Petrozavodsk State University
Department of Computer Science

Sergei Marchenkov

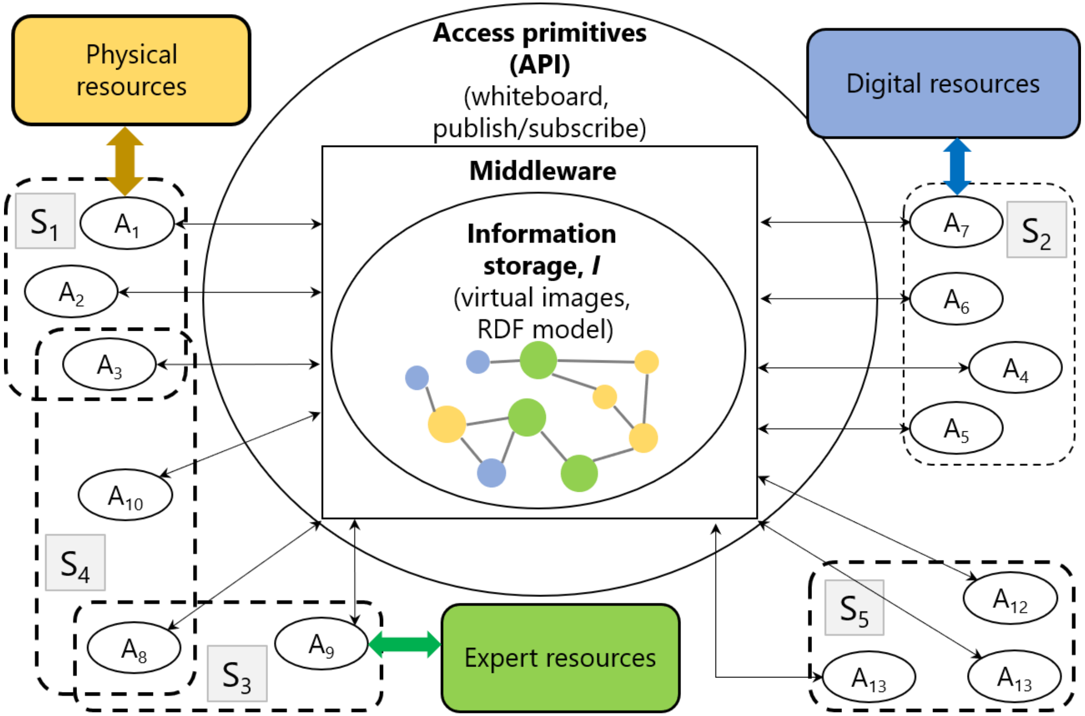
Automated Code Generation of Multi-Agent Interaction for Constructing Semantic Services

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.

UBICOMM 2020
October 25-29, 2020 – Nice, France

Smart Spaces (SmS) approach

- The construction of sementic 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.

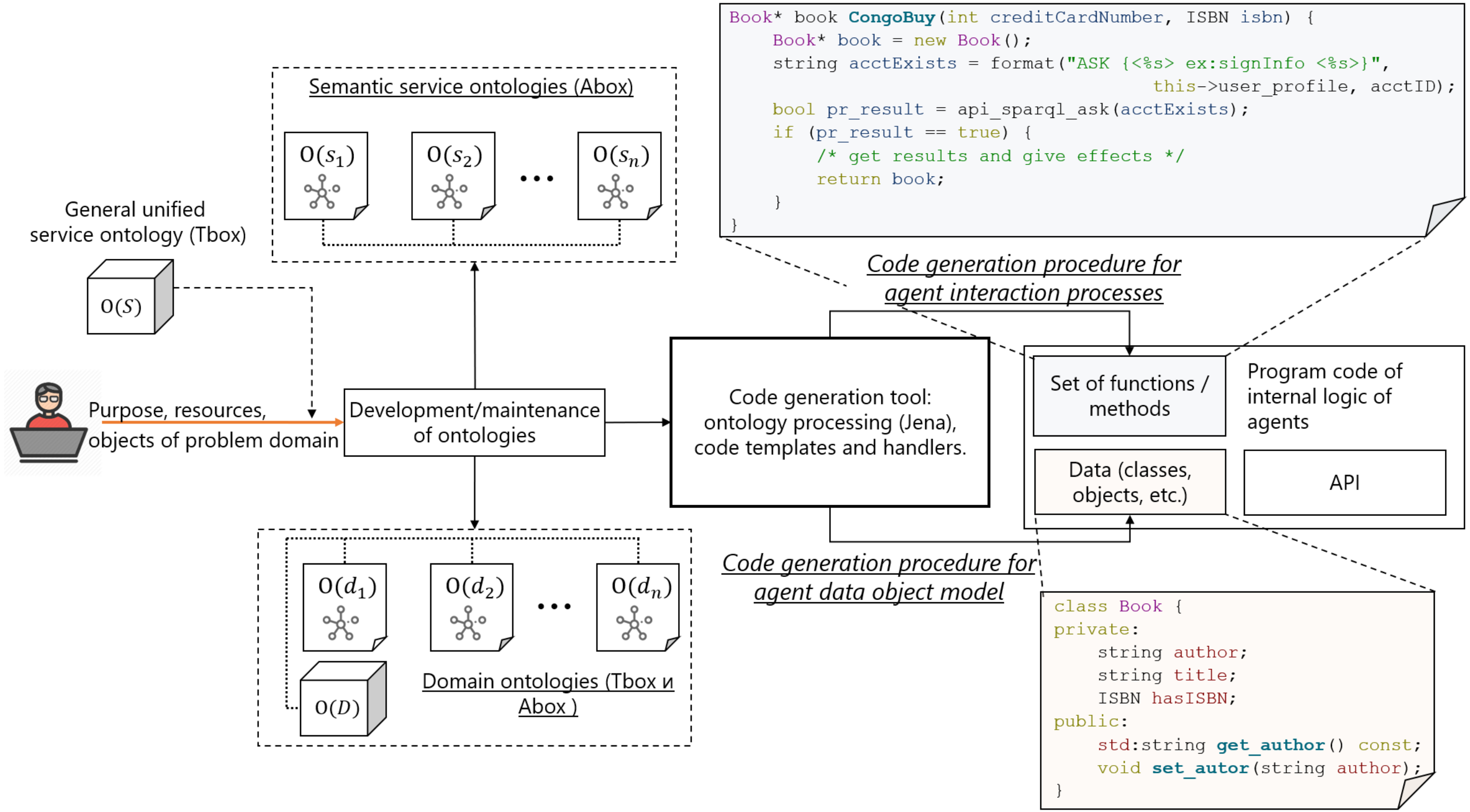


Requirements for SmS applications

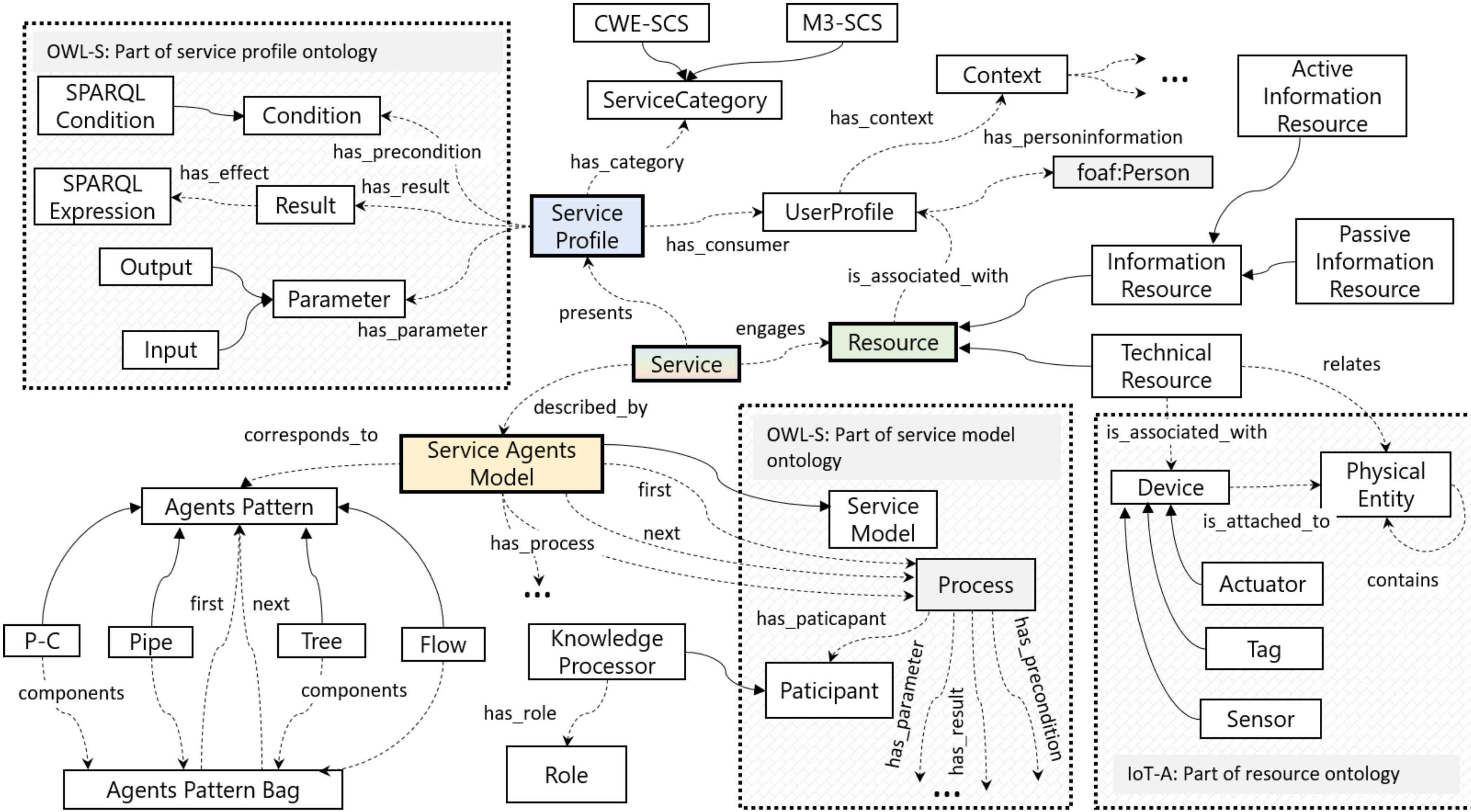
- **Unified service ontology**. The design of semantic services should be based on a general unified ontology. It is possible to achieve the integration of both the SmS themselves and their applications for solving collaborative tasks.

- **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.

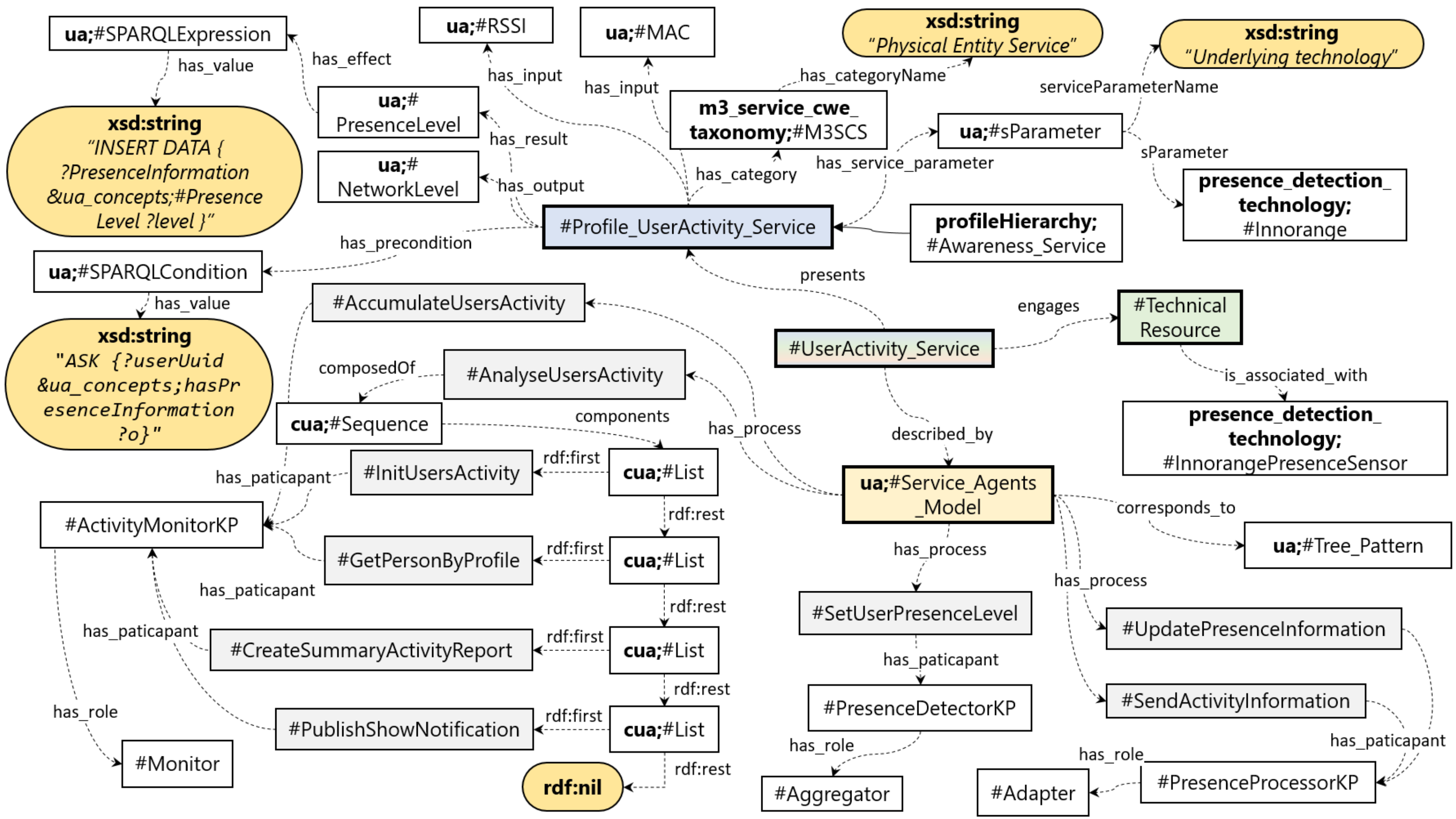
Automated code generation process of multi-agent interaction



Ontological model of semantic service in SmS (TBox)



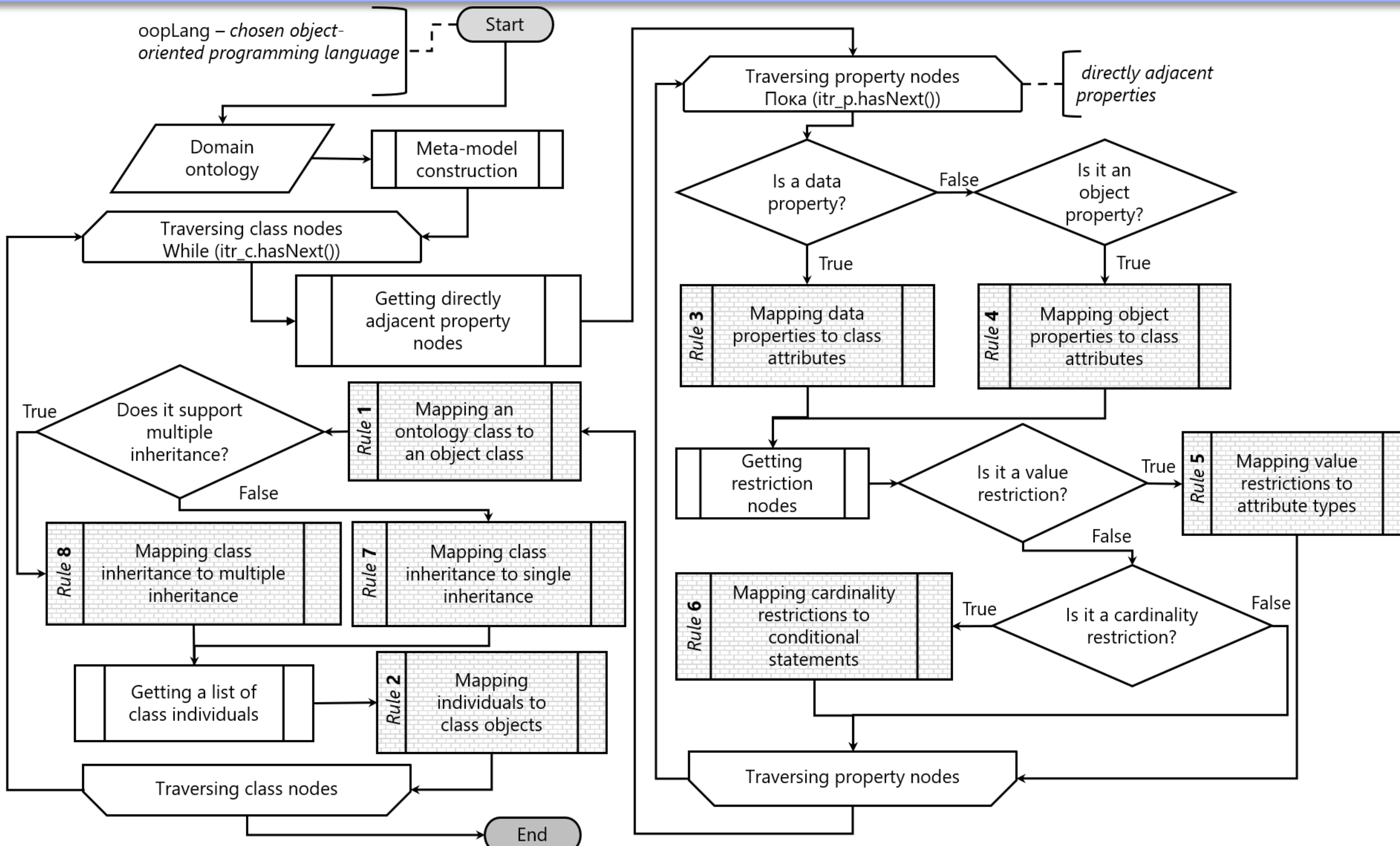
ABox of ontology for user presence and activity service (S_{prs})

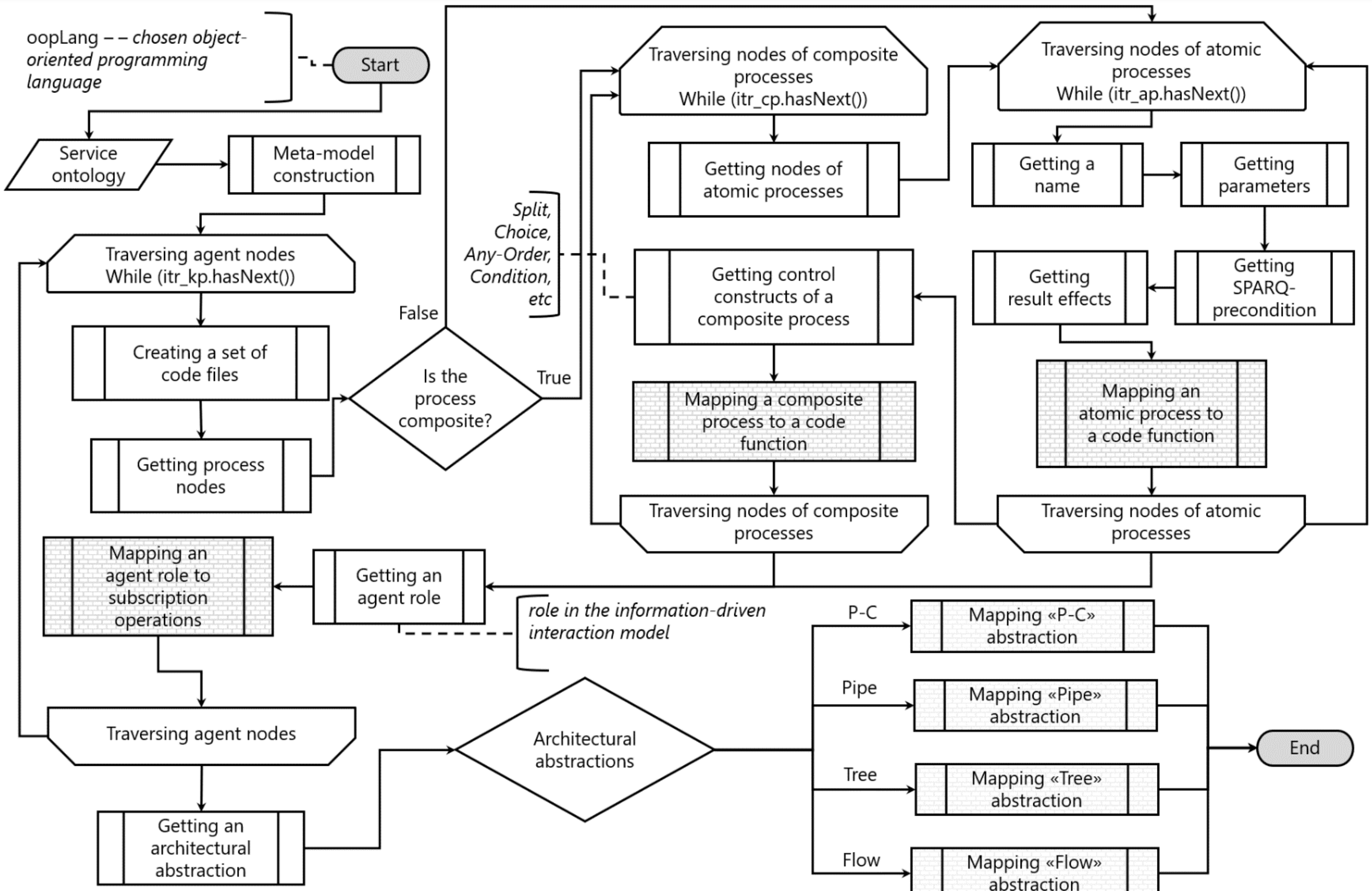


Mapping rules (input: ontology; output: object model)

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





oopLang – – chosen object-oriented programming language
Start
Service ontology
Meta-model construction
Traversing agent nodes While (itr_kp.hasNext())
Creating a set of code files
Getting process nodes
False
Is the process composite?
True
Split, Choice, Any-Order, Condition, etc
Traversing nodes of composite processes While (itr_cp.hasNext())
Getting nodes of atomic processes
Getting control constructs of a composite process
Mapping a composite process to a code function
Traversing nodes of composite processes
Traversing nodes of atomic processes While (itr_ap.hasNext())
Getting a name
Getting parameters
Getting SPARQ-precondition
Getting result effects
Mapping an atomic process to a code function
Traversing nodes of atomic processes
Mapping an agent role to subscription operations
Getting an agent role
role in the information-driven interaction model
Traversing agent nodes
Getting an architectural abstraction
Architectural abstractions
P-C
Mapping «P-C» abstraction
Pipe
Mapping «Pipe» abstraction
Tree
Mapping «Tree» abstraction
Flow
Mapping «Flow» abstraction
End

.owl

```
<process:AtomicProcess rdf:ID="SetUserPresenceLevel">
  ...
  <process:hasInput rdf:ID="userUuid">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
    &xsd;#string
    </process:parameterType>
  </process:hasInput>
  <process:hasPrecondition>
    <expr:SPARQL-Condition rdf:ID="prInfoExists">
      <expr:expressionObject>
      "ASK {?userUuid
      &ua_concepts;hasPresenceInformation ?o}"
      </expr:expressionObject>
    </expr:SPIN-Condition>
  </process:hasPrecondition>
  <process:hasOutput rdf:ID="prInfo“>
     <process:parameterType rdf:datatype="&xsd;#anyURI">
     &THIS;#PresenceInfo
     </process:parameterType>
  </process:hasOutput>
  ...
</process:AtomicProcess>

<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">
    ...
  </owl:DatatypeProperty>
...
</owl:Class>
```

.cpp

```
PresenceInfo* prInfo SetUserPresenceLevel
                                         (string userUuid)
{
   /* local variables */
   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;
   }
}
class PresenceInfo {
   unsigned long int title;
   string presenceLevel;
   ...;
}
```

Function name

Parameter names

Precondition:
SPARQL query

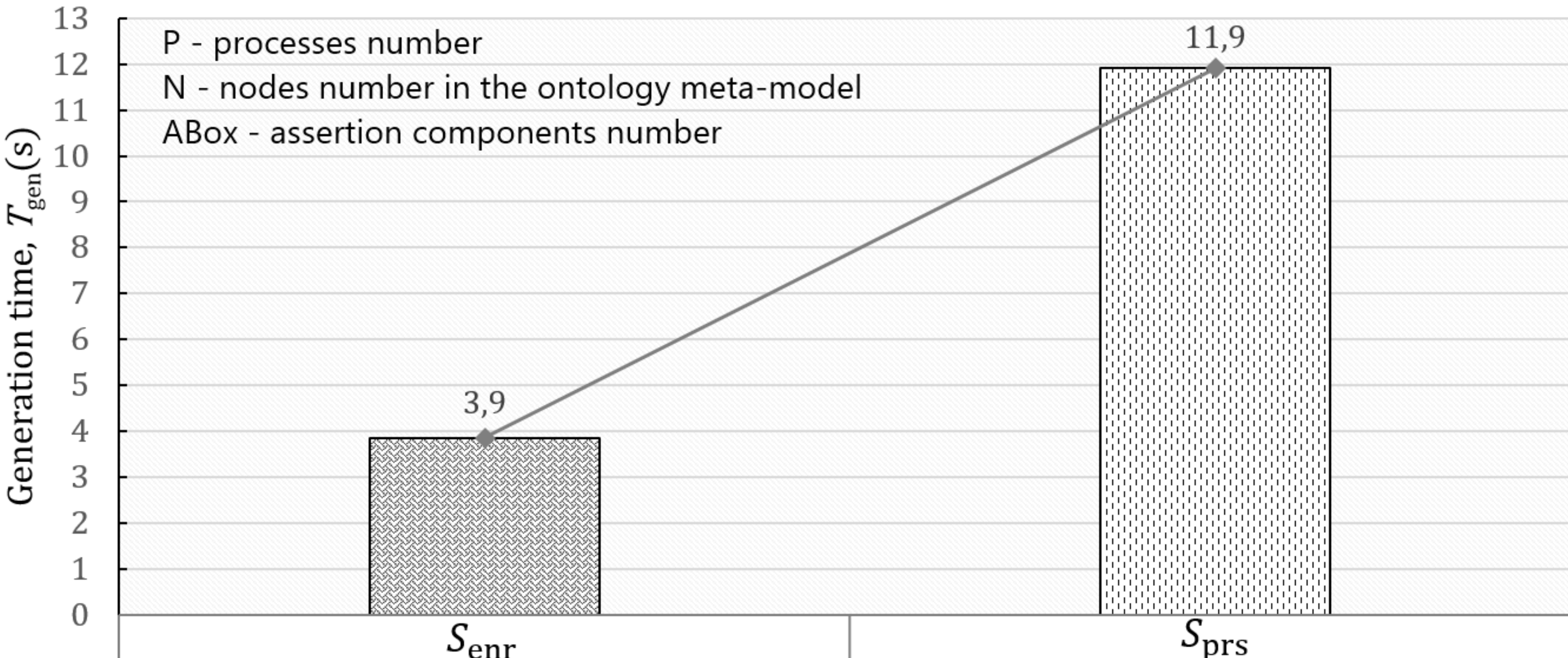
Return value name

Domain Object Model

Proportion of generated code for service implementations

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**

Time to generate program code for service implementations



	S_{enr}	S_{prs}
P	12	24
N	534	1010
ABox	254	733

Quality metrics of generated code

.cpp

```
PresenceInfo* prInfo SetUserPresenceLevel
                                    (string  userUuid )
{
   /* local variables */
   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;
   }
}
class PresenceInfo {
   unsigned long int title;
   string presenceLevel;
   ...;
}
```

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**

Conclusion

- 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