Put the User in Control: Ontology-driven Meta-level Interaction for Pervasive Environments

Geert Vanderhulst Kris Luyten Karin Coninx
Hasselt University – transnationale Universiteit Limburg – IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{geert.vanderhulst,kris.luyten,karin.coninx}@uhasselt.be

Abstract

Pervasive computing environments are complex to interact with due to the dynamic assembly of interaction resources and the need to adapt to sudden changes in the environment configuration. When the complexity of such an environment is masked by the underlying computing system, end-users are often left with limited or no control over their interactive space. This brings up the need to make users aware of their surroundings and to provide them with runtime control over the environment configuration. We present a semantic meta-layer that encapsulates a model, view and controller to support the design of context-aware pervasive applications that can be controlled and evaluated by the end-users at runtime.

1. Introduction

The devices people carry around and the appliances embedded in their surroundings give rise to a pervasive interactive space that interconnects users, physical resources and computational entities. The dynamic composition of such a space poses various challenges to its designers, related to both the system and the end-user support. However, we witness pervasive frameworks primarily focus on either the system support or the user support, leaving a gap between the design of domain-specific applications and the end-user interaction with the pervasive infrastructure. Many pervasive systems serve as middleware frameworks to simplify the development of context-aware applications [4, 1, 5] while others focus on ambient interaction techniques on a case-per-case basis [9, 7]. These ad-hoc approaches for assembling interactive spaces suggest the need for runtime control over the environment setup.

Coutaz introduced the concept of a meta-user interface (meta-UI) to denote a kind of interactive system that allows users to control, mould and understand interactive spaces [2]. However, in current interactive infrastructures system designers still tend to automate the adaptation of a pervasive application to the context of use. A mixed-initiative approach could avoid confusion of the end-user when adaptation strategies are only suggested by the system and not carried out automatically, instead handing over control to the end-users to finetune their interaction spaces. Pervasive computing middleware should not only target interaction with applications, but also focus on interaction with the computing system itself, denoted as *meta-interaction*. From a user's point of view, a pervasive computing system can be considered as an interactive application that allows to communicate with the environment.

In this paper, we present a dynamic meta-layer that integrates system and user support for context-aware pervasive applications. This layer is designed according to the Model-View-Controller (MVC) pattern: the model captures the execution context of the environment (section 3); the view presents a meta-UI constructed from information in the model (section 5); the controller processes user and system actions that might invoke changes in the model which are reflected in the view (section 4). This meta-layer is heavily inspired by ontologies which are the driving force behind our context-aware infrastructure. We illustrate how ontologies help to provide runtime control over pervasive applications by reducing the complexity to discover, observe, couple and manipulate the resources in a pervasive world.

2. Related work

Grimm. et al. identify three requirements system support for pervasive systems must meet: embrace contextual change, encourage ad-hoc compositions and recognize sharing as the default [4]. These requirements are also relevant at the meta-level, i.e. beyond the domain-dependent ser-

978-0-7695-3542-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ONTORACT.2008.15

vices that support human activities. Applications need to adapt to a changing context, but should leave the option to address the new situation to the end-user. Besides, manual control should be provided to dynamically assemble physical resources and software services in the user's vicinity. This is for example illustrated in Huddle [9] where appliances are connected using a content flow diagram supplied by the user. Furthermore, as users collaborate, they should be able to share information. We support collaboration at the task-level: a task is a resource that can be shared by multiple entities.

In [7] Hellenschmidt et al. propose ad-hoc device ensembles that carry out execution strategies spontaneously and behave like single devices the user can interact with. Our meta-layer also supports the allocation of device ensembles to enable pervasive applications, though these are not self-organizing by default: the user is involved in the assembly of the ensemble. Previous research on mixed-initiative user interfaces suggests agent entities that ask users about their goals and needs instead of guessing these [8].

We use ontologies to describe the context of use; an approach that has been researched extensively before. In [10], [1] and [5] an architecture based on ontologies is proposed to enable the development of context-aware pervasive applications. Chen et al. [1] designed a rich ontology for ubiquitous and pervasive applications (SOUPA) that is exploited in a broker-centric agent architecture to support knowledge sharing and context reasoning. In [5] context discovery and knowledge sharing are supported using an ontology-based context model and an OSGi-based middleware infrastructure. Opposed to this work, we do not only use ontologies to capture and query the execution context, but also to provide runtime control over a pervasive application. Coutaz showed the importance of providing runtime control over the user interface configuration by the end-user in [2]. The complexity of these environments can confuse the end-user and can benefit from additional tools to query and configure the user interface at runtime. Related work by Demeure et al. underlines the importance of using additional semantic information to increase the plasticity of the user interface at runtime [3].

3. A distributed context model

Pervasive applications rely on the ability to query, store and exchange contextual information. Ontologies have proven to be valuable for describing context and making context information accessible to software components. As context information is provided by different computing resources, we need to decide whether the context information is managed by a central instance or remains distributed over the environment. When all available context information is stored in a central datastore, a trade-off has to be made between the amount of information that is modelled and the extra overhead caused by contextual change. A knowledge base can provide powerful querying facilities, but also results in lots of update operations when the environment context emerges. Resources often produce more context information than the active pervasive applications actually consume, rendering the propagation of context inefficient in a centralized approach. In our approach, we choose for a distributed model that delivers context on demand while maintaining a shared view on the environment topology.

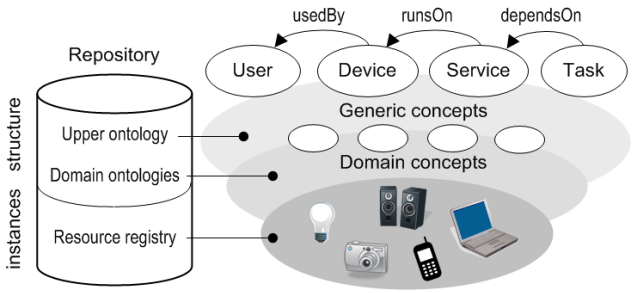


Figure 1. The environment model.

As depicted in figure 1, our context model spans a central environment repository and a set of heterogeneous computing nodes. The environment repository stores information about the *structure of resources*, captured by OWL DL ontologies. An upper environment ontology defines a base structure upon which ontologies for specific domains are constructed. These domain ontologies are aggregated at runtime and shared amongst the software components that depend on their knowledge base. Furthermore, the environment repository includes a registry with references to *instances of resources* whose execution context resides on distributed computing nodes. The registry facilitates the discovery of resources: software entities can interrogate the meta-information stored in references (e.g. type and location of a resource) and use this information to acquire the full context of the resource.

The separation of structural information and instance data demands for distributed data management and query processing across federated devices, which is discussed in the next sections.

3.1. Resource management

Devices that enter a pervasive computing environment can serve multiple purposes: some are used as interaction device while others are better suited to visualize information. A device's specific role is dynamically assigned by an application after the device is integrated in the environment as a heterogeneous computing node that offers a platform to publish resources (e.g. the device itself, the services that

run on the device, etc). A resource is identified by a URI and published as a data object on a computing node where it manages its own state. A resource can for instance store its context in a chip, main memory, a database on the internet, etc. It is important to note that a resource's context is not continuously replicated in a semantic datastore, but serialized *on demand* into a semantic representation. This ensures its state can be queried against an ontology, regardless of where and how it is stored internally. Additionally, software entities can subscribe to sensors associated with a resource to get notified of specific context changes that occur in the resource. These sensors and the data they produce are also described in an ontology so that interested entities know how to consume the information. The combination of pull (query for context changes) and push (propagate context changes through sensors) allows us to develop responsive context-aware applications.

Published resources are advertised with a reference in the registry that represents the resource during its lifetime. To avoid the registry becoming polluted with deprecated references, e.g. because a service fails or when a device and its local resources leave the environment without proper announcement, resources are regularly probed to verify whether they are still available.

3.2. Query answering over distributed data

Federated query solutions such as DARQ [11] and KAONp2p [6] rely on distributed query processing and aggregation of query results. These solutions scale well in environments where large server-side datastores are addressed, but they are costly to integrate in a dynamic heterogeneous environment. Embedded computing nodes are constrained in processing power and hence are less suited for query processing. Since the context of individual resources is expected to be limited in size, the benefit of distributed query processing will be minimal after all. Besides, when computing nodes are part of the environment configuration, their relations with other resources still require query answering over distributed instances. In our approach, queries are directed to a query manager layered on top of the environment repository which processes a query in two steps:

1. **Data aggregation**: a temporary model is prepared prior to query evaluation. This model shares domain knowledge from the environment repository and includes context information fetched from distributed nodes. A dedicated query that is derived from the final query selects references to resources whose context should be resolved and included in the model.

2. **Query evaluation**: with all relevant context data aggregated in a temporary model the query can be evaluated against this model.

```
Q  : SELECT ?d WHERE {?d :usedBy <:PersonX>}
Q_r : SELECT ?r WHERE {?r :refType ?t . ?t rfds:subClassOf :Device}
```

Listing 1. SPARQL queries to interrogate distributed resources in the context model.

Listing 1 shows a query Q that asks for the devices that are used by 'PersonX'. Since the registry only contains references to resources, the environment repository does not include the required context information (i.e. instance data) to evaluate the query. By analyzing the domain of the 'usedBy' predicate in the ontology, information can be derived to determine what type of resources need to be dereferenced and included in the virtual model in order to guarantee the final query results are complete. As the domain of the 'usedBy' relation defines device resources, references will be selected using a query Q_r (listing 1) which returns a reference to all device resources in the environment. The query Q_r can also be specified manually to avoid the reasoning step over the final query and thus increase the execution speed.

4. Tasks: controllers of a pervasive application

Users think in terms of the goals they want to achieve in the world surrounding them [7, 8]. The environment should be sensitive to the user's goals and adapt its configuration to support these goals. We consider a task as a concept that can be understood by either the user and the system. A task is approached from both a system perspective and an end-user perspective.

A *system task* orchestrates a number of resources in the environment. The component that defines a system task consists of glue that combines the functionality of various services with the interaction capabilities of different devices in order to reach a user's goal. The ensemble of devices allocated by a system task forms an heterogeneous interaction space on which user tasks, presented by a user interface, can be deployed. A system task process acts as the controller of a pervasive application: it reconfigures an application when changes in the environment setup could improve or compromise the application's workflow. The task process monitors the environment to detect the presence of new resources suited for its goal and deals with resources involved in the task that leave the environment, e.g. by redistributing a user task from a lost device to another suited device.

A *user task* corresponds to an activity for which a user interface is available, e.g. playing music, navigating slides, etc. It provides an extra abstraction over a user interface to stimulate context-awareness: a suitable representation for a user task is selected at runtime based on the context of use, e.g. a graphical user interface or a speech user interface.

4.1. From user to user task

The available resources support different user activities which are either presented by a user or a system task. Figure 2 illustrates user interaction with either of these tasks. Interaction with a user task relies on the initiative of the

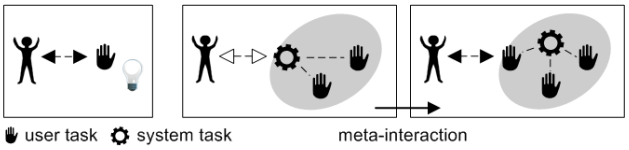


Figure 2. The user directly interacts with a resource through a user task or participates in a pervasive application through a user task that is assigned after meta-interaction with the application's system task.

user: a user interface is migrated to the user's interaction device and interaction is carried out. This enables the user to directly observe and manipulate the state of a resource. Interaction with a system task is more complicated because it relies on a mixed-initiative of user and system. A system task has its own dynamic execution context, composed by the ensemble of resources involved in the task process. In order to involve the user in the configuration of the ensemble and to decide on distribution strategies of user tasks in the interaction space, we propose a set of operators to interact with a system task, outlined in table 1.

start	A new system task process is initiated.
stop	A system task process is terminated.
setup	A user interface to configure the system task is migrated to the end-user's device.
suspend	The full context of the application, managed by its system task process, is serialized into a task profile and the application is suspended.
resume	A previously stored application context is restored from a task profile.
share	A system task is shared and can be joined.
invite	A user is invited to join a collaborative task.
join	A user participates in an application.
leave	A user stops interacting with an application.

Table 1. Meta-operators help to configure pervasive applications.

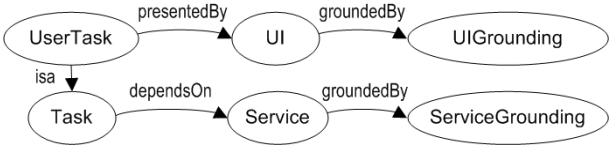
A system task corresponds to a process that coordinates a pervasive application. Such a process is respectively initiated and terminated using start and stop operators. A setup operator maps on an application-dependent user task that allows the end-user to configure the application context. Hence the user is in control of her interaction space whilst the task process can still develop and propose adaptation strategies when change occurs. We believe the balance between user and system initiative should be evaluated on a per application basis.

Tasks can be dispersed over time when users decide to temporarily interrupt their activities. In order to guarantee the continuity of the task, its execution context should be suspended and resumed (using suspend and resume operators). We use task profiles as a medium to store and traverse an application's execution context. If an application is suspended by its initiator, its related user tasks are dismissed and allocated resources are released. When resuming the task (i.e. feeding the task profile to a new task process), the environment context might have changed and the services and interaction devices that were previously allocated might not be available anymore. In this case, the system task will need to reconfigure itself and adapt to the new environment context, for instance by suggesting new interaction possibilities to the end-user(s).

The operators share, invite, join and leave target collaborative interactions at the task-level. The meta-layer acts as a platform to setup collaborations between users independent of the application domain. Concrete roles (i.e. user tasks) are assigned at runtime; joining and leaving a task is considered as contextual change and is also handled by the system and/or user.

4.2. From user task to user interface

When a user task is assigned to a device, the bootstrap software installed on the target device will lookup a suitable user interface (UI) in the model to present the task. The upper environment ontology defines a 'UserTask' and a 'UI' concept which are interconnected through a 'presentedBy' relation, as shown in figure 4.2. Since a user task can



be presented by various UIs (SwingUI, WebUI, ...), we do not enforce a single 'generic' UI format (e.g. a markup language that can be renderered on a wide range of devices), but rather leave the option to provide different types of UIs to accommodate a broader range of applications. When a device chooses to render a UI, it should understand the

grounding specifics for this type of UI which are also described in the ontology. The grounding concept is adopted from OWL-S, where a grounding dictates how to access a (Web) service. Likewise, software services are grounded so they can be accessed remotely. For example, when a UI is rendered to enable a user task, proxies to interact with the task's related services are automatically constructed and passed to the UI component. For more information on the technical details of the grounding process, we refer to [12].

5. A view on the environment

Users immersed in a pervasive environment should become aware of the tasks they can execute using the resources present in the environment. Therefore they should also discover the resources that can be used to accomplish the task. Moreover, users should be able to manage the tasks they are involved in, i.e. (re)configure them or switch between tasks. Pervasive environments also demand for collaborative interactions between users in order to reach common goals. Figure 3 presents a meta-UI, layered on the environment's context model, that accommodates these requirements. The meta-UI provides a view on the resources

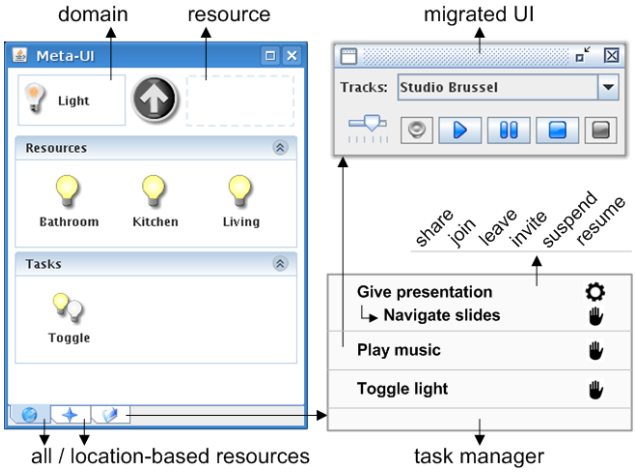


Figure 3. A meta-UI provides a view on the environment and allows end-users to explore the available resources and to execute their supported tasks.

the user can interact with. This view is rendered from information in the underlying model and can be displayed on devices carried by the end-user. As a resource can range from a software service (e.g. weather service) to a physical resource (e.g. digital fridge), the meta-UI provides uniform access to various task-supporting entities. The resources present in the environment are structured into domains that correspond to the ontologies in which they are defined. For example, a domain-specific ontology that models a light resource along with services to operate a light gives rise to a 'Light' domain that provides access to all available light resources (see figure 3).

5.1. Location-based view

As the number of resources in a computer-augmented environment could grow large, it can be hard to navigate to a specific resource. Besides, some resources will only support certain tasks when the user is near the resource, e.g. switching stations on a television device or playing a collaborative game on a shared screen. By tracking the location of the user and the resources present in the environment, the system can discover the resources in the user's vicinity and present those in a personal, location-based view. For example, when resources are labeled with RFID tags they can be sensed when approached by the user's device, triggering the meta-UI to navigate to the resource and present its supported tasks.

5.2. Task management

While navigating through the meta-UI, the user is presented with the tasks that are supported by a resource or a domain in general. These are user tasks or tasks coordinated by the system that are linked with resources in the model and that can be accessed through the meta-UI. For example, a task to operate a light can be attached to a light resource. When a user task is selected, a user interface is migrated to the user's interaction device as outlined in section 4.2. Selecting a system task initiates a new task process to which the meta-operators introduced in table 1 can be applied. A task process could for instance coordinate a number of resources in order to support the goal of giving a presentation. In this case, a specific user task can be to navigate the slides of the presentation which is assigned after meta-interaction with the system task (see figure 2).

All tasks being executed are listed in a task manager component, allowing the end-user to switch between tasks and manage them at runtime. When a task is suspended, its context is automatically stored on the requesting interaction device and the task is marked as idle. Besides, users can share the system tasks they initiate which can then be joined by other participants through the meta-UI. When a task is shared, it is made available to all users in the meta-UI. Users can also choose to setup a private interaction space by sending an invite to join a task to a number of selected users. Hence the meta-UI supports collaborative interaction at the task level.

6. Conclusions

In this work we address the gap between system and end-user support for pervasive computing environments. We have presented an ontology-based distributed environment model that captures the context of use along with a meta-UI layered on this model that maintains a view on the available resources and their related tasks. The meta-UI fulfils the role of a traditional application menu and task manager found on single-user workstations, yet reshaped towards a pervasive environment to provide end-users with runtime control over the environment configuration. Moreover, we explained how tasks act as a controller between model and view and as a mediator between user and user interface. The model, view and controller are encapsulated in a meta-layer that enables the development of pervasive context-aware applications that can be monitored and operated by the end-users at runtime. The proposed approach was successfully applied in several practical cases, e.g. a pervasive paint application and an arcade game [12].

Directions for future work include support for personalized behaviors and improved window management of migrated user interfaces on heterogeneous devices. Information in a user profile could be exploited to personalize the behavior of a pervasive application and to apply a personal look and feel to its migrated interfaces. A difficulty we currently experience is the proper positioning of these UIs on the screens of the end-user devices. It might be useful to integrate a layout manager in the meta-UI and/or define meta-information in the ontology to specify how a UI should be presented (e.g. fullscreen). Further research is also needed to find an optimal balance between the initiative of the system and the end-user to deal with changes in the environment configuration, i.e. to minimize the cognitive load of the user. The meta-UI could learn from the user's inputs and become predictive in the future.

Acknowledgments

Part of the research at EDM is funded by EFRO (European Fund for Regional Development) and the Flemish Government. Funding for this research was also provided by the Research Foundation – Flanders (F.W.O. Vlaanderen, project number G.0461.05).

References

[1] H. Chen, T. Finin, and A. Joshi. Semantic Web in the Context Broker Architecture. In *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PERCOM'04)*, pages 277–286, 2004.

[2] J. Coutaz. Meta-User Interfaces for Ambient Spaces. In *Proceedings of the 5th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA'06)*, pages 1–15, 2006.

[3] A. Demeure, G. Calvary, J.-S. Sottet, and J. Vanderdonckt. A Reference Model for Distributed User Interfaces. In *Task Models and Diagrams for User Interface Design: Proceedings of the Forth International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA'05)*, pages 79–86, 2005.

[4] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. System Support for Pervasive Applications. *ACM Trans. Comput. Syst.*, 22(4):421–486, 2004.

[5] T. Gu, H. K. Pung, and D. Q. Zhang. Toward an OSGi-Based Infrastructure for Context-Aware Applications. *IEEE Pervasive Computing*, 3(4):66–74, 2004.

[6] P. Haase and Y. Wang. A Decentralized Infrastructure for Query Answering over Distributed Ontologies. In *Proceedings of the 22nd ACM Symposium on Applied Computing (SAC'07)*, pages 1351–1356, 2007.

[7] M. Hellenschmidt and T. Kirste. DynAMITE - The Approach to Ubiquitous Computing within dynamic ad-hoc Device Ensembles. *Computer Graphik Topics*, 15(6):24–25, 2003.

[8] E. Horvitz. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'99)*, pages 159–166, 1999.

[9] J. Nichols, B. Rothrock, D. H. Chau, and B. A. Myers. Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06)*, pages 279–288, 2006.

[10] S. Peters and H. E. Shrobe. Using Semantic Networks for Knowledge Representation in an Intelligent Environment. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM'03)*, page 323, 2003.

[11] B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *Proceedings of the 5th European Semantic Web Conference (ESWC'08)*, 2008.

[12] G. Vanderhulst, K. Luyten, and K. Coninx. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring. In *Proceedings of the 4th IET International Conference on Intelligent Environments (IE '08)*, 2008.