Perceptive Services Composition using semantic language and distributed knowledge

Rémi Emonet, Dominique Vaufreydaz

Perceptive Services Composition using semantic language and distributed knowledge

Rémi Emonet, Dominique Vaufreydaz.

PRIMA - INRIA Rhône-Alpes, Zirst Montbonnot, 655 avenue de l'Europe, 38334 Saint Ismier cedex, France
{Remi.Emonet, Dominique.Vaufreydaz}@inrialpes.fr

Abstract. Building applications composing perceptive services in a pervasive environment can lead to an inextricable problem: they were built by several people, using different programming languages and multiple conventions and protocols. Moreover, services can be volatile, so appear or disappear during running time of the application. This paper proposes the use of a dedicated human-readable semantic language to describe perceptive services. After converting this description into a more common language, one can recruit services using inference engines to build complex applications. In order to increase robustness of the whole system, descriptions of services are distributed over the network using a crosslanguage crossplateform open-source middleware of our own called OMiSCID.

Keywords: semantic language, distributed description, services composition, pervasive environment, dynamic availability.

1 Problematic

In pervasive environment, one may find many different devices or sensors (PDAs, beamers, cameras, microphones, etc.) and perceptual software (visual tracking systems, speech recognition systems, acoustic localization ...) [1]. Our aim is to construct automatically and dynamically application using several components or services available over the network. Pervasive environments have some specific constraints that are making this task complex:

• dynamic availability of services: services may appear and disappear anytime while applications are running.

• heterogeneous infrastructure: one must cope with many devices and computers with different configurations (operating system, programming language, etc.), each with specific sensing and actuating capabilities.

For these reasons, designing applications in pervasive environments necessitate some methods that facilitate awareness and dynamic integration of newly available devices and services. User needs are also dynamic: expressing it and converting it to computing service requirements is also mandatory.

Some other research projects are dealing with this problem. The CHIL project [2] is aiming at providing a programming language independent API to give access to a

knowledge base server. This server is then used as a service registry in pervasive environments. Their approach is centralized and chooses to represent all knowledge in the base, even some very low level one (camera position, room shape, inter-component message formats ...). As the knowledge is really detailed and in a centralized base, the inference process can be slowed down.

The Service Binder from the Gravity project proposes to simplify the design of applications where dynamic availability is present [3,4]. One little drawback of their proposal is that it is based on OSGi and so is limited to the Java platform. The Gravity project also lacks some expressive power in functionality description: they are using plain Java interface name as functionality description and are currently not able to do some inference and reasoning about provided functionalities. Moreover, their approach has no distinction between service functionality and access protocol.

2 Proposed Approach

Our approach is based on two concepts: the use of a service oriented architecture (SOA) to handle the dynamic availability of services over the network and the intensive use of semantic description of the services functionalities.

The SOA vision is to build systems as an assembly of loosely coupled software services. All services are self-content and their descriptions can be found using services discovery mechanisms. We propose to describe the functionalities provided by a service at a semantic level: we will express what a service does but not how it does it. The basic idea is to construct dynamically a knowledge base (ontology) that contains the functionalities provided by the running services. The functionalities ontology will be the meeting point between different services and between services and users. In our proposal, each software or hardware element is packaged as a service: a person tracker, a microphone or a video camera, etc. Each service is described in term of functionalities using words chosen by the service designer. The matching between terms is done by adding some correspondences in the entire ontology: it could be done “manually” or automatically using ontology alignment methods. Compositions or combinations of functionalities can also be expressed to lead to another functionality.

A functionality corresponds to a functional concept, not its implementation nor its protocol. Two services can expose a same functionality while using different protocols. This decoupling between protocol and functionality is one of the keys of the approach: using correspondences between ontologies we can infer that a given service *S* provides a given functionality *F*, it is however unlikely that the protocol used by the designer of the service *S* matches exactly the protocol used by the original designer of the functionality *F*. In such a case, it will then be possible to have the system say something like: “If I can find a protocol adapter between *protocol1* and *protocol2*, I can use this service”. The separation between protocol and functionality also contributes to the tractability of inferences.

2.1. Approach Details

This section gives some details about the description of functionalities. Inspired by the semantic web languages such as Resource Description Framework (RDF) or Web Ontology Language (OWL), we propose a textual language that can be easily written by any user. This language is compiled into RDF to leverage the existing tools such as inference engines.

2.1.1 Functionality Description Meta-model. We cannot ask the user to know a language such as OWL or to have the mastering of OWL editors such as Protégé [5]: the language must be simple and easy to write and understand. Our functionalities metamodel is architectured around three main concepts:

• resources: resources can be seen as identifiers for entities such as services and physical or virtual objects relevant to the applications. I.e. a camera service is a resource but it could expose some related virtual objects such as the 3D referential attached to the camera in the space and the 2D referential of the camera image pane.

• facets: facets represent the functionalities. One can define facet classes that are some kind of functionalities and can associate some facet to a resource. A given resource can have several facets and can even expose multiple facets of the same facet class. This would be improperly represented using OWL classes and properties.

• properties: facet properties represent functionalities attributes. Properties can be defined at the class level, giving a name for the property and an indicative type. One can say the resource representing the camera service has an “ImageSource” facet with its ”format” property set to “RGB888”.

2.1.2 Language Syntax Overview. We propose to use existing knowledge representation languages but only under cover. This way we can leverage existing tools while providing a language that is adequate for the task. This kind of reasoning is behind the philosophy of Domain Specific Language (DSL) approaches. The first part of our language enables us to declare resources, facets and properties as seen below. Here is a simple example:

```
namespace is
 http://cmppc07/eg#
this isa Camera
 with refImage = this|image
 with refCamera = this|home
```

```
a ExternalCameraCalibration
 having camera = ?cam
 having object = ?obj
isa ChangeOfReferenceFrame
 with from = ?obj
 with to = ?cam.refCamera
```

In the first column, the first 2 lines define the default namespace to be used. The usage of namespaces used to avoid name clashes. Given the default namespace, the lines 3 to 5 are associating a *Camera* facet to the *this* resource. *this* represents the resource associated to the service exposing the knowledge. Lines 4 and 5 are in charge of affecting facet properties. Correspondences between different functionalities can also be expressed as we can see in the second column. This snippet says that any resource having an *ExternalCameraCalibration* facet *f* will also have a *ChangeOfReferenceFrame* facet with a *from* property set to the value of the object

property of *f* and, a *to* property set to the resource pointed by the *refCamera* property of the camera property of *f*.

2.1.3 Integration in a Service Oriented Middleware. The functionality description is constructed as a layer over a service oriented middleware. We propose to rely on an opensource middleware designed for crosslanguage crossplatform pervasive applications: OMISCID [6]. The grounding in the middleware is done by adding to each functionality facet an automatically propagated property named *grounding*. This property must be set when registering a service against a functionality and is automatically propagated during functionality inference. The grounding form depends on the concepts present in the underlying middleware: in our case, grounding can be either a reference to a service variable or a reference to a service connector.

3 Current status and future work

This work is under progress. We are currently finishing the first step of the implementation. First non trivial examples have been designed successfully, for example, the composition of several reference frame transformation services. Indeed, each perceptive service is working using its own frame of reference. Composing dynamically transformations can lead to use any corrected output as an input for another service. The next application planned to be developed using this approach is a multimodal person tracker dynamically composed with all possible sources of localization and identity information: video, audio, Wifi, Bluetooth, RF-ID, etc.

4 References

1. Macho D., Padrell J., Abad A., Nadeu C., Hernando J., McDonough J., Wolfel M., Klee U., Omologo M., Brutti A., Svaizer P., Potamianos G., Chu S.M.: Automatic Speech Activity Detection, Source Localization, and Speech Recognition on the Chil Seminar Corpus. In IEEE International Conference on Multimedia and Expo, Amsterdam, (January 2005).
2. Alexander Paar, Jürgen Reuter, John Soldatos, Kostas Stamatis, and Laz aros Polymenakos: A formally specified ontology management API as a registry for ubiquitous computing systems. In 3rd IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI) 2006, Athens, Greece, Athens - Greece, (June 2006).
3. Humberto Cervantes and Richard S. Hall: Autonomous adaptation to dynamic availability using a service-oriented component model. In International Conference in Software Engineering, 614–623, Edinburgh – Scotland, (May 2004).
4. Cervantes Humberto and Hall Richard: Automating Service Dependency Management in a Service-Oriented Component Model. In ICSE CBSE6 Workshop, Portland - USA (May 2003).
5. Standford University of Medicine: The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/.
6. Emonet, R. and Vaufreydaz, D. and Reignier, P. and Letessier, J.: O3MiSCID: an Object Oriented Opensource Middleware for Service Connection, Introspection and Discovery. In 1st IEEE International SIPE Workshop, Lyon – France, (June 2006).