Adapting the Obtrusiveness of Service Interactions in Dynamically Discovered Environments

William Van Woensel[1], Miriam Gil[2], Sven Casteleyn[1,2], Estefanía Serral[2], and Vicente Pelechano[2]

[1] Vrije Universiteit Brussel,
Pleinlaan 2, 1000 Brussels, Belgium
{william.van.woensel}@vub.ac.be
[2] Centro de Investigación en Métodos de Producción de Software,
Universitat Politècnica de València, Camino de Vera, 46022 Valencia, Spain
{sven.casteleyn}@upv.es, {mgil,eserral,pele}@pros.upv.es

Abstract. Due to the ubiquity of mobile devices, mobile service interactions (e.g., agenda notifications) may occur in any situation, leading to potential obtrusiveness (e.g., while in a meeting). In order to effectively adapt interaction obtrusiveness to suit the user's situation, the user's different situations should be defined in an unambiguous, generic and fine-grained way, while being valid across previously unknown, dynamically discovered environments. To realize this, we put the user in charge of defining his own situations, and exploit rich, descriptive environment information for defining and determining user situations. Our concrete approach aligns and extends two approaches, namely AdaptIO and SCOUT, to autonomously adapt mobile interactions in new, dynamically discovered environments. We supply a mobile user interface for defining situations, and validate it via an initial study with end-users.

Keywords: interaction adaptation, obtrusiveness adaptation, dynamic environment discovery.

1 Introduction

Mobile devices are an integral part of our lives. Improved battery life, screen resolution, input capabilities and computing power, as well as increased WiFi and 3G/4G coverage, have made them powerful and quasi-permanently connected computing devices. As a result, mobile devices are used at any time and everywhere, for instance to run general-purpose, resource-intensive applications (e.g., office applications, games) or to access online information and services.

Mobile service interactions comprise any interaction between mobile users and mobile services, where a service may proactively notify the user (e.g., agenda notification) or the user may directly contact the service (e.g., buying tickets from an e-ticket service). Because of their ubiquitous nature, mobile service interactions occur during a variety of situations, thus increasing their potential for obtrusiveness; for instance, loud notifications while the user is at a meeting or in a theatre. The necessity to reduce the obtrusiveness of mobile interactions is well recognized [1, 2]. In order to determine interaction obtrusiveness, most approaches currently either rely on

K. Zheng, M. Li, and H. Jiang (Eds.): MOBIQUITOUS 2012, LNICST 120, pp. 250–262, 2013.

(semi-)automatic learning techniques [3] or on designer knowledge [4]. Automatic learning techniques require training data and do not support cold-starts [5]; also, they require time to adjust to new user behavior. On the other hand, the designer cannot capture all situations that influence interaction obtrusiveness for all users, especially in a priori unknown environments without well-defined context or location models (e.g., at a theatre or at work). The only stakeholder with the required knowledge to define such situations accurately and unambiguously is the user himself. Furthermore, in order to effectively define situations across a priori unknown environments, we need to rely on rich environment data. In contrast, solely relying on local context, collected by sensors (e.g., microphones) or applications (e.g., agenda) [6], can lead to inaccuracy and ambiguity; e.g., simply turning up the ring volume in loud areas would not work while watching an action movie in a theatre. By relying on descriptive environment data, the user can specify he is in a "quiet-place" whenever he is inside a place of type "Theatre", thus defining situations in a more fine-grained and generic way.

Our goal is to adapt the obtrusiveness of mobile interactions in a priori unknown environments. To achieve this, the user is put in charge of defining his own situations, while rich environment context is exploited to define and determine user situations. Our approach aligns AdaptIO [7], a mobile obtrusiveness adaptation approach, with SCOUT [8], a mobile framework that dynamically discovers new (smart) environments, and autonomously collects context data. To ensure autonomy in any environment (potentially lacking middleware), all the components run on the mobile Android platform. Moreover, the AdaptIO approach has been extended in several ways. A user situation inferencing component has been added, which derives the user's current situation based on rich environment context. Furthermore, AdaptIO is extended with expressive user support for defining situations, via a user-friendly mobile interface. To validate the AdaptIO extension, where the user becomes a major stakeholder in mobile interactions adaptation, we evaluate the expressivity and usability of the interface by means of an initial study with real users. The developed software can be found at http://www.pros.upv.es/adaptio/dynamicenvironments.

2 Related Work

Some studies [3, 9] have been conducted on automatically adapting the modality configurations of mobile devices, based on user context. However, their focus is on context recognition, not on the modality configuration and how it influences obtrusiveness. Moreover, they rely on the designer to define the different user situations. In the same area of context-aware adaptation, [10] provides users with a UI to manually define new interactions in smart phones (e.g., gestures) and link them to device actions; however, interaction adaptation is not provided.

In the area of mobile interaction obtrusiveness, research focuses on minimizing unnecessary interruptions for the user [11]. This problem has been addressed directly by means of models of importance and willingness [4]. Also, [2] uses context-aware mobile devices to calculate the adequate timing for interruptions. Sensay [6] infers user's context from a variety of sensed data and determines whether or not the phone should interrupt the user while in regular communications. This research focuses primarily on determining *when* to interrupt for a particular application. In contrast,

our approach dynamically adapts the obtrusiveness of interruptions to suit the user's current situation. Furthermore, as far as we know, no approach provides support for newly discovered services.

A number of approaches aim to facilitate mobile devices in interacting with newly discovered smart environments. For instance, the SOFIA project [12] interacts with new, heterogeneous smart environments (e.g., with legacy services, different data formats) by providing mobile applications with shared, interoperable information spaces. In [13], personalized service access is supported across different, heterogeneous environments. However, these approaches require environments to be outfitted with extra middleware, deploying their specific software. On the other hand, mobile ad-hoc networks (MANETS) allow mobile applications and services to directly discover and communicate with each other, without requiring an existing infrastructure (e.g., via event-based communication) [14, 15]. MANETS allow powerful ad-hoc and loosely coupled communication with newly discovered services; however, their approach-specific software needs to be deployed on each component.

In contrast, we rely on open, minimally outfitted and standards-based environments that do not require middleware; instead, services are semantically described, and any coordination work is delegated towards the client. In addition, by relying on well-known standards, any client can discover new services and interact with them, without requiring support for specific approaches.

3 Architecture Overview

Our approach adapts the obtrusiveness of mobile interactions in previously unknown environments. To determine and define fine-grained and generic situations in such environments, our approach relies on rich and descriptive environment context. Furthermore, the user is made responsible for specifying his own situations, allowing for accurate and unambiguous situation definitions.

Our integrated system (see Fig. 1) comprises three layers: the *environment discovery and management layer*, which utilizes SCOUT to discover and manage previously unknown environments; the *services layer,* comprising interactive mobile services; and the *obtrusiveness adaptation layer,* which employs AdaptIO to adapt mobile interaction obtrusiveness. The AdaptIO system has been extended to support our goals, and its components moved to the mobile platform to ensure autonomy. Below, we elaborate on each of the layers.

3.1 Environment Discovery and Management Layer

This layer discovers a priori unknown (smart) environments, interacts with them, and collects context data. To achieve this, it relies on SCOUT, a mobile, client-side framework for the development of context-aware applications. SCOUT runs autonomously on the mobile device and utilizes technologies such as Quick Response (QR) codes, RFID/NFC and GPS to dynamically discover new environments and collect information on the user's surroundings. Based on this detected information, SCOUT builds a client-side, integrated view on the user's environment called the Environment Model, which is expressed using Semantic Web technology.

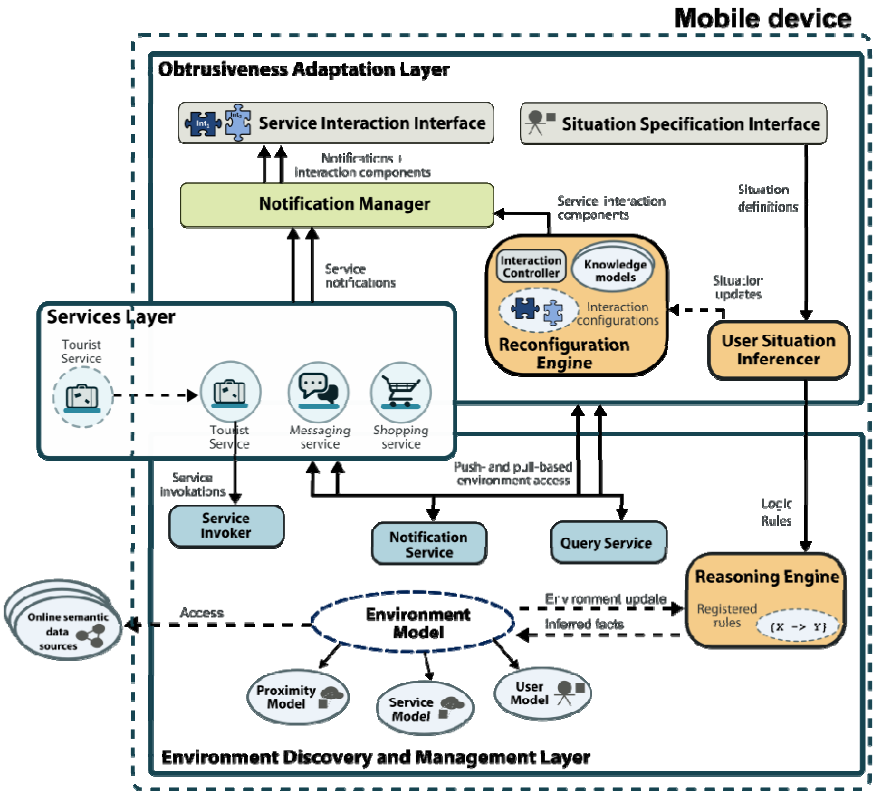

Fig. 1. System architecture overview

As shown in Fig. 1, four main information sources compose the Environment Model:

1/ *User Model:* This model contains the user's personal profile, including preferences, characteristics and device information (using ontologies such as FOAF).

2/ *Proximity Model:* This model encodes positional information about the user's environment, specifying which people, places, things and services are nearby.

3/ *Service Model:* This model keeps semantic descriptions of detected services in the user's environment (see below).

4/ *Online semantic data sources:* This includes RDF(S)/OWL data sources and semantically annotated websites, describing nearby people, places, things and services. SCOUT obtains references to these sources from the user's environment; for instance, by automatically reading URLs from QR codes or RFID tags, and by utilizing open online datasets such as LinkedGeoData[1], which link absolute coordinates (e.g., the user's GPS position) to online semantic information.

The SCOUT API provides mobile applications with access to the Environment Model in a push- and pull-based way, respectively via the **Notification Service** and **Query Service**. Data selection and caching techniques are in place to optimize data access

[1] `http://linkedgeodata.org/`

[16]. Finally, SCOUT provides applications with a general-purpose **Reasoning Engine**. Each time the user's environment changes, the engine (re-)evaluates the registered rules, potentially inferring new environment facts.

Regarding services support, SCOUT focuses on *lightweight* smart environments, i.e., environments outfitted with sensing, actuation and information services containing only the required service hardware and no external middleware. This way, SCOUT aims to support a wide range of smart environments that are cheap and easy to setup. SCOUT relies on environments that are fully standards-based and contain semantically described services; this way, any discovery, invocation and orchestration work can be delegated towards the client. In order to interact with newly discovered smart environments, SCOUT relies on the following semantic service stack. The W3C Semantic Annotations for WSDL and XML Schema[2] (SAWSDL) defines mechanisms to complement technical service descriptions (written using the W3C Web Service Description Language[3] or WSDL) with concrete semantics. WSMO-Lite[4] exploits the SAWSDL mechanisms, and utilizes a concrete ontology to semantically describe services. SCOUT converts and adds the online semantic service descriptions to the Service Model (see before) in RDF format, making it part of the Environment Model. In order to be alerted when certain services become nearby, mobile applications can register a discovery query with the Notification Service (or use the Query Service), to find services useful (nearby) services offering specific functionality. Applications interact with discovered services via the **Service Invoker**.

In order to convert WSDL descriptions (with SAWSDL annotations) to RDF, part of the SOA4ALL iServe[5] project code was extended and ported to Android. The Service Invoker uses the kSOAP2 library to interact with SOAP services, while the Reasoning Engine is based on the Androjena[6] general-purpose rule engine. We refer to [8] for more information on the SCOUT implementation.

3.2 Services Layer

This layer comprises local and remote services (see Fig. 1) that interact with the mobile user. Typically, plenty of local services or applications are running on a user's mobile device (e.g., agenda), which may for instance notify the user in case of important events (e.g., agenda deadline approaching). Remote services can also be plugged in, making their interaction capabilities available on the device. For instance, in Fig. 1, a local tourism application enables discovered remote tourist services to provide the user with information on good nearby hotel deals, and nearby points-of-interest. Such local applications register a discovery query with the Notification Service from the environment discovery and management layer (see Section 3.1). In case a relevant remote service is encountered, the application is notified, and utilizes the Service Invoker for remote communication. Based on the received data, the application provides notifications, for instance informing the user of good deals.

[2] http://www.w3.org/2002/ws/sawsdl/
[3] http://www.w3.org/TR/wsdl
[4] http://www.w3.org/Submission/WSMO-Lite/
[5] http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/iserve/
[6] http://code.google.com/p/androjena/

Local services can also utilize the environment discovery and management layer to enhance their own functionality. For instance, the shopping service (see Fig. 1) notifies the user in case a shop that sells products on his digital shopping list becomes nearby. To achieve this, the service registers a query with the Notification Service, to be alerted in case such shops become nearby. A number of service discovery scenarios and queries can be found on http://wise.vub.ac.be/Mobiquitous2012/.

3.3 Obtrusiveness Adaptation Layer

This layer adapts the obtrusiveness of mobile service interactions received from the *services layer*, depending on the user's current situation. This layer utilizes and extends the AdaptIO system, a mobile adaptation approach that adapts service interaction obtrusiveness at runtime. It is a model-based approach, where a service designer declaratively specifies the service's interaction adaptation behavior in knowledge models (see Section 4.1; for more information, we refer to [7]). In a nutshell, AdaptIO intercepts notifications from mobile services, chooses appropriate interaction resources (e.g., dialog, sound), and presents them to the user. Below, we elaborate on the main components (see Fig. 1).

Firstly, AdaptIO is extended with the **User Situation Inferencer**, which determines the user's current situation and notifies other components of changes. The user-supplied situation definitions (see Situation Specification Interface), expressed as logic rules, are passed to the environment discovery and management layer (Reasoning Engine), which uses them to accurately infer the user's situation. If a new situation is inferred, that layer's Notification Service notifies this component.

The **Reconfiguration Engine** determines which high-level interaction resources should be used for each service's interaction, based on the user's current situation. When alerted by the User Situation Inferencer of a new user situation, the engine consults the aforementioned *knowledge models* to retrieve the interaction resources that best suit the user's new situation. The **Interaction Controller** converts these abstract interaction resources (e.g., dialog) to concrete platform-specific (e.g., Android) interaction components, thus decoupling the models from the platform.

The **Notification Manager** receives notifications from mobile services and relays them, together with the service's latest interaction components (obtained from the Reconfiguration Engine), to the **Service Interaction Interface**. This interface displays the notifications to the user, employing suitable interaction components.

Finally, AdaptIO is extended with a **Situation Specification Interface**. This interface allows users to expressively define their situations, utilizing the environment context from the environment discovery and management layer. Situation definitions are passed to the User Situation Inferencer. In Section 4.2, we elaborate on the UI.

The Reconfiguration Engine is based on MoRE [12], which was ported to Android and is based on Autonomic Computing principles [17]. To query the knowledge models at runtime, we rely on a ported version of the Eclipse Modeling Framework Model Query[7] plugin. The model-handling operations are described in [18].

[7] `http://www.eclipse.org/modeling/emf/`

4 Methodology

In this section, we elaborate on the approach methodology and detail the tasks that need to be performed by the two stakeholders: the service designer and user. The designer is responsible for creating the knowledge models, which capture the service's desired behavior for adapting interaction obtrusiveness. On the other hand, the user is in charge of specifying his situations across which the obtrusiveness of interactions differ (e.g., in a meeting, in free-time). Service designers are not able to specify these situations for all users, especially in a priori unknown environments, without well-defined context or location models. To support this, our approach provides a mobile interface (called the Situation Specification Interface), which exploits environment context. Below, we elaborate on the designer's tasks. Section 4.2 discusses the Situation Specification Interface.

4.1 Service Designer: Adaptation Behavior Specification

In order to model the interaction obtrusiveness of services, we use the conceptual framework for implicit interactions presented in [19]. This framework defines two dimensions to characterize interactions: initiative and attention. Regarding the initiative factor, our approach focuses on proactive interactions (or notifications), where the system takes initiative and the user is potentially interrupted. The attention factor concerns an interaction's attentional demand, which can be represented on an axis. For the purpose of this paper, we divided the attention axis in three segments: *invisible* (user does not perceive the interaction), *slightly-appreciable* (user does not perceive the interaction, unless he makes an effort), and *user-awareness* (user is completely aware of the interaction, even while performing other tasks).

In our approach, the service's potential levels of interaction obtrusiveness correspond to the attention axis segments. Depending on the user's situation, the service's current (interaction) obtrusiveness level will vary. To capture this behavior, the designer creates the first knowledge model, namely an *obtrusiveness model*, which contains a state machine. Each state corresponds to an obtrusiveness level, and the guard conditions of the state transitions reference a user situation. The services' obtrusiveness models are checked by the Reconfiguration Engine (see Fig. 1) whenever it receives a new user situation (see Section 3.3); if any transition matches the new situation, it is fired, leading to a new obtrusiveness state for the service. In Fig. 2, we show the state diagram of a service that displays incoming messages.

Fig. 2. Obtrusiveness state diagram for a messaging service

When the user arrives at work (@*work* situation), the messaging service passes to the *slightly-appreciable* state, thus reducing notification obtrusiveness when the user is working. When the system determines that the user is no longer working (@*free-time* situation), the service goes back to the *user-awareness* state, increasing notification obtrusiveness. In case the system determines the user is in a meeting (@*meeting*), the service passes to the *invisible* obtrusiveness state, making sure the user is not disturbed. In addition, if the user is in the company of others (@*with-company*) while the service is at maximum notification obtrusiveness (i.e., *user-awareness*), the messaging service transitions to the *slightly-appreciable* level, so the user is not overly disturbed while socializing.

Furthermore, each of the obtrusiveness states is supported by the appropriate interaction resources. In the second knowledge model, the *interaction model*, the designer associates interaction resources with one or more obtrusiveness levels (e.g., slightly appreciable: status bar icon, vibration). A service's interaction model is consulted by the Reconfiguration Engine (see Fig. 1) in case a transition fires and leads to a new state (see before); this way, the engine can retrieve interaction resources suiting the new obtrusiveness level.

The knowledge models are represented in XML Metadata Interchange standard (XMI)[8]. Examples of obtrusiveness and interaction models can be found on http://wise.vub.ac.be/Mobiquitous2012.

4.2 Service User: Situation Specification

In order to guarantee accurate and unambiguous situation definitions, the user is put in charge of defining his own situations. We developed a mobile interface that allows users to specify their situations in a generic and fine-grained way, based on environment context. To increase usability and support nomadic users in a wide range of environments, the interface also supports directly *capturing* user situations. Below, we first discuss how the user can manually specify situations, and then how he can use the "capture" functionality.

4.2.1 Manually Defining Situations

In the first screen (see Fig. 3), the user can choose to define a still undefined situation (referenced in a service obtrusiveness model), or edit an already defined one. In the second screen (see Fig. 4), he can define the chosen situation using two aspects: location and time. A third, more advanced "free-form" option allows the user to place arbitrary constraints on his environment (see below). Using the location option, the user describes the location(s) he is in while being in the chosen situation. For each location (see Fig. 5), the user specifies whether he is *inside* or *nearby* a certain place, person or thing (i.e. physical entity) in that situation, and provides a way to identify that physical entity via its type and/or unique identification (URI). The user is aided via auto-complete functions: the type field suggests terms from well-known ontologies, as well as synonyms of the ontology terms (provided by WordNet); while

[8] `http://www.omg.org/spec/XMI`

the URI field suggests URIs that identify physical entities the user has encountered (this information is obtained from the environment discovery and management layer; see Section 3.1). The user can also specify time intervals (i.e., days of the week and time span) during which he is in the situation (see Fig. 6).

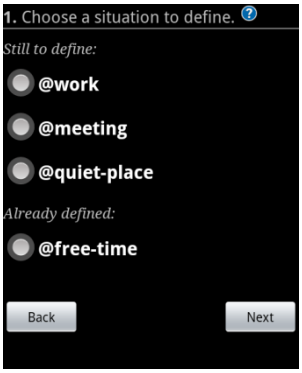

Fig. 3. Situation overview

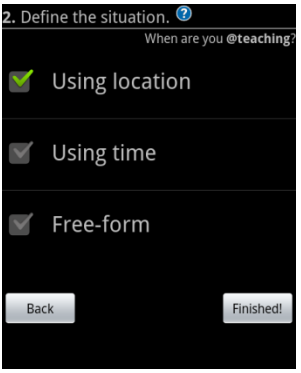

Fig. 4. Specification options

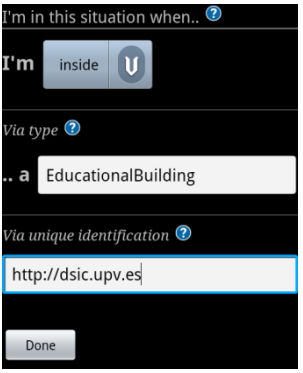

Fig. 5. Define via location

The advanced, "free-form" option (see Fig. 7) allows defining situations in a more powerful and expressive way, by placing arbitrary constraints on the user's environment context. A constraint consists of a property and a value field. A user may arbitrarily constrain a property value, either by providing a concrete string or by linking its connector to other constraints. This free-form option, with connectable components, resembles the popular Yahoo! Pipes online mashup tool. In this example, the user is inside his office during the *@work* situation. To describe this in a generic way, the user specifies the *inside* property, and creates two constraints on the place he should be inside of. The first constraint states the *type* of the place should be "Office", while the second specifies the place is the user's office (via the *housesPerson* property). Using the constraint's connectors, the user connects the two new constraints to the first constraint's value field. The property and value fields are respectively backed by the same auto-complete functions mentioned above.

4.2.2 Capturing Situations

The "capture" option exploits the user's current environment to quickly and easily specify situations. In this option, the user takes a snapshot of his environment, fine-tunes it, and attaches it to a situation. For example, the user is sitting in a movie theatre, and one of the services produces a loud notification. The *obtrusiveness model*, defining the service's adaptation behavior (see Section 4.1), makes sure notifications are handled at the *invisible* obtrusiveness level in an *@quiet-place* situation (e.g., classroom). However, mobile users are typically nomadic and move in a wide range of (previously unknown) environments, making it difficult even for them to foresee every situation-relevant environment (e.g., movie theatre). After quickly (and manually) turning off the device's sound, the user selects the capture option. This option re-uses the screens from the previous "define" option (see previous section) and populates them, based on the user's current environment. The user selects the

location aspect (see Fig. 8), and sees that the "inside MovieTheatre" location is present, as well as some other captured locations (e.g., "nearby Cafe"). He then removes the irrelevant locations, and also unchecks the time and free-form option, since they are not relevant in this case. In the final screen, the user attaches the fine-tuned context to the *@quiet-place* situation, thus making sure the *invisible* obtrusiveness level will be utilized in movie theatres as well.

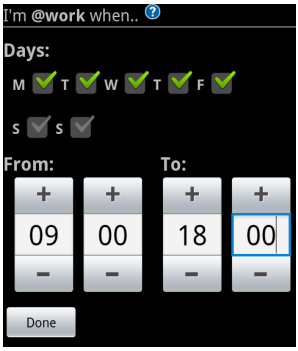

Fig. 6. Define via time

Fig. 7. Free-form option

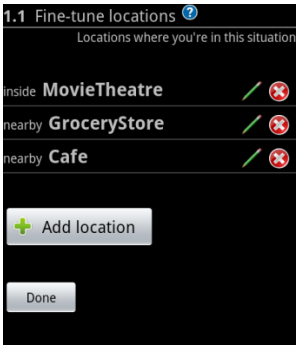

Fig. 8. Captured locations

5 Evaluation

We validated the usability and expressivity of the Situation Specification Interface by means of a user evaluation, where users had to specify six situations of varying complexity via the definition and capturing options (see Section 4.2). The final, most complex situation required using the more advanced free-form option. For detailed information on these situations, we refer to http://wise.vub.ac.be/Mobiquitous2012/.

In each user session, we took five minutes to shortly explain the interface, and then let the user specify the described situations. We noted the required time, as well as any encountered difficulties and errors during their task. After performing their task, the users filled out the Post-Study System Usability Questionnaire (PSSUQ) [20]. This questionnaire is a 19-item instrument for assessing user satisfaction with system usability. Specifically, it studies the following four dimensions: overall satisfaction with the system, its usefulness, information quality, and interface quality. A total of 8 subjects participated in the experiment (5 male and 3 female), between the ages of 25 to 35. All of them had a strong background in computer science, being students or researchers; they were also familiar with the use of a smartphone, and 4 out 8 owns an Android-based smartphone similar to the one used in the experiment.

5.1 Evaluation Results

Fig. 9 shows a summary of the PSSUQ questionnaire results; the complete dataset can be downloaded from http://wise.vub.ac.be/Mobiquitous2012/. Overall, users found the interface simple to use (questions 1, 2) and very easy to learn (7), while they also felt they could complete tasks effectively (3) and quickly become productive using

the system (8). Users found the provided information more or less clear (11), easy to find (12) and understand (13), and clearly organized (15). Overall, around 80% of the users found the interface pleasant (16), and 75% liked using the interface (17). Averaging the questionnaire results, on a scale from 1 (strongly agree) to 7 (strongly disagree); overall satisfaction was 3.09, usefulness was 3.2, information quality was 3, and interface quality was 3.04.

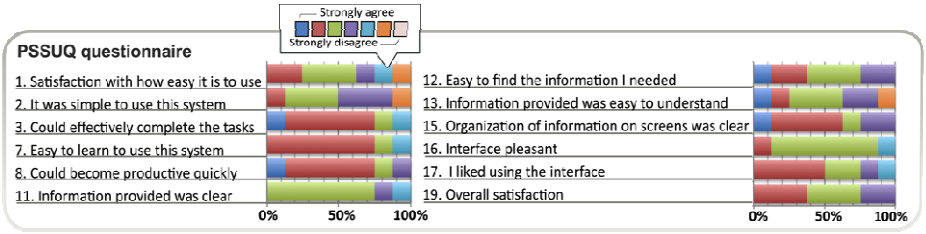

Fig. 9. Summarized questionnaire results

On average, users took about 13 minutes to specify all situations; since "capturing" the first five situations proved trivial, they were left out the remaining evaluations of the capture option. One user had initial difficulty with the location method (see Section 4.2.1); but the other users had no problems. Six out of seven users had difficulty using the free-form method while testing the define option; on the other hand, they found using this method easier when "capturing" situations.

In conclusion, the evaluations show the interface to be usable and expressive, allowing users to specify non-trivial situations within a short time span. Moreover, the capture option makes it very simple to specify most situations. The free-form method, which allows for more generic and complex situation definitions, proved to be more difficult to use and have a steep learning curve. However, using this option becomes much easier in the capture option, since users are able to fine-tune a given free-form specification instead of creating one from scratch. We do note that this is a preliminary evaluation, and additional experiments, with a larger and more heterogeneous user group, are needed to confirm and generalize these results.

6 Conclusion and Future Work

In this paper, we presented an autonomous approach that adapts mobile interaction obtrusiveness across dynamically discovered, a priori unknown (smart) environments. The user represents a major stakeholder, being the only one with the required knowledge to accurately and unambiguously define his own situations. We provide the user with a mobile interface, where he can manually define or directly capture situations, based on expressive environment context. Our approach runs entirely on the mobile device, and does not require orchestrating or adaptation middleware.

Regarding future work, we are considering fine-tuning the mobile interface based on user feedback; more specifically, by improving the usability and learnability of the free-form option. Further user tests are still required to fully assess the usability of the

interface, with a focus on users without computer science backgrounds. A major challenge is to allow users to specify interaction adaptation behavior as well, a task now exclusively reserved for the service designer. This way, the user could express custom adaptation behavior not initially foreseen by the designer.

Acknowledgments. This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011 and co-financed with ERDF, in the grants program FPU. Sven Casteleyn is supported by an EC Marie Curie Intra-European Fellowship (IEF) for Career Development, FP7-PEOPLE-2009-IEF, N° 254383.

References

1. Chittaro, L.: Distinctive aspects of mobile interaction and their implications for the design of multimodal interfaces. Multimodal User Interfaces 3, 157–165 (2010)
2. Ho, J., Intille, S.S.: Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In: Proc. of CHI 2005, pp. 909–918. ACM (2005)
3. Valtonen, M., Vainio, A.-M., Vanhala, J.: Proactive and adaptive fuzzy profile control for mobile phones. In: Proc. of PERCOM 2009, pp. 1–3. IEEE Computer Society (2009)
4. Chen, H., Black, J.P.: A quantitative approach to non-intrusive computing. In: Proc. of Mobiquitous 2008, pp. 1–10 (2008)
5. Serral, E., Valderas, P., Pelechano, V.: Improving the cold-start problem in user task automation by using models at runtime. In: Proc. of ISD 2010, pp. 648–659. Springer (2010)
6. Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., Wong, F.L.: Sensay: A context-aware mobile phone. In: Proc. of ISWC 2003, p. 248 (2003)
7. Gil, M., Giner, P., Pelechano, V.: Personalization for unobtrusive service interaction. Personal and Ubiquitous Computing 16(5), 543–561 (2012)
8. Van Woensel, W., Casteleyn, S., Paret, E., De Troyer, O.: Mobile Querying of Online Semantic Web Data for Context-Aware Applications. IEEE Internet Comp. 15(6), 32–39 (2011)
9. Assad, M., Carmichael, D.J., Kay, J., Kummerfeld, B.: Personisad: distributed, active, scrutable model framework for context-aware services. In: LaMarca, A., Langheinrich, M., Truong, K.N. (eds.) Pervasive 2007. LNCS, vol. 4480, pp. 55–72. Springer, Heidelberg (2007)
10. Korpipaa, P., Malm, E.-J., Rantakokko, T., Kyllonen, V., Kela, J., Mantyjarvi, J., Hakkila, J., Kansala, I.: Customizing user interaction in smart phones. IEEE Perv.Comp. 5, 82–90 (2006)
11. Ramchurn, S.D., Deitch, B., Thompson, M.K., Roure, D.C.D., Jennings, N.R., Luck, M.: Minimising intrusiveness in pervasive computing environments using multi-agent negotiation. In: Proc. of MobiQuitous 2004, Los Alamitos, CA, USA, pp. 364–372 (2004)
12. Toninelli, A., Pantsar-Syväniemi, S., Bellavista, P., Ovaska, E.: Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability. In: International Workshop on Middleware for Pervasive Mobile and Embedded Computing, pp. 5:1–5:4. ACM, New York (2009)

13. Retkowitz, D., Armac, I., Nagl, M.: Towards Mobility Support in Smart Environments. In: 21st International Conference on Software & Knowledge Engineering, pp. 603–608 (2009)
14. Hadim, S., Al-Jaroodi, J., Mohamed, N.: Trends in Middleware for Mobile Ad Hoc Networks. Journal of Communications 1(4), 11–21 (2006)
15. Meier, R., Cahill, V.: STEAM: Event-Based Middleware for Wireless Ad Hoc Network. In: 22nd International Conference on Distributed Computing Systems, pp. 639–644 (2002)
16. Van Woensel, W., Casteleyn, S., Paret, E., De Troyer, O.: Transparent Mobile Querying of online RDF sources using Semantic Indexing and Caching. In: Bouguettaya, A., Hauswirth, M., Liu, L. (eds.) WISE 2011. LNCS, vol. 6997, pp. 185–198. Springer, Heidelberg (2011)
17. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. Computer 36, 41–50 (2003)
18. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer 42(10), 37–43 (2009)
19. Ju, W., Leifer, L.: The design of implicit interactions: Making interactive systems less obnoxious. Design Issues 24(3), 7–84 (2008)
20. Lewis, J.R.: Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. Int. J. Hum.-Comput. Interact. 7(1), 57–78 (1995)