Application Ontology Manager for Hydra

Ján Hreňo[1] , Peter Kostelník[1], Martin Sarnovský[2]

[1] Ekonomická fakulta TU Košice, Boženy Němcovej 32, 04200 Košice, Slovakia
[2] Fakulta elektrotechniky a informatiky, Letná 9, 04200 Košice, Slovakia
{Jan.Hreno, Peter.Kostelnik, Martin.Sarnovsky}@tuke.sk

Abstrakt. The article describes main functionalities of an Ontology Manager software developed for the Hydra project. The Ontology Manager is a service based set of tools used to semantically enhance a middleware for networked embedded system of devices. The core services of the Ontology Manager are based on Open RDF Sesame. User interface is developed as an Eclipse plugin.

Keywords: Model driven development, Service driven architecture, Semantic devices.

1 Introduction

Application Ontology Manager is a set of tools designed to run in the application part of the Hydra project Service driven architecture [1]. It is based on the Sesame - the open source Java framework for storage and querying of RDF data. User interface using Eclipse plugin architecture was developed together with the service. In this article we describe core functionalities of the Ontology Manager. Integral part of the ontology manager is a set of basic ontologies describing devices [2].

2 Model driven device editor

Most of sophisticated applications working with various devices would require searching of devices satisfying several requirements. Semantic descriptions of the device models created in the device enabling process represent only the basic information necessary for the device functionality. This information can be further extended using the ontology administration tools included in the Integrated Development Environment (IDE), which serves as the ontology and annotation editor. Based on the most frequent application functionality and internal requirements [3], the ontology has been extended with models of hardware, events provided by device, energy profiles, and quality of service properties or security properties. Device ontology was also extended by properties used to annotate the extended information to device models. Using extended semantic descriptions, the devices and services have the full semantic support and are accessible through query interfaces and whole process of knowledge extension is guided by the ontology. Two ontologies supporting the annotation process were created.

Static taxonomy model - the container for all information, which serves as the taxonomies containing instances, which can be annotated to the devices, these static instances, when annotated to device parts, can be further used as values in searches.

Annotation property model - to be able to decide, which behaviour of IDE should be used in what context, the owl:DatatypeProperty and owl:ObjectProperty classes was extended with the custom classes specifying the annotation or form properties (**Fig. 1**). The properties extending owl:ObjectProperty are of two types:

- FormProperty ensures that IDE will automatically generate the form including all literal properties of the class assigned as the property range.
- For AnnotationProperty, the IDE generates the tree browser with the root class assigned as the property range.

The *owl:DatatypeProperty* is extended by the class *FormFieldProperty*, which was created in order to define some special literals, which can be edited.

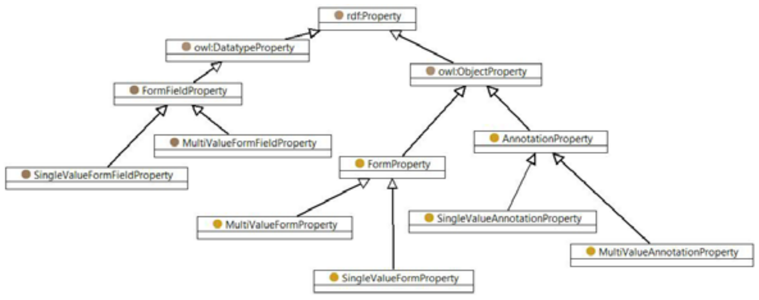

Fig. 1. The annotation property model.

FormFieldProperty enables to select, which properties can be edited. The difference from *FormProperty* is, that *FormProperty* expect all literals to be editable. *FormFieldProperty* allows specifying, which particular properties can be edited. All properties may be single or multi value, so IDE will add or replace the newly added property value.

3 Querying with expectations and requirements

Application developers need to query ontology for various devices or their services actually presented in runtime e.g. to retrieve specified device or service properties, which should be used for further computations in the application logics. For purposes of model driven query building and query evaluation, the simple query language was developed. Each query is composed of comma-separated clauses. Each clause contains the slash-separated sequence of properties starting from device/service instance and defines the target value – the last item in the sequence, which serves as the filter for results. The target value of the clause may be of two types:

1. Property without defined value: serves as the existence filter – device has the

property attached, does not matter, what is the value of the property.

2. Property with defined value: serves as the value filter – device has the property set to concrete value. The value may be the value of literal property or the instance from static taxonomy.

The requirements structure is similar; the only difference is that requirements must not contain the property values definitions. The reason is, that requirements may only specify what properties must be retrieved (not what should be the property value). Queries are translated to SPARQL queries and executed. Then, each retrieved result is queried for each requirement and the value of requirement is attached.

4 Semantic devices

Each physical device provides a set of specific services, which can be directly used by the application developer. The concept of semantic devices brings the idea of specifying the application specific behaviour achieved as the composition of several devices organized into complex units. Semantic devices can include physical, but also other semantic devices. Each semantic device is defined by a set of semantic services. Each semantic service is composed by a set of requirements in terms of preconditions. There are two kinds of preconditions. Static preconditions represent the list of persistent identifiers of concrete devices, which will appear in the application and will be used in semantic service execution in runtime. Dynamic preconditions used in the runtime to generate the candidate devices matching the requirements specified by the query. Developer has to define and implement semantic device services using the DDK (Device Development Kit) tool. In this case, the preconditions defined for each service are used to automatically generate the class of proxy implementation using the configuration attached to the semantic models of used devices. At the runtime, each time a new device joins the application, the semantic devices are rediscovered and the required devices satisfying defined preconditions are automatically tied with the semantic devices.

5 Application context awareness

We have developed ontologies containing the application model, which can be instantiated in the IDE. User can select the required application type, create the instance and create the application entities using the model driven application editor. A device is attached to an application entity using the PID (Persistent ID) identifier, as in application modelling phase it is required to know exactly which devices are in relation with the particular application entity. Application entities are related to the device PIDs using instances of specific class named *rule:DevicePID* having only one string property *rule:PID*. This information represents, which devices are related to application entities using PIDs. For querying purposes, when searching for required devices, also inverse relation is needed. This information has to be inferred in the runtime; therefore a resolver was developed responsible for generating this information. We created an ontology defining the *rule:generatesInverseProperty*, which can be attached to any *owl:ObjectProperty* instance. This property defines which property has to be generated as the inverse property. It is expected, that application property, such as:

```
Person ownsDevice DevicePID
```

Generate inverse properties, such as:

```
ownsDevice generatesInverseProperty ownedBy.
```

When new device gets the PID information, all relations to application entities are identified and attached to the new device run-time instance. Then it is possible to ask queries containing application context information:

```
device:hasHardware/hardware:hasDisplay,
application:locatedIn/application:name;"MyLivingRoom"^^xs
d:string
```

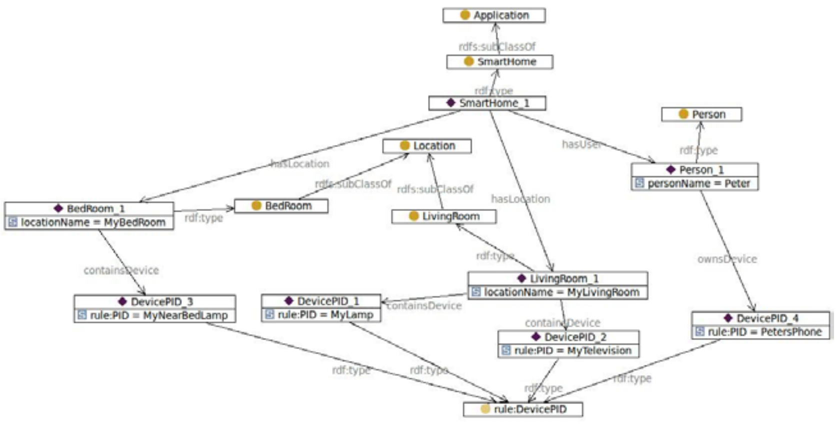

Fig. 2. Example of application.

Acknowledgments. The work was supported within the FP6 IST-2005-034891 HYDRA (50%), VEGA 1/0042/10 (25%) project and is the result of the project implementation: Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF (25%).

6 References

1. Eisenhauer, M., Prause, Ch., Schneider, A., Scholten, M., Zimmermann, A.: Initial architectural design specification, Public deliverable D3.4, Hydra project, http://www.hydramiddleware.eu
2. Sarnovský, M., Kostelník, P., Hreňo, J., Butka, P.: Device Description in HYDRA Middleware. In Proceedings of the 2nd Workshop on Intelligent and Knowledge oriented Technologies 2007, WIKT 2007, Košice, Slovakia, November 2007 (published 2008), pp.71-74, ISBN 978-80-89284-10-8
3. Zimmermann, A., Jahn, M.: Updated Systems Requirements Report, Public deliverable D2.7, Hydra project, http://www.hydramiddleware.eu