Automated Code Generation of Multi-Agent Interaction for Constructing Semantic Services

Sergei A. Marchenkov

Department of Computer Science
Petrozavodsk State University (PetrSU)
Petrozavodsk, Russia
e-mail: marchenk@cs.petrsu.ru

***Abstract*—This paper proposes a solution to the problem of simplifying the development and maintenance of smart space applications by creating tools for automated code generation of multi-agent interaction for constructing semantic services. The general scheme of automated code generation process of multi-agent interaction for constructing semantic services is introduced. By expanding the Web Ontology Language for Services (OWL-S), a unified ontological description of the semantics of service constructing processes is introduced. The code generation procedures for agent data object model and interaction processes are presented. The efforts, in automated development of semantic services through the use of the proposed unified service ontology ,and the code generator are investigated based on estimation of time to generate program code and the quality metrics of generated code.**

Keywords–semantic services; code generation; ontology-driven development; information-driven interaction.

I. Introduction

The Smart Spaces (SmS) approach to smart application development combines Internet of Things technologies with the Semantic Web to create a class of ubiquitous environments [1]. The smart nature of this approach is due to the need to provide participants in SmS with services in the conditions of their mass use, the presence of heterogeneity of computing devices and software components, their physical distribution, a variety of resources used and possible means of network communications for transferring data between participants. The interaction participants are software agents, who are consumers and producers of shared information storage.

Each agent in SmS application works in accordance with a specific domain area and model of information-driven interaction with other agents in the process of constructing and delivering services [2]. The agent logic developer uses the Application Programming Interface (API) middleware to access SmS information storage. From the point of view of agents, information storage is organized as an Resource Description Framework (RDF) graph, as a rule, in accordance with some ontology defined using the Web Ontology Language (OWL) description language.

SmS middleware (platform) is a software layer that allows agents to share content. A middleware supports a variety of semantic interoperable access primitives, including ontology-oriented ones. Currently, there are many available software implementations of platforms for creating such SmS: OpenIoT [3], Neo4j [4], SEPA [5], FIESTA-IoT [6]. As an example, the paper discusses the CuteSIB [7] platform using an ontology-oriented approach to service development. The environments of the smart museum [8] and collaborative work [9] environments are considered as examples of applications.

The construction of services in SmS is implemented as a distributed computing process, that allows creating more complex system solutions based on information-driven interaction of agents. In other words, services are created as a result of agents working together. Such agents perform a step-by-step process of changing the shared information storage based on ”publish/subscribe” models in order to implement the application function and ensure interaction with the resources of the computing environment. The consumers of services are often users. Therefore, the process of knowledge extraction and service delivery, as a rule, is personalized taking into account the priorities and preferences of users, considered in the context of the current situation and the state of the environment.

The elaboration on the Semantic Web concept and related concepts, such as Web 3.0 [10] and the Semantic Web of Things [11], defines the direction for the elaboration on SmS services towards semantic services. The description of a semantic service is represented by a machine-interpreted service ontology. SmS services can be defined as semantic services, which must have uniquely described semantics, be available among other heterogeneous environments, be suitable for automated search, composition, proactive construction and proactive delivery. The use of semantic services in the SmS approach changes the requirements for the development of services, and therefore, SmS applications. In connection with the constantly growing and dynamically changing set of participants in the SmS environment, the complexity of the phases of development and maintenance of services increases.

Unified service ontology. The design of semantic services should be based on a general unified ontology. Such an ontology defines not only the service interface in terms of transmitted data and return values, but also defines the purpose of the service, describes the process of its construction, and uniquely determines its semantics. With this consistent design approach, the services of different SmS applications can interact with each other regardless of the domain area and environment. Providing in this way the network interaction of the SmS environments, it is possible to achieve the integration of both the SmS themselves and their applications for solving collaborative tasks based on semantic services.

Automation of agent programming processes. The way to develop applications is needed that allows to reduce the amount

of program code generated by an application developer during routine tasks through the use of computer-aided design and programming tools. In particular, the automation of agent programming processes when constructing services can be achieved through the use of semantic service ontologies. Ontologies are accepted as input parameters to generate an object model and code templates for object-oriented programming languages. By understanding the semantics of the service, as well as information about the available resources of the environment, ontology-based self-organization of agents can be achieved by defining their functional roles, interaction models and operations/functions in the process of constructing and delivering a service.

The paper proposes a solution to the problem of simplifying the development and maintenance of SmS applications by creating tools for automated code generation of multi-agent interaction for constructing semantic services.

The rest of the paper is organized as follows. Section II introduces the general scheme of automated code generation process of multi-agent interaction for constructing semantic services. Section III provides the ontology of semantic service in SmS. Section IV proposes the code generation procedures for agent data object model and interaction processes. Section V evaluates the developer efforts in automated development of semantic services through the use of the proposed unified service ontology and the code generator. Finally, Section VI concludes the paper.

II. AUTOMATIC ONTOLOGY-DRIVEN DEVELOPMENT

The development of SmS applications follows the principles of ontology-driven software development. According to these principles, the design phase is reduced to the creation of a specification for a specific domain and services in the form of an OWL/RDF description. The use of ontologies allows to achieve a common understanding of the structure of information storage between agents to facilitate knowledge reuse through concepts already defined in other ontologies, as well as support for formal logic and logical reasoning [12]. Thus, it is beneficial to use the features of the ontology-driven approach in the case of using ontologies at all phases of development.

To automate the development stages, traditional design and development methods are being replaced by methods that facilitate the implementation of an approach with extensive use of Computer-Aided Design (CAD) and Computer-Aided Programming (CAP) tools. Such tools support application prototyping.

CAD tools are are being used to automate processes aimed at creating and maintaining various ontological and graphical representations during of application systems design. In turn, CAP tools are being used to simplify the task of programming agents. Rather than directly coding up executable programs for software agents, the developer provides an ontology with a problem domain and service specification allowing code generation algorithms to create correct code functions, data structures, and other elements of the specified programming language. The integrating efforts of CAD and CAP tools will bring automated program-code generation directly from design-phase specifications. CAD and CAP tools can also be distributed together with a middleware providing an integrated environment for building/deploying and managing applications, such as in the OpenIoT middleware [3].

The design phase is reduced to creating a specification of a problem domain and services as an RDF/OWL description. There is a large number of works aimed at solving CAD problems at the design phase for ontology-driven software development [12][13]. For example, ontology development tools, such as Protégé [14] and OWL-S Editor [15] allow users to create these specifications and provide guidance to find mistakes based on validation mechanisms. These tools serve as rapid prototyping environments, in which ontology designers can instantly create individuals of their ontology and experiment with semantic restrictions, and enable developers to visualize descriptions in a graphical manner are even able to generate user interfaces that can be further customized for knowledge acquisition in a particular domain.

At the implementation phase, which involves the use of programming languages to encode the resulting design solutions, software agents from the design specifications and models are created. At this phase, solving the CAP tool [16] is not enough for an automated task of ontology-driven programming of agents. Existing solutions for ontology-driven software development solve this problem in part by mapping OWL classes, their instances and properties to programming language classes, their objects, and fields, respectively [14].

Obviously, this approach is difficult both for practical implementation and for use in the case of statically typed compiled programming languages. However, it is convenient for dynamically typed interpreted and object-oriented programming languages. This approach is primarily intended for creating data structures and elements of the object model of an agent problem domain. However, it does not allow creating methods, functions and other elements of internal program logic. One of the examples of this approach is SmartSlog CodeGen, which is part of the SmatSlog ontology library designed for creating SmS applications. Its mechanisms allow creating data structures for particular OWL ontology entities.

A solution is proposed, aimed at creating the program code generator for agents based on ontology using object-oriented programming languages (e.g., C++, Java, Python). The general scheme of the program code generation process is shown in Figure 1. The main features of the proposed code generation scheme are: (i) use as input ontologies, together with OWL domain ontologies, service ontologies for SmS based on the OWL-S ontology [15]; (ii) generation, in addition to data structures, elements of the program logic of agent interaction based on the API of the SmS middleware for the purpose of constructing and delivering services, as well as an object model of the domain.

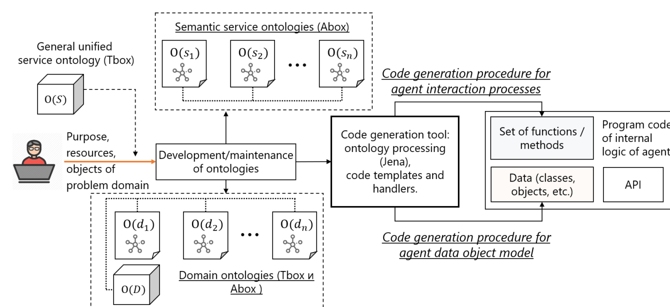

Figure 1. The general scheme of automated code generation process of multi-agent interaction

An agent developer provides a problem domain specification as an OWL description and a specification of SmS services as an OWL-S-based description. The generator uses a static pattern of templates and handlers. Code templates are "pre-code" of data structures, classes and functions that implement OWL-S-based and OWL ontologies entities and their properties.Handlers can transform one or more templates into the final code replacing specialized tags with names and elements taken from ontologies. The code can be generated for several agents, depending on the agents involved in constructing and delivering the services described with the OWL-S-based specification. The transformation should occur during ontology RDF graph traversal based on Jena OWL framework. Jena constructs a meta-model to represent the graph. The generator should traverse this model comprehensively, and those nodes are visited that a handler needs to transform its templates into final code.

III. Ontological Semantic Services Model

SmS allow effectively organizing the interaction of its participants and their sharing of information [1]. In particular, a virtual digital image is created for each participant, which is semantically linked with other participants and resources. The service ontology describes the context of the environment and its participants, interacting agents and the resources involved. The ontological model is created on the basis of a conceptual description of semantic web services using models for organizing information-driven interaction of agents, defined within the SmS approach. Due to such a unified ontological description of service construction processes, an increase in the quality of service design is achieved. Services become suitable for automated search, composition, construction and delivery to users by describing the interface (service purpose, inputs and outputs, etc.) and the processes that occur.

The World Wide Web Consortium (W3C) has introduced an ontology built on top of OWL to describe semantic web services. This ontology is called OWL-S [17]. The main purpose of OWL-S is to allow users to discover, invoke, compose, and monitor with a high degree of automation Web resources offering particular services and having particular properties,using a minimalistic approach for describing semantic Web Services. OWL-S describes the characteristics of a service by using three top-level concepts: a service profile, a service model, and a service grounding.

The purpose of a service profile is to define a service in a uniform way for future use, detailing the content of queries and conditions under which specific results will occur, and, if necessary, step-by-step processes leading to these results. The process model describes how to access a service and what happens when the service is executed. The service process model describes a service as a collection of atomic and composite processes. An atomic process corresponds to a "one-step" procedure that takes input parameters, processes them, and then returns a result. Composite processes decompose into other processes; by defining their decomposition using control constructs (for example, *If-Then-Else*, *Split*, *Repeat-While*). The service grounding defines how the service is invoked by the consumer of the service, including the communication protocol and message format. In the case of the M3 architecture [18], the service foundation ontology is an optional part, since the interaction of agents is organized using the Smart Space Access Protocol (SSAP) protocol, which does not require a separate description.

Figure 2. Ontological model of semantic service in SmS.

A unified ontological description of the service semantics for SmS is proposed, containing a set of terminological axioms (see Figure 2). A service profile describes a service using Inputs, Outputs, Preconditions, Effects (IOPEs). The *has_output* property is used to represent the output from a service resource to a consumer. If a service needs to specify input data for processing by a resource or control action in the form of notifications, then this is determined using the *has_input* property.

The initial state of the shared information storage for initializing the process of constructing a service is set using a precondition (*has_precondition*). The desired state of the storage is determined using the resulting condition (*has_result*). Preconditions and results are specified using expressions. There are several possible approaches that allow using rules and logic for the RDF/OWL view [19]. The main idea is to treat expressions as literals using Protocol and RDF Query Language (SPARQL) expressions, which are defined using the SPARQL-Expression class.

The *has_category* property describes service categories based on SmS service categories. Categories are determined based on criteria, such as impact, underlying resource, way of integration, function performed. The category defines a typical model of information-driven interaction of agents [1].

OWL-S ontology is very flexible, but OWL-S solutions are not enough to use it fully to describe the semantics of a SmS service. In order to take into account the priorities and preferences of users (if the customer is a human being) providing the personification of a service, the existing service profile is expanded by the *has_consumer* property and its *UserProfile* class object, which is described both in terms of user contextual information. Such information reflects the state of the user's computing environment (*has_context*), as well as his preferences, interests, personal information represented by part of the Friend of a Friend (FOAF) ontology (*has_personinformation*). The OWL-S ontology also does not support a multi-agent approach for describing a service process model. The vision of agents in the OWL-S ontology is reduced: agents are considered primarily as consumers or "seekers" of services. At the same time, the process of constructing a service in the SmS approach is a process of interaction of several agents.

The service process ontology is extended by introducing the *ServiceAgentsModel* class to describe the process of

constructing services for SmS. The process is described by a model of information-driven interaction of agents, which is determined depending on the service category. The interaction of each agent (*KnowledgeProccessor*), represented as an extension of the participant's class, is determined by its functional role (*has_role* property) performed in the model. A functional role is an abstract description of the functional properties of an agent. The role of the agent defines the general principles of implementation of the individual internal logic of the agent, as well as the principles of interaction with other agents. The logic of a separate agent leads to the interaction of agents during the construction of a service based on interaction patterns (*AgentsPattern*) described using such architectural abstractions as Provider-Consumer (PC), Pipe, Tree, Flow. The process of interaction between agents can consist of several patterns presented in a certain sequence (*AgentsPatternBag*).

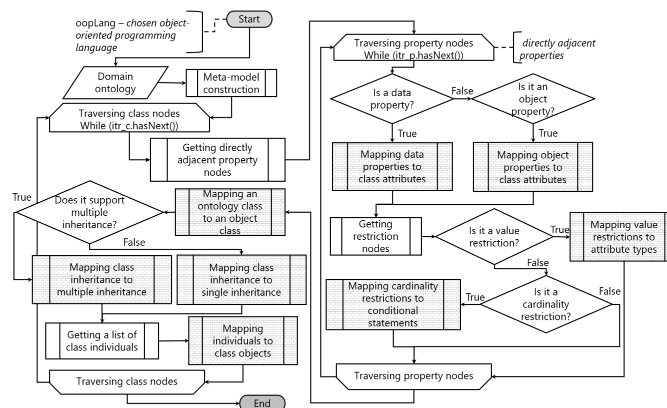

Figure 3. The code generation procedure for an agent data object model.

IV. AUTOMATING PROGRAMMING OF MULTI-AGENT INTERACTION BASED ON CODE GENERATION

The automation of programming processes for agents involved in constructing and delivering services is achieved through the use of a program code generator. As a result of the design stage of SmS application, the developer has a set of ontologies, which are divided into two groups: (1) the domain ontology and (2) the service ontology for SmS based on OWL-S. Ontologies provide the necessary semantics that are used to generate code in object-oriented programming languages. The generator uses algorithms for automating agent programming processes to implement the structures of the object model and for the agents interaction. The code generation procedure for an agent data object model takes the domain ontology as an input parameter. The code generation procedure for agent interaction processes takes as an input parameter the service ontology to generate blocks of agent program code.

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. 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 [20].

The code generation procedure for an agent data object model is presented in the flowchart (see Figure. 3). he 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 the structures of an object-oriented programming language.

Constructing SmS service can be viewed as a set of calls to agents' software functions. The service ontology for SmS based on OWL-S provides a declarative, computer-interpreted description that includes the semantics of the IOPEs model that must be specified for each process. The process entity can be used to generate procedures, functions and other elements of the target programming language that implement information-driven interaction and the necessary internal logic of agents. The code generation procedure for agent interaction processes in the flowchart is shown in Figure 4.

Instances of the *AtomicProcces* class are used to generate function code. The *rdf:ID* attribute of the AtomicProcess class defines the function name. Each *has_input* property with *rdf:ID* attribute corresponds to the input parameters of the function. The type of the input parameter can be obtained by extracting at the *parameter_type* property.

The functions internal logic is implemented using SPARQL queries and sets of program statements. The *has_precondition* property with a SPARQL expression defines a precondition code block that describes the initial state of the information storage. A precondition block is required to initialize the service construction. A code block is generated that calls the API middleware (SmS platform) to execute a SPARQL query (usually an ASK query) and verifies the query result using an "if-then-else" statement. A similar generation process is performed for a code block representing a result condition (*has_result*) — a set of actions performed at the end of a function call.

The *has_output* property with the *parameter_type* property corresponds to the output parameter and defines the function return value. The required data types (string, unsignedLong, etc.) are described using an XML Schema Definition (XSD) schema. Elements of the object model can be used as input and output parameters. The *CompositeProcess* class, by analogy with a composite process, defines a function that calls other functions within itself, which are described by *CompositeProcess* or *AtomicProcess* entities. In this case, calls to internal functions can be specified using control constructs (*If-Then-Else*, *Repeat-While*, etc.) which can be transformed into the corresponding statements of the programming language.

An instance of the ServiceAgentsModel class is used by agents to define their role in the process of constructing and delivering services, as well as a method of information-driven interaction based on the publish subscribe model. For this purpose, in advance, in the code of each agent a block is formed with the necessary subscription operations using internal functions, handlers, and API functions. Each subscription operation query is specified by a SPARQL query that can be obtained from the *has_subprecondition* property, where a subscription expression is specified using SPARQL queries that use domain classes and properties. In addition,

Copyright (c) IARIA, 2020.

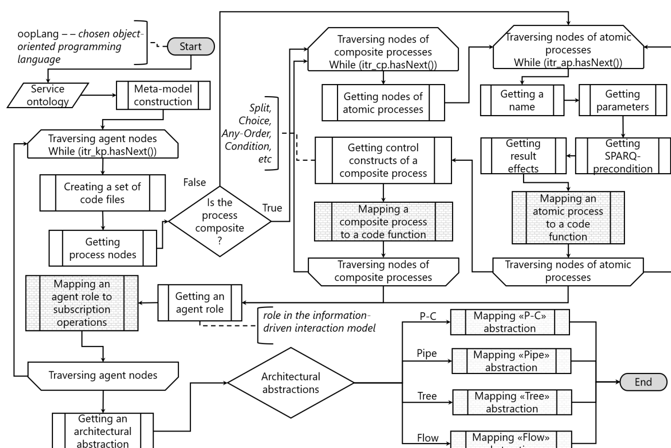

Figure 4. The code generation procedure for agent interaction processes.

subscription operations can be defined based on the execution results of agent processes defined in the *Result* classes. Each such class defines what domain changes are produced during the execution of an agent. During the execution of agents, their self-organization can occur as each agent is aware of its purpose in the interaction model based on interaction patterns (*AgentsPattern* class).

V. Effort Estimation

Efficiency is the relationship between the results achieved, and the resources used. Efficiency of the proposed solutions is determined on the basis of effort estimation in automated development of semantic services through the use of the unified service ontology and the generator of the agent interaction program code. For effort estimation, any unit of measurement of the duration of ongoing processes can be used. The effort in semantic service development are considered in two phases: design and implementation.

The main stages in a design phase of multi-agent software systems are:

1) defining the roles of agents and their functional description;
2) conceptual modelling of inter-role interaction based on the selected protocol;
3) modelling the interaction between the user and the system, and defining the access interface;
4) creating the code structure for each agent and the system as a whole.

The use of the proposed solutions by the developer makes it possible to fix the obtained design decisions (agent roles, protocol and model of agent interaction, service interface) while directly creating an ontological description of services in unified terms. It is known that the use of ontologies at the design phase increases the developer's efforts to create design solutions. However, some additional efforts can be minimized, while others provide additional opportunities at the next development phases (e.g., programming automation, agents self-organization). One way to minimize efforts is to use existing computer-aided design tools (such as Protégé), which provide a software environment for rapid prototyping.

Additional design efforts allow obtaining uniform service ontologies that define the interaction interface and describe the execution semantics. With this unified design approach, services across domains can interact with each other independently of the computing environment. The use of the solutions is not limited to the design phase. The service ontologies are used to automate further service programming processes (creating an object model, code functions).

TABLE I. Proportion of generated code for services.

Service	Agents and their roles		Object data model		Information-driven interaction		Internal logic		Total	
			SLOC	%	SLOC	%	SLOC	%	SLOC	%
User presence and activity service ($\mathbf{S_{prs}}$)	Presence processor adapter-agent	all	26	8,8	78	26,4	191	64,8	295	100
		gen.	21	7,1	40	13,6	9	3	**70**	**23,7**
	Presence detector aggregator-agent	all	18	11,2	44	27,5	98	61,2	160	100
		gen.	14	8,7	18	11,3	5	2,8	**37**	**23,1**
	Activity monitor-agent	all	74	13,2	88	15,7	392	71,1	560	100
		gen.	52	9,3	75	13,4	13	2,3	**140**	**25**
Historical data enrichment service ($\mathbf{S_{enr}}$)	External finder-agent	all	65	9,6	162	23,9	451	66,5	678	100
		gen.	50	7,4	103	15,2	7	1	**160**	**23,6**
	Semantic controller-agent	all	222	12,6	515	29,2	1028	58,2	1765	100
		gen.	148	8,4	339	19,2	23	1,3	**510**	**28,9**
	Enrichment aggregator-agent	all	391	11,9	898	27,3	2001	60,8	3290	100
		gen.	296	9	513	15,6	26	0,8	**835**	**25,4**

Programming effort is investigated based on the ratio of the total number of lines of agent source code to that automatically generated using the proposed implementation of the program code generator for the following service: user presence and activity service (S_{prs}) and historical data enrichment service (S_{enr}). Table I provides the percentage of generated program code for agents involved in the implementation of services. The program code, regardless of the role of the software agent, consists of the following blocks:

1) structures of the object data model and methods for working with them;
2) information-driven interaction based on supported operations for middleware;
3) internal logic, including local processing of general information.

The average share of generated program code was 23.4%, with the greatest results falling on the "information-driven interaction" block. For the object data model, the generator also shows high rates, the average coverage percentage of the corresponding source code with the generated code is 68%.

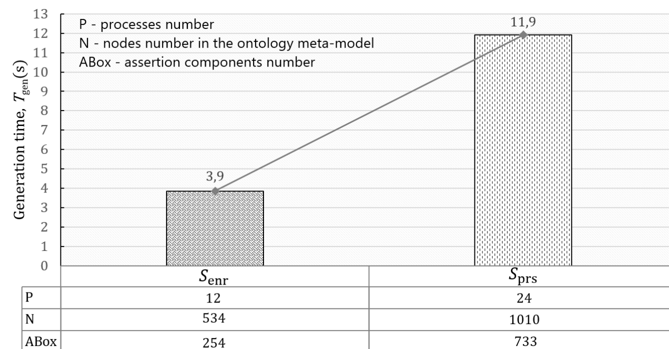

Figure 5. Time to generate program code for service implementations.

The process of programming services imposes additional efforts associated with the time spent on the process of generating program code. The generator accepts the developed service

ontologies and the domain ontology as input parameters. The generation time of the program code depends on ontological metrics (cyclomatic complexity, vocabulary size) that characterize the complexity of traversing the ontology meta-model (ontological graph). Experiments have shown (see Figure 5) that when processing the ontology for the user presence and activity service (S_{prs}), which has the highest metrics from the developed ontologies, the time for traversing the meta-model and generating code does not exceed 12 s.

TABLE II. QUALITY METRICS OF GENERATED CODE.

Metric	*Value*
Total number of generated code lines	**49**
Cyclomatic complexity	**4**
Number of distinct operators: η_1	13
Number of distinct operands: η_2	17
Total number of occurrences of operators: N_1	26
Total number of occurrences of operands: N_2	41
Halstead vocabulary: $\eta = \eta_1 + \eta_2$	30
Halstead program length: $N = N_1 + N_2$	67
Program volume: $V = N * \log_2 \eta$	406
Program difficulty: $D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$	15.67
Programming effort: $E = D * V$	6362
Programming time (seconds): $T = \frac{E}{18}$	**353.5**

The quality of the generated code is investigated on the basis of calculating and evaluating the quality metrics of the code. The following quality metrics of the generated program code are measured: cyclomatic complexity and Halstead's metrics [21]. The measured metrics made it possible to estimate: the complexity of maintaining the generated code, efforts to create the code manually. Table II shows the measured metrics for one simple process of S_{prs} service. The cyclomatic complexity is 4, the estimated time to create such a code manually is 353.5 seconds.

VI. CONCLUSION

This paper proposed a solution to the problem of simplifying the development and maintenance of smart space applications by creating tools for automated code generation of multi-agent interaction for constructing semantic services. The general scheme of automated code generation process of multi-agent interaction for constructing semantic services was introduced. By expanding the OWL-S ontology, a unified ontological description of the semantics of service constructing processes was introduced. The code generation procedures for agent data object model and interaction processes were presented. The efforts in automated development of semantic services were investigated based on estimation of time to generate and the quality metrics of generated code.

ACKNOWLEDGMENT

The reported research study is supported by RFBR (research project # 19-07-01027). The work is implemented within the Government Program of Flagship University Development for Petrozavodsk State University (PetrSU) in 2017–2021.

REFERENCES

[1] D. G. Korzun, S. I. Balandin, A. M. Kashevnik, A. V. Smirnov, and A. V. Gurtov, "Smart spaces-based application development: M3 architecture, design principles, use cases, and evaluation," International Journal of Embedded and Real-Time Communication Systems (IJERTCS), vol. 8, no. 2, 2017, pp. 66–100.

[2] D. Korzun, "On the smart spaces approach to semantic-driven design of service-oriented information systems," in International Baltic Conference on Databases and Information Systems. Springer, 2016, pp. 181–195.

[3] J. Soldatos et al., "Openiot: Open source internet-of-things in the cloud," in Interoperability and Open-Source Solutions for the Internet of Things. Springer International Publishing, 2015, pp. 13–25.

[4] J. Guia, V. G. Soares, and J. Bernardino, "Graph databases: Neo4j analysis." in ICEIS (1), 2017, pp. 351–356.

[5] L. Roffia et al., "Dynamic linked data: A sparql event processing architecture," Future Internet, vol. 10, no. 4, 2018, p. 36.

[6] J. Lanza et al., "A proof-of-concept for semantically interoperable federation of iot experimentation facilities," Sensors, vol. 16, no. 7, 2016, p. 1006.

[7] I. Galov, A. Lomov, and D. Korzun, "Design of semantic information broker for localized computing environments in the Internet of Things," in Proc. 17th Conf. of Open Innovations Association FRUCT. IEEE, Apr. 2015, pp. 36–43.

[8] D. Korzun, S. Yalovitsyna, and V. Volokhova, "Smart services as cultural and historical heritage information assistance for museum visitors and personnel," Baltic Journal of Modern Computing, vol. 6, no. 4, 2018, pp. 418–433.

[9] S. A. Marchenkov, A. S. Vdovenko, and D. G. Korzun, "Enhancing the opportunities of collaborative work in an intelligent room using e-tourism services," Trudy SPIIRAN, vol. 50, 2017, pp. 165–189.

[10] A. Gyrard, M. Serrano, and G. A. Atemezing, "Semantic web methodologies, best practices and ontology engineering applied to internet of things," in 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015, pp. 412–417.

[11] P. Kujur and B. Chhetri, "Evolution of world wide web: Journey from web 1.0 to web 4.0," International Journal of Computer Science and Technology, vol. 6, Jan. 2015.

[12] 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, Feb. 2017, pp. 668–679.

[13] 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, 2015, pp. 863–869.

[14] 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.

[15] D. Elenius et al., "The owl-s editor–a development tool for semantic web services," in European Semantic Web Conference. Springer, 2005, pp. 78–92.

[16] A. Lomov, "Ontology-based kp development for smart-m3 applications," in 2013 13th Conference of Open Innovations Association (FRUCT). IEEE, 2013, pp. 94–100.

[17] D. Martin et al., "Bringing semantics to web services with owl-s," World Wide Web, vol. 10, no. 3, 2007, pp. 243–277.

[18] 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.

[19] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," Journal of Web Semantics, vol. 5, no. 2, 2007, pp. 51–53.

[20] 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.

[21] T. Hariprasad, G. Vidhyagaran, K. Seenu, and C. Thirumalai, "Software complexity analysis using halstead metrics," in 2017 International Conference on Trends in Electronics and Informatics (ICEI). IEEE, 2017, pp. 1109–1113.

Copyright (c) IARIA, 2020. ISBN: 978-1-61208-811-2