

# *Stratified Composition of Web Services<sup>1</sup>*

Riina Maigre

Institute of Cybernetics at TUT

Theory Days at Kääriku 2009

---

<sup>1</sup>Based on: R. Maigre, P. Grigorenko, P. Kungas , E. Tyugu. Stratified Composition of Web Services. In: M. Virvou, T. Nakamura (eds.) Knowledge-Based Software Engineering. Proc. 8th JCKBSE. IOS Press, 2008, p. 49 - 58

# *Outline*

## *Motivation*

Large information systems with service oriented architecture

## *Knowledge architecture of the composition tool*

User knowledge level

Logical level

Service implementation level

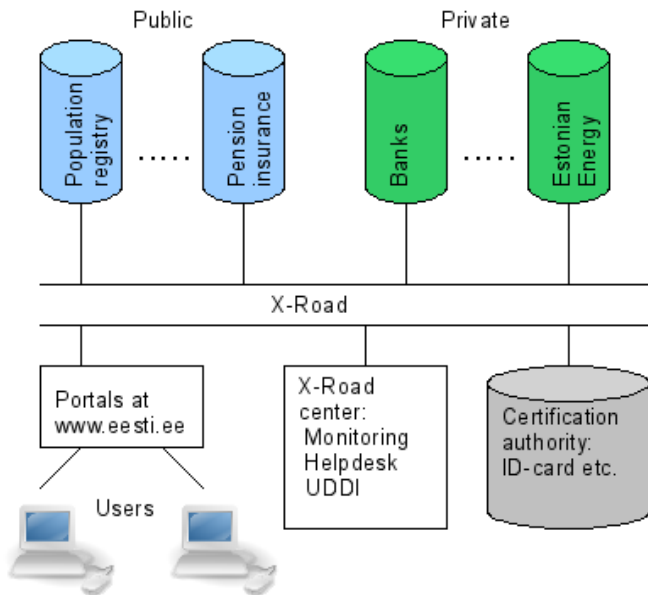
## *Motivation*

- ▶ Goal of this work is to overcome the complexity of service composition on very large sets of atomic services.
- ▶ To achieve this goal we propose an automatic service composition methodology with three distinguished knowledge levels:
  - ▶ user knowledge level
  - ▶ logical level
  - ▶ implementation level

## *Example application domain*

- ▶ Estonian e-government information system has a service-oriented architecture.
  - ▶ Services accessible through this information system are described in WSDL.
  - ▶ More than thousands of services are provided by information systems of different institutions.
- ▶ Secure access is provided (over X-Road) to nearly all public (i.e. governmental) but also some private databases.
- ▶ All Estonian residents having the national ID card can access these services through X-Road within their limits of authority.
- ▶ In this work we refer to the whole system as X-Road.

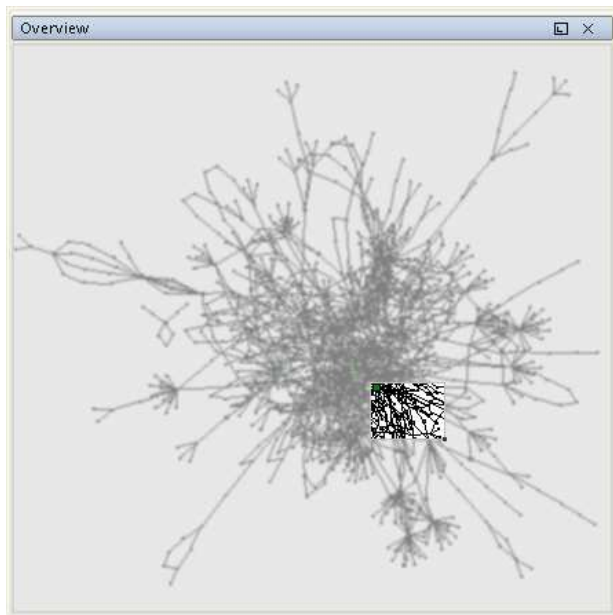
# *X-Road*

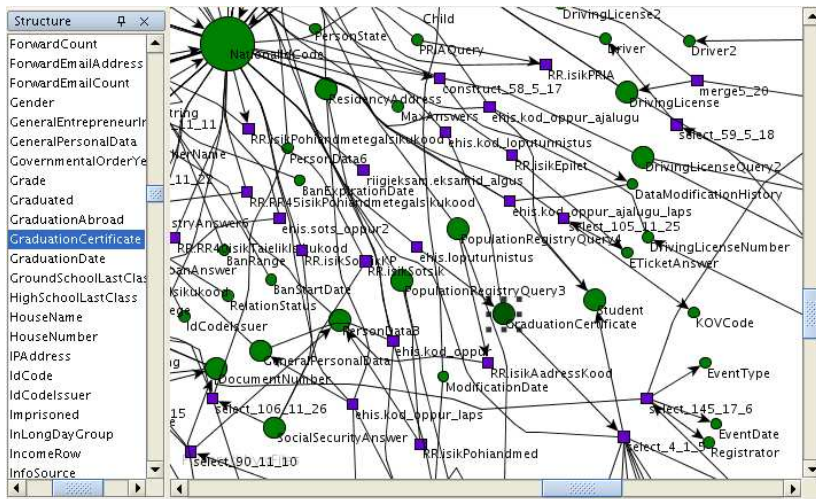


## *Example model*

- ▶ We have performed experiments on a part of X-road information system.
- ▶ Service model containing about 300 atomic services and about 600 references to semantic resources has been created.
- ▶ Totally more than thousand of atomic services are available.

## *Model of the application domain*







## *Querying X-Road*

- ▶ Only predefined queries can be done through X-Road portals
- ▶ Combined queries between information systems of different institutions are not possible.
- ▶ Possible semantic relationships between queries are ignored.

## *Example: querying person's contact addresses*

(Without complex queries:)

From the 1th DB: *NationalIdCode*  $\rightarrow$  *Address*

From the 2nd DB: *NationalIdCode*  $\rightarrow$  *Address*

...

From the n-th DB: *NationalIdCode*  $\rightarrow$  *Address*

## *Example complex query*

*NationalIdCode →  
AddressString, EstonianAddressString,  
OwnerAddressString, ResidencyAddress,  
ResponsibleUserAddress*

## *Service composition tool*

- ▶ We have created a tool for service composition that supports the stratified composition methodology and hides complexity of composition from the end user.
- ▶ Tool is created in the software development environment CoCoViLa that supports automatic synthesis of programs and generates Java code from visual and textual specifications.

## *Knowledge architecture of the composition tool*

Knowledge system – module of knowledge architecture.

- ▶ knowledge language
- ▶ knowledge handling mechanism
- ▶ method for associating meanings to knowledge objects

|   |
|---|
| S |
| M |

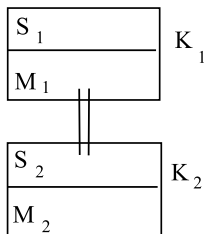
S – knowledge objects (notations)

M – set of meanings (denotations)

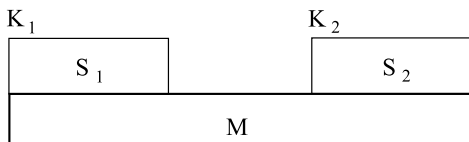
## *Connections of knowledge systems*

Knowledge systems can be composed into larger knowledge architecture by using:

- ▶ Hierarchical connections (a)
- ▶ Semantic connections (b)
- ▶ Operational connections



a)

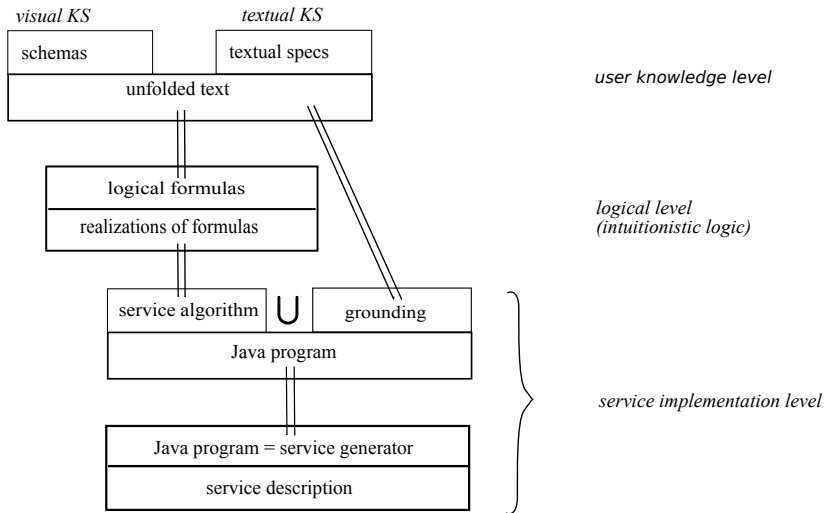


b)

## *Knowledge levels*

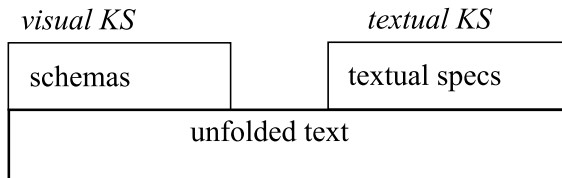
1. User knowledge level
2. Logical level
3. Service implementation level

# Service composition tool



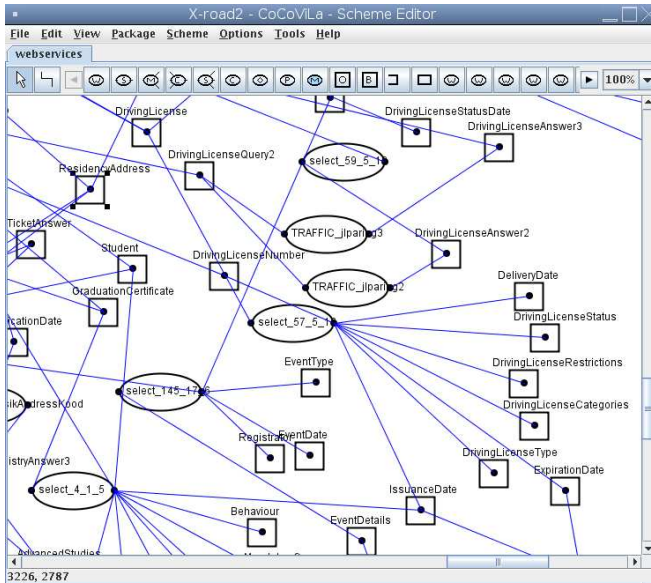


## *1. User knowledge level*



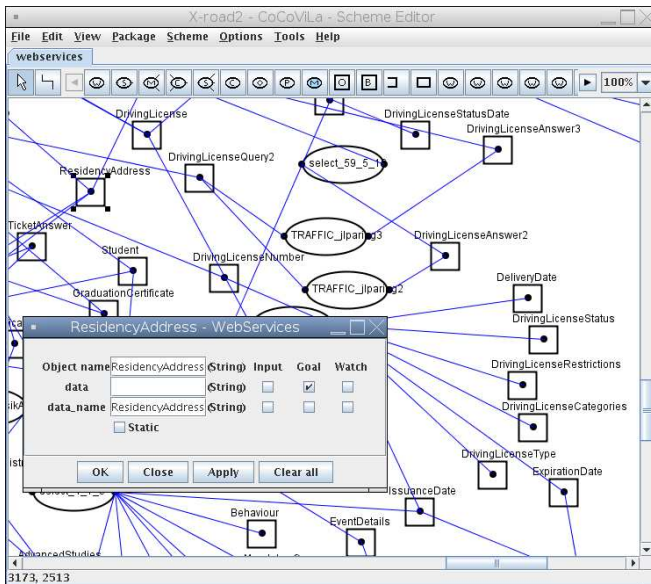
# 1. User knowledge level

Visual knowledge representation.



# 1. User knowledge level

Specifying a complex service on visual representation.



# 1. User knowledge level

Visual and textual representation of knowledge.

The screenshot displays the X-road2 WebServices - Specification tool interface. The top window, titled "X-road2", shows a visual network diagram with nodes like "DrivingLicense", "ResidencyAddress", "TicketAnswer", "Student", and "DrivingLicen". The bottom window, titled "WebServices - Specification", has tabs for "Specification", "Program", and "Run results". The "Specification" tab is active, showing a BPEL code snippet. Below the code, there is a table for "ResidencyAddress - WebServices" with columns for "Object name", "data", "data\_name", "Input", "Goal", and "Watch".

**ResidencyAddress - WebServices**

| Object name      | data | data_name        | Input                    | Goal                                | Watch                    |
|------------------|------|------------------|--------------------------|-------------------------------------|--------------------------|
| ResidencyAddress |      | ResidencyAddress | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
|                  |      |                  | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |

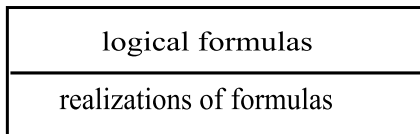
Buttons: OK, Close, Apply, Clear all

**BPEL Specification:**

```
1 public class WebServices extends BPEL {
2     /*@ specification WebServices super BPEL {
3         ehis_kod_oppur ehis_kod_oppur;
4         ehis_kod_oppur.name = "ehis_kod_oppur";
5         ehis_kod_oppur.input = {"str*"};
6         ehis_kod_oppur.output =
7         {"GeneralPersonalData", "ResidencyAddress",
8          "BaseStudies", "AdvancedStudies"};
9         select select_591;
10        select_591.name = "select_0_1_1";
11        select_591.input = "GeneralPersonalData";
12        select_591.output = {"NationalIdCode",
13                             "BirthDate", "Gender",
14                             "LastName", "Citizenship",
15                             "IsProtege", "IsWaif"};
16        lect_592;
17        t_592.name = "select_1_1_2";
18        t_592.input = "ResidencyAddress";
19        t_592.output = {"CountryName",
20                       "reetName", "HouseNumber",
21                       "ber", "PostalCode"};
22        lect_593;
23        t_593.name = "select_2_1_3";
24        t_593.input = "BaseStudies";
25        t_593.output =
26        {"EducationAtAdmission", "AdmissionYear",
27         "StudyStartDate", "School", "StudyProgram",
28         "StudyLanguage", "StudyForm", "FinancingSource",
29         "Class", "ParallelIndication",
30         "CompoundClassIndication", "SchoolLevel",
31         "ClassType", "DistanceFromSchool",
```

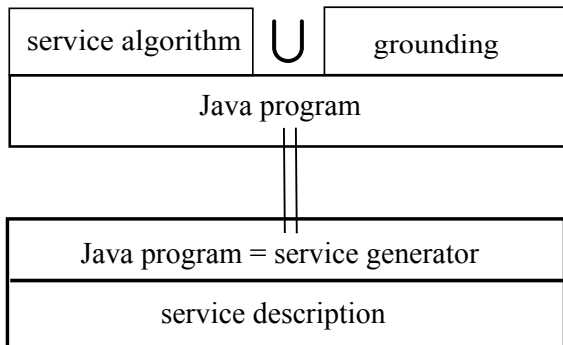
## 2. *Logical level*

Formal representation of knowledge and automatic composition of new services.



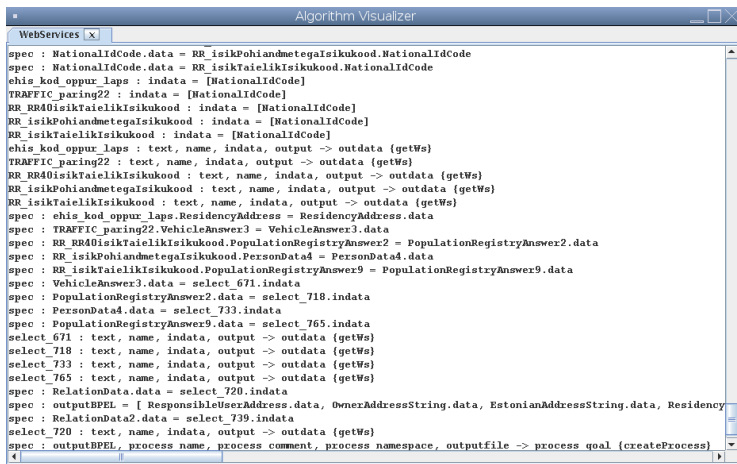
- ▶ Knowledge about atomic services, goal  $\Rightarrow$  algorithm of expected service
- ▶ Structural synthesis of programs is used to synthesize a structure of a new complex service

### 3. *Service implementation level*



### 3. Service implementation level

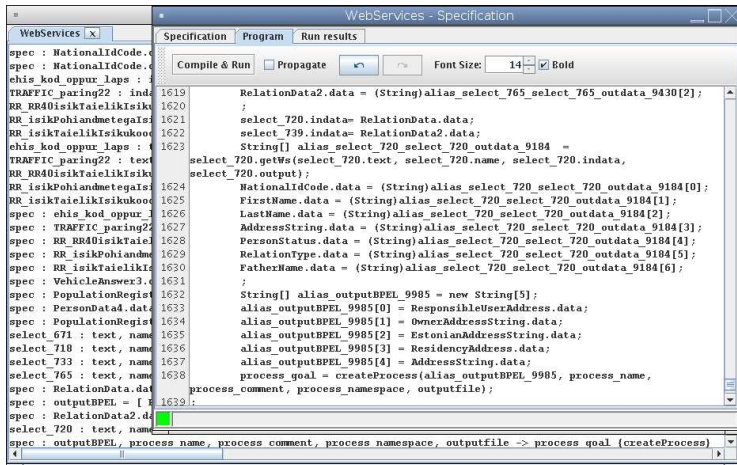
Synthesizing a structure of the complex service.



```
Algorithm Visualizer
WebServices
spec : NationalIdCode.data = RR_isikPohiandmetegaIsikukood.NationalIdCode
spec : NationalIdCode.data = RR_isikTaelikIsikukood.NationalIdCode
ehis kod oppur laps : indata = [NationalIdCode]
TRAFFIC_paring22 : indata = [NationalIdCode]
RR_RR40IsikTaelikIsikukood : indata = [NationalIdCode]
RR_isikPohiandmetegaIsikukood : indata = [NationalIdCode]
RR_isikTaelikIsikukood : indata = [NationalIdCode]
ehis kod oppur laps : text, name, indata, output -> outdata {getWs}
TRAFFIC_paring22 : text, name, indata, output -> outdata {getWs}
RR_RR40IsikTaelikIsikukood : text, name, indata, output -> outdata {getWs}
RR_isikPohiandmetegaIsikukood : text, name, indata, output -> outdata {getWs}
RR_isikTaelikIsikukood : text, name, indata, output -> outdata {getWs}
spec : ehis kod oppur laps.ResidencyAddress = ResidencyAddress.data
spec : TRAFFIC_paring22.VehicleAnswer3 = VehicleAnswer3.data
spec : RR_RR40IsikTaelikIsikukood.PopulationRegistryAnswer2 = PopulationRegistryAnswer2.data
spec : RR_isikPohiandmetegaIsikukood.PersonData4 = PersonData4.data
spec : RR_isikTaelikIsikukood.PopulationRegistryAnswer9 = PopulationRegistryAnswer9.data
spec : VehicleAnswer3.data = select_671.indata
spec : PopulationRegistryAnswer2.data = select_718.indata
spec : PersonData4.data = select_733.indata
spec : PopulationRegistryAnswer9.data = select_765.indata
select_671 : text, name, indata, output -> outdata {getWs}
select_718 : text, name, indata, output -> outdata {getWs}
select_733 : text, name, indata, output -> outdata {getWs}
select_765 : text, name, indata, output -> outdata {getWs}
spec : RelationData.data = select_720.indata
spec : outputBPEL = [ ResponsibleUserAddress.data, OwnerAddressString.data, EstonianAddressString.data, Residency
spec : RelationData2.data = select_739.indata
select_720 : text, name, indata, output -> outdata {getWs}
spec : outputBPEL, process name, process comment, process namespace, outputfile -> process goal {createProcess}
```

### 3. Service implementation level

Grounding of services - calling services and performing actual computations.



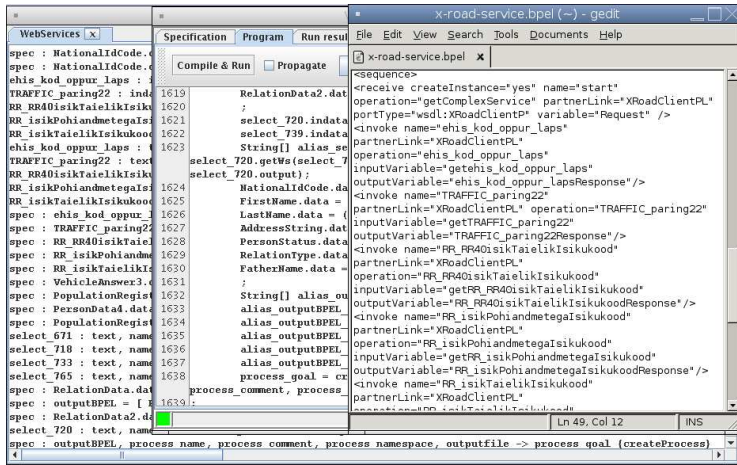
The screenshot shows a window titled "WebServices - Specification" with three tabs: "Specification", "Program", and "Run results". The "Specification" tab is active, displaying a list of service specifications on the left and their corresponding implementation code on the right. The specifications include various data types and operations, such as "NationalIdCode", "TRAFFIC\_paring22", "RR\_RR40isikTaelikIsiku", "RR\_isikPohiandmetegaIsi", "RR\_isikTaelikIsikukoo", "ehis\_kod\_oppur\_laps", "VehicleAnswer3", "PopulationRegist", "PersonData4", "select\_671", "select\_718", "select\_733", "select\_765", "RelationData2", "outputBPEL", and "process\_name". The implementation code on the right shows the corresponding data structures and operations, such as "RelationData2.data", "select\_720.indata", "String[] alias\_select\_720\_select\_720\_outdata\_9184", "NationalIdCode.data", "FirstName.data", "LastName.data", "AddressString.data", "PersonStatus.data", "RelationType.data", "FatherName.data", "String[] alias\_outputBPEL\_9985", "alias\_outputBPEL\_9985[0]", "alias\_outputBPEL\_9985[1]", "alias\_outputBPEL\_9985[2]", "alias\_outputBPEL\_9985[3]", "alias\_outputBPEL\_9985[4]", "process\_goal", "process\_comment", "process\_namespace", "outputfile", and "process\_name".

```
spec : NationalIdCode.e
spec : NationalIdCode.e
ehis_kod_oppur_laps : i
TRAFFIC_paring22 : inda
RR_RR40isikTaelikIsiku
RR_isikPohiandmetegaIsi
RR_isikTaelikIsikukoo
ehis_kod_oppur_laps : t
TRAFFIC_paring22 : text
RR_RR40isikTaelikIsiku
RR_isikPohiandmetegaIsi
RR_isikTaelikIsikukoo
spec : ehis_kod_oppur_laps
spec : TRAFFIC_paring22
spec : RR_RR40isikTaeli
spec : RR_isikPohiandme
spec : RR_isikTaelikIsi
spec : VehicleAnswer3.e
spec : PopulationRegist
spec : PersonData4.data
spec : PopulationRegist
select_671 : text, name
select_718 : text, name
select_733 : text, name
select_765 : text, name
spec : RelationData2.data
spec : outputBPEL = [
spec : RelationData2.d
select_720 : text, name
spec : outputBPEL, process name, process comment, process namespace, outputfile -> process goal (createProcess)
```



### 3. Service implementation level

Generating BPEL from Java code.



## *Conclusion*

- ▶ We have described the architecture of the complex knowledge-based tool with stratified knowledge levels.
- ▶ The main features of the presented architecture are
  - ▶ user friendly upper level
  - ▶ precisely defined mappings between levels
- ▶ We hope to make the tool usable first of all to the developers

Thank you for listening!

Questions?

Supporters:

Estonian Science Foundation

Tiger University

