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
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

<table>
<thead>
<tr>
<th>Example implementation</th>
<th>C KPI</th>
<th>Smarthlog ontology library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hello World Without subscription</td>
<td>Hello World Synchronous subscription</td>
</tr>
<tr>
<td>Lines of code</td>
<td>49</td>
<td>144</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>13</td>
<td>37</td>
</tr>
</tbody>
</table>

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


