Multilingual Ontology Library Generator for Smart-M3 Application Development

Aleksandr A. Lomov, Pavel I. Vanag, Dmitry G. Korzun

Petrozavodsk State University
Department of Computer Science

9th FRUCT Conference, April 25–29, Petrozavodsk, Russia
# Table of Contents

1. Smart-M3 platform and applications
2. SmartSlog tool
3. Ontology manipulations
4. Optimizations for generated code
5. Conclusion
Smart-M3 platform

A kind of publish/subscribe system

- Semantic information brokers (SIBs) maintain SS content in low-level RDF triples

- Application consists of several knowledge processors (KPs) running on each device

- Smart space access protocol (SSAP) for SIB↔KP communication

- Smart-M3: Multidomain, Multidevice, Multivendor
## KP development tools

<table>
<thead>
<tr>
<th>Low-level (RDF triple)</th>
<th>High-level (OWL object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whiteboard, Whiteboard-Qt</td>
<td>Smart-M3 ontology to C-API generator</td>
</tr>
<tr>
<td>(C/Glib, C/Dbus, C++/Qt) (Smart-M3)</td>
<td>(C/Glib, C/Dbus) (Smart-M3)</td>
</tr>
<tr>
<td>Smart-M3 Java KPI library</td>
<td>Smart-M3 ontology to Python generator</td>
</tr>
<tr>
<td>Java (University of Bologna and VTT)</td>
<td>Python (Smart-M3)</td>
</tr>
<tr>
<td>M3-Python KPI (m3_kp)</td>
<td>SmartSlog</td>
</tr>
<tr>
<td>Python (Smart-M3 distribution)</td>
<td>ANSI C, C# (Petrozavodsk State University)</td>
</tr>
<tr>
<td>C# KPI library</td>
<td></td>
</tr>
<tr>
<td>C# (University of Bologna)</td>
<td></td>
</tr>
</tbody>
</table>
The Problem

- **Simplifying KP code using high-level OWL terms**
  - SIB uses low-level RDF triples
  - KP uses high-level abstractions

- **Speed development of huge amount of KPs**
  - Multilingual support
  - Cross-platform code generation

- **Target devices could be low-performance**
  - Subset of ANSI C version
  - Modest code schemes

These criteria are controversial, efficient tradeoff is a challenging problem
High-level scheme

SmartSlog tool

Problem domain specification.
OWL ontology
- classes
- relations

Jena
OWL framework (Java based)

SmartSlogCodeGen
Based on Smart-M3 ontology to C-API generator

SmartSlogCodeGen
- templates
- handlers

MetaModel
Representation of ontology
RDF graph

is visited according to a set of rules

uses

is input to

SmartSlog ontology Lib
- classes
- operations
- API

uses

uses

1 provides

think in terms of

developer

Optionally:

Makefile, KP template

2 writes

exploits

Device
running KP

- insert/remove/update
- subscribe/unsubscribe
- query

Low-level KPI
- SSAP operations
- RDF triplets support

Aleksandr Lomov
SmartSlog
FRUCT 9th 29.04.2011 6 / 18

Smart-M3 Space
- instances of classes
defined in the OWL ontology
Code Generation

Refers to a class of source code generators
Transformation approach of automatic programming

- Java-based CodeGen
- Static templates/handlers scheme

- **Templates** are “pre-code” of data structures
  - implementation of ontology classes
  - implementation of properties for classes
  - tags ⟨name⟩ instead of proper ontology names
  - dependence on the mediator library (KP ↔ SIB)
    (SmartSLog uses KPI_low library)

- **Handlers** transform templates into final code
  - Replacing tags with the names taken from the ontology
  - Executed when the ontology graph is analyzed
    (CodeGen calls Jena framework)
## OWL mapping to code

Multilingual library generator for **Smart Space ontology**

- ANSI C ontology library (low-performance devices)
- C# ontology library (.NET framework, Windows OS)

<table>
<thead>
<tr>
<th>ontological class structure</th>
<th>ANSI C code</th>
<th>C# code</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct class_s {...} class_t;</td>
<td>class OntClass {...}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ontological property of class structure</th>
<th>ANSI C code</th>
<th>C# code</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct property_s {...} property_t;</td>
<td>class Property {...}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ontological individual structure</th>
<th>ANSI C code</th>
<th>C# code</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct individual_s {...} individual_t;</td>
<td>class Individual {...}</td>
<td></td>
</tr>
</tbody>
</table>
Easy to develop: mobile phone KP code

1. Individual creation, property setting and sending it to SS:

   individual_t *mobile = new_individual(CLASS_MOBILEPHONE);
   set_property(mobile, PROPERTY_NAME, "mob");
   set_property(mobile, PROPERTY_ISCALLING, "false");
   ...   
   ss_insert_individual(mobile);

2. Waiting for income call and property updating:

   while(1) {
     wait_call();
     ss_update_property(mobile, PROPERTY_ISCALLING, "true");
     wait_call_ending();
     ss_update_property(mobile, PROPERTY_ISCALLING, "false");
   }
Easy to develop: music player KP code

1 Individual creation, property setting and sending it to SS:

```c
individual_t *mobile = new_individual(CLASS_MOBILEPHONE);
individual_t *player = new_individual(CLASS PLAYER);
```

2 Subscribe to property:

```c
subscription_container_t *container =
new_subscription_container();
add_data_to_list(prop_list, PROPERTY_ISCALLING);
add_individual_to_subscribe(container, player, prop_list);
ss_subscribe_container(container, true);
```

3 Check if phone calling and turn on/off volume:

```c
while(1) {
    is_calling = get_property(mobile, PROPERTY_ISCALLING);
    if (is_calling == "true") set_sound_on(false);
    else set_sound_on(true);
    wait_subscribe(container_counter);
}```
Implemented optimizations

1. Generating ontology dependent part
2. Available ontology independent part (.a or .so library)
3. Memory control
4. Local data structures
5. Threading
Ontology composition

- Ontology integration:
  - Complete integration
  - Partial integration

Developer can manipulate with several knowledge sets in the smart space via a single KP.
Ontology composition

- **Bridge properties:**
  - Many properties represented by one
  - Access the same data through active property
  - Background transformation of data
Ontology composition

- KP controller:
  - Controls ontology entities that are shared other KPs
  - Controls state of other KP
  - Decides further control actions

Case studies:
- Smart conference
- Smirnov et al. 2009 (KP is used for resolving the problem of simultaneous access to the smart space content)
Knowledge patterns

- A data model that allows defining ontological objects
  - filtering locally available objects
  - searching new objects in the smart space

- Evaluation
  - correctness of defining objects
  - efficiency of processing

Filtering is used for transferring/delivering necessary parts of objects to/from the smart space.

Searching is used to deliver (search) new objects, existing in SS.
Synchronization

- SmartSlog data synchronization is subscription
  - blocking (synchronous subscriptions)
  - none-blocking (asynchronous subscriptions)

- Two ways to define objects to synchronize
  - explicit pointing
  - auto-marking for synchronization

- Mechanism for determination of synchronization moments
Performance evaluation

Starting performance evaluation

- OWL mapping performance
- Synchronization performance

Example of results:
Conclusion

- SmartSlog is a tool that supports efficient programming such devices for participating in smart space applications
  - Cross-platform and Multilingual
  - Code is compact due to high-level ontology style

- Future directions
  - Ontology manipulations
  - Optimizations

- Developers wiki:
  http://oss.fruct.org/wiki/SmartSlog/

- Open source code:
  http://sourceforge.net/projects/smartslog/

Thank you!