ReWiRe: Designing reactive systems for pervasive environments

Non Peer-reviewed author version

VANDERHULST, Geert; LUYTEN, Kris & CONINX, Karin (2008) ReWiRe: Designing reactive systems for pervasive environments. In: INTERACTIVE SYSTEMS: DESIGN, SPECIFICATION, AND VERIFICATION, PROCEEDINGS., 1536, p. 155-160.

DOI: 10.1007/978-3-540-70569-7_15
Handle: http://hdl.handle.net/1942/8508

ReWiRe: Designing Reactive Systems for Pervasive Environments

Geert Vanderhulst, Kris Luyten, and 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. The design of interactive software that populates an ambient space is a complex and ad-hoc process with traditional software development approaches. In an ambient space, important building blocks can be both physical objects within the user's reach and software objects accessible from within that space. However, putting many heterogeneous resources together to create a single system mostly requires writing a large amount of glue code before such a system is operational. Besides, users all have their own needs and preferences to interact with various kinds of environments which often means that the system behavior should be adapted to a specific context of use while the system is being used. In this paper we present a methodology to orchestrate resources on an abstract level and hence configure a pervasive computing environment. We use a semantic layer to model behavior and illustrate its use in an application.

1 Introduction

Although pervasive computing environments have gained much importance over the last years, they remain among the most complex environments to develop interactive software for. Generic development environments that explicitly target ambient spaces are scarce because of several reasons:

Lack of engineering approaches : most pervasive applications are ad-hoc coded and hence are only applicable in just one situation [5].

New middleware requirements : generic middleware is required to abstract hardware, deal with distributed computing resources, steer the migration of user interfaces [7].

Support for situation-aware human-computer interaction : the context in which tasks are executed affects the user's interaction with the system [2].

In this paper we report on the *ReWiRe* framework [8] which supports the dynamic composition and adaptation of behavior rules in a pervasive environment. With services and devices that enter and leave the user's environment, the ability to support the dynamic composition of the interactive system is a strong requirement. Our approach relies on a semantic layer that captures the context of the entire environment (its users, devices, services, etc) and uses this information to configure the behavior of resources (section 3). Since orchestration is performed at an abstract level, we can mask the underlying service technologies (section 4). To accomplish this, we have underpinned our

framework with semantic Web frameworks such as RDF, OWL and OWL-S [8]. We demonstrate our approach by means of a test-bed that illustrates how services can be orchestrated and (re)wired at runtime to take advantage of changes in the environment configuration (section 5).

2 Related Work

The emergence of Web services has lead to different solutions to coordinate distributed business processes, e.g. BPEL [1]. Pervasive services demand for similar orchestration tools that take into account the full environment context. This goes beyond dealing with preconfigured service compositions, but also involves runtime adaptation of the environment configuration whilst users are interacting with it. Muñoz et al. [5] propose a model-driven approach for the development of pervasive systems. A domain specific language (PervML) is used to specify the system using conceptual primitives suitable for the target domain.

Mokhtar et al. [4] also study highly dynamic pervasive computing environments where users need to perform tasks anytime anywhere, using the available functionality of the pervasive environment. Grimm [3] identified three requirements that should be fulfilled by systems that support these dynamic interactive pervasive environments: support for a continuously changing context of execution and make this explicit in the system design, support for ad-hoc composition of devices and services and collaboration among users should be supported out-of-the-box. With *ReWiRe* we tackle exactly these requirements.

3 Environment and Behavior Model

We use a semantic layer to describe the context of use of an interactive software system during its lifetime. This layer includes both an environment and a behavior model which are described by an ontology. Several (domain-specific) ontologies can be merged at runtime and offer a dynamic schema that evolves when new software components become available. The system's configuration is linked with an instance of these ontologies. Figure 1 presents the environment and behavior ontology together with the OWL-S ontologies (note that 'a' labels correspond to 'isa' relationships). The OWL-S ontology describes a service in terms of what it does (*profile*), how it is used (*model*) and how to interact with it (*grounding*). Although OWL-S services are usually considered to be semantically enriched Web services, their scope is not limited to Web services only. A service can be any arbitrary piece of functionality that can be used in the environment. With OWL-S one can describe a service (e.g. its inputs and outputs) in a uniform way and define a custom grounding that provides details on how to invoke that service. We use OWL-S service descriptions to attach functionality to 'resources' in the environment model. A resource represents everything that can be included in this model, e.g. users who interact with the surroundings, devices that offer computing power, storage and input modalities, etc. Domain-specific ontologies that introduce new concepts such as light resources are merged with an upper environment ontology at runtime. The environment ontology defines 'sensors' and 'actions' to interact with these resources:

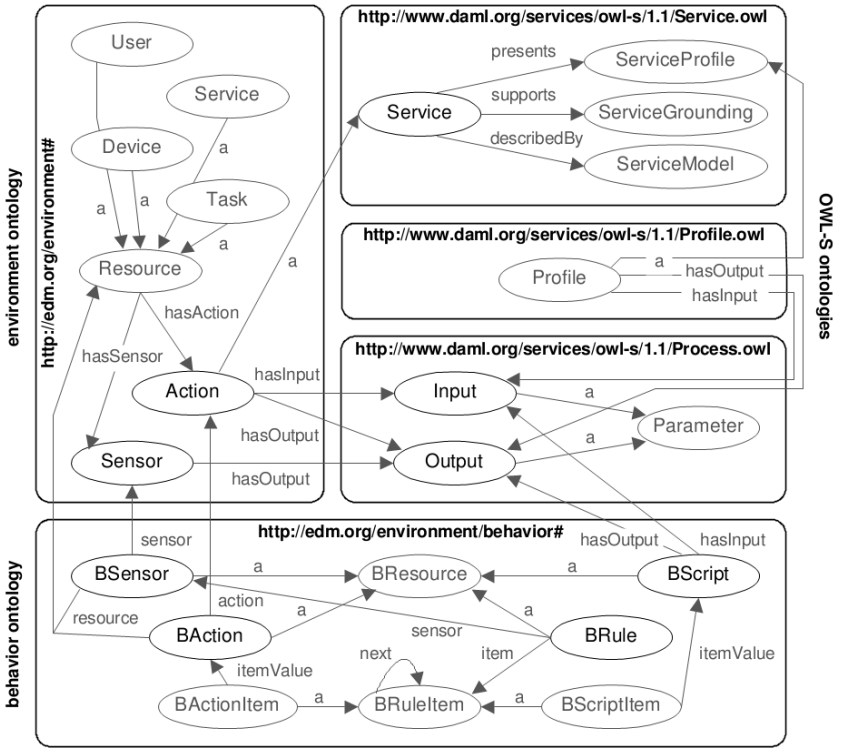


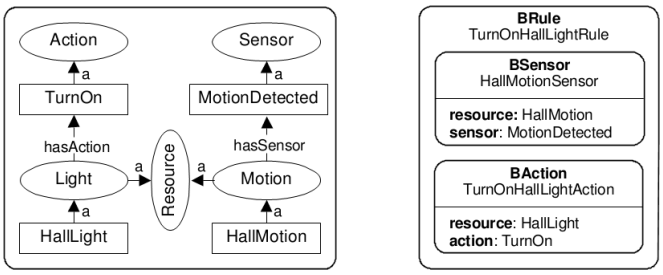
Fig. 1. The environment and its behavior are described using ontologies.

- **Sensor**: A Sensor publishes context events that occur in a resource in the environment. In other words, a sensor provides remote context events to interested resources in the environment.
- **Action**: An Action has a one-to-one correspondence with an OWL-S service. We introduce the term 'Action' to differentiate between the definition of a service in the environment model and an OWL-S service. For example, 'DoSearch' and 'DoSpellingSuggestion' are two OWL-S services (i.e. actions) that belong to the service 'GoogleService'.

While sensors and actions allow interaction with resources, their output data often lacks context w.r.t. other resources. Consider for example a 'LocationService' that triggers a sensor each time the location of a tracked object changes. This sensor outputs plain coordinates which have few meaning to other resources. Hence we introduce context-aware sensors and actions in the behavior model, such as a 'NearWhiteboard' sensor that is triggered when a tracked object approaches the whiteboard in a room. This sensor interprets coordinates produced by the location sensor and thus adds a concrete meaning to this data. Semantically enriched sensors and actions act as building blocks to compose Event-Condition-Action (ECA) rules and are defined in the behavior ontology. We distinguish the following concepts in this ontology:

– **BSensor**: A BSensor represents a resource/sensor pair, optionally linked with a script that acts as a filter on the base sensor: only if certain conditions are met, the behavior sensor is triggered (e.g. a script could check if the sensor's output parameters match certain values).
– **BAction**: A BAction represents a resource/action pair.
– **BScript**: A BScript encapsulates script code (e.g. JavaScript) that is dynamically interpreted. Scripts also have input and output parameters that are read and set using dedicated variables (`$in`, `$out`).
– **BRule**: A BRule relates a BSensor with a chain of actions and scripts. When the sensor is triggered, this chain is executed. The output of either sensor, action or script can be passed as input to subsequent actions/scripts in the chain.

Consider for example the behavior rule listed in figure 2 which will automatically turn on the light in the hall when motion is detected at this place.



(a) Environment model (part). (b) Behavior rule.

Fig. 2. A behavior rule (b) connects independent resources in the environment model (a): a light is automatically switched on when motion is sensed.

4 Orchestrating Resources

To achieve a desired behavior in an ambient space, the objects in this space need to adapt to a (new) context of use. Hence different software services that were not initially designed to collaborate, should be orchestrated and become aware of each other. Our orchestration approach is based on semantic matching of Web services capabilities [6]. Semantic matching is a key element to establish late binding and a service-oriented architecture (SOA) has proven to be useful for this purpose in highly dynamic pervasive environments [4].

In *ReWiRe* the behavior of the environment is described by a set of rules $R_0, \ldots, R_n$ that all contain a reference to a behavior sensor S and a set of executable items $I_0, \ldots, I_n$ with I_i either a behavior action or a behavior script. When a rule's sensor is triggered, its behavior items are executed one by one in the specified order, consuming and producing data. The inputs and outputs of behavior resources are described by OWL-S

parameters in a similar way as the parameters of a semantic Web service are described. OWL classes and OWL's built-in XML schema types (xsd:string, xsd:integer, ...) describe a parameter's datatype. Parameter $p1$ matches parameter $p2$ if both parameter types are equivalent or if the parameter type of $p1$ subsumes the parameter type of $p2$. In other words, the parameter type of $p2$ is either an exact match of the parameter type of $p1$ or it is a 'super class' (in terms of OWL class equivalence) of $p1$. A service is only invoked if all input parameters that have no (default) value are set. Otherwise a service call will usually lead to a malfunction.

5 Collaborative Paint Application

A proof-of-concept application built using our framework aims to improve the experience of painting in the digital world. We try to mimic a real-world multi-user painting setup by supporting heterogeneous federations of devices. For example, figure 3 shows a user painting on a canvas projected on a touch-sensitive whiteboard, using a PDA to select and mix colors. The whiteboard represents the painter's easel while the PDA acts as his mobile color palette. Users can use their own devices or make use of the resources already present in the environment (e.g. tabletop device, tablet interface, ...) to participate in the painting process.

Fig. 3. A user is creating a painting on the whiteboard using his PDA as a mobile palette, whilst another user is painting using a tablet interface.

While this application could be realized using traditional development approaches, this would involve a lot of ad-hoc coding. Using our framework, one has to provide a functional core ('PaintService') along with user interface components leveraging this functionality and a set of behavior rules to orchestrate paint resources in the environment. Note that legacy paint applications can be (re)used as a functional core in our framework and benefit from *ReWiRe*'s distribution capabilities. By differentiating between an engineering and a modeling step, we promote code reuse whilst being able to alter the behavior of resources at runtime. In an exemplary scenario, we linked a sensor that is triggered when a new device enters the environment (discovered by the middleware) with a distribution request for the paint canvas interface, provided that the target-device is capable of running this component. Besides, we installed an RFID tag near the whiteboard that triggers a 'PaletteTagScanned' sensor when it is scanned

(through a 'RFIDService'). A behavior rule that is invoked when this sensor provides new data, executes an action that migrates a user interface for the color palette to the device that scanned the tag (e.g. a PDA). This allows a user to move his PDA (equipped with an RFID reader) near the RFID tag to have a palette distributed to it.

6 Conclusion

The development of pervasive applications remains difficult due to several reasons, e.g. lack of engineering approaches, new middleware requirements and situation-aware human-computer interaction. In this paper we presented a model-driven approach to coordinate the behavior of a pervasive application.

Our future work includes improving the behavior model and its tool support. While the behavior rules are currently composed as a linear list of orchestrated actions/scripts, more complex behavior rules require a more advanced structure, e.g. to model conditional tests on output values. A remaining challenge is to integrate this system-oriented orchestration with a more user-oriented task modeling approach.

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. Charlton Barreto et al. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, 2007.
2. José L. Encarnação and Thomas Kirste. Ambient Intelligence: Towards Smart Appliance Ensembles. In *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments*, pages 261–270, 2005.
3. Robert Grimm. One.world: Experiences with a Pervasive Computing Architecture. *IEEE Pervasive Computing*, 03(3):22–30, 2004.
4. Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. *J. Syst. Softw.*, 80(12), 2007.
5. Javier Muñoz, Vicente Pelechano, and Carlos Cetina. Software Engineering for Pervasive Systems. Applying Models, Frameworks and Transformations. In *International Workshop on Software Engineering for Pervasive Services (SEPS '06)*, 2006.
6. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic Matching of Web Services Capabilities. In *Proc. of the 1st International Semantic Web Conference on The Semantic Web (ISWC '02)*, pages 333–347, 2002.
7. Geert Vanderhulst, Kris Luyten, and Karin Coninx. Middleware for Ubiquitous Service-Oriented Spaces on the Web. In *Proc. of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW '07)*, pages 1001–1006, 2007.
8. Geert Vanderhulst, Kris Luyten, and Karin Coninx. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring. In *Proc. of the 4th IET International Conference on Intelligent Environments (IE '08)*, 2008. to appear.