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.
Smart Spaces (SmS) approach

- The construction of semantic services is performed by software agents.
- Agents create and maintain shared information store for semantic linking of resources.
- Virtual images of resources, users and ongoing processes.
- Semantic interoperable access primitives.
- Software infrastructure:
  - software agents ($A_n$) with information-driven interaction;
  - services ($S_n$) - systems of interacting agents;
  - agent programming tools.
Requirements for SmS applications

- **Unified service ontology.** The design of semantic services should be based on a general unified ontology. It is possible to achieve the integration of both the SmS themselves and their applications for solving collaborative tasks.

- **Automation of agent programming processes.** Reduce the amount of program code created by an application developer during routine tasks through the use of computer-aided design and programming tools.
Automated code generation process of multi-agent interaction

Semantic service ontologies (Abox)

General unified service ontology (Tbox)

Development/maintenance of ontologies

Domain ontologies (Tbox and Abox)

Code generation tool: ontology processing (Jena), code templates and handlers.

Code generation procedure for agent interaction processes

Set of functions / methods

Program code of internal logic of agents

Data (classes, objects, etc.)

API

Class Book {
    private:
        string author;
        string title;
        ISBN hasISBN;
    public:
        std::string get_author() const;
        void set_author(string author);
    }

Book* book CongoBuy(int creditCardNumber, ISBN isbn) {
    Book* book = new Book();
    string acctExists = format("ASK {<%s> ex:signInfo <%s>}
        this->user_profile, acctID);
    bool pr_result = api_sparql_ask(acctExists);
    if (pr_result == true) {
        /* get results and give effects */
        return book;
    }
}
Ontological model of semantic service in SmS (TBox)
ABox of ontology for user presence and activity service ($S_{prs}$)
Mapping rules (input: ontology; output: object model)

1. Ontology classes → Object classes
2. Instances → Objects
3. Data type properties or slots → Data attribute variables & get/set methods
4. Object type properties or slots → Object attribute variables & get/set methods
5. Value-type/space facets → Attribute variables types & if-then-else statements
6. Cardinality facets → Additional attributes & if-then-else statements
7. Single class inheritance → Single object inheritance
8. Multiple inheritance → Single inheritance & multiple interface inheritance
Code generation procedure for agent data object model

Start

oopLang – chosen object-oriented programming language

Domain ontology

Meta-model construction

Traversing class nodes
While (itr_c hasNext())

Getting directly adjacent property nodes

Does it support multiple inheritance?

Rule 1
Mapping an ontology class to an object class

Rule 2
Mapping individuals to class objects

Rule 3
Mapping data properties to class attributes

Rule 4
Mapping object properties to class attributes

Rule 5
Mapping value restrictions to attribute types

Rule 6
Mapping cardinality restrictions to conditional statements

Rule 8
Mapping class inheritance to multiple inheritance

Traversing property nodes
Π̃π (itr_p hasNext())

Is a data property?
False

Is it an object property?
True

Getting restriction nodes

Is it a value restriction?
True

Is it a cardinality restriction?
False

Getting a list of class individuals

Traversing class nodes

End
Automated Code Generation of Multi-Agent Interaction

1. oopLang – chosen object-oriented programming language
2. Service ontology
3. Meta-model construction
4. Traversing agent nodes While (itr_kp.hasMoreElements())
5. Creating a set of code files
6. Getting process nodes
7. Mapping agent role to subscription operations
8. Getting an agent role
9. Traversing agent nodes
10. Getting an architectural abstraction

Start

Split, Choice, Any-Order, Condition, etc

Is the process composite?

- False
  - Getting nodes of atomic processes
  - Getting a name
  - Getting parameters
  - Getting result effects
  - Getting SPARQL-precondition

- True
  - Mapping a composite process to a code function
  - Traversing nodes of composite processes

Traversing nodes of atomic processes While (itr_ap.hasMoreElements())

Mapping an atomic process to a code function

Traversing nodes of atomic processes

Architectural abstractions

P-C: Mapping «P-C» abstraction
Pipe: Mapping «Pipe» abstraction
Tree: Mapping «Tree» abstraction
Flow: Mapping «Flow» abstraction

End
Automated Code Generation of Multi-Agent Interaction

Sergei Marchenkov

Domain Object Model

**.owl**

```owl
<owl:Class rdf:ID="PresenceInfo">
  <owl:DatatypeProperty rdf:ID="lastSeen">
    <rdfs:domain rdf:resource="#PresenceInfo"/>
    <rdfs:range rdf:resource="#xsd#unsignedLong"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="presenceLevel">
    <rdfs:domain rdf:resource="#PresenceInfo"/>
    <rdfs:range rdf:resource="#xsd#string"/>
  </owl:DatatypeProperty>
  ...;
</owl:Class>
```

**.cpp**

```cpp
PresenceInfo* prInfo = new PresenceInfo();

/* precondition SPARQL query */
string prInfoExists = format("ASK {<%s> &ua;hasPresenceInformation <%s>}, userUuid, this->prinfo},
"/* query execution by an API function */
bool pi_result = api_sparql_ask(prInfoExists);

if (pi_result == true) {
  /* get results and give effects;
     check and set user presence level */
  ...;
  return prInfo;
} else {
  /* the condition was not met, a fault occurred */
  ...;
  return NULL;
}
```

Function name: `SetUserPresenceLevel`

Parameter names:
- `userUuid` (string)

Precondition: SPARQL query

Return value name: `prInfo`
## Proportion of generated code for service implementations

<table>
<thead>
<tr>
<th>Service</th>
<th>Agents and their roles</th>
<th>Object data model</th>
<th>Information-driven interaction</th>
<th>Internal logic</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SLOC</td>
<td>%</td>
<td>SLOC</td>
<td>%</td>
</tr>
<tr>
<td>User presence and activity service ($S_{prs}$)</td>
<td>Presence processor adapter-agent</td>
<td>all</td>
<td>26</td>
<td>8,8</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>21</td>
<td>7,1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Presence detector aggregator-agent</td>
<td>all</td>
<td>18</td>
<td>11,2</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>14</td>
<td>8,7</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Activity monitor-agent</td>
<td>all</td>
<td>74</td>
<td>13,2</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>52</td>
<td>9,3</td>
<td>75</td>
</tr>
<tr>
<td>Historical data enrichment service ($S_{sem}$)</td>
<td>External finder-agent</td>
<td>all</td>
<td>65</td>
<td>9,6</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>50</td>
<td>7,4</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Semantic controller-agent</td>
<td>all</td>
<td>222</td>
<td>12,6</td>
<td>515</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>148</td>
<td>8,4</td>
<td>339</td>
</tr>
<tr>
<td></td>
<td>Enrichment aggregator-agent</td>
<td>all</td>
<td>391</td>
<td>11,9</td>
<td>898</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen.</td>
<td>296</td>
<td>9</td>
<td>513</td>
</tr>
</tbody>
</table>
Time to generate program code for service implementations

- $P$ - processes number
- $N$ - nodes number in the ontology meta-model
- ABox - assertion components number

<table>
<thead>
<tr>
<th></th>
<th>$S_{enr}$</th>
<th>$S_{prs}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>$N$</td>
<td>534</td>
<td>1010</td>
</tr>
<tr>
<td>ABox</td>
<td>254</td>
<td>733</td>
</tr>
</tbody>
</table>
## Quality metrics of generated code

```cpp
class PresenceInfo {
    unsigned long int title;
    string presenceLevel;
    ...
}
```

### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of generated code lines</td>
<td>49</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>4</td>
</tr>
<tr>
<td>Number of distinct operators: $\eta_1$</td>
<td>13</td>
</tr>
<tr>
<td>Number of distinct operands: $\eta_2$</td>
<td>17</td>
</tr>
<tr>
<td>Total number of occurrences of operators: $N_1$</td>
<td>26</td>
</tr>
<tr>
<td>Total number of occurrences of operands: $N_2$</td>
<td>41</td>
</tr>
<tr>
<td>Halstead vocabulary: $\eta = \eta_1 + \eta_2$</td>
<td>30</td>
</tr>
<tr>
<td>Halstead program length: $N = N_1 + N_2$</td>
<td>67</td>
</tr>
<tr>
<td>Program volume: $V = N \ast \log_2 \eta$</td>
<td>406</td>
</tr>
<tr>
<td>Program difficulty: $D = \frac{\eta_1}{2} \ast \frac{N_2}{\eta_2}$</td>
<td>15.67</td>
</tr>
<tr>
<td>Programming effort: $E = D \ast V$</td>
<td>6362</td>
</tr>
<tr>
<td>Programming time (seconds): $T = \frac{E}{18}$</td>
<td>353.5</td>
</tr>
</tbody>
</table>
This paper proposed 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.

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 services were investigated based on estimation of time to generate and the quality metrics of code.

Thank you for your attention
Any questions are welcomed
marchenk@cs.petrsu.ru