swipe gestures with three or more fingers. In such a case, one possible ensemble might comprise two IPs combining two fingers multitouch swipe with one foot-based motion swipe to replace the original three fingers swipe. In principle, the potential for building useful scenarios for interaction plugins and ensembles is only limited by the creativity of interface designers and engineers developing sensing devices in various forms and shapes to be used in future ambient scenarios. For further illustration, this section presents a full description of an interactive demonstration scenario, called the AmbientRoom, to showcase the dynamic aspects of dynamic interaction deployment at runtime.

The AmbientRoom is a STAGE-enabled mobile application for controlling the ambient lighting of smart rooms. The application provides three simple functionalities, namely dimming the light, changing light's color (i.e., rotate between the rainbow colors), and switching the light on/off. The application relies on our implementation of the Art-Net Light Controller plugin, which uses the Art-Net protocol to send DMX512 data for controlling lighting equipment over the Internet Protocol networks. The Art-Net plugin is a conventional Dynamix context plugin that acts as an actuator in this scenario, extensive discussion about this type of plugins is covered in [4]. Therefore, it should not be mixed with the IP concept discussed in this paper.

Because the application is STAGE-enabled, it adapts its interactivity to the user's physical capabilities. In the case of difficulties touching the interface controls with fingers (due to finger injury), the AmbientRoom application searches for alternative and adequate IPs allowing the user to perform the interaction tasks required by the application. In this scenario, the applications finds two motion-based interaction plugins for replacing the position (i.e., navigation) interaction task with a shaking IP (shaking the device for positioning the focus on the next GUI component) and the selection interaction task with an IP utilizing head movement respectively. Those plugins are found and dynamically deployed at runtime.

The rest of this section will demonstrate the flow of the various processes involved in this scenario. Once the application is started, the application will show

the user with a disabled GUI presenting the application main functional capabilities, as in Figure 6 (1). These capabilities are initially looked or disabled until the required plugins are requested by the application and dynamically deployed and activated by dynamix. In this case, the application starts by requesting the Art-Net Light plugin. Once the plugin is found, downloaded, and activated successfully by Dynamix, the application GUI controls will be activated, as shown in Figure 6 (2).

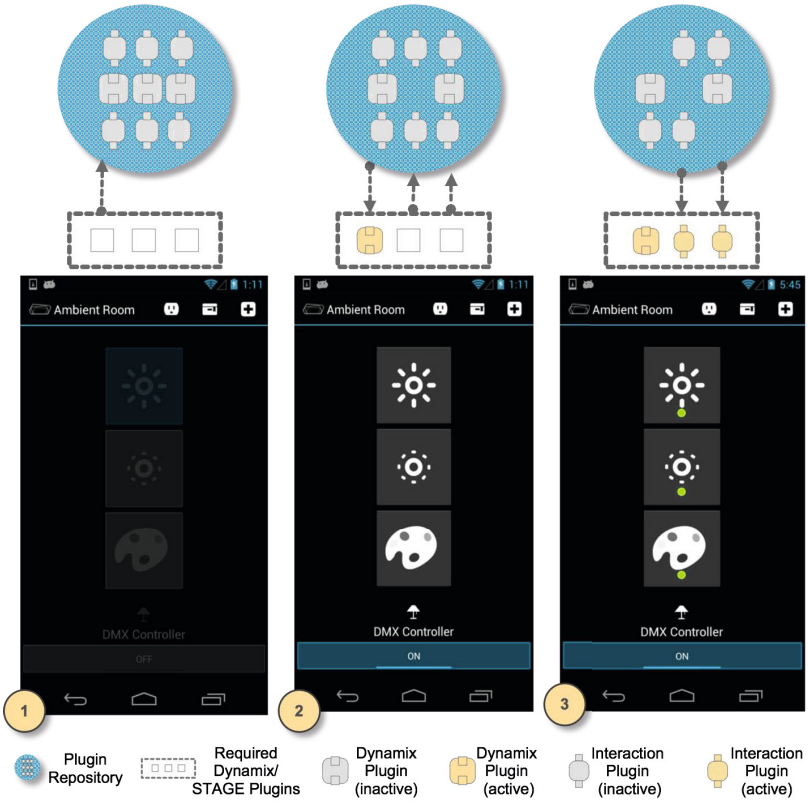

Fig. 6. AmbientRoom application (screenshots): (1) deactivated GUI controls; (2) activated GUI controls after Art-Net plugin runtime deployment; (3) GUI controls are marked after IPs runtime deployment

Next, the AmbientRoom application requests two motion-based IPs to enable users to interact with the application using two motion gestures. The first IP relies on the mobile device built-in accelerometer for detecting device shaking gesture to position the focus on the next button. The second IP relies on the Asus Xtion PRO LIVE[9] motion sensor for detecting head movements to select

[9] `http://www.asus.com/Multimedia/Xtion_PRO_LIVE/`, latest access on 26.06.2014.

(i.e., click) the selected button. Upon a successful deployment of the two plugins, the corresponding GUI control are marked with a green symbol to indicate the possibility to be controlled using motion gestures in addition to the conventional multitouch interface as shown in Figure 6 (3).

Figure 7 demonstrates the NUI style interactions with motion gestures enabled by the deployed IP. Figure 7 (1) shows the NUI-enabled buttons. Using the shaking gesture, the user positions the focus on the control of choice (each device shake will position the focus on the next button) as illustrated in Figure 7 (2). Using the head movement, the user is able to select (i.e., push) the button in focus as illustrated in Figure 7 (3).

Despite its limited scope, the AmbientRoom scenario illustrates the possibility to turn a mobile device into an interaction hub to enable the user to control ambient resources (i.e., lighting) using natural interactions techniques (i.e., different motion gestures). Most importantly, it demonstrates the potential and power of the soft wiring approach where interaction capabilities are deployed (as plugins) at runtime using the STAGE system.

5 Provisioning Interaction Plugins

Provisioning IP undergoes a number of phases as illustrated in Figure 8. In the first phase, the interaction techniques passes the usual design and implementation processes. In the second phase, the interaction developer (i.e., designer,

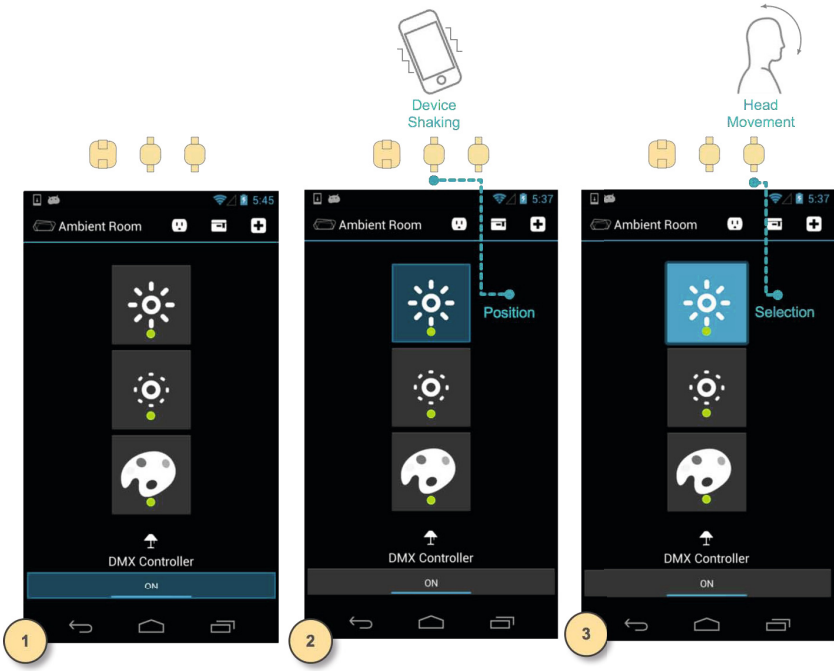

Fig. 7. AmbientRoom application NUI-enabled Interactions

Fig. 8. IP provisioning lifecycle

developer, or team) makes sure that the techniques satisfies the function and utility defined in the interaction design. In the third phase, the interaction developer should define the interaction's movement profile, based on an acceptable level of movement description; define the interaction's ability profile, based on the most important physical qualities that impact the interaction; and assign the interaction semantics, based on the envisioned utility of the technique. In the fourth phase, the interaction developer wraps the interaction's internal logic as a Dynamix plugin. In the fifth phase, the IP is published to an accessible repository. Dynamix supports two types of accessible plugin storage locations (file-system or network). Our implementation supports the later type as shown in Figure 1. The repository hosts the plugins OSGi bundles and the corresponding context plugin description XML documents.

6 Conclusions and Future Work

This paper argues that a large-scale dynamic runtime deployment of existing and future interaction techniques for Natural User Interfaces remains an enduring challenge for engineering interactive systems for ambient spaces. Only few research projects aim to target this problem in the context of HCI. In this paper, we have presented the architecture and implementation of the STAGE system, which enables interaction techniques to be constructed as standalone dynamically deployable objects (*Interaction Plugins*) for Android mobile platforms during runtime. The presented implementation is one of many possible ways to realize the proposed concept. Hence, the proposed implementation should not be necessarily considered as an ultimate implementation solution, instead the implementation aims to demonstrate and evaluate the feasibility of the approach. The current implementation utilizes the wide adoption of mobile devices for rich personalization, customization, and context acquisition in ambient spaces. Hence, this work fosters the use of mobile devices as customized and personalized interaction hubs. As future work continuation to this paper, we are working on an extended performance evaluation of the Interaction Plugins and rich feasibility tests by implementing a number of demonstration scenarios. We also aim at increasing the richness of Interaction Plugins and improving plugin's filtering and matching according to the user's physical context.

Acknowledgement. This work was partially supported by the Graduate School for Computing in Medicine and Life Sciences funded by Germany's Excellence Initiative [DFG GSC 235/1].

References

1. Altakrouri, B., Gröschner, J., Schrader, A.: Documenting natural interactions. In: CHI 2013 Extended Abstracts on Human Factors in Computing Systems, CHI EA 2013, pp. 1173–1178. ACM, New York (2013), http://doi.acm.org/10.1145/2468356.2468566
2. Altakrouri, B., Schrader, A.: Towards dynamic natural interaction ensembles. In: Devina Ramduny-Ellis, A.D., Gill, S. (eds.) Fourth International Workshop on Physicality (Physicality 2012) co-located with British HCI 2012 Conference, Birmingham, UK (September 2012)
3. Calvary, G., Coutaz, J., Thevenin, D.: Embedding plasticity in the development process of interactive systems. In: 6th ERCIM Workshop "User Interfaces for All". Also in HUC (Handheld and Ubiquitous Computing) First workshop on Resource Sensitive Mobile HCI, Conference on Handheld and Ubiquitous Computing, Florence, Italy (October 2000)
4. Carlson, D., Altakrouri, B., Schrader, A.: Ambientweb: Bridging the web's cyber-physical gap. In: 2012 3rd International Conference on the Internet of Things (IOT), Wuxi, China, pp. 1–8 (October 2012)
5. Carlson, D., Altakrouri, B., Schrader, A.: An ad-hoc smart gateway platform for the web of things. In: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, Beijing, August 20-23, pp. 619–625 (2013)
6. Carlson, D., Altakrouri, B., Schrader, A.: Reinventing the share button for physical spaces. In: IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), San Diego, California, USA, March 18-22, pp. 318–320 (2013)
7. Carlson, D., Schrader, A.: Dynamix: An open plug-and-play context framework for android. In: Proceedings of the 3rd International Conference on the Internet of Things (IoT2012), Wuxi, China (October 2012)
8. Consortium, I.M.: Óist-music: Context-aware self-adaptive platform for mobile applications (2010), http://ist-music.berlios.de
9. England, D.: Whole Body Interactions: An Introduction. In: England, D. (ed.) Whole Body Interaction, pp. 1–5. Springer London (2011)
10. Fogtmann, M.H., Fritsch, J., Kortbek, K.J.: Kinesthetic interaction: revealing the bodily potential in interaction design. In: Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat, OZCHI 2008, pp. 89–96. ACM, New York (2008), http://doi.acm.org/10.1145/1517744.1517770
11. Iacolina, S., Lai, A., Soro, A., Scateni, R.: Natural interaction and computer graphics applications. In: Puppo, E., Brogni, A., Floriani, L.D. (eds.) Eurographics Italian Chapter Conference, pp. 141–146. Eurographics Association, Genova (2010), http://publications.crs4.it/pubdocs/2010/ILSS10
12. Pruvost, G., Heinroth, T., Bellik, Y., Minker, W.: User Interaction Adaptation within Ambient Environments. In: Next Generation Intelligent Environments: Ambient Adaptive Systems edn., ch. 5, pp. 153–194. Springer, Boston (2011)

13. Rousseau, C., Bellik, Y., Vernier, F., Bazalgette, D.: A framework for the intelligent multimodal presentation of information. Signal Process. 86(12), 3696–3713 (2006), http://dx.doi.org/10.1016/j.sigpro.2006.02.041
14. Schrader, A., Carlson, D.V., Busch, D.: Modular framework support for context-aware mobile cinema. Personal Ubiquitous Comput. 12(4), 299–306 (2008), http://dx.doi.org/10.1007/s00779-007-0151-6
15. Wachs, J.P., Kölsch, M., Stern, H., Edan, Y.: Vision-based hand-gesture applications. Commun. ACM 54, 60–71 (2011), http://doi.acm.org/10.1145/1897816.1897838
16. Wang, A.I., Ahmad, Q.K.: Camf - context-aware machine learning framework for android. In: Rey, M.D. (ed.) Iasted International Conference on Software Engineering and Applications, SEA, CA, USA, November 8-10 (2010)