Dynamic Service Synthesis on a Large Service Models of a Federated Governmental Information System

Riina Maigre[1], Peep Küngas[2], Mihhail Matskin[3,4], Enn Tyugu[1]
[1]Institute of Cybernetics at Tallinn University of Technology, Tallinn, Estonia
[2]SOA Trader, Ltd, Tallinn, Estonia
[3]Royal Institute of Technology – KTH, Stockholm, Sweden
[4]Norwegian University of Science and Technology – NTNU, Trondheim, Norway
riina@cs.ioc.ee, peep@soatrader.com, misha@imit.kth.se, tyugu@cs.ioc.ee

Abstract

In this paper we describe our experiments with large syntactic Web service models of a federated governmental information system for automatic composition of services. The paper describes a method for handling syntactic service models for synthesis of compound services. The method's implementation as a visual tool developed in software environment CoCoViLa is explained on an example from e-government domain. Given a specification and a goal, the tool automatically synthesizes a program that generates a required composite service description in BPEL or OWL-S.

Keywords: automatic service composition; large service models; e-government services.

1. Introduction

The present paper describes experiences of composition of Web services on very large syntactic Web service models, and it is an extended version of the paper [1] at the 3rd International Conference on Internet and Web Applications and Services. The lessons learned presented here are based on a work with software developed for providing services to citizens by a number of governmental agencies. A federated e-government information system [2], complying to service-oriented architecture, has been developed in Estonia during the recent years. A syntactic service model of a part of the system exists and can be used for automating composition of new services. However, this process is still too complicated for end users and can only be useful, first of all, for software experts developing and maintaining the system.

Developing the federated information system has been a complicated task that has required cooperation of a number of government agencies that already provide services for citizens. Analysis of the system has resulted in a unified service model which includes about three hundred of atomic services [3], including a number of rather primitive data transformers needed for interoperability of databases. Totally more than a thousand atomic services are available, which could be included in the service model and composed into complex services. To determine a structure of a new complex service we are going to use a syntactic service model that describes only inputs and outputs of atomic services and includes references to the semantics of these services.

Although there does not seem to be any literature available considering usage of automated composition tools for federated governmental information systems, many EU countries have started their public sector semantic interoperability initiatives.

The primary aim of this work is development of a tool for automatic composition of Web services that involve atomic services from several governmental institutions. It is easy for the end user to get a service from a governmental agency through the agency's portal. Operations get more complicated when one needs to use services from several agencies, i.e., from multiple providers. Currently an end-user has to get information from one provider and forward the results manually to the second provider's service. The user has to know exactly which data has to be passed from one portal to another.

Manual construction of a new complex service from atomic services is a challenging task even for the software developers, because service descriptions from different providers are published on different servers and the number of possible inputs, outputs and their combinations is large. Our tool is intended to automate this process for software developers. To achieve our goal we use a visual programming environment CoCoViLa [4] that uses automatic synthesis of algorithms and can generate Java code from both visual and textual specifications. The syntactic service model is presented as a specification to the tool developed in CoCoViLa. For each requested service, a goal is given that specifies the input and output data of the service. From this information, an algorithm of the service is composed, and a service description is generated in BPEL (or in OWL-S, if requested).

The paper is structured as follows. The process of service composition is described in Section 4 and Section 5 after discussing the federated e-government information system and its service model in Section 2 and Section 3. Related work is presented in Section 6 and concluding remarks in Section 7.

2. X-road

The central part of Estonian e-government information system is the infrastructure, called X-Road, guaranteeing secure access to nearly all Estonian national databases over the Internet [2]. It is the environment through which hundreds of services are provided to the citizens, entrepreneurs and public servants on the 24/7 bases. These services are available through domain-specific portals to a variety of user groups (citizens, entrepreneurs, public servants). All Estonian residents having a national ID-card can access these services through X-Road. The number of requests per month exceeds currently 3 million. For brevity we are going to call the whole information system from now on as X-Road.

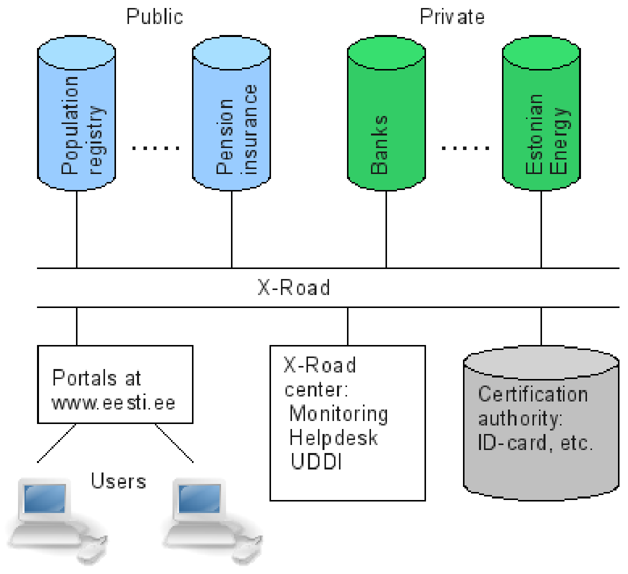


Figure 1. X-Road connects public and private information service providers.

Integration of databases developed by different developers at different times has been a difficult task that started in 2001 and has resulted in a widely used system at the present. During this time a number of standard tools have been developed to enable the creation of e-services capable of simultaneously using the data from different national and international databases. These services enable to read and write data, develop business logic based on data, etc. In X-Road data exchange is handled by SOAP messages, Web services are described in WSDL and service descriptions are published at a UDDI repository.

Figure 1 shows the simplified structure of Estonian information system based on X-Road. As demonstrated in the figure, X-Road connects besides public databases also some private ones. These are, for instance, main banks and some privately owned infrastructure enterprises. Some services are provided by the X-Road infrastructure itself that includes PKI infrastructure, help-desk, monitoring, etc. Users connect to the system through portals where they can execute predefined services. All queries have to be done one by one, even if semantic connections exist between services.

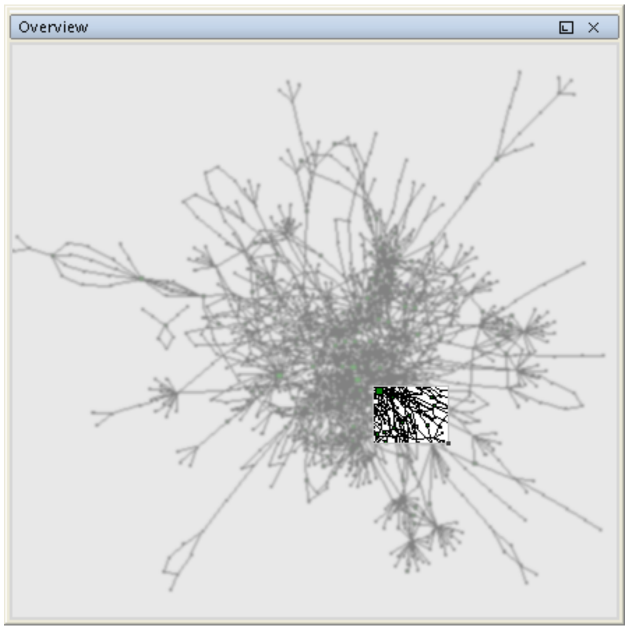


Figure 2. X-Road service model.

This large system is continuously changing and also its maintenance requires often new software updates. When new organization is joining the X-Road it means that they will make their services available through X-Road and/or need to access services offered through the X-Road. In the former case number of new services will be published and in the latter case specialized queries may be needed. This accentuates the problem of automation of composition of services on the service model of X-Road.

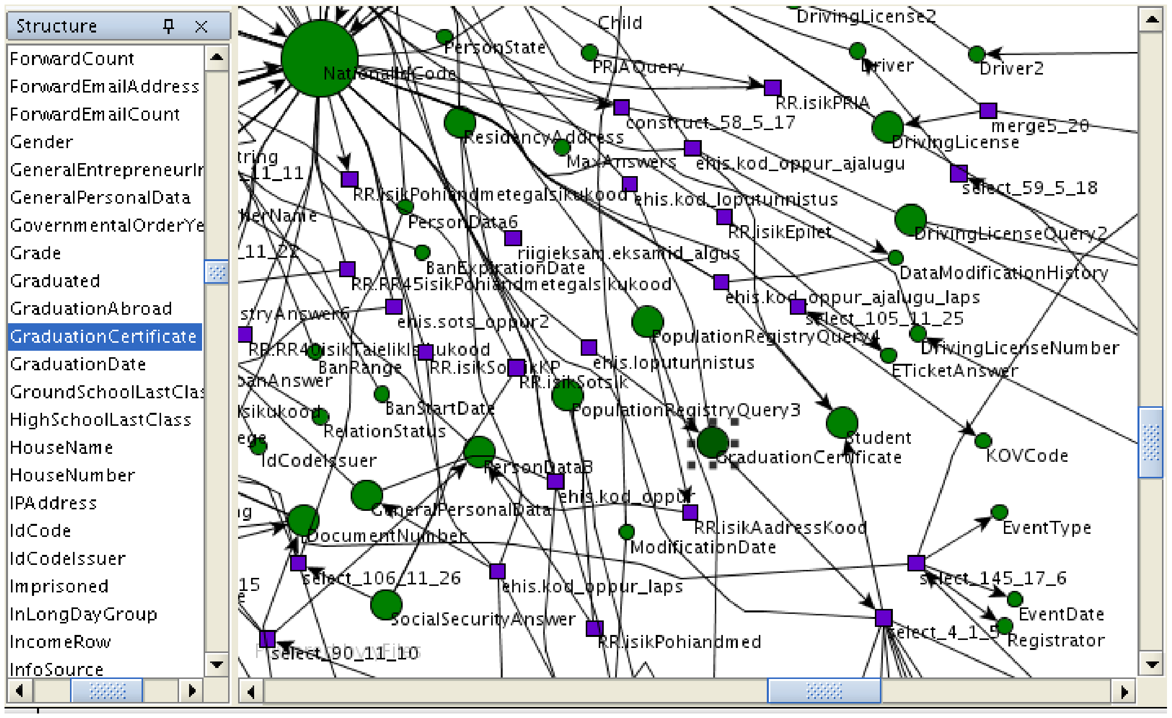


Figure 3. Zoomed-in part of X-Road service model.

3. X-Road service model

Analysis [3] of already operational X-Road resulted in a service model that included about 300 atomic services and about 600 unique references to semantic resources. These services have been annotated and descriptions of their interfaces constitute the syntactic service model shown in Figure 2. This figure shows a visual representation of the whole service model visualized by Java graph editor yEd. It is a large graph where nodes are atomic services and resources.

A small part of the model shown by a rectangle in Figure 2 is enlarged in the in Figure 3. List in the left pane of the user interface fragment shows a scrollable list of all resources. One can see atomic services as rectangles and inputs and outputs as circles (e.g., *GraduationCertificate*, *Student*, etc.) connected to services. The highlighted resource *GraduationCertificate* is input for the atomic services *select_4_1_5* and *ehis.kod_loputunnistus*, and output for the service *ehis.loputunnistus*. This is shown by particular arrows. Size of a circle of a resource shows its relative importance (connectivity to services). A resource with the largest value in the current model is *NationalIdCode* that is not surprising, because it is used in most of the services.

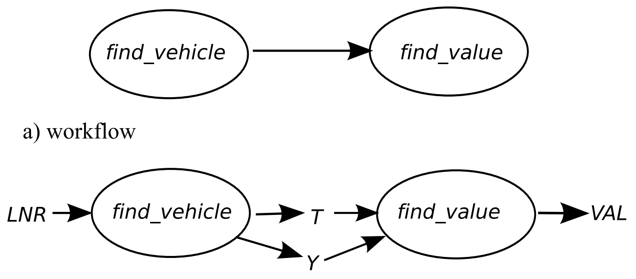
The model presented here is the basis for automatic synthesis of services. However, it must be transformed into another format in order to be applicable as a specification for the synthesis. This format is prescribed by the tool we use. It will be discussed in Section 5 on an example. Before going to explain the tool, we present the logical basis of synthesis in the next section.

4. Automatic handling of a Web service model

In the present section, we describe a logic-based method of automatic composition of services that is the basis of the tool we are using. The method is based on structural synthesis of programs (SSP) [5] and has been in use in several programming tools [4]. In this setting, a service is considered as a computational problem – computing a desired output from a given input. The problem is described for SSP by a set of formulas automatically extracted from a specification, and a goal that the expected result is computable is formulated as a theorem to be proven. A proof of solvability of the problem is built, and a program for solving the problem is extracted from the proof. SSP uses intuitionistic logic that is a constructive logic, i.e., a logic where any proof of existence of an object also is supported by an algorithm that enables one to construct the object [6].

Let us explain the synthesis of services on an example. The problem is to find an estimate of total value of a company's vehicles that we will denote by *RESULT* from the company's official registry number *RNR*. The first and simple task can be to start with a license number *LNR* of a vehicle, find its type *T* and production year *Y*, and to calculate an estimated value *VAL* of a car for *T* and *Y*.

To be able to represent complex services with control structures (loops, choices, etc.) we are using higher order workflow (HOWF) with data dependencies [7]. We translate workflows in a logic that enables us to reason about the reachability of goals on workflow models. Workflow with data dependencies includes explicitly represented inputs and outputs of every atomic service – data items. An ordinary workflow graph and the respective workflow graph with data dependencies for our simple task are shown in Figure 4.



b) workflow with data dependencies

Figure 4. Two representations of a workflow.

In a workflow with data dependencies one does not need extra arrows for representing the order of execution of atomic services, because the data dependencies determine the required order. (If no data are passed between two services and their execution order is still important, then one can add a dummy data item to determine the execution order.)

To solve the whole example problem, one has first to find a list of all vehicles *VEH* from a given registry number *RNR* of a company. Let us call the respective node *all_vehicles*. Thereafter, in order to compute *RESULT*, one has to repeat the task shown in Figure 4 for each vehicle found for the company. This requires introduction of some control in the composed service. This control will be represented by a node called *loop* that will pass *LNR* to the workflow shown in Figure 4 and collect *VAL* as the result of the task performed by this workflow. It is important to note that the data items *LNR* and *VAL* are not input and output of the node *loop*. They are input and output for the part of the workflow that solves a subtask – computes *VAL* from *LNR*, and they are bound with the *loop* node by dotted arrows, see Figure 5. The complete workflow for solving the example problem is shown in Figure 5. The node *loop* is a higher order node – one of its inputs is a subtask "compute *VAL* from *LNR*" and the workflow is a higher order workflow (HOWF). More on using HOWF for representing Web services can be found in [7].

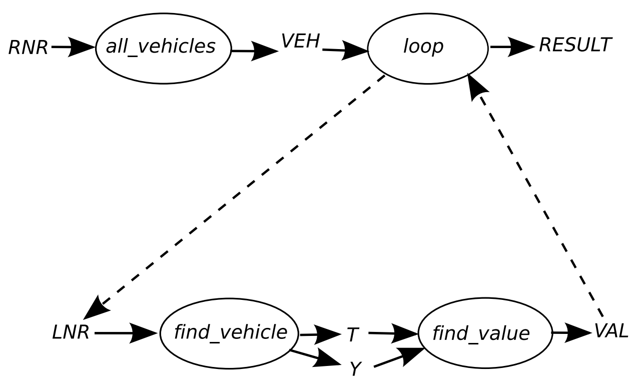


Figure 5. Higher order workflow.

We have not yet discussed the synthesis of services – only their representation by means of HOWF has been described. Now we will show that a HOWF in its turn can be represented in a propositional logic. Let us consider any data name *X* in a workflow as a proposition "there is a way to find the value of *X*". Then a service that computes, for instance, *VEH* from *RNR*, can be encoded in the intuitionistic logic as an implication

$$RNR \supset VEH\{all\_vehicles\},$$

because this implication says "from the fact that there is a way to find the value of *RNR* follows that there is a way to find the value of *VEH*". The name in curly brackets denotes in constructive logic a function that realizes the implication. In our case it is the atomic service *all_vehicles*. The general rule is that a service can be represented by an implication where its inputs are conjuncts on the left side and outputs are conjuncts on the right side. This is also how all services from a service model can be represented in the logic. But we have higher order nodes as well. In this case one has to consider a subtask as an extra input. A subtask itself is represented by an implication, in our example it is

$$LNR \supset VAL\{\varphi\},$$

where it has an unknown realization denoted by a functional variable φ. Adding this implication to inputs of *loop* node, we get the following formula for the *loop*:

$$(LNR \supset VAL)\{\varphi\} \wedge VEH \supset RESULT\{loop(\varphi)\}$$

A logical description of the complete workflow from Figure 5 is the following:

$$\left.\begin{array}{c} RNR \supset VEH\{all\_vehicles\} \\ (LNR \supset VAL)\{\varphi\} \wedge VEH \supset RESULT\{loop(\varphi)\} \\ LNR \supset T \wedge Y\{find\_vehicle\} \\ T \wedge Y \supset VAL\{find\_value\}. \end{array}\right\} (*)$$

In our implementation, the whole service model (hundreds of atomic services) is represented in the logic as above.

$$\dfrac{RNR \supset VEH\{\textbf{all\_vehicles}\} \qquad \dfrac{(LNR \supset VAL)\{\varphi\} \wedge VEH \supset RESULT\{\textbf{loop}(\varphi)\} \qquad \dfrac{LNR \supset T \wedge Y\{find\_vehicle\} \quad T \wedge Y \supset VAL\{\textbf{find\_value}\}}{LNR \supset VAL\{find\_vehicle; find\_value\}}(SSP2)}{VEH \supset RESULT\{loop(find\_vehicle; find\_value)\}}(SSP1)}{RNR \supset RESULT\{all\_vehicles; loop(find\_vehicle; find\_value)\}}(SSP2)$$

Figure 6. Proof of $RNR \supset RESULT$.

When a new composite service with inputs $X_1, \ldots, X_m$ and outputs $Y_1, \ldots, Y_n$ has to be built, a goal is given in the form of an implication

$$X_1 \wedge \ldots X_m \supset Y_1 \wedge \ldots Y_n\{\psi\},$$

where ψ is a functional variable denoting the composite service that has to be found. Let us have the goal

$$RNR \supset RESULT\{\psi\},$$

and assume that the formulas (*) are included in the service model as a part of the model. Then our tool searches for a proof of the goal, and after finding it, translates the proof into required form of the service description. The proof in our case includes only three steps as we see in Figure 6. The first step is deriving $LNR \supset VAL$ from $LNR \supset T \wedge Y$ and $T \wedge Y \supset VAL$ using a rule of structural synthesis denoted by *SSP2*, and described below, etc.

One can see that at deriving a new formula also its realization is built, and the whole program appears gradually step by step: first *find_vehicle; find_value*, then *loop(find_vehicle; find_value)* and finally ψ*=all_vehicles;loop(find_vehicle; find_value)*.

A special feature of this proof is that its every step corresponds to an application of at least one atomic service (shown in bold font in Figure 6). This is achieved due to a special form of inference rules – the SSP rules. These rules are admissible rules of intuitionistic logic, i.e., they enable one to construct only logically correct proofs. From the other side, these rules are very good for proof search, because there is no need to make a separate step for every logical connective (conjunction or implication). A notation of a respective rule is shown at each derivation step. The SSP rules are shown here with metasymbols *A,B,C,D,G,Z,W* denoting conjunctions of propositional variables and *X* denoting a conjunction of propositional variables and implications that represent subtasks:

$$\frac{(A \supset B) \wedge X \supset Z : f \quad A \wedge W \supset B : g}{X \wedge W \supset Z : f(g)} \qquad \text{(SSP1)}$$

$$\frac{A \supset B \wedge C : f \quad B \wedge D \supset G : g}{A \wedge D \supset C \wedge G : f; g} \qquad \text{(SSP2)}$$

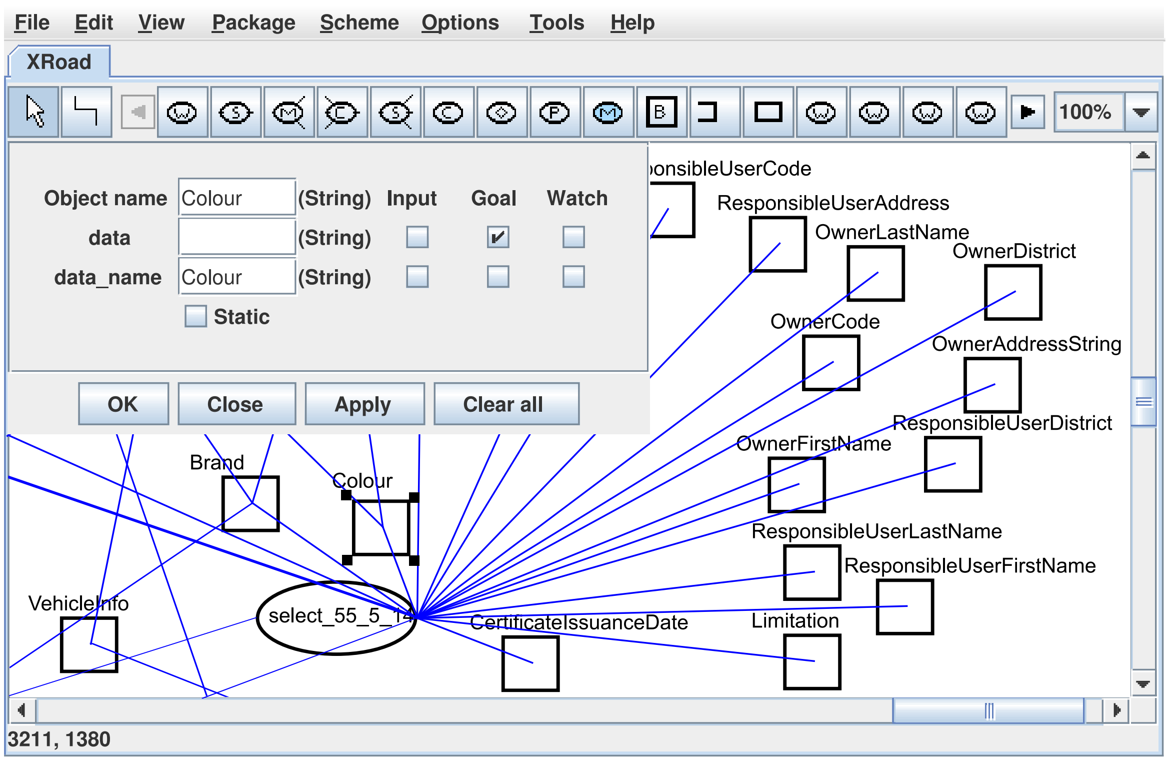
This is a brief explanation of the logic used for synthesis of compositions of atomic services. The tool for automatic composition of services has been implemented in the software environment CoCoViLa that includes an SSP-based algorithm synthesis part and is able to handle specifications given in the visual or textual form [8]. Internal representation of logical formulas in CoCoViLa is not a text, i.e., not formulas as we see them here, but a complex data structure with cross-references, designed with the aim of providing the required performance even in the case of a large number of atomic services. However, the implementation respects precisely the logic explained here. An example in the next section demonstrates the usage of this tool in more detail.

5. Implementation

We have implemented a prototype tool for automatic composition of services in the software environment CoCoViLa that includes an SSP-based algorithm synthesis part and is able to handle specifications given in the visual or textual form.

A visual language for representing Web services, control nodes, data resources and their connections has been developed. By using the visual language, visual service models can be constructed. Known inputs and desired outputs from which the goal is formulated can be defined on the model. A visual model and the goal are automatically translated into a textual specification that is used to generate a Java program if the proof of solvability of the problem can be built. By running the Java program, we can generate BPEL or OWL-S description of the complex service or execute complex service from the Java code.

Figure 7 shows the order of steps that are done by our composition tool in order to compose a new complex service. To use the composition system, developer of new complex service has to know how to define the goal, i.e., desired outputs and needed inputs of the complex service. After inputs and outputs have been defined, a service composition algorithm will be synthesized automatically. The structure of this algorithm already represents the structure of the complex service to be generated. However, CoCoViLa



			Input	Goal	Watch
Object name	Colour	(String)			
data		(String)	☐	✓	☐
data_name	Colour	(String)	☐	☐	☐

☐ Static

3211, 1380

Figure 8. Specifying a compound X-Road service on CoCoViLa service model.

produces only a Java code, but we need a representation in BPEL or OWL-S. Therefore the steps of the code use preprogrammed generators of BPEL or OWL-S. When this code is run, a service description corresponding to the structure of the synthesized algorithm is generated using additional information from the initial specification.

All intermediate steps, that is steps between defining a goal on the model and getting complex service description as an output, are done automatically by the composition tool. However, if necessary, intermediate steps (e.g., a structure of the complex service or a generated Java code) can be visualized for the developer, for instance, for debugging purposes.

We have visualized a syntactic service model for X-Road services with our service composition tool. Model used in our composition tool is generated from the one described in Section 1. Figure 8 shows a small part of the X-Road model and data properties window for a resource. Services are represented by ovals and data resources by squares in our visual language.

To illustrate the composition process described in previous section, let us consider a task where an official has to identify graduate's home address, occupation area and some attributes of its car, e.g., license number and color of the car. Input is the graduation certificate of the person. Getting a required service includes the following manual steps.

1. Using an ontology/dictionary, find the name of the considered data items in the service model. These names in the present case happen to be the following: *GraduationCertificate*, *EstonianAddressString*, *OccupationArea*, *RegistrationMark*, *Colour*. Note that these data items belong to different databases managed by different organizations.

2. Mark the input (*GraduationCertificate*) and requested outputs (*EstonianAddressString*, *RegistrationMark*, *Colour*, *OccupationArea*) on a visual representation of the service model.

3. Define the settings for formatting output (e.g., BPEL) and some information related to the generated complex service, for instance, its name, filename the service description is written to, etc. The rest will be done by our composition tool.

A developer can define output by marking it to be a goal as shown in the Figure 8. The model can be zoomed in and adjusted for more detailed analysis. Composition tool includes a search window, to ease the finding of data resources

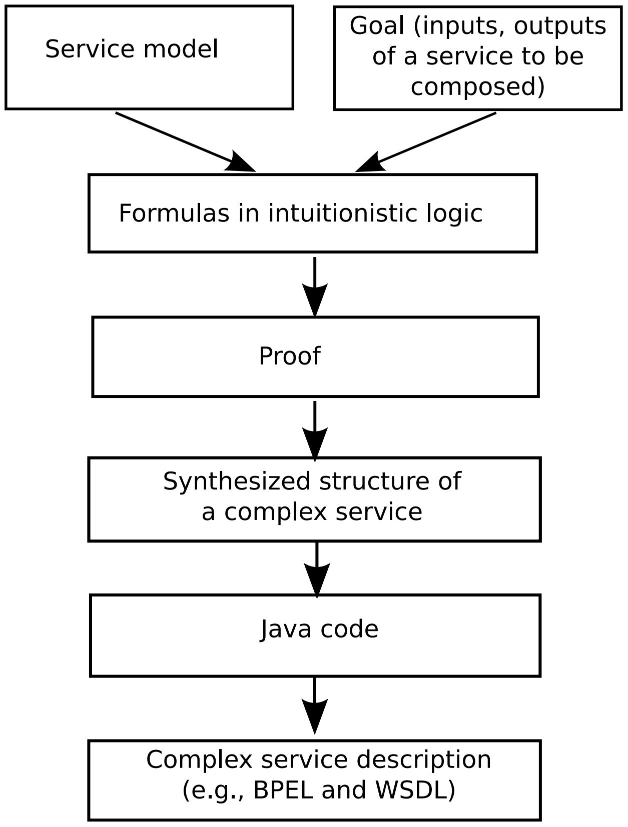

Service model
Goal (inputs, outputs of a service to be composed)
Formulas in intuitionistic logic
Proof
Synthesized structure of a complex service
Java code
Complex service description (e.g., BPEL and WSDL)

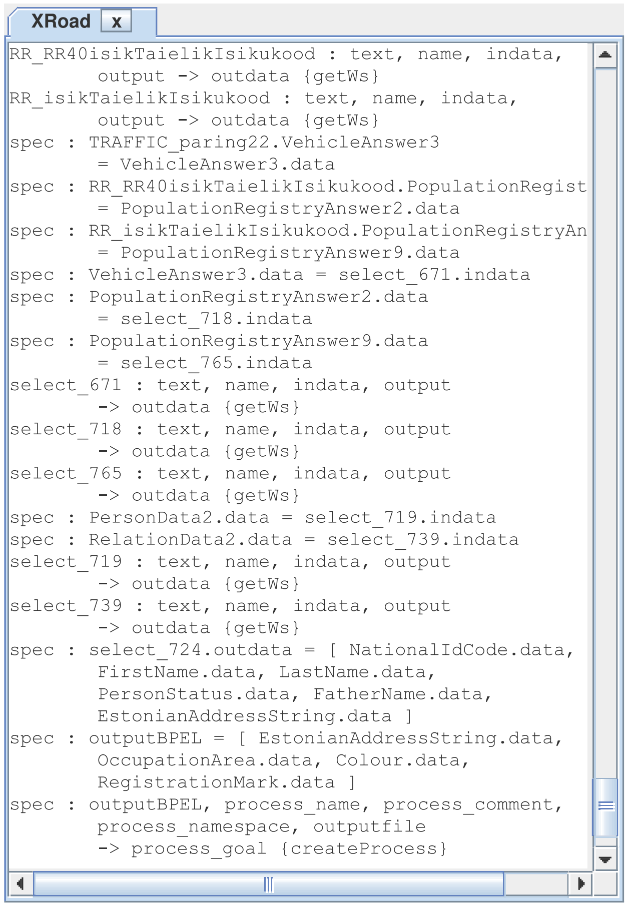
Figure 7. Automatic steps in composition process.

or services. If a goal of getting output data from the input data is provable, a complex service and its specification in BPEL will be generated.

The proof of reachability of the goal is constructed automatically by CoCoViLa as described in Section 4. The proof obtained gives the structure of the service to be constructed, this is also an algorithm to construct the service. In the present example, the algorithm includes 1434 lines, and part of it is visualized by CoCoViLa in a separate window shown in Figure 9. Algorithm gives an order in which services should be executed. Lines starting with service name (e.g., *RR_RR40isikTaielikIsikukood*, *select_739*), are representing services that need to be invoked in order to compute the goal. After the service name there is an implication and a function implementing it:

$$text, name, indata, output- > outdata\{\texttt{getWs}\}.$$

This line specifies that having an input (*text*, *name*, *indata*, *output*), we know that output – *outdata*, is computable using a function `getWS`. Functions referred to from the algorithm are Java functions that will be used in Java code and are executed when the code is run.



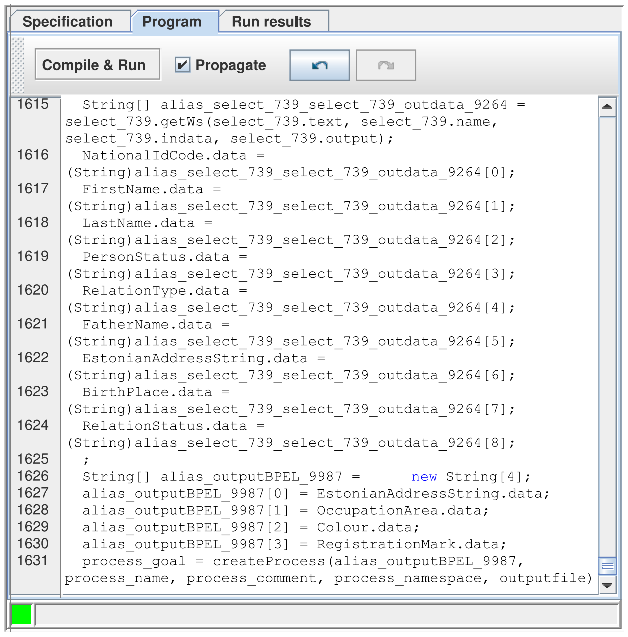
```
RR_RR40isikTaielikIsikukood : text, name, indata,
          output -> outdata {getWs}
RR_isikTaielikIsikukood : text, name, indata,
          output -> outdata {getWs}
spec : TRAFFIC_paring22.VehicleAnswer3
          = VehicleAnswer3.data
spec : RR_RR40isikTaielikIsikukood.PopulationRegist
          = PopulationRegistryAnswer2.data
spec : RR_isikTaielikIsikukood.PopulationRegistryAn
          = PopulationRegistryAnswer9.data
spec : VehicleAnswer3.data = select_671.indata
spec : PopulationRegistryAnswer2.data
          = select_718.indata
spec : PopulationRegistryAnswer9.data
          = select_765.indata
select_671 : text, name, indata, output
          -> outdata {getWs}
select_718 : text, name, indata, output
          -> outdata {getWs}
select_765 : text, name, indata, output
          -> outdata {getWs}
spec : PersonData2.data = select_719.indata
spec : RelationData2.data = select_739.indata
select_719 : text, name, indata, output
          -> outdata {getWs}
select_739 : text, name, indata, output
          -> outdata {getWs}
spec : select_724.outdata = [ NationalIdCode.data,
          FirstName.data, LastName.data,
          PersonStatus.data, FatherName.data,
          EstonianAddressString.data ]
spec : outputBPEL = [ EstonianAddressString.data,
          OccupationArea.data, Colour.data,
          RegistrationMark.data ]
spec : outputBPEL, process_name, process_comment,
          process_namespace, outputfile
          -> process_goal {createProcess}
```

Figure 9. Synthesized structure of a complex service.

Some checks about availability of relevant information (service name, output filename, etc.) necessary for composed service description are also done. This is shown in the last line (starting with *spec*), which shows that given goal is provable (this is indicated with *process_goal* as output), if the following inputs are given: text generated so far (*outputBPEL*) and information about complex service (*process_name*, *process_comment*, *process_namespace*). To compute the goal, method *createProcess* will be used. Lines starting with *spec* contain assignments that need to be done in order to prove the goal.

Java code is extracted from the synthesized structure of the complex service and data from the initial model (e.g., grounding of services and requested composition description language). Screenshot of a piece of the Java code made visible to the service developer can be seen in Figure 10. Program includes more than 1600 lines of code.

Extracted Java code is not the code of complex service, but a complex service description generator. Functions shown in curly brackets, e.g., `getWs`, as realizations of implications in Figure 9 are part of the generated Java code. See for example line number 1615, where object *select_739* has function `getWs`, with arguments that were given as input for the implication shown in Figure 9. Function `getWs` is taking BPEL construct given in *text*, includes information specific for the given service (given in *name*, *indata* and *output*) and adds generated construct to the BPEL output generated before. Similarly *process_goal* will be computed with `createProcess` function. This is shown in line 1631.



```
1615   String[] alias_select_739_select_739_outdata_9264 =
     select_739.getWs(select_739.text, select_739.name,
     select_739.indata, select_739.output);
1616   NationalIdCode.data =
     (String)alias_select_739_select_739_outdata_9264[0];
1617   FirstName.data =
     (String)alias_select_739_select_739_outdata_9264[1];
1618   LastName.data =
     (String)alias_select_739_select_739_outdata_9264[2];
1619   PersonStatus.data =
     (String)alias_select_739_select_739_outdata_9264[3];
1620   RelationType.data =
     (String)alias_select_739_select_739_outdata_9264[4];
1621   FatherName.data =
     (String)alias_select_739_select_739_outdata_9264[5];
1622   EstonianAddressString.data =
     (String)alias_select_739_select_739_outdata_9264[6];
1623   BirthPlace.data =
     (String)alias_select_739_select_739_outdata_9264[7];
1624   RelationStatus.data =
     (String)alias_select_739_select_739_outdata_9264[8];
1625   ;
1626   String[] alias_outputBPEL_9987 =      new String[4];
1627   alias_outputBPEL_9987[0] = EstonianAddressString.data;
1628   alias_outputBPEL_9987[1] = OccupationArea.data;
1629   alias_outputBPEL_9987[2] = Colour.data;
1630   alias_outputBPEL_9987[3] = RegistrationMark.data;
1631   process_goal = createProcess(alias_outputBPEL_9987,
     process_name, process_comment, process_namespace, outputfile)
```

Figure 10. Extracted Java program.

The final step of the composition is generation of the Web service description in the final form, e.g., in BPEL as requested in the example. This is done by compiling the synthesized Java code and running it. Figure 11 shows a fragment of the BPEL output (about 50 lines totally) of the generated service. Figure includes first half of the BPEL sequence with nine service invocations to different databases (four of which are shown), that need to be done in order to satisfy the simple goal that we were using as an example. In addition to BPEL sequence, represented in the figure, it is possible to generate other BPEL constructs (e.g., *while*, *condition*). It is also possible to create generators for other languages. So far we have only experimented with BPEL and OWL-S. It is important to notice that the steps of proving, compiling and service text generation are performed automatically, without interference of the user.

The synthesis algorithm has linear time complexity and can be applied to very large syntactic models. The time spent for solving the example here was about one second on a laptop with 1.2 GHz Intel processor.

```
...
<sequence>
<receive createInstance="yes" name="start"
operation="getComplexService"
partnerLink="XRoadClientPL"
portType="wsdl:XRoadClientP"
variable="Request" />
<invoke name="select_4_1_5"
partnerLink="XRoadClientPL"
operation="select_4_1_5"
inputVariable="getselect_4_1_5"
outputVariable="select_4_1_5Response"/>
<invoke name="TRAFFIC_paring22"
partnerLink="XRoadClientPL"
operation="TRAFFIC_paring22"
inputVariable="getTRAFFIC_paring22"
outputVariable="TRAFFIC_paring22Response"/>
<invoke name="RR_RR40isikTaielikIsikukood"
partnerLink="XRoadClientPL"
operation="RR_RR40isikTaielikIsikukood"
inputVariable=
"getRR_RR40isikTaielikIsikukood"
outputVariable=
"RR_RR40isikTaielikIsikukoodResponse"/>
<invoke name="RR_isikTaielikIsikukood"
partnerLink="XRoadClientPL"
operation="RR_isikTaielikIsikukood"
inputVariable="getRR_isikTaielikIsikukood"
outputVariable=
"RR_isikTaielikIsikukoodResponse"/>
...
```

Figure 11. BPEL fragment of a composed service.

6. Related work

A number of methods for dynamic composition of Web services have been proposed since the introduction of Web services standards. Majority of them fall into one of the following two categories: methods based on pre-defined workflow models and methods, which build the workflows from scratch. For the methods in the first category, the user should specify the workflow of the required composite service, including both nodes and the control flow and the data flow between the nodes. The nodes are regarded as abstract services that contain search templates. The concrete services are selected and bound at runtime according to the search recipes (see [9] and [10], for instance).

The second category includes methods related to AI planning, automated theorem proving, graph search, etc. They are based on the assumption that each Web service is an action which alters the state of the world as a result of its execution. Since Web services (actions) are software components, the input and the output parameters of Web services act as preconditions and effects in the planning context. After a user has specified inputs and outputs required by the composite service, a workflow (plan) is generated automatically by AI planners or other tools from the scratch.

Theoretically any domain-independent AI planner can be applied for Web service composition. In [11] SHOP2 planner is applied for automatic composition of DAML-S services. Other planners, which have been applied for automated Web service composition, include [12],[13],[14],[15], just to mention a few of them.

Waldinger [16] proposes initial ideas for a deductive approach for Web services composition. The approach is based on automated deduction and program synthesis and has its roots in the work presented in [17]. Initially available services and user requirements are described with a first-order language, related to classical logic, and then constructive proofs are generated with Snark [18] theorem prover. From these proofs workflows can be extracted.

Although only conjunctions are allowed for describing services and user requirements in most cases of deductive composition methods, Lämmermann [19] takes advantage of disjunctions in intuitionistic logic as well. Disjunctions are used to describe exceptions, which may be thrown during service invocations.

McDermott [20] tackles the closed world assumption in AI planning while composing Web services. He introduces a new type of knowledge, called value of an action, which allows modeling resources or newly acquired information – entities, which until this solution were modeled extra-logically. Anyway, while using resource-conscious logics, like linear logic, applied by Rao et al [21], or transition logic, this problem is treated implicitly and there is no need to distinguish informative and truth values. Since linear logic is not based on truth values, we can view generated literals as references to informative objects.

Hull and Su [22] present a short overview of tools and models for Web service composition. The models include OWL-S, the Roman model [23] and the Mealy machine [24]. While OWL-S includes a rich model of atomic services and how they interact with an abstraction of the "real world", the Roman model and the Mealy machine use a finite state automata framework for representing workflows.

Hashemian and Mavaddat [25] combine breadth-first graph search and interface automata [26] for automating Web service composition. While graph search is used for finding a path with minimum length from identified input nodes to identified output nodes, interface automata is applied for composing paths into a composite Web services. Graph search operates over a directed graph, where edges represent available Web services and nodes represent inputs/outputs of particular Web services.

Although there is enormous amount of literature available describing different composition methods and methodologies, not so many graphical composition environments such as CoCoViLa have been described and implemented so far. Sirin et al [27] propose a semi-automatic Web service composition scheme for interactively composing new Semantic Web services. Each time a user selects a new Web service, the Web services, that can be attached to inputs and outputs of the selected service, are presented to the user. Much manual search is avoided in this way. The process could be fully automated by applying our methodology if user requirements to the resulting service are known a priori. CoCoViLa complements this tool by providing a GUI for supporting specification of user requirements and synthesis of solutions based on them.

Gómez-Pérez et al [28] describe another graphical tool for Semantic Web service composition. This tool enables the user to specify graphically the input/output interactions among the sub-services that constitute the required service. Once the design has been checked, wrappers perform the translations from the instances of framework ontologies into the OWL-S specification.

Rao et al [29] describe a tool for mixed initiative framework for semantic Web service discovery and composition that aims at flexibly interleaving human decision making and automated functionality in environments where annotations may be incomplete and even inconsistent. An initial version of this framework has been implemented in SAPs Guided Procedures, a key element of SAPs Enterprise Service Architecture (ESA). This is a graphical tool for aiding composition if no or only partial semantic annotations of Web services are given.

Hakimpour et al [30] present a tool based on the model that supports a user-guided interactive composition approach, by recommending component Web services according to the composition context. This tool is based on a model for composition of Web services, which complements the WSMO orchestration in IRS-III – a framework for semantic Web services based on WSMO specification.

Since service composition is an application and an integral part of semantic interoperability, we shall enlist here some of these initiatives as well. In addition to German initiative Deutschland Online [31], Italian initiative in public administration [32] there are Finnish semantic initiative FinnONTO [33], and Semantic Latvia project [34]. The scope and accent of these initiatives are quite different – some focus on consolidating semantic assets in several governmental institutions already in place into semantic portals, some on building full-scale national semantic web infrastructures, others target syntactic or semantic descriptions of data schemas, some are on the level of human-oriented descriptions of assets, others try to reach automatic use. Estonian semantic interoperability initiative [35] is focused on providing semantic descriptions of public Web services and thus provides a valuable source for evaluating automated composition tools in large scale.

There are also pan-European initiatives, which include SEMIC (SEMantic Interoperability Centre Europe) [36], led by the European Commissions IDABC program [37], and semanticGov [38]. This shows a real need for automation of composition of governmental Web services.

7. Concluding remarks

We have shown in the present work the feasibility of automatic composition of Web services on a very large syntactic service model of governmental services. This approach can be used without any changes for composition of services for large companies as soon as a federated syntactic service model can be built. It could be useful, first of all, for software developers who are extending and modifying a large existing information system that in the present case provides Web-based services for citizens. The advantages of this approach are, first, provable ease of introduction of new services and, second, guaranteed correctness of the services. This approach is scalable to very large service models as can be seen from the synthesis time of services and space requirements of the service model. The available tool supports easy maintenance of the service model – it can be modified on the fly.

In principle, the described tool, after adjusting its user interface, could be given to end users, so that they could develop wanted services by themselves. However, in the present form it is impossible, because it would require more skills that an average citizen has. Making composition of services publicly available may include also security risks. Our experience shows that the main difficulty that a user would have is the unsolvability of the synthesis task – more inputs will be needed than given in the initial service description. This is a semantic debugging problem where partial evaluation could give some help. The partial evaluation on models similar to syntactic service models has been investigated in [39]. The presented work concerns only stateless services. In the case of stateful services, several services have to work simultaneously and have to be orchestrated respectively. Experiments based on CoCoViLa support this by means of higher-order service schemas [7]. However, in the present work the services are stateless, and we do not use this feature.

Acknowledgments

This work is partially supported by the Estonian Science Foundation Grant No. 6886 and Estonian Information Technology Foundation's Tiger University program.

References

[1] R. Maigre, P. Küngas, M. Matskin, and E. Tyugu. Handling large Web services models in a federated governmental information system. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 626–631, Washington, DC, USA, 2008. IEEE Computer Society.

[2] Information technology in public administration of Estonia. Yearbook 2006, Estonian Ministry of Economic Affairs and Communication, 2007.

[3] P. Küngas and M. Matskin. From Web services annotation and composition to web services domain analysis. *International Journal of Metadata, Semantics and Ontologies*, 2(3):157–178, 2007.

[4] P. Grigorenko, A. Saabas, and E. Tyugu. Visual tool for generative programming. *ACM SIGSOFT Software Engineering Notes*, 30(5):249–252, 2005.

[5] G. Mints and E. Tyugu. Justifications of the structural synthesis of programs. *Sci. Comput. Program.*, 2(3):215–240, 1982.

[6] S. C. Kleene. *Introduction to metamathematics*. Elsevier, 1980.

[7] M. Matskin, R. Maigre, and E. Tyugu. Compositional logical semantics for business process languages. In *Proceedings of Second International Conference on Internet and Web Applications and Services (ICIW 2007)*. IEEE Computer Society, 2007.

[8] M. Matskin and E. Tyugu. Strategies of structural synthesis of programs and its extensions. *Computing and Informatics*, 20:1–25, 2001.

[9] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE 2000)*, volume 1789 of Lecture Notes in Computer Science, pages 13–31. Springer-Verlag, 2000.

[10] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE 2000)*, volume 1789 of Lecture Notes in Computer Science, pages 247–263. Springer-Verlag, 2000.

[11] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web services composition using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, 2003.

[12] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proceedings of the 11th International Conference on Artificial Intelligence, Methodologies, Systems, and Applications (AIMSA 2004)*, volume 3192 of Lecture Notes in Computer Science, pages 106–115. Springer-Verlag, 2004.

[13] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS Web services. In *Proceedings of 2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 293–301, 2005.

[14] M. Sheshagiri, M. desJardins, and T. Finin. A planner for composing services described in DAML-S. In *Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering*, 2003.

[15] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *Proceedings of 3rd International Semantic Web Conference (ISWC 2004)*, volume 3298 of Lecture Notes in Computer Science, pages 380–394. Springer-Verlag, 2004.

[16] R. Waldinger. Web agents cooperating deductively. In *Proceedings of FAABS 2000*, volume 1871 of Lecture Notes in Computer Science, pages 250–262. Springer-Verlag, 2000.

[17] Z. Manna and R. J. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[18] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In *Proceedings of 12th International Conference on Automated Deduction (CADE 1994)*, volume 814 of Lecture Notes in Artificial Intelligence, pages 341–355. Springer-Verlag, 1994.

[19] S. Lämmermann. *Runtime service composition via logic-based program synthesis*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, 2002.

[20] D. McDermott. Estimated-regression planning for interaction with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*. AAAI Press, 2002.

[21] J. Rao, P. Küngas, and M. Matskin. Composition of semantic Web services using linear logic theorem proving. *Information Systems, Special Issue on the Semantic Web and Web Services*, 31(4-5):340–360, 2006.

[22] R. Hull and J. Su. Tools for composite Web services: A short overview. *SIGMOD Record*, 34(2):86–95, 2005.

[23] D. Berardi, D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proceedings of the First International Conference on Service-Oriented Computing (ICSOC 2003)*, volume 2910 of Lecture Notes in Computer Science, pages 43–58. Springer-Verlag, 2003.

[24] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of 12th International World Wide Web Conference (WWW 2003)*, pages 403–410, 2003.

[25] S. V. Hashemian and F. Mavaddat. A graph-based approach to Web services composition. In *Proceedings of 2005 IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005)*, pages 183–189. IEEE Computer Society, 2005.

[26] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering(ESEC 2001)*, pages 109–120. ACM Press, 2001.

[27] E. Sirin, B. Parsia, and J. Hendler. Composition-driven filtering and selection of Semantic Web services. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series*, pages 129–136. AAAI Press, 2004.

[28] A. Gómez-Pérez, R. Gonzalez-Cabero, and M. Lama. A framework for design and composition of Semantic Web services. In *In Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series*, pages 113–120. AAAI Press, 2004.

[29] J. Rao, D. Dimitrov, P. Hofmann, and N. Sadeh. A mixed initiative approach to semantic web service discovery and composition: Sap's guided procedures framework. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 401–410, Washington, DC, USA, 2006. IEEE Computer Society.

[30] F. Hakimpour, D. Sell, L. Cabral, J. Domingue, and E. Motta. Semantic web service composition in irs-iii: The structured approach. In *CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*, pages 484–487, Washington, DC, USA, 2005. IEEE Computer Society.

[31] Deutschland Online. `http://www.deutschland-online.de/DOL_en_Internet/broker.jsp`. [May 15, 2009].

[32] Italian initiative in public administration. `http://www.cnipa.gov.it/site/it-IT/`. [May 15, 2009].

[33] E. Hyvönen, K. Viljanen, J. Tuominen, and K. Seppälä. Building a national semantic web ontology and ontology service infrastructure -The FinnONTO approach. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008*, pages 95–109, 2008.

[34] G. Barzdins, R. Balodis, K. Cerans, A. Kalnins, M. Opmanis, and K. Podnieks. Towards semantic Latvia. In *Communications of 7th International Baltic Conference on Databases and Information Systems*, pages 203–218. Technika, 2006.

[35] H.-M. Haav, A. Kalja, P. Küngas, and M. Luts. Ensuring large-scale semantic interoperability: The estonian public sector's case study. In H.-M. Haav and A. Kalja, editors, *Databases and Information Systems V – Selected Papers from the Eighth International Baltic Conference, DB&IS 2008*, pages 117 – 128. IOS Press, 2008.

[36] SEMantic Interoperability Centre Europe. `http://www.semic.eu/semic/view/snav/About_SEMIC.xhtml`. [May 15, 2009].

[37] European Commission's IDABC program. `http://ec.europa.eu/idabc/`. [May 15, 2009].

[38] T. Vitvar, M. Kerrigan, A. van Overeem, V. Peristeras, and K. Tarabanis. Infrastructure for the semantic pan-european e-government services. In *Proceedings of the 2006 AAAI Spring Symposium on The Semantic Web meets eGovernment (SWEG)*, 3 2006.

[39] P. Küngas and M. Matskin. Detection of missing Web services: The partial deduction approach. *Special Issue on Recent Innovations in Web Services Practices, International Journal of Web Services Practices*, 1(1-2):133–141, 2005.