Evaluation of Program Code of Smart-M3 Knowledge Processors Developed Using the SmartSlog Tool

Aleksandr A. Lomov and Dmitry G. Korzun
Petrozavodsk State University (PetrSU)
Petrozavodsk, Russia
{lomov, dkorzun}@cs.karelia.ru

***Abstract*—The SmartSlog is a software development tool for programming Smart-M3 agents (Knowledge Processors). SmartSlog applies the model-driven code generation approach. Given OWL ontologies of agent problem domain, SmartSlog produces the ontology library as middleware for agent developer. SmartSlog ontology library allows easier constructing the program code. The developer thinks in agent domain ontology terms (classes, relations and individuals) instead of RDF triples, as it happens in the low-level development approach of Smart-M3. In this talk, we focus on performance evaluation and on measurement of program code metrics to show effectiveness of SmartSlog tool.**

I. Introduction

A smart space is a virtual, service-centric, multi-user, multi-device, dynamic interaction system that applies a shared view of resources in a given computing environment [1], [2]. Smart-M3 [3] is an open software platform that implements smart spaces, focusing on the multi-device, multi-vendor, and multi-domain properties of modern computing environments and advanced digital services. Software agents, which are called knowledge processors (KP), run on devices of the computing environment. A smart space (SS) is served by a semantic information broker (SIB). KPs act cooperatively accessing and generating information. SS content is represented using the RDF model from the Semantic Web. Interaction of KPs is supported by the publish/subscribe model, and any KP can persistently detect changes in the shared content of the KP's interests [4].

SmartSlog [5] is a software development tool for programming KPs and their interaction in the smart space. There are two approaches for the KP development (Fig. 1). KP logic is programmed on the level of RDF triples or on higher level using domain terms represented in the code as ontological classes, properties, and individuals. On the RDF level, the SS access primitives are provided by KP interface (KPI). On the high-level, SS access primitives are ontology-oriented and provided by middleware. The

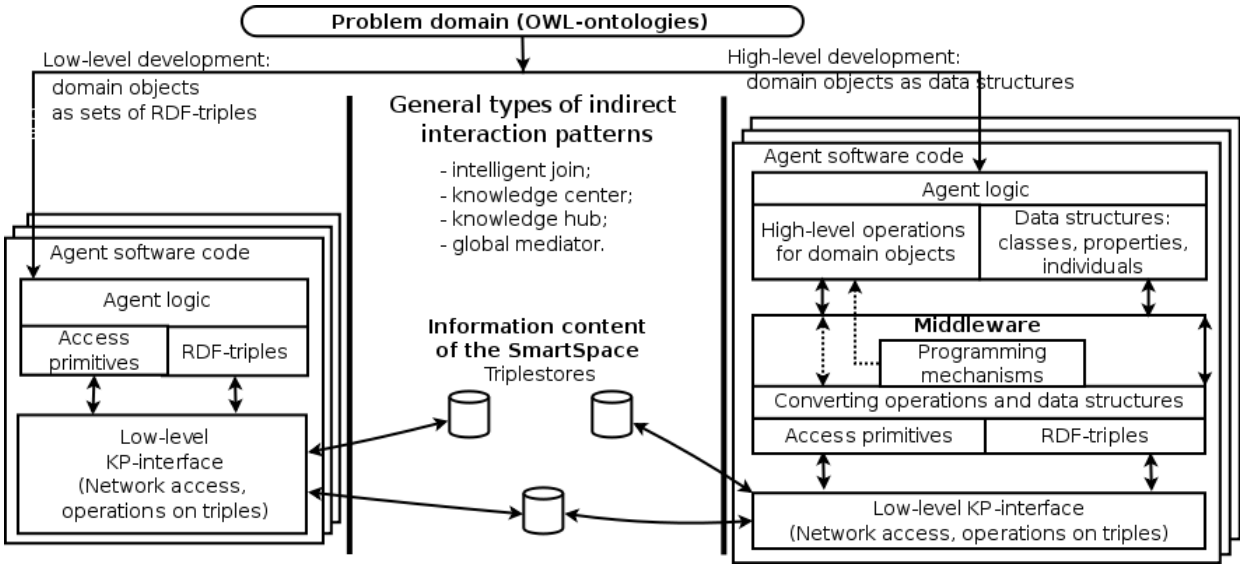

Fig. 1. Approaches for smart spaces agent development

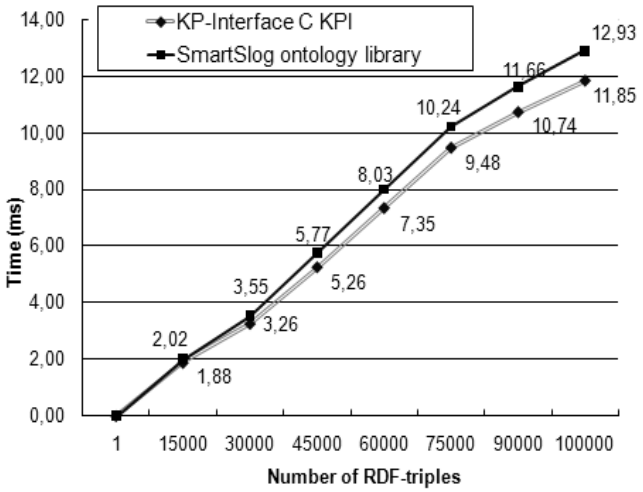

Fig. 2. CPU-Time for processing RDF-triples incoming from smart space

middleware converts operations to access primitives of the given KPI and data structures to RDF triples.

SmartSlog includes a code generator for such middleware. The latter is called SmartSlog ontology library. In particular, it implements programming mechanisms to automate programming of interaction of KS in a smart space. There are four general interaction patterns [6]: intelligent join, knowledge center, knowledge hub, and global mediator. Note that such interaction is indirect; instead of direct exchange the information is shared in the smart space.

To check the effectiveness of SmartSlog for KP development three groups of experiments were performed. The first group of experiments aims at evaluation of the performance. In KP runtime execution, its SmartSlog ontology library needs extra CPU resources to convert ontology-oriented operations and structures to RDF-oriented access primitives of KPI and vice versa. Fig. 2 shows the time required to process incoming triples depending on the number of locally stored RDF triples (for low-level C KPI) or objects that represent these RDF-triples (for SmartSlog). The average decrease of performance is 7% (0.15 ms). Note that this property only slightly increases the execution time and does not affect the SmartSlog programmability of KPs for various devices.

The second group of experiments measures the amount of programming operations to implement indirect interaction. An operation is defined as one complete action (creating an individual or a set of triples). The experiment result is shown in Fig. 3. For SmartSlog ontology library an operation is a line of code, for C KPI developer such an operation takes several lines of code. The average reduction in the number of operations to program is about 39%.

The third group of experiments shows how KP's program code can be simplified if using Smartslog ontology library. We estimate the cyclomatic complexity [7] for basic examples of KPs, as shown in the Table I. The "Hello Word" example for C KPI with synchronous subscription is more complex than the same example for SmartSlog with asynchronous subscription. To use asynchronous subscription with C KPI the developer manually organizes concurrent threads in the KP's logic code. In this case the cyclomatic complexity is much higher. Notably that the "GPS" example, which uses some features of SmartSlog ontology library (subscription to properties and classes, tracking network errors), has almost the same complexity as the simple example "Hello World" for C KPI.

For further experiments with SmartSlog, we consider to extend the third group with Halstead [8], [9] and Jibs [10] metrics. The Halstead metric counts operators, keywords (return, if, continue), identifiers, and constants. The Jibs metric is defined as saturation of the program code with such expressions as IF-THEN-ELSE. These metrics allow determining which parts of low-level KP code can be programmed more effectively. Also, these metrics can be used for improvement of SmartSlog ontology library by changing some of the functionality to provide the developer more effective ways for KP programming.

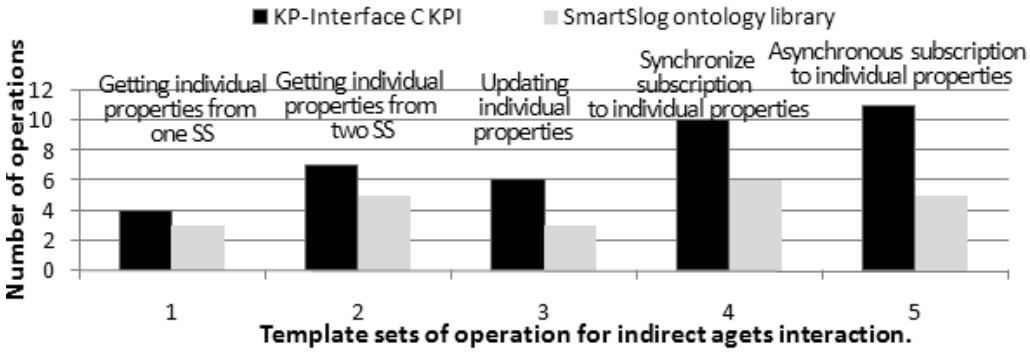

Fig. 3. Cyclomatic complexity for base KP examples

TABLE I. NUMBER OF OPERATIONS FOR VARIOUS ACTIVITIES OF THE INDIRECT INTERACTION

Example implemetation	C KPI		SmartSlog ontology library		
	Hello World		Hello World	GPS	
	Without subscription	Synchronous subscription	Asynchronous subscription	Asynchronous subscription	Connection reconnect
Lines of code	49	144	68	173	194
Cyclomatic complexity	13	37	19	38	40

ACKNOWLEDGMENT

This research is financially supported by project # 1481 from the basic part of state research assignment # 2014/154 of the Ministry of Education and Science of the Russian Federation and by research project # 14-07-00252 of the Russian Fund for Basic Research. The authors are grateful to the Open Innovations Association FRUCT for its support and R&D infrastructure. We would also like to thank Alexander V. Smirnov, Andrey A. Pechnikov, Andrew A. Krizhanovsky, and Alexey M. Kashevnik for their feedback and expertise.

REFERENCES

[1] A. Smirnov and A. Kashnevik and N. Shilov and I. Oliver, S. Balandin and S. Boldyrev, "Anonymous agent coordination in smart spaces: Stateof-the-art", in *Proc. 9th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'09) and 2nd Conf. Smart Spaces (ruSMART'09)*, LNCS 5764. Springer-Verlag, 2009, pp. 42–51.

[2] S. Balandin and H. Waris, "Key properties in the development of smart spaces," in Proc. 5th Int'l Conf. Universal Access in Human-Computer Interaction (UAHCI '09*). Part II: Intelligent and Ubiquitous Interaction Environments*, LNCS 5615. Springer-Verlag, 2009, pp. 3–12.

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

[4] A. A. Lomov and D. G. Korzun, "Subscription operation in Smart-M3*", Proc. 10th Conf. of Open Innovations Association FRUCT and 2nd Finnish–Russian Mobile Linux Summit. SUAI*, Nov. 2011, pp. 83–94.

[5] G. Korzun and A. Lomov and P. Vanag and S. Balandin and J. Honkola. "Generating Modest High-Level Ontology Libraries for Smart-M3", *Proc. 4th Int'l Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, October 25 - 30, 2010, Florence, Italy, pp. 103-109.

[6] M. Murth and E. Kühn. "Knowledge-Based Interaction Patterns for Semantic Spaces", *In Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '10). IEEE Computer Society*, pp. 1036-1043.

[7] A. H. Watson and T. J. Mccabe and D. R. Wallace. *"Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric",* Technical Report NIST Special Publication 500-235, National Institute of Standards and Technology (NIST), September, 1996.

[8] H. Halstead, *Elements of Software Science. Amsterdam: Elsevier North-Holland*, 1977.

[9] A. Serebrenik, Software metrics, Web: http://www.win.tue.nl/~aserebre/2IS55/2010-2011/10.pdf

[10] R. B. Hassan. Automatic Measurement of Source Code Complexity. Master of Science Computer Science and Engineering. Web: https://pure.ltu.se/portal/files/33040726/LTU-EX-2011-32994156.pdf