Template-based ontology population for Smart Environments configuration

Template-based ontology population for Smart Environments configuration

Sebastián Aced López, Dario Bonino, and Fulvio Corno

Dipartimento di Automatica ed Informatica.
Politecnico di Torino
Torino, Italy
{sebastian.acedlopez, dario.bonino, fulvio.corno}@polito.it

Abstract. Smart Environments is one of several domains in which Semantic Web technologies are applied nowadays. Ontologies, in particular, are used as core modeling languages for representing devices, systems and environments. Developing such ontologies, that typically involve several device descriptions (individuals) and related information, i.e., individuals of classes contributing to the device model, is often done by a manual, time consuming, and error-prone approach.
This paper presents a template based approach, which increases accuracy, ease of use, and time-effectiveness of the ontology population process by reducing the amount of user-given information of about an order of magnitude, with respect to the fully manual approach. User-required information only pertains device features (e.g., name, location, etc.) and never implies knowledge of Semantic Web technologies, thus enabling end-user configuration of smart homes and buildings. Experimental results with a prototypical implementation confirm the viability of the approach on a real-world use case.

Keywords: Ontology population, OWL templates, Instantiation, Smart Environments, Configuration

1 Introduction

Semantic Web technologies have allowed cities, workplaces and homes to become *smarter* over the years by supporting explicit context representation, expressive context querying, and flexible context reasoning [1]. Ontologies can be used to model agents, contexts and behaviors, while SPARQL querying helps to easily retrieve data from them, and reasoners can use these data to infer relations and describe complex scenarios, enriching the model capabilities.

Modeling Smart Environments (SmEs) by means of ontologies enables the creation of a layer of abstraction, in which reality is represented in terms of classes, properties and instances, and allows developers to work with conceptualizations of real entities instead of dealing with low-level representations of them.

Adding new instances into the ontology is known as populating (or instantiating) the ontology, and it is an essential part of almost every ontology based application, especially in the SmE field, because the only way of configuring a specific SmE, such as *my-particular-room*, is by creating specific instances, such as *my-particular-floor* or *my-particular-lamp*. However, as it will be explained later, most of the available methods to populate ontologies are error prone and time consuming.

This paper proposes an OWL-template based approach that allows accurate, fast and semi-automatic population of ontologies that can be used in general, but specifically helps in the configuration of SmEs.

2 Related Works

Even if many ontologies have been created and are actually being used in a variety of fields such as SmE modeling, better methods to populate them are still object of research due to the challenging nature of the task. However, like for the content and the structure, population techniques (listed in [2]) change a lot from one type of ontology to another. The approaches for ontology population found in literature can be divided into two types depending on how the information to generate the individuals is gathered: either through *Information extraction* or *User-given* data.

Information extraction approaches assume that the needed information is already available somewhere, it could be on text documents, Internet pages, databases etc., so they extract and process it to identify pieces that fit as instances of some reference ontology[1]. This is the case of the Artequakt system [3], which automatically extracts biographical information from the web to instantiate a reference ontology and generate artists biographies.

Approaches based on user-given data, on the converse, gather the information needed for instantiation directly from the user, as proposed in [4] and in [5]. This is the case in SmE context, since the needed information is specific for each particular environment configuration and cannot be mined elsewhere. However, few of the approaches that gather information from the user, focus on the generation of new individuals. Instead, they aim to support the design and creation of new concept classes. See [6].

On the other hand, this paper presents an approach to enhance the creation process of new individuals from already defined classes, exploiting different techniques (inherited from the software engineering) based on automatic code generation and template modeling, which have already been proven appropriate for working with ontologies [7].

[1] The term *Reference ontology* refers to an ontology containing the classes from which the instances are going to be created

3 Background

In this paper, examples and experimentation exploit a publicly available ontology for smart environments named DogOnt[2] [8]. It is organized in 5 main hierarchies of concepts:

- Building Environment (BE): The concepts below this hierarchy are used to describe the architectural spaces of built environments (Garage, Flat, Room, etc.)
- Building Thing (BT): The concepts below this hierarchy are used to describe the controllable (i.e., devices) and uncontrollable objects (e.g., furniture) of a given environment.
- State: The concepts below this hierarchy are used to describe the working configurations (observable status) that controllable BT objects can assume.
- Functionality: The concepts below this hierarchy are used to describe the controllable BT objects capabilities.
- Network Component: The concepts below this hierarchy are used to describe the technology-specific information needed for describing real-world devices.

Users configure specific SmEs by instantiating BE and BT concepts and by connecting them according to the ontology-defined domain semantics.

4 Problem Statement

The population process, as stated previously, is time-consuming and error-prone [2], mainly because of two challenging tasks it encompasses: *instance properties determination* and *implicitly derived instantiation*. Figure 1 presents a DogOnt fragment example to help to illustrate those concepts.

Instance properties determination

When a class is instantiated, it is not easy to identify which properties must be included in the new instance description, for it to be valid and logically consistent, and which properties can, instead, be omitted. The former type of properties, namely *Mandatory properties* need to be created for the model to be valid (under a logic and semantics standpoint) whereas the latter properties (*Optional properties*) are not strictly required from a formal standpoint but might be crucial for the model to be usable in the real world (e.g., the location of a given device).

Classifying the properties of a class as mandatory or optional, helps to determine which of them *must* and which of them *may* be part of a new instance description. More in detail:

[2] `http://elite.polito.it/ontologies/dogont`

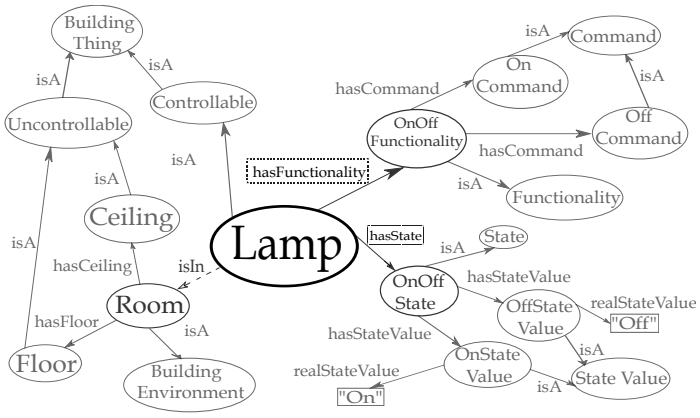

Fig. 1. Ontology fragment showing a DogOnt Lamp class.

- *Mandatory properties*: A property is mandatory if, in the class definition of a given individual, its Cardinality Constraint is at least one. In other words, the class mandatory properties are those that an individual must include in its description, to be considered a valid instance of such class. Figure 1 shows the mandatory properties of Lamp class: *hasState* and *hasFunctionality*.

- *Optional properties*: Optional properties may be included in an individual description but are not required for it to be a valid instance of any class. The user decides whether to include optional properties in a particular instance description. In the example of Figure 1, the optional property *isIn* is represented by a dashed arrow.

Implicitly derived instantiation

Assuming that the properties of a new individual have been established somehow and it has been determined which of them can be automatically generated, the *implicitly derived instantiation* problem has to be solved. Implicitly derived instantiation refers to the fact that sometimes the *explicit* creation of a new instance, *implicitly* leads to the generation of additional individuals. This happens when an explicitly created instance is described by *object properties*, because they describe instances by associating them with other URI resources (classes or individuals), that also need to be created in order to produce valid associations and valid descriptions. This is illustrated better by Figure 1: when an instance is created explicitly, for example a Lamp instance, and it has an *object property*, such as *hasState*, implicitly another instance has to be created, in this case a new *OnOffState* individual.

Implicitly derived instantiation becomes a problem in large and highly interrelated ontologies because the creation of one instance can start a chain reaction of instantiations, making of the population task a long and complex process.

5 Proposed solution

In order to improve the ontology population process, and in particular to tackle the inherent complexity of the tasks discussed in the previous section, a two stage approach is proposed which aims at reducing the cardinality of information needed from the user and at hiding ontology formalisms by only requiring actually needed data (e.g., device types and names instead of device instance definitions). The two stages exploit different techniques based on automatic code generation and template modeling, respectively.

The information cardinality reduction is based on the identification of which objects/properties require external information to be created. For example, in DogOnt, to describe a room instance, the Room class property *hasFloor* must be filled with a *Floor* individual manually defined by the user (two rooms can share the same floor). Instead, other classes, e.g., OnOffStateValue, are completely specified and individuals creation can be automatically carried. Moreover, if a given class has mandatory properties that refer to fully specified classes, individuals of such a class can also be generated automatically, by implementing a suitable recursive mechanism. In this way the amount of instances that need manual creation can be greatly reduced, depending on the ontology branching and the adopted modeling approach: highly specified models experience greater improvements with respect to loosely specified ones. SmE ontologies typically fall in the former typology.

Formalism hiding, instead, is obtained through a template-based mechanism which models repetitive syntactical structures in OWL, (e.g., type definitions) and replaces information that must be given by users with suitable placeholders to be filled at configuration time. In such a way, the actual data that users are required to fill decreases (contributing to an additional cardinality reduction) and ontology constructs are completely hidden and exposed as free parameters to be filled.

In such a way, this approach helps to populate ontologies more quickly (due to cardinality reduction) and more accurately (templates are validated once and ensure syntactical correctness) than in the current state of the art. More in detail, the overall approach is divided in the following three phases, and is illustrated in Figure 2:

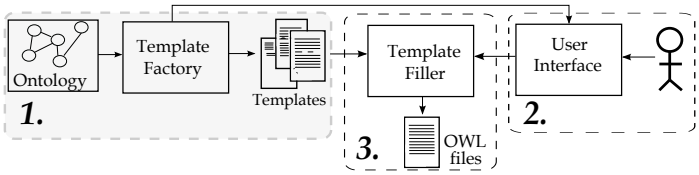

Fig. 2. Block diagram of the proposed approach.

5.1 Template generation

The template generation phase, which is executed offline only once (unless the reference ontology itself changes) by the Template Factory, aims to create a template for each target class.

Templates are divided into two parts: a *main block* and a *secondary block*. The main block contains the description of the target class instance (*the main instance*) for whom the template is created. The secondary block contains the descriptions of all the *implicitly* derived instances (*secondary instances*).

In order to obtain a template, two steps have to be followed: *reference ontology exploration*, to determine and classify the properties used in the instance descriptions and *the template writing* in which those instance descriptions are structured and written in a template body.

Reference ontology exploration To create a template of a target class, the reference ontology is recursively explored to find all the classes and properties "connected" to such a class, identifying which information shall be filled by the user and which one can be generated by an unsupervised process. Such an exploration is based on SPARQL querying. The query process retrieves the properties that can be used to describe target classes , and the information needed to classify those properties as mandatory or optional.

Exploration queries should be designed to exploit the specific characteristics of each reference ontology. As an example, in the DogOnt ontology used throughout this paper for illustrating the proposed approach, mandatory properties do not have free parameters, whereas the optional object properties always have them. Consequently, its particular exploration query only returns the necessary information to classify properties in the following groups:

- *Mandatory object properties*
- *Mandatory datatype properties*
- *Optional object properties*

Template Writing Once the template information is gathered and the class properties properly classified, the template can be written. As stated before a template is structured in blocks (main and secondary), each one describing an instance. In general, such blocks contain:

- *Namespace* and *main instance name* placeholders.
- Static OWL statements: Corresponding to the mandatory properties with no free parameters.
- Parametrized OWL statements: Corresponding to the mandatory properties with free parameters, i.e., placeholders.
- Optional Statements: Corresponding to IF statements enclosing the optional properties.
- A `rdf:type` property stating the class of the instance described in the block.

5.2 User input Information

After the template generation phase, information from the user is required to resolve the Optional Statements and to assign proper values to the template placeholders. Such information could be gathered through any user interaction mechanisms, e.g. through a graphical user interface (GUI). Figure 2 shows an arrow that goes from the Template Factory to the UI block, to indicate that the former must supply information, such as the placeholder labels, to configure the latter.

5.3 Ontology consolidation

The last phase of the overall process, consists in merging the template-encoded information with the data entered by the user. The latter, in particular, replaces all the template placeholders providing a valid and fully consistent OWL instance definition. The steps to consolidate it are very simple:

1. Replace all the *namespace* and *main instance name* placeholders.
2. Resolve the IF statements (if present) to determine which optional properties to include in the instance description.
3. Replace the rest of the template placeholders.
4. Write the output OWL file.

6 Experimental Results

The template based approach exposed along this document has been initially tested in a real world case of ontology-based smart environment configuration. More precisely, experiments were carried to populate a specific SmE: the *Simple Home* [9] which is based on the DogOnt ontology (1835 classes) and describes a flat with several (114) domotic devices modeled by 1408 concept instances.

The prototype tool developed for experimentation uses the Jena framework to manage the ontologies and the ARQ engine to issue SPARQL queries at the template generation phase. User information is collected through a dynamically generated JavaFX application which also drives the consolidation process. User given information, is mapped to a set of automatically generated Java Beans accompanying each template and providing the additional information to check the correctness of filled data, e.g., the allowed placeholder filler classes. This set of beans, is then used in the ontology consolidation phase by a Velocity Template Engine to fill the templates.

Experimental results confirm that by following the approach presented in this document, the effort and time that users spent manually populating such a large ontology was significantly reduced by using templates: the entire population process took, in fact, less than one day (vs. over a week in the fully manual case) and only required to fill free parameters for the 114 devices (roughly 300 parameters) instead of manually describing the 1408 required instances (amounting to about 7000 triples), with a cardinality reduction of over one order of magnitude.

7 Conclusions

This paper discussed a general template-based approach for effective ontology population, with a particular focus on the smart environment domain. While the general problem of implicitly derived instantiation affecting current tools (e.g., general editors as Protégé) cannot be fully solved as ontology modeling implies the creation of related instances, with a complexity that depends on the ontology branching factor, template-based solutions, as the one presented, allow to greatly reduce the cardinality of user-given information and, at the same time hide ontology-specific formalisms from the end users.

Preliminary experimental results, confirmed the viability of the proposed solution with over an order of magnitude reduction in the cardinality of information required to users: about 300 parameters vs over 7000 triples.

Future works will involve extensive experimentation with users, by exploiting different ontology models for SmEs and a thoroughly study of user interfaces for filling free parameters.

References

1. Wang, X., Dong, J.S., Chin, C., Hettiarachchi, S., Zhang, D.: Semantic space: An infrastructure for smart spaces. IEEE Pervasive Computing **3**(3) (2004) 32–39
2. Maleshkova, M., Martínez, I.: Ontology instantiation state of the art report. Technical report (2008)
3. Alani, H., Kim, S., Millard, D.E., Weal, M.J., Hall, W., Lewis, P.H., Shadbolt, N.: Using protege for automatic ontology instantiation. In: 7th International Protégé Conference. (2004) Event Dates: July 6-9.
4. Kawamoto, K., Kitamura, Y., Tijerino, Y.: Kawawiki: A semantic wiki based on rdf templates. In: WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology, Washington, DC, USA, IEEE Computer Society (2006) 425–432
5. Doherty, L., Kumar, V., Winne, P.: Assisted ontology instantiation: a learningkit perspective. In: Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on. (2007) 265–267
6. Jupp, S., Horridge, M., Iannone, L., Klein, J., Owen, S., Schanstra, J., Stevens, R., Wolstencroft, K.: Populous: A tool for populating templates for owl ontologies. In Burger, A., Marshall, M.S., 0001, P.R., Paschke, A., Splendiani, A., eds.: SWAT4LS. Volume 698 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
7. Parreiras, F., Gröner, G., Walter, T., Staab, S.: A model-driven approach for using templates in owl ontologies. In Cimiano, P., Pinto, H., eds.: Knowledge Engineering and Management by the Masses. Volume 6317 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 350–359
8. Bonino, D., Corno, F.: Dogont - ontology modeling for intelligent domotic environments. In Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K., eds.: International Semantic Web Conference. Volume 5318 of Lecture Notes in Computer Science., Springer (2008) 790–803
9. Bonino, D., Corno, F.: Dogsim: A state chart simulator for domotic environments. In: PerCom Workshops, IEEE (2010) 208–213