Data Consistency Management in an Open Smart Home Management Platform

Jie Song, Silvia Calatrava Sierra, Jaime Caffarel Rodríguez, Jorge Martín Perandones,
Guillermo del Campo Jiménez, Jorge Olloqui Buján, Rocío Martínez García and Asunción Santamaría Galdón

***Abstract*—In this paper, the authors introduce a novel mechanism for data management in a middleware for smart home control, where a relational database and semantic ontology storage are used at the same time in a Data Warehouse. An annotation system has been designed for instructing the storage format and location, registering new ontology concepts and most importantly, guaranteeing the Data Consistency between the two storage methods. For easing the data persistence process, the Data Access Object (DAO) pattern is applied and optimized to enhance the Data Consistency assurance. Finally, this novel mechanism provides an easy manner for the development of applications and their integration with BATMP. Finally, an application named "Parameter Monitoring Service" is given as an example for assessing the feasibility of the system.**

Keywords-data consistency; ontology; data access object; annotation; smart home;

I. INTRODUCTION

Smart Home is a field which has been the focus of research for many years. A good deal of work has been done on the topic of automated control, sensor technologies, communication protocols, etc. However, the complexity of device selection and installation makes it difficult for users with little technological knowledge to easily apply a smart home system in their homes. Therefore, CeDInt-UPM has designed an open smart home management platform (BATMP) in order to facilitate the installation and configuration of a home control system for those users. BATMP is a middle-ware which integrates different building automation technologies and provides services such as the control of devices or ambient parameter monitoring. [1]

Through the implementation of this project, vast amounts of information can be collected, such as electrical energy consumption, daily room temperature trends, and even user preferences and actions such as switching lights on/off or leaving/getting home. How to store and share these data effectively and efficiently has become one of the key concerns of BATMP. The following issues regarding data management in BATMP should be solved: first, enabling the data collected from different resources to understand and interact with each other. To achieve this, we have applied ontology technology, considering its capability to create a unified domain vocabulary, which enables the integration of different data sources such as weather prediction [7]. However, considering that the relational databases have a significantly faster processing speed and good stability, we would also like to use them to store those data that are only used locally in one BATMP. We therefore decided to build a Data Warehouse which consists of both a relational database and the semantic ontology triple store. The details will be explained in Section 2.

On the basis of this decision, we focused on the second issue regarding how to ensure the data consistency between different data storage systems and to interact with both of them simultaneously. To solve this, building an annotation system to give instructions regarding the storage process finally turned out to be the best solution, the details of which will be illustrated in Section 3. In parallel, for processing the information provided in the annotations, we have built a system following the Data Access Object (DAO) pattern. Section 4 explains the application and optimization of DAOs and how they also help to guarantee the data consistency.

The combination of annotation and DAOs also helps us to deal with the third issue: to provide the software developers with an easy way to integrate their data into BATMP. In Section 5, an application is given as example to show how this integration is achieved.

Finally, Section 6 shows a summary of the work and offers conclusions. The DAO API and the annotation API developed will be published in the future so that third parties can develop their own applications and integrate them into BATMP, which will significantly increase the scalability of the system.

II. BATMP INFRASTRUCTURE AND COMPONENTS

A. BATMP infrastructure

BATMP is a software system which can be installed in a Raspberry Pi, MiniPC, PC, etc. Fig. 1 shows part of the BATMP infrastructure and its components.

Each BATMP comprises five components: Gateway Manager, Technology Manager, Application Manager, Model Manager and Data Warehouse. All the components communicate with each other through a list of parameters. [1]

- The Gateway Manager is in charge of the interaction between users and BATMP. It contains various services

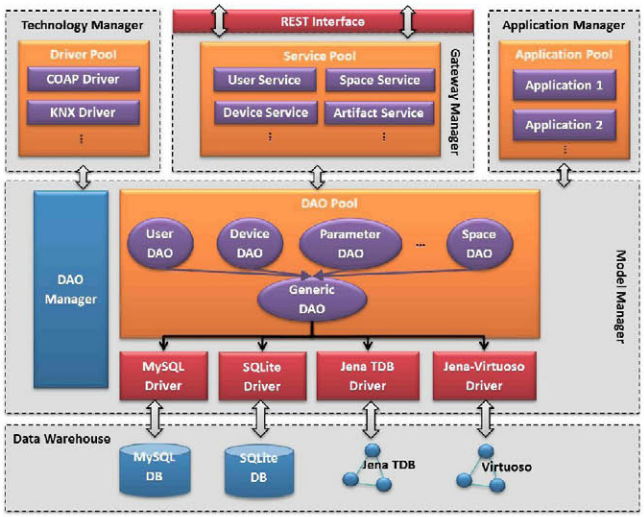

Fig. 1. BATMP infrastructure

to enable the communication between the User Interface and the Model Manager. For instance, a user service is used for parsing the login information received through REST and calling User DAO for authentication.

- The Technology Manager is in charge of the interaction between devices and BATMP. It contains a list of drivers to communicate BATMP with devices of different protocols.
- The Application Manager is in charge of the management of applications in BATMP.
- The Model Manager collects and stores data in BATMP. The above three components send the data they generate to the Model Manager for storing and request the data they need from the Model Manager. The Model Manager is responsible for the communication among all the BATMP components.

B. Data Warehouse.

All the BATMP data are stored in the Data Warehouse, including information about users, devices, artifacts, spaces, profiles, etc. Nowadays, in many projects [2][3][4], only a single type of data storage is applied, either a relational database or a semantic ontology triple store. Sometimes technologies are developed to map the data or convert the format between these two kinds of data management systems [5]. But there is rarely a methodology to realize the mapping and unification between these two kinds of storage at the software level. A discussion about the advantages and disadvantages of these two storage methods are given as follows:

- Speed of Processing: in [6], it is clearly illustrated that a relational database such as MySQL has a highly superior performance in processing speed than those ontology triple stores (such as JenaTDB and Virtuoso), both in load time of triples and the query processing execution.
- System Stability: due to their mature development, relational databases have a better stability compared to ontology storage. By using JenaTDB, we have met some problems such as triple store crash, which leads to the loose of data. And using Virtuoso, the server occupied too much space when a Mini PC is used to host BATMP. On the other hand, MySQL has very good stability during the test period and an easy data backup system.
- Security Mechanism: in general, a relational database has a relatively complete security mechanism. For instance, a relational database such as MySQL has an Access Privilege System to authenticate a user who connects from a given host and associate that user with some privileges on the database. Another commonly used relational database SQLite hooks built-in for database encryption. However, for the ontology, due to the principles of openness and data sharing, less security mechanisms are included.
- Data sources integration: One of the key contributions of an Ontology is to create a common vocabulary in a specific domain for easy knowledge sharing and reusing. It is able to integrate the information from different data sources easily, while relational databases are not designed for this function.
- Data publishing: For easily publishing and sharing data, there is already a list of tools such as the endpoints in SPARQL. Relational databases are relatively closed though and do not publish the data they contain.

BATMP has been designed for integrating different kinds of data, including those related to users, spaces, devices, measurements, etc. The requirements for data storage depend on the data. For instance, user authentication data such as username and password, which requires a higher security level, should be stored in the relational database as a local resource. However, the person-description information such as age and gender may be stored in the ontology for data publishing and further data mining such as user behaviour analysis. As a result, we decided to apply both storage methods in BATMP Data Warehouse for storing different data according to their usage and properties. Currently, MySQL is chosen as the default relational database and Jena TDB as the default ontology storage.

III. Annotation System

As illustrated above, Data Warehouse consists of both a relational database and an ontology store. Therefore, a mechanism is necessary for instructing the Model Manager about where and how to persist a Java object and ensure the consistency of data between both of the storage systems. To achieve this goal, an annotation system has been developed. With the help of annotations, an instance of Java class can be persisted to both the relational database and ontology in the same transaction.

BATMP requires that those Java Classes to be persisted should follow the pattern of Plain Old Java Object (POJO), namely, only have variables and the getters and setters for each variable. As these objects should be a reflection for the data persisted in the Data Warehouse, this convention is not going to result in the sacrificing of expressiveness.

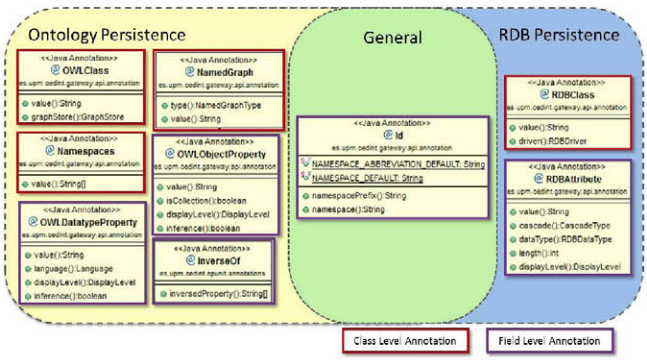

Fig. 2. Annotation API

By adding annotations in a Java class, the developer is able to integrate the class into BATMP and benefit the data persistence process of Model Manager without changing its class structure. The instances of the Java class will be persisted to the relational database or the ontology triple store according to the annotations set. If the developer decides to change the storage location or format of a class or a field, for instance, from the relational database to the ontology, he/she can just change the annotations without worrying about how the storage process is achieved.

The annotation API (Fig. 2) consists of three parts, general annotations, annotations for the relational database and annotations for the ontology. This annotation system is able to solve many issues which concern us, including data storage instruction, ontology concepts registration and most importantly, data consistency guarantee in Data Warehouse.

A. Data Storage instruction and configuration

1) Having a class to persist, the first problem to solve is which storage(s) to choose. Two class-level annotations have been defined: @OWLClass and @RDBClass. If a Java class needs to be persisted partly or fully in the ontology triple store, the annotation @OWLClass should be provided with the corresponding concept name, for instance, @OWLClass("spatiaCore:User"). The annotations @OWLDatatypeProperty and @OWLObjectProperty inside of a class will be ignored if the class does not have @OWLClass. Similarly, if @RDBClass("user") is provided, the class will be persisted to the table named "user" in database.
2) As long as the storage method is chosen, the connection driver should be selected. BATMP provides four drivers: Driver MySQL and SQLite for relational database and Driver Jena TDB and Jena-Virtuoso for the ontology store.

B. Concept Registration in BATMP

In order to register a class in the Ontology Structure of BATMP, it is necessary to provide the information by using @Namespaces, @NamedGraph and @OWLClass:

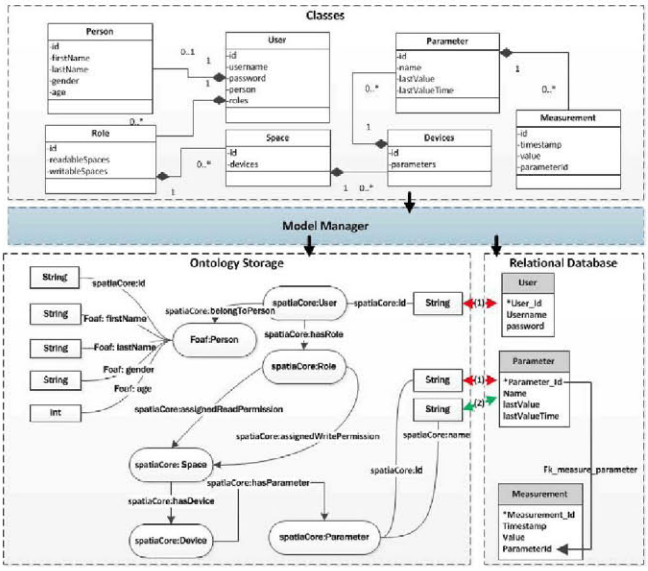

Fig. 3. Ontology Storage and Relational Database Schema

1) @Namespaces provides a list of mapping between the prefixes and the corresponding URLs. For instance, "spatiaCore:www.cedint.upm.es/residentialOntology.owl♯". If the defined prefix is already registered with another URL, an exception will be thrown. This prevents the potential conflict as the prefix is used frequently and a convention should be created for them.
2) @NamedGraph indicates the graph to which the concept is stored. It is recommended that the relatively close concepts be stored in the same graph while the others are stored in other graphs. By doing this, the furthering data query and inference efficiency could be improved.
3) @OWLClass provides the concept URL corresponding to the class. If this concept does not exist in the BATMP ontology structure, it will be added.

C. Data consistency between ontology and relational database

In the case where a class is stored in both the database and the ontology, a unique identifier is necessary. The annotation @Id has been defined to be used in such cases. It is compulsory to use @Id just for one Primitive/Primitive Wrapper field of the class. The annotated field will be used as the primary key in the database and as the reference to generate the individual URL in the ontology. Two objects with the same @Id field will be identified as the same object. Fig. 3 illustrates how the same Java object is persisted into both storages. During the persistence process, data consistency is guaranteed as the data come from the same object. Meanwhile, during the retrieval process, Model Manager retrieves the data of each field according to the same ID passed and unifies them in the same object. If the data of the same field is different in the two storages (Arrow 2), an exception will be thrown.

D. Storage of concepts' attributes and relations among them

In a Java class, a field could be of Primitive/Primitive Wrapper type (1) or an object/collection of objects (2). The field in case (1) will be stored in the ontology as an attribute of the concept if the annotation @OWLDatatypeProperty("xxx") is provided, while if @RDBAttribute("xxx") is provided, the field will be stored as a column in the corresponding table. In case (2), if a @OWLObjectProperty() is provided, this relationship will be stored as an object property between the annotated field and the current class. In relational databases the cases could be even more complicated. Considering the different cases of fields inside a Java class, the option cascade() is provided for @RDBAttribute, indicating the manner to store the data:

- TOSTRING, indicates that the field is converted as a String to be stored in the corresponding column;
- MERGE, indicates that all the fields inside this annotated field will be retrieved and stored in the same table of the class;
- RECURSIVE, indicates that the annotated field will be stored in a separate table with a foreign key consistent with the primary key of the current class.

E. Data Retrieve confidentiality and efficiency

With regard to the efficiency and security of data retrieval, the option displayLevel() is provided in the annotations @RDBAttribute, @OWLDatatypeProperty and @OWLObjectProperty. It indicates the level to which the field value is exposed:

- CONFIDENTIAL, used for the information that should never be exposed, such as a password;
- PUBLIC, used for the fields that should be shown or retrieved in all kinds of methods;
- SPECIFIC, the default option, which is used for those fields whose exposure is not so important when several objects are retrieved at the same time. For instance, one field with this option will be retrieved in the method findById, while in the method findAll, this field will be set to null, without making any query to Data Warehouse.

In the Model Manager, those methods for retrieving data can define the displayLevel. The information will be exposed only if its displayLevel is higher than the level specified in the retrieve method, the sequence is PUBLIC>SPECIFIC>CONFIDENTIAL.

IV. Data Persistence and Retrieval

The Model Manager is used to ensure that the data sent by the different BATMP components are stored correctly in the Data Warehouse and that the stored data are queried in an effective and secure way by each component. For easing the process of data persistence, the software pattern Data Access Object (DAO) is employed. Model Manager includes a DAO Manager, for registering and retrieving DAOs; DAO Pool, which contains various DAOs for handling the persistence of different Java classes; and several Data Warehouse connection drivers for the use by DAOs.

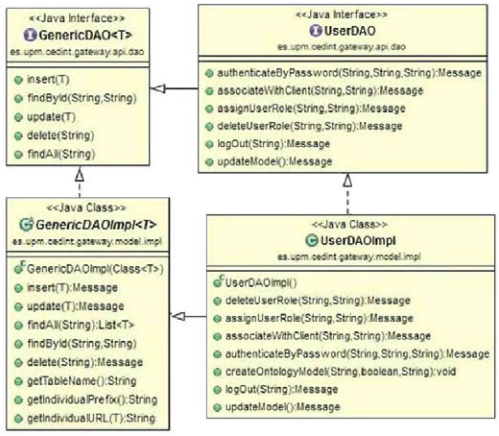

Fig. 4. Generic DAO and DAO Structure

A. Data Access Objects (DAOs)

Data Access Object is a software design pattern introduced by Sun Microsystems for providing data operations without exposing details of the database [11]. Data Access Object is an interface used to provide access to the persistent storage layer. This makes it possible that the changes of logic at the storage side will not affect the operation of other modules. These changes could be database type (relational databases, ontology, flat files, etc.), database driver (MySQL, SQLite, Jena TDB, Jena-Virtuoso, etc.), database location, etc. In general, by using DAOs, the process of storage is eased significantly and separated from the main business logic of the system.

B. Use of Generic DAO

In BATMP, each DAO is responsible for handling one specific class and its subclasses. For instance, User DAO is responsible for User class, and ParameterDAO is responsible for the class Parameter and its subclasses: ParameterReadOnly, ParameterModifiable and ParameterSelectable. Following the rule of not repeating code, a Generic DAO is constructed for carrying out the basic data CRUD (Create, Read, Update, Delete) functions and storing the key information of persistent storage. Generic DAO works with the generic type T [9] and provides five basic functions: insert, update, delete, findById and findAll by using Java Reflection programming [10]. Each DAO inherits the five basic functions from Generic DAO and contains its own methods according to the different needs for handling the data of the corresponding class. For instance, in User DAO, a method authenticateByPassword() is created for validating the user's authentication information. Fig. 4 shows the construction of Generic DAO and an example of DAO, User DAO. Besides the default five methods, GenericDAO also provides a list of methods for the general use purposes, such as findByField(), findByQuery(). Each DAO is able to connect to one relational database and one ontology triple store at the same time. All the DAOs are kept in a DAO pool and each component retrieves those DAOs that interest it from there.

C. DAO creation, registration and retrieval

For creating and retrieving DAOs properly and efficiently, a DAO Manager is employed. This DAO Manager maintains a list of mappings between the Java classes to be persisted and their corresponding DAOs. For instance, for persisting the instances of User class, User DAO is needed. Every time a DAO is created, it needs to be registered in the DAO Manager by adding a mapping between the target class and the DAO. If a class has subclasses, the mapping between each subclass and the DAO should also be added. Sometimes, an instance may contain variables which are instances of other classes. In this case, during the reading and writing process, the DAO Manager will first search its mapping list. If a corresponding DAO is found for this object, this DAO is applied. If no proper DAO is found, a generic method is provided for persisting the field. For instance, a user contains a list of roles. As the Role DAO is registered, it is used for handling the role instances inside the user instance. This mechanism has two advantages. First, each DAO contains some basic information such as the ID field, the table name, the individual URL prefix, etc. which needs to be obtained by looping all fields of the class. Therefore, this mechanism reduces the iteration process to work more efficiently. Second, some classes need special operations for data persistence and retrieval. For instance, those classes which have subclasses need to keep a mapping of subClass-URL or subClass-Indicator in the corresponding DAOs. As a result, it is necessary to use this mechanism for ensuring that the instances of each class are persisted by the correct DAO.

D. DAOs for applications

Besides the default DAOs created in BATMP, the platform allows its applications to create their own DAOs and register them in the DAO Manager, taking advantage of the easiness for integrating their data with BATMP. Before the DAO is registered successfully, DAO Manager will validate the corresponding class, checking if all the annotations are set appropriately; if the name spaces provided are already registered; if the table already exists, etc. For security considerations, all the data from the application will be stored in a separate database in Data Warehouse.

E. Control of Data Consistency by DAOs

As the annotation contains all the information about exactly where and how to persist the data, the initial database and the ontology can be empty unless it is required for some backup file to be loaded first. In the following section, the collaboration between DAO and annotations will be explained. The Data Consistency is guaranteed by the DAOs from three aspects.

1) Data Consistency during initial step. Subclass Registration and Database/Ontology construction: this automatic construction mechanism prevents the potential inconsistency among a manually created database, ontology structure and the persisted Java class. It also avoids the extra workload that might occur if any of these three parts were modified. When creating a DAO, the target class T should be assigned to the DAO. If the class has subclasses, they should be registered in the DAO so as to know to which URL a subclass corresponds. Once the registration has taken place, the DAO will check the annotations of the class. If @OWLClass is found, a new concept corresponding to this class will be created in the ontology, while if @RDBClass is found, a table will be created in the relational database. Hereafter, the DAO scans each field until it finds the one with @Id. This field will be used to generate the individual URL in the ontology and/or be used as primary key in the database. Finally, the DAO scans all the fields and creates the properties in the ontology and/or the columns in the database accordingly.

2) Data Consistency during Data Storage: the data storage is accomplished by the methods insert() and update(). When a new instance needs to be persisted, the DAO reads the identifier field and creates a new individual in ontology and/or a new row in database accordingly. Afterwards, the DAO reads all the fields, adding new properties and/or updating the row accordingly following the annotations. If it succeeds, a success message containing the object ID will be returned. As the information for storing in both places comes from the same instance, the data consistency is guaranteed. If it fails, an error message containing the cause will be returned followed by a rollback in the transaction with both relational database and ontology store. In this case, data consistency is assured.

3) Data Consistency during Data Retrieval: the data retrieval is accomplished by the methods findById() and findAll(). Taking findById() for instance, the DAO reads all the fields of the target class and constructs an Ontology query (SPARQL) and/or a database query (SQL) according to the annotations. After making the queries, the DAO creates an instance of target class and assigns the retrieved values to each corresponding field. If it succeeds, the instance is returned. If it fails for some reason such as that the given object ID is not valid, a null will be returned. If the data retrieved from the two storages is not consistent, such as that the database is changed by accident, an exception will be thrown during the data retrieval process, which ensures the Data Consistency.

V. Development of applications for BATMP

On the basis of BATMP, several applications have been developed, such as a Parameter Monitoring Service, an application to monitor all the parameters in BATMP; BatStreetLight, an application for intelligent street lighting control; and GreenLabs, an application for greenhouse monitoring. It has been proved that this data management system eases and shortens the process of application development and integration significantly. In this section, we will take the Parameter Monitoring Service as an example to illustrate how to develop an application for BATMP, taking advantage of this developed Data Management mechanism. Fig. 5 shows one of the application screens. In this application, two new classes ParameterConfig and Graph are introduced. ParameterConfig represents the configuration of one parameter regarding the way in which the user wants to see its measurements, as

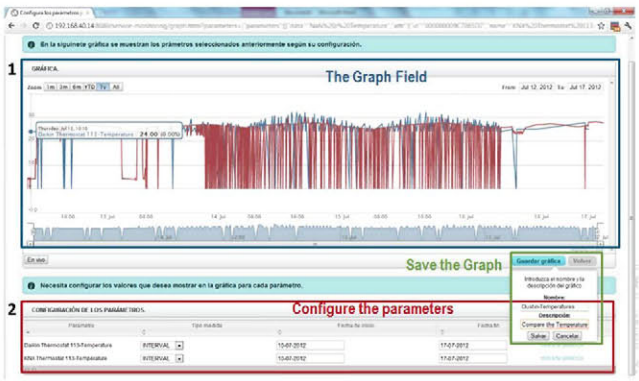

Fig. 5. Application for Parameter Monitoring of BATMP

```
@RDBClass("graph")
public class Graph {
  @Id
  @RDBAttribute(value = "graphId", length = 30, displayLevel = DisplayLevel.PUBLIC)
  private String id;
  @RDBAttribute(value = "name", length = 30, displayLevel = DisplayLevel.PUBLIC)
  private String name;
  @RDBAttribute(value = "graphId", casacade = CascadeType.RECURSIVE)
  List<ParameterConfig> parameterConfigList;
}

@OWLClass("spatiaCore:ParameterConfig")
@Namespaces({"spatia","www.cedint.upm.es/residentialOntology.owl#"})
@RDBClass("parameter_config")
public class ParameterConfig {...}
```

(a)

```
public interface ParameterConfigDAO extends GenericDAO<ParameterConfig>{...}

public class ParameterConfigDAOImpl
      extends GenericDAOImpl<ParameterConfig> implements ParameterConfigDAO{
  public ParameterConfigDAOImpl(){
    super(ParameterConfigDAO.class);
  }
}
```

(b)

Fig. 6. Example of annotations and DAO creation

denoted in part 2. Graph represents a single graph in which the selected parameters are shown together, as denoted in part 1. A graph includes a list of ParameterConfigs. The application allows users to create their own graphs, store and retrieve them. To achieve this, it is necessary to create two DAOs, ParameterConfigDAO and GraphDAO for dealing with these two classes. Meanwhile, the two classes should be annotated properly for instructing the DAOs about the persistence.

Fig. 6(a) shows an example of the annotations for the two new classes. Considering the use of ParameterConfig and Graph, we decided to store both of them in the relational database for the local use of users. As a result, @RDB-Class is provided for both classes. Meanwhile, we might want to publish part of the ParameterConfig data such as parameterId for the further user behaviour analysis. Following the instruction of the annotations, two tables named "graph" and "parameter_config" will be created in the relational database and a new concept "spatia:ParameterConfig", namely, "www.cedint.upm.es/residentialOntology.owl♯ParameterConfig" will be added to the BATMP ontology structure. The columns in the database will be created according to the annotations @RDBAttribute provided, while the properties in the ontology will be created according to the annotations @OWLDataProperty and @OWLObjectProperty provided. Once the new classes have been annotated, their corresponding DAOs should be created. Fig. 6(b) shows an example of DAO creation. As can be observed, just using a couple of lines, a new DAO with the basic functionality (insert, update, findById, findAll and delete) is created. After the DAOs are created, it is necessary to register these DAOs in the DAO Manager, namely, add mapping

VI. Conclusion and Future Work

In conclusion, due to the complexity of an open smart home management system such as different data sources, multi-user, frequent data CRUD process, it turns out that a combination of a relational database and an ontology is a good solution. In this paper, the authors have designed a novel solution for data persistence with more than one type of storage at the same time, including an annotation system for providing the persistence instruction and list of DAOs for processing the data according to the annotations. By implementing this mechanism, the Data Consistency is guaranteed appropriately. Meanwhile, the third-party developers can develop their applications easily taking advantage of the existing data access methods and publish their data for the benefit of other developers.

However, to improve the current system, some work remains to be done. For instance, the annotation system can be improved with the feedback from the developers in order to make the system more flexible. Moreover, more drivers for both the relational database and the ontology should be provided later.

References

[1] Jaime Caffarel, Guillermo del Campo-Jimenez, Jorge M. Perandones, Csar Gomez-Otero, Rocio Martnez and Asuncin Santamara: Open Multi-Technology Building Energy Management System. In: Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress, pp.397–404. St. Petersburg (2012).

[2] Bonino Dario, Fulvio Corno: Dogont-ontology modeling for intelligent domotic environments. The Semantic Web-ISWC 2008. Springer Berlin Heidelberg. 790-803 (2008).

[3] Mossberg Walt: SmartThings Automates Your House Via Sensors, App. (2014).

[4] universAAL Project, UNIVERsal open platform and reference Specification for Ambient Assisted Living, 7h Framework Programme of the European Union, Grant Agreement No. 247950, 2010-2014, http://www.universaal.org. [Accessed on 18th June 2014].

[5] Anuradha Gali1, Cindy X. Chen1, Kajal T. Claypool1 and Rosario Uceda-Sosa: From ontology to Relational Databases. In: Workshop Concept.-Model Driven Web, pp. 278–289. Shanghai, China (2004).

[6] Christian Bizer, Andreas Schultz: The Berlin SPARQL Benchmark. In: International Journal on Semantic Web and Information Systems, Vol. 5, Issue 2, Pages 1-24 (2009).

[7] Ghislain Atemezing, Oscar Corcho, Daniel Garijo, José Mora, Mara Poveda Villalón, Pablo Rozas, Daniel Vila-Suero, Boris Villazón-Terrazas. Transforming Meteorological Data into Linked Data. In: Semantic Web Interoperability, Usability, Applicability an IOS Press Journal. 1570-0844 (2012).

[8] Guillermo del Campo, Eduardo Montoya, Jorge Martín, Igor Gómez, Asunción Santamaría: BatNet: A 6LoWPAN-Based Sensors and Actuators Network. Ubiquitous Computing and Ambient Intelligence. 7656, 58–65 (2012).

[9] Generic Types, http://docs.oracle.com/javase/tutorial/java/generics/types.html. [Accessed on 18th June 2014].

[10] The Reflection API, http://docs.oracle.com/javase/tutorial/reflect/ [Accessed on 18th June 2014].

[11] Core J2EE Patterns - Data Access Objects, http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. [Accessed on 18th June 2014].