An Approach to Generating Ontology–Based Object Model for Smart-M3 platform

Kirill Kulakov, Sergei Marchenkov
Petrozavodsk State University (PetrSU)
Petrozavodsk, Russia
{kulakov, marchenk}@cs.karelia.ru

Sergey Tishkov
Karelian Research Centre of the
Russian Academy of Sciences (KarRC RAS)
Petrozavodsk, Russia
insteco_85@mail.ru

***Abstract*—The developing of software agents for the Smart-M3 platform requires a deep knowledge of specific details from the developer. The important task is to provide the developer with high-level programming tools. This work presents an approach to generating ontology-based object model for the Smart-M3 platform. The ontology-based object model is implemented as a source code generator (SmartJavaLog) for the Java language. SmartJavaLog generates Java classes for objects and the necessary infrastructure for interacting with the semantic information broker (connection, query, subscription). The proposed approach is demonstrated on a small smart service consisting of three software agents.**

I. Introduction

Based on the smart spaces approach to developing a service-oriented environment [1] the information service is represented as a semantic information broker (SIB) and set of knowledge processors (KP) [2]. The semantic information broker stores data of the smart space and provides the possibility of interaction between KP's through a low-level programming interface (insert, remove, query, subscribe). The agents interact in the smart space over the shared information storage to construct a semantic network based on the available information objects and their semantic relations obtained from various sources. At the same time, the interaction between KP and SIB has no limitations. This leads to the need for interaction rules: which knowledge objects and object relations can be available in the SIB, how to add, remove or update a knowledge object and notify other KP about it.

One way to define interaction rules is to construct an ontological model [2]. The ontological model allows describing objects, relationships between objects, data properties, instances of objects, etc [3]. There are generally accepted models for a number of areas and a service ontology model may include parts of third-party models. For example, location data can be represented using Basic Geo (WGS84 lat/long) Vocabulary (https://www.w3.org/2003/01/geo/).

The knowledge objects are represented in the SIB as triplets "object–predicate–subject". The programming interface of the semantic information broker allows to input a set of triplets that describes an inserted, updated or removed knowledge object. The output of the SIB is also a set of triplets, which describes modifications of the smart space. Therefore, the KP developing requires the implementation interaction with SIB programming interface: convert internal objects into set of triplets for sending to SIB and convert a set of triples received from SIB to the internal objects.

Basically, KP implements a programming interface using a library with "low level" functions (e.g. kpi_low [4], java KPI [5], python KPI [6]). This library implements the SSAP protocol and allows to access the basic functions of the SIB: add or remove triplets, send queries and subscribe on notifications. Control over the correct representation of data in the form of triples, the correspondence of the ontological model, and the determination of the interaction between KP's rests with the KP developer.

There is also an approach to the development a high-level interaction of KP with SIB based on ontologies. In this case, the library includes some additional functions, for example, reconciliation with the dictionary or representation data in the objects or structures [1]. The most known library is SmartSlog [7], [8]. The Smart-M3 platform also contains Smart-M3 ontology to C-API generator and Smart-M3 ontology to the Python generator [9]. As a rule, a library that implements a high-level interaction is represented as a source code generator. The resulting code can contain information from the ontological model: dictionary, objects, instances.

Unlike other popular programming languages there are no known library for Java with a high-level interaction implementation. Typically, a Java developer uses the low-level interoperability feature through the Java KPI or Java Native Interface (JNI): implementation interaction with the SIB as a C module and connection to the Java application [10]. Unfortunately, in the first case, the developer must control the correctness of interaction with the SIB manually, and in the second case the result is not a Java-only application. In addition, if an error occurs in the JNI module, a KP failure occurs.

This paper describes an approach to generating an object model for the Java language based on an ontology. Low-level tasks, such as converting triples to and from an object, sending change notifications and tracking changes in object instances, are included in the generated object code and executed automatically. Thus, the KP implementation is performed in the traditional object model using the capabilities of the Java language.

The rest of the paper is organized as follows. Section II presents an overview of related work. Section III introduces the main approaches of data presentation: triples, structures and objects. Section IV contains a description of the main ways to interact with the SIB. Section V describes the SmartJavaLog implementation: the source code generation process, the high-level architecture and the source code templates used. Section VI shows the results of SmartJavaLog for the smart service “GeoCode”: ontology and code fragments used. Section VII ccontains a discussion of the pros and cons of SmartJavaLog. Section VIII concludes the paper.

II. Related work

There is a large number of works aimed at developing computer-aided programming tools used to automate simplify the task of ontology-driven software development. One of the popular related research areas is mapping between ontologically derived concepts and object-oriented language constructs. Evermann and Wand [11] propose mapping rules to guide the construction of object-oriented conceptual models from ontological conceptual models. This work indicates that object-oriented languages are expressive enough to model real-world application domains. Batanov and Vongdoiwang [12] consider the similarities and differences between object models and ontologies. The most substantial difference is while ontologies represent well structured description of mutually related terms, the object model is to represent a system as structure of objects ready for implementation in software.

Another popular research area is to create software based on code a generation approach. This approach is commonly used for dynamically typed, interpreted, and object-oriented program languages (e.g. Java). The Protéjé source code generator [13], [14] is a module for Protéjé open-source ontology editor. It generates factory that serves as the entry point to the generated code providing access to existing individuals in the ontology and the ability to create new individuals in the ontology. The Vocabulary class collects all used IRI’s. Each object represents as interface with list of methods.

The OWL-DL to C(glib) API generator [9] generates a C API, abstracting the ontology and the SIB communication. The KP template can be used by the developer to easily start using the generated API. It contains all needed code to join the SIB. The code can be inserted into the kpMainLoop. The generated code includes functions and structures for the KP developer for accessing the generated ontology mappings (generic.h, generic.c, etc.), mediator for accessing the SIB that also contains a local triplestore (mediator.h, mediator.c).

RDFReactor [15] is a open-source code generator which transforms a given ontology in RDF Schema into a familiar, dynamic, object-oriented Java API (classes and etc). At runtime, objects of these classes act as stateless proxies on the RDF model. This enables developers to interact with java proxy objects, thus allowing them to stay in their own world and at the same time to make use of the advantages RDF capabilities.

SmartSlog (Smart Space Ontology) [16], [8] is a software/application development kit (SDK or ADK) for programming Smart-M3 agents (Knowledge Processors, KPs) that consume/produce smart space content according with its high-level ontological representation . SmartSlog applies the code generation approach: given an OWL ontology description, the ontology programming library is produced. The latter provides API to access the smart space via a Smart-M3 Semantic Information Broker (SIB) and data structures and functions to represent and maintain locally in KP code all ontology classes, relations, properties, and individuals.

III. Object model

KPs in smart space applications are consumers or producers of a shared information storage, that is is organized as an RDF graph. The development of smart space applications follows the principles of ontology-driven software development, when the design phase is reduced to creating a specification of a certain problem domain and services as an OWL/RDF description [17], [18]. Ontologies are used to share common understanding of the structure of information among KPs, to enable reuse of domain knowledge, and to analyze knowledge. This design approach is used in various problem domains such as collaborative work [19], e-Tourism [20], and smart cities [21]. In this case, the process can be accompanied by the use of tools for rapid developments of ontologies, such as Protégé [14], in which designers can instantly create classes, properties, and individuals of their ontology and experiment with semantic restrictions.

One of the features of SmartJavaLog is to create a Java object model for KPs from design ontology models. The object model merges data and functionality into an abstract variable type – an object. The object model provides a more realistic representation of objects that the end user can more easily understand. While an ontology structure contains definitions of concepts (classes) and relationship between concepts and attributes (properties, aspects, parameters), an object model uses classes to represent objects and functions to model relationships of objects and the attributes. The similarity of concepts in an ontology with an object model determines the applicability of an object-oriented approach to ontology modeling [11], [12]. However, ontology represents a more richer information model than Java objects by supporting such distinctive features as inheritance of properties, symmetric/-transitive/inverse properties, full multiple inheritances among classes and properties [22].

Taking into account the above aspects and constraints, Table I provides a summary of mapping rules used in SmartJavaLog during the generation of the object model source code from ontology. This approach based on these rules to generating ontologybased object model is convenient in such class-based and object-oriented program languages as Java. The main idea of the ontology-object mapping is to create a set of classes and objects in such a way that each ontological class with their instances, properties, slots, and facets has its equivalent in a Java class/object.

Thereby, for implementation interaction between KP and SIB it is necessary to determine objects and object properties. In smart-m3 terms object represented as a ID (string) and type (URI). Object property may have a simple data type or complex type. The simple data types can be easily determine in ontological model: it supports owl and owl2 data types like xsd:long or xsd:string. The complex data type may be aggregation of simple data types, another object and so on. There may also be restrictions on data types, for example, facets for reals. Summing up KP developer needs an opportunity to create object or load object from SIB, set object properties with data type checking mechanism and remove object.

TABLE I. SUMMARY OF MAPPING RULES (INPUT: ONTOLOGY; OUTPUT: OBJECT MODEL)

Rule notation	Explanation
Ontology classes → Object classes	Classes in an ontology are close to object classes in an object model representing abstract groups of physical or logical objects. Whereas a object class may be viewed as a type definition for objects. Ontology classes are mapped as object classes.
Instances → Objects	Ontology instances are used to represent specific elements of classes. Object class serves as a pattern of describing its object. Ontology instances are mapped as objects.
Data type properties or slots → Data attribute variables & get/set methods	Data type slots represent the data type properties (attributes) of the ontology class. These slots are described by primitive types (integer, boolean, string, etc.) and sets of values of those types. attribute variables. Data attribute variables describe the characteristics objects by various data types. Data type properties or slots are mapped as data attribute variables along with a combination of get/set methods.
Object type properties or slots → Object attribute variables & get/set methods	Object type properties or slots are used to describe a relationship between two concepts. The first must be an instance of the class that is the domain of the slot; the second must be an instance of the class that is the range of the slot. Object attribute variables represent the relation between object classes from an object model. Object type properties or slots are mapped as object attribute variables along with a combination of get/set methods.
Value-type/space facets → Attribute variables types & if-then-else statements in set methods	A value-type/space facet in an ontology is some kind of binary relation, that is attached to the slot and describes what types of value can fill in the slot and what value-space restrictions it has. These facets are applicable to attribute variables in an object model, for example, xsd:type facets can be used to designate a variable type, and xsd:lenght facets can be represented by a set of if-then-else statements in a set method of a corresponding attribute variable constraining its value space.
Cardinality facets → Additional attributes & if-then-else statements in set methods	Cardinality facets define how many values a property or slot can have distinguish single cardinality (at most one value) and multiple cardinality (any number of values). Some cardinality facets allow specifying a minimum and maximum cardinality to describe the number of slot values more precisely. Additional attribute variables in a object model can be introduced to specify cardinality facets. Some of these facets can be represented by a set of if-then-else statements in a set method of a corresponding attribute variable constraining its cardinality.
Multiple inheritance → Single inheritance & multiple interface inheritance	Java does not support multiple inheritances in classes because it can lead to diamond problem. However, the use of interfaces can solve this problem and single inheritance can also be applied in simple situations.

IV. KP'S INTERACTION METHODS

The interaction between KPs is a key function of the IoT service. The general approach offers interaction through a semantic information broker (SIB) [2]. The semantic information broker provides an application programming interface for two interaction methods.

- "Query—Answer" interaction. KP sends a request for the required data and the SIB receives all triplets found.
- "Subscription—Notification" interaction. The KP subscribes to a specific data set and the SIB sends all changes until the KP unsubscribes.

The semantic information broker supports two types of queries: a basic query (define a triple part) or an extended query in the SPARQL language [2]. The basic query has a strict limitation: the subject, the predicate and the object can be precisely defined or have any value. A SPARQL query should return the result as a set of triples i.e. the query should have three return values.

The SIB provides subscription using a triple template or a SPARQL query (not supported in some SIB implementations). The subscription operation has the same restriction as the query operation. After the subscription operation is completed, the SIB sends the first notification with all the triples found (as after the query operation). When data changes in the SIB, the KP receives a notification. Each notification, except the first, includes a set of new or modified triples and a set of deleted triples.

In practice, the implementation of the subscription operation has additional restrictions. If the response or notification contains a large set of triples (for example, after the query "give me all triples") the SIB can be overloaded [23]. A knowledge processor also should not use a large number of subscriptions because the SIB can be overloaded.

The one of solutions is to use "smart notifications" [24] when KP sends to SIB additional "notification" triplets with object changes. Other KPs can subscribe to receive "notification" triplets. If the SIB notifies of new "notification" triplets, then KP can make a query to obtain all individual data.

There are a number of studies devoted to the problem of constructing smart spaces based on ontological models. In paper [19] authors presents ontological representation of SmartRoom system with e-Tourism services information space. In paper [21], the authors present the ontological model for Smart Planning of Urban Solid Waste Management service. The paper [20] describes the ontological model of Cultural Trip Planning Service. It can be noted that the most of ontological models are represented in the form of a tree-like structure, using the terms "object" and "object property".

Based on architectural abstractions and the smart space access primitives [25] we can determine the data flows between KP's and SIB.

- Insert object. KP generates object identifier and sends to SIB one triple with object type definition and zero or more triples with object property values.
- Update object. KP gets object identifier and sends to SIB one or more triples with object property values.
- Remove object. KP removes all triples from SIB matching the object identifier.
- Search one or more objects. KP sends request to SIB based on triple template or SPARQL query. SIB returns zero or more triples corresponding request.
- Object inserting notification. SIB notifies subscribed KP about new objects in smart space.
- Object updating notification. SIB notifies subscribed KP about object property changed.
- Data updating notification. SIB notifies subscribed KP about data changes in smart space.

Thus, it can be noted that from a practical point of view, the implementation of both interaction methods ("Query–Answer" and "Subscription–Notification") is required.

V. IMPLEMENTATION

The ontology-based object model is implemented by a source code generator (SmartJavaLog) [26]. The ontology description format is very flexible. Object properties can be represented as a set of axioms. Therefore, we need a two-step algorithm: analysis of data from ontology and generation of source files. The generation process has the following steps (see Fig. 1):

1) parse ontology file;
2) generate files with common Java source code;
3) generate files with ontology specific Java source code.

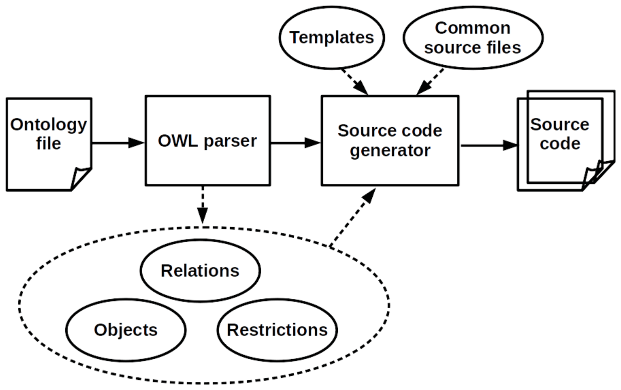

Fig. 1. Source code generation process scheme

We use the OWL API library [27] to parse ontology. The library uses "Visitor" pattern (implements in OntologyVisitor class, see Fig. 2) and presents ontology file as a set of nodes and axioms. All gathered data is stored in a single instance of OntologyFactory class ("Factory" pattern). OntologyFactory includes Maps of objects (OntologyObject class), properties (OntologyProperty class), types (OntologyComplexDataType class) and comments. The IRI uses as a key in map structure.

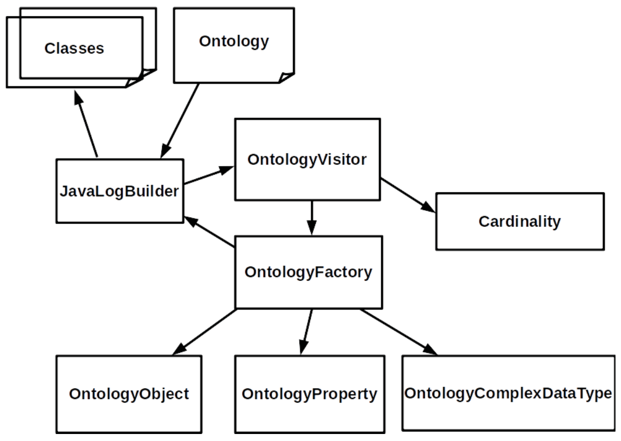

Fig. 2. SmartJavaLog high-level architecture

Common java source code uses JavaKPI library for interaction with SIB and implements core mechanisms, like asynchronous work, interfaces, factories. The common java source code includes the following items:

- class BaseRDF — parent class for all ontology objects, includes triples store, work with listeners methods and methods for interaction with SIB;
- class KPIproxy — JavaKPI library wrapper, implements asynchronous add and remove triples, join, leave and query processes;
- interface QueryListener — notification interface, includes add item notification;
- class SIBFactory — main point to work with one or more SIBs, uses "Factory" template;
- class SIBQueryTask — parent class for asynchronous access to SIB;
- class SIBSubscribeTask — parent class for subscription processes;
- class SubscribeQuery — main point of subscriptions, implements wrapper for JavaKPI subscriptions;
- interface SubscribeListener — subscription notification interface;
- class TaskListener — parent class for all tasks;
- interface UpdateListener — object changes notification interface.

SmartJavaLog uses template approach for source code generation. Ontology specific templates includes the following items (see fig. 3):

- class.java — template for object class based on BasRDF;
- object-property.java — template for object property methods (get, set);
- data-property.java — template for simple type data property with common get and set methods;
- set-data-property.java — template for extended set methods for simple type data property;
- complex-data.java — implements complex data type;
- update-data-property.java — fragment of code for update data property value in triple store;
- update-object-property.java — fragment of code for update object property value in triple store.

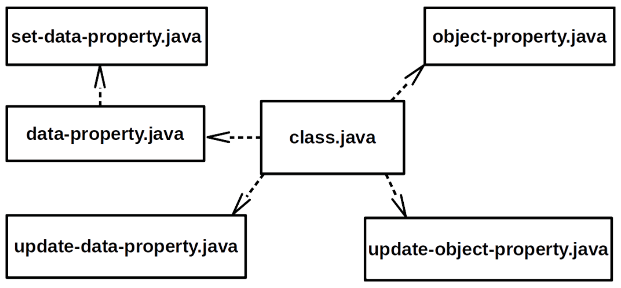

Fig. 3. Templates relationships

During the implementation of KP for Android, we were faced with a significant limitation: any work with network should not be in main thread. The original version of Java KPI don't use threads for communication with SIB. Therefore, SmartJavaLog implements wrappers for Java KPI function.

The wrapper creates SIBAsyncTask (extends AsyncTask class) based thread and call Java KPI function. When Java KPI function ends SmartJavaLog notifies all listeners.

Another problem is that non-android platforms has not AsyncTask class. For this platforms we implement simple similar class with single-thread work.

VI. Usage example

The SmartJavaLog usage example was carried out on a simple IoT service "GeoCode". Service consisted of three KP:

- GeoCode test — geo point generation KP;
- GeoCode teacher — KP of adding geo-dependent information to geo point;
- GeoCode android — KP for showing result to user.

GeoCode based on ontology with two objects: Place and Point (see Fig. 4). GeoCode test generates pair Place–Point

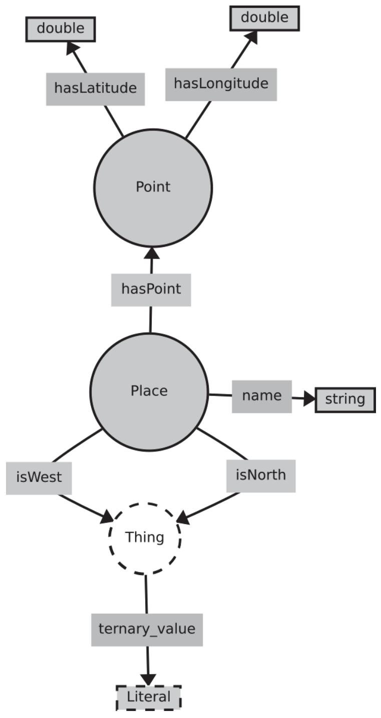

Fig. 4. GeoCode ontology

with random coordinates, GeoCode teacher determines a direction of the world based on Point coordinates and record it to the Place instance.

The connection procedure is very simple, see code fragment in listing 1. As interaction with SIB throw network is an asynchronous process, we add a event listener.

Listing 1. Fragment of code to connect to SIB
```
SIBFactory.getInstance().getAccessPoint().
    setAddr("localhost", 10101);

// register used classes before connection
Point.getClassUri();

// connect to SIB
SIBFactory.getInstance().getAccessPoint().
    connect().addListener(new TaskListener() {
        @Override
        public void onSuccess(SIBResponse
            response) {
        // interaction with SIB was here
        }
        @Override
        public void onError(Exception ex) {
        // do something when connection was
            not established
        }
}
```

The fragment of source code for adding Point and Place object instances is shown on listing 2. It is ay be noted the presence of abstraction from implementation details, i.e. the developer does not delve into the intricacies of implementing RDF triples.

Listing 2. GeoCode test source code
```
Place gp = Place.getInstance();
Point pt = Point.getInstance();
gp.setHasPoint(pt);
gp.setName("Generated point");
pt.setHasLatitude(Math.random() * 180 - 90);
pt.setHasLongitude(Math.random() * 180 - 90);

// update point
pt.update().addListener(new TaskListener() {
        @Override
        public void onSuccess(SIBResponse
            response) {
                // update place
                gp.update().addListener(new
                    TaskListener() {
                @Override
                public void onSuccess(
                    SIBResponse response) {
                // it's OK
                }
                ....
                }
                ....
        }
        ....
}
```

Implementation of subscription to class is shown on listing 3. This code structure is similar to the approaches offered in the platforms like Firebase.

Listing 3. Subscription source code
```
SubscribeQuery.getInstance().addSubscription(
    Place.getClassUri(), new SubscribeListener<
    Place>() {
        @Override
        public void addItem(Place item) {
                // item was added to SIB
        }
     @Override
        public void removeItem(Place item) {
                // item was removed from SIB
        }
        @Override
        public void onError(Exception ex) {
                // something happen
        }
});
```

Also developer can track changes in the object. The example of source code is shown on listing 4. This approach looks logical and does not require the developer to know about the features of the subscription implementation.

Listing 4. Subscription source code
```
item.addListener(new UpdateListener() {
        @Override
        public void onUpdate() {
                // item properties was updated
                    in SIB
        }
        @Override
        public void onError(Exception ex) {
                // something happen
        }
});
```

VII. Discussion

The SmartJavaLog implements object-oriented source code generation based on the ontology description. This approach has its pros and cons. The first advantage is that a KP uses identical communication mechanism. Most of the code for "Subscription–Notification" interaction is generated by SmartJavaLog. The second advantage is that the developer uses familiar objects and code structures. Working with the SIB occurs at a higher level and does not require the developer to know the features of the low-level implementation. The developer has the opportunity to use low-level access to JavaKPI for specific tasks. The main disadvantage is the impossibility of describing all the required tasks using object models. For example, the usual placement of a triple with data without specifying the type of object violates the object model. Also, the task of presenting the results of a SPARQL query in the form of objects and properties requires additional research. The second disadvantage is that ontology is very flexible and can include a variety of domain knowledge. Some of them are very difficult to use in the object model, for example, transitive property.

The proposed approach to generating ontologybased object model can be extended to generate program code for smart space M3 services. Rather than directly coding up executable programs for software agents (KPs) and their services, the developer can provide an ontology with a problem domain and service specification allowing code generation algorithms to create the object model with correct code functions with internal logic by API to access a smart space information storage. This ontological model should determine the implementation of a service based on the multi-agent approach, providing agents with a common view on the service purpose, the model of its construction, and how to interact with it. The basic idea to create such a comprehensive specification is the use the OWL-S as upper ontology [28].

Constructing a M3 service can be viewed as a collection of KPs procedures calls. OWL-S-based markup provides a declarative, computer-interpretable description that includes the semantics of the IOPEs (inputs, outputs, preconditions, effects) model to be specified when executing these calls. The process entities (AtomicProcess and CompositeProcess classes) from the OWL-s process ontology can be used to generate a Java object model methods. The IOPEs process model corresponds to the concept of Java functions/methods. The main internal logic of methods can be implemented using SPARQL queries generated from SWRL conditions and expressions. The control constructs of CompositeProcess entity (If-Then-Else, Repeat-While, etc) can be transformed into the corresponding statements of Java. The additional InteractionModel class can be used to determine KPs roles in the process of constructing a M3 service based on the publish/subscribe model. For this purpose in the code of each KP a block is formed with the necessary subscription operations using objects, internal functions, handlers, and API functions.

VIII. Conclusion

This paper presents an approach to generating ontology–based object model for the Smart-M3 platform. The ontology-based object model is implemented as a source code generator (SmartJavaLog) for the Java language. The SmartJavaLog generates Java classes for objects and the necessary infrastructure for interacting with SIB (connection, query, subscription). The SmartJavaLog will be useful for Java developers, especially in the use on the Android platform. This approach can be extended to other object-oriented languages, such as C# and PHP.

Acknowledgment

The research was financially supported by the Ministry of Education and Science of Russia within project # 2.5124.2017/8.9 of the basic part of state research assignment for 2017–2019. The reported study was funded from Russian Fund for Basic Research according to research project # 19-07-01027. The results were implemented by the Government Program of Flagship University Development for Petrozavodsk State University in 2017–2021.

References

[1] D. G. Korzun, S. I. Balandin, V. Luukkala, P. Liuha, and A. V. Gurtov, "Overview of Smart-M3 principles for application development," in *Proc. Congress on Information Systems and Technologies (IS&IT'11), Conf. Artificial Intelligence and Systems (AIS'11)*, vol. 4. Moscow: Physmathlit, Sep. 2011, pp. 64–71.

[2] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform," in *Proc. IEEE Symp. Computers and Communications (ISCC'10)*. IEEE Computer Society, Jun. 2010, pp. 1041–1046.

[3] J. Davies, R. Studer, and P. Warren, *Semantic Web technologies: trends and research in ontology-based systems*. John Wiley & Sons, 2006.

[4] "Download KPI_low software for free at SourceForge.net," Oct. 2012. [Online]. Available: http://sourceforge.net/projects/kpilow/

[5] "Smart-m3 java kpi library," Oct. 2014. [Online]. Available: https://sourceforge.net/projects/smartm3-javakpi/

[6] "Download smart m3 python kpi iddi/sofia wiki," Aug 2012. [Online]. Available: https://github.com/iddi/sofia/wiki/Download-smart-m3-python-kpi

[7] A. A. Lomov, "Ontology-based KP development for Smart-M3 applications," in *Proc. 13th Conf. of Open Innovations Association FRUCT and 2nd Seminar on e-Tourism for Karelia and Oulu Region*, S. Balandin and U. Trifonova, Eds. SUAI, Apr. 2013, pp. 94–100.

[8] "SmartSlog: free development software downloads at SourceForge.net," Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smartslog/

[9] "Smart-M3: Free development software downloads at SourceForge.net," Release 0.9.5beta, Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smart-m3/

[10] "Java native interface specification: Contents," 2017. [Online]. Available: https://docs.oracle.com/en/java/javase/11/docs/specs/jni/index.html

[11] J. Evermann and Y. Wand, "Ontology based object-oriented domain modelling: fundamental concepts," *Requirements engineering*, vol. 10, no. 2, pp. 146–160, 2005.

[12] D. N. Batanov and W. Vongdoiwang, *Using Ontologies to Create Object Model for Object-Oriented Software Engineering*. Boston, MA: Springer US, 2007, pp. 461–487. [Online]. Available: https://doi.org/10.1007/978-0-387-37022-4_16

[13] H. Knublauch, "Ontology-driven software development in the context of the semantic web: An example scenario with Protege/OWL," in *1st International workshop on the model-driven semantic web (MDSW2004)*, 2004, pp. 381–401.

[14] A free, open-source ontology editor and framework for building intelligent systems. [Online]. Available: https://protege.stanford.edu/

[15] M. Volkel and Y. Sure, "Rdfreactor – from ontologies to programmatic data access," in *Poster Proceedings of the Fourth International Semantic Web Conference*, 2005.

[16] D. Korzun, A. Lomov, P. Vanag, J. Honkola, and S. Balandin, "Generating modest high-level ontology libraries for Smart-M3," in *Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, Oct. 2010, pp. 103–109.

[17] C. W. Yang, V. Dubinin, and V. Vyatkin, "Ontology driven approach to generate distributed automation control from substation automation design," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 668–679, Feb. 2017.

[18] S. Isotani, I. I. Bittencourt, E. F. Barbosa, D. Dermeval, and R. O. A. Paiva, "Ontology driven software engineering: a review of challenges and opportunities," *IEEE Latin America Transactions*, vol. 13, no. 3, pp. 863–869, 2015.

[19] A. Vdovenko, S. Marchenkov, and D. Korzun, "Enhancing the smart-room system with e-tourism services," in *Proc. 17th Conf. Open Innovations Framework Program FRUCT*, Apr. 2015, pp. 237–246.

[20] K. Kulakov and O. Petrina, "Ontological model of multi-source smart space content for use in cultural heritage trip planning," in *Proc. 17th Conf. Open Innovations Framework Program FRUCT*. IEEE, Apr. 2015, pp. 96–103.

[21] V. Catania and D. Ventura, "An approch for monitoring and smart planning of urban solid waste management using smart-m3 platform," in *Proceedings of 15th Conference of Open Innovations Association FRUCT*, April 2014, pp. 24–31.

[22] W. V. Siricharoen, "Ontologies and object models in object oriented software engineering," *IAENG International Journal of Computer Science*, vol. 33, no. 1, pp. 19–24, 2007.

[23] A. S. Vdovenko, D. G. Korzun, and I. V. Galov, "Simulation performance evaluation of Smart-M3 applications for Internet of Things environments," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2017 9th IEEE International Conference on*, vol. 2. IEEE, 2017, pp. 994–999.

[24] I. Galov and D. Korzun, "A notification model for Smart-M3 applications," in *Proc. 14th Int'l Conf. Next Generation Wired/Wireless Networking and 7th Conf. on Internet of Things and Smart Spaces (NEW2AN/ruSMART 2014), LNCS 8638*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Springer-Verlag, Aug. 2014, pp. 121–132.

[25] D. Korzun, "Service formalism and architectural abstractions for smart space applications," in *Proc. 10th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR 2014)*. ACM, Oct. 2014, pp. 19:1–19:7.

[26] "Smart spaces ontology java code generator," 2018. [Online]. Available: https://github.com/seekerk/smartjavalog

[27] "Owl api main repository," 2018. [Online]. Available: https://github.com/owlcs/owlapi

[28] (2004) Owl-s: Semantic markup for web services. [Online]. Available: https://www.w3.org/Submission/OWL-S/