Interactive Discovery and Composition of Complex Web Services

Sergey Stupnikov[1], Leonid Kalinichenko[1], and Stephane Bressan[2]

[1] Institute of Informatics Problems, Russian Academy of Science
{ssa, leonidk}@synth.ipi.ac.ru
[2] National University of Singapore
steph@nus.edu.sg

Abstract. Among the most important expected benefits of a global service oriented architecture leveraging web service standards is an increased level of automation in the discovery, composition, verification, monitoring and recovery of services for the realization of complex processes. Most existing works addressing this issue are based on the Ontology Web Language for Services (OWL-S) and founded on description logic. Because the discovery and composition tasks are designed to be fully automatic, the solutions are limited to the realization of rather simple processes. To overcome this deficiency, this paper proposes an approach in which service capability descriptions are based on full first order predicate logic and enable an interactive discovery and composition of services for the realization of complex processes. The proposed approach is well suited when automatic service discovery does not constitute an absolute requirement and the discovery can be done interactively (semi-automatically) with human expert intervention. Such applications are, for instance, often met in e-science. The proposed approach is an extension and adaptation of the compositional information systems development (CISD) method based on the SYNTHESIS language and previously proposed by some of the authors. The resulting method offers a canonical extensible object model with its formal automatic semantic interpretation in the Abstract Machine Notation (AMN) as well as reasoning capabilities applying AMN interactively to the discovery and composition of web services.

1 Introduction

The current Web service infrastructure offers syntactic interoperability by means of widely accepted standards such as the Web Services Description Language (WSDL1.1 [27]), the Universal Description, Discovery and Integration language (UDDI [26]) and the Simple Object Access Protocol (SOAP [21]).

Yet semantic interoperability is one of the most important expected features of a global service oriented architecture. In order to reach the necessary level of automation of service discovery, composition, verification, monitoring and recovery, a large body of research works aims at devising richer specifications providing for semantically well-founded reasoning about services. Many such approaches

Y. Manolopoulos, J. Pokorný, and T. Sellis (Eds.): ADBIS 2006, LNCS 4152, pp. 216–231, 2006.

are based on OWL-S [17], a language for the semantic specification of services building upon OWL [16], the Ontology Web Language proposed by W3C. In OWL-S each service is provided with an advertisement containing three descriptions: *service profile* ("what the service does"), *service model* ("how the service works"), and *service grounding* ("how to access the service"). OWL-S and other similar languages assume mechanisms for their combined use with existing Web service standards such as UDDI and WSDL. The solutions proposed realize simple "on the fly" dynamic discovery – usually referred to as service matchmaking – and composition of services based on the service's capability descriptions, i.e. inputs, outputs, pre-conditions and effects (IOPEs) of a service [20].

Because of the limitations of the OWL description and of the reasoning mechanism based on description logic, and because the discovery and composition tasks are designed to be fully automatic, the solutions are limited to the realization of rather simple processes. Indeed, full automation applies to tractable problems (checking credit, simple procurement, etc.). Most importantly, such approaches do not apply to the problems of discovery and composition of services for the realization of complex processes that are in use in numerous application domains such as e-science.

In order to overcome this deficiency, this paper[1] proposes an approach in which IOPEs service capability descriptions based on full first order predicate logic that enable an interactive discovery and composition of services for the realization of complex processes. The proposed approach is well suited when automatic service discovery does not constitute an absolute requirement and the discovery can be done interactively (semi-automatically) with human expert intervention. The proposed approach extends and adapts the Compositional information systems Development (CISD) method [4] devised and proposed by some of the authors.

CISD is a method for the semantically correct composition of software components into coherent application. The CISD method is originally designed for object- oriented platforms such as CORBA, RMI and J2EE. CISD leverages an ontological model and a canonical object model, both based on the SYNTHESIS language [10], to offer a unified representation of both the new application (specification of requirement) and pre-existing components. Discovery and composition of components relevant to the application is realized in the framework offered by the domain ontology and the canonical object model.

In order to apply the CISD method to Web services, we have preliminarily defined in [5] a mapping of WSDL specifications into the canonical model. The basic steps for composing Web services by refining a specification of requirement were also demonstrated in [5].

In this paper, we concentrate on the discovery based on the IOPEs capabilities of services, putting aside other service properties (e.g., non functional properties) defined by other metadata facilities of the augmented UDDI. We use the canonical extensible object model of SYNTHESIS [10], its formal automatic

[1] This research has been partially supported by the grants 05-07-90413, 06-07-89188-a of the Russian Foundation for Basic Research and by NUS Eastern Europe Research Scientists & Students Exchange & Collaboration Programme (EERSS).

semantic interpretation in the Abstract Machine Notation (AMN) [2] and reasoning mechanisms obtained by applying AMN interactively. If a service is defined in WSDL or UDDI augmented with OWL-S, such definition is also assumed to be mapped into the canonical model. How to do such mapping for OWL is considered in detail in [12,15].

The paper is structured as follows. In Section 2 we give a brief introduction to the canonical model. We define the operations of the type composition calculus based on the *refinement*[2] relation. We show the need for a formal proof of the refinement relation between type specifications. B-Technology that implements AMN [1] is used for that purpose. We give a short characterization of AMN as well as of the program facilities that have been recently developed [24] for the automatic mapping of the canonical model specifications into AMN.

In Section 3, we present an example with two Web services. The first Web service is specified in OWL-S and WSDL. Pre-conditions and effects in this service are defined using Semantic Web Rule Language (SWRL) [25] which is tractable. The second Web service is described with formulae expressed in the canonical model to describe pre-conditions and effects. Such formulae are generally not tractable (as they make use of full first order predicate logic). Appropriate composition of services is realized by using the type specification calculus [11]. To show that the discovery of Web services is correct, we prove that a relevant type of specification of requirements is refined by the composition of Web services obtained. In Section 4, we illustrate an approach to the formal proof of this condition showing also some details of the mapping of the canonical model into AMN. In section 5, we survey the related work. In the conclusion section, we summarize the contribution of the paper.

2 Complex Service Discovery for Compositional IS Development

In CISD, the SYNTHESIS language is intended to provide a uniform (canonical) representation of heterogeneous data, programs and processes for their use as interoperable entities. Strongly typed, object-oriented subset of the language (as required in this paper) contains a universal constructor of arbitrary abstract data types, a comprehensive collection of the built-in types, as well as type expressions based on the operations of type calculus.

All operations over typed data in the SYNTHESIS language are represented by functions. Functions are given by predicative specifications expressed by mixed pre- and post-conditions formulae of typed first order predicate logic.

In the SYNTHESIS language the type specifications are syntactically represented by frames, their attributes by slots of the frames. Frames are embraced by figure brackets { and }, slots are represented as pairs $\langle$*slot name*$\rangle$: $\langle$*slot value*$\rangle$ (a frame can be used as a slot value). Slots in a frame are separated by semi-colons.

[2] A non-formal definition of refinement is as follows. Type *A refines* type *B* if *A* can be substituted instead of *B* so that a user does not notice the difference.

Compositional development is a process of systematic manipulation and transformation of specifications. Type specifications of the SYNTHESIS language are chosen as the basic units for such manipulation. The manipulations required include decomposition of type specifications into consistent fragments, identification of reusable fragments, composition of identified fragments into specifications refining [2] the requirements, justification of the refinement relation reached. The compositional specification calculus [11], designed for such manipulations, uses the following concepts and operations.

Definition 1. *A* **signature** Σ_T *of a type specification* $T = \langle V_T, O_T, I_T \rangle$ *includes a set of operation symbols* O_T *indicating operation arguments and result types and a set of predicate symbols* I_T *for invariants. Conjunction of all invariants in* I_T *constitutes the type invariant* Inv_T*.* O_T *is a union of type state attributes* Att_T *and type methods* $Meth_T$*. Extent* V_T *of the type* T *(carrier of the type) is modeled by a set of admissible instances of the type. Each instance of the type is a tuple of pairs* $\langle a, v \rangle$ *such that* a *is a state attribute of the type* ($a \in Att_T$) *and* v *is a value of the attribute. Every instance must satisfy the invariant* Inv_T*.*

Definition 2. *A* **type reduct** R_T *is a subspecification of an abstract data type* T *specification. A signature* Σ'_T *of* R_T *is a subsignature of* Σ_T *including the extent* V_T*, a set of operation symbols* $O'_T \subseteq O_T$*, a set of symbols of invariants* $I'_T \subseteq I_T$*.*

The identification of a fragment of an existing component type that may be reused in the implementation of another type requires a *most common reduct* of these types to be constructed.

Definition 3. Most common reduct $R_{MC}(T_1, T_2)$ *for types* T_1 *and* T_2 *is a reduct* R_{T_1} *of* T_1 *such that there exists a reduct* R_{T_2} *of* T_2 *such that* R_{T_2} *refines* R_{T_1} *and there can be no other reduct* R'_{T_1} *such that* $R_{MC}(T_1, T_2)$ *is a reduct of* R'_{T_1}*,* R'_{T_1} *is not equal to* $R_{MC}(T_1, T_2)$ *and there exists* R'_{T_2} *of* T_2 *that refines* R'_{T_1}*.*

Definition 4. *Type* C *is a* **refinement** *of type* R *iff:*

- *there exists an injective mapping* $Ops : O_R \to O_C$*;*
- *there exists a total abstraction function* $Abs : V_C \to V_R$*;*
- *for all* $v \in V_C$*,* $Inv_C(v)$ *implies* $Inv_R(Abs(v))$*;*
- *for all* $m \in Meth_R$*,* m *is refined by* $Ops(m)$*.*

To establish a method refinement m_1 *refines* m_2 it is required that method pre-condition $pre(m_2)$ implies pre-condition $pre(m_1)$ and post-condition $post(m_1)$ implies post-condition $post(m_2)$.

The type calculus operations (such as *meet* and *join*) are used for the composition of identified reusable fragments into specification refining the specification of requirements. The *meet* operation $T_1 \sqcap T_2$ produces a type T as an "intersection" of specifications of the operand types. The *join* operation $T_1 \sqcup T_2$ produces a type T as a "join" of specifications of the operand types [11]. To save space, we provide a complete definition only for *join* operation.

Definition 5. *Generally type T – a result of an operation $T_1 \sqcup T_2$ – includes a merge of specifications of T_1 and T_2. Common elements of specifications of T_1 and T_2 are included into the merge (resulting type) only once. Common elements of the types are defined by most common reducts $R_{MC}(T_1, T_2)$ and $R_{MC}(T_2, T_1)$. More formally $O_{T_1 \sqcup T_2}$ is defined as*

$$O_{T_1 \sqcup T_2} = (O_{T_1} \setminus O_{R_{MC}(T_1, T_2)}) \cup (O_{T_2} \setminus O_{R_{MC}(T_2, T_1)})$$

Type invariant of T is defined as a conjunction of operand types invariants $Inv_{T_1} \& Inv_{T_2}$.

The most important stage of CISD is the proof of the refinement relation between requirement type R and component composition of type C. Since predicative specifications of functions in the SYNTHESIS language are based on an undecidable first order logic, special methods and tools should be used for establishing of the refinement. In CISD for this purpose B-Technology [1] is used. B-Technology provides an implementation for formal specification language – Abstract Machine Notation (AMN) as well as tools (B-Toolkit[28]) for automatic/interactive proof of the refinement.

AMN [2] is based on the first order predicate logic and Zermelo-Frenkel set theory and enables to consider state space specifications and behavior specifications in an integrated way. The system state is specified by means of state variables and invariants over these variables, system behavior is specified by means of operations defined as generalized substitutions – a sort of predicate transformers. Refinement of AMN specifications is formalized as a set of refinement proof obligations – theorems of first order logic. Generally speaking in terms of pre- and post-conditions of operations, refinement of AMN machines means weakening pre-conditions and strengthening post-conditions of corresponding operations included in these constructions. Proof requests are automatically generated by B-Toolkit and should be proven with the help of B-Toolkit theorem prover.

To reduce refinement of SYNTHESIS type specifications to refinement of AMN specifications, specific program facilities were recently developed for automatic mapping of the canonical model (SYNTHESIS) specifications into AMN. These facilities were developed as a part of CISD tool [4] [5] and include a graphical user interface enabling an expert to select an appropriate type from the meta-information repository to map into AMN and a translator of the SYNTHESIS specifications into AMN. The paper introduces the principles of the canonical model into AMN mapping (section 4) and demonstrates their use for formal verification of complex services discovery and composition.

3 Specifications of Requirement and Web Services

Compositional development includes several stages mentioned on Fig. 1. In this paper we concentrate on the stage of formal verification of the refinement assuming the previous stages to be done and results obtained. The stage of formal verification is marked in grey on Fig. 1. Some details about the stages omitted including ontological relevance check can be found in [4][5].

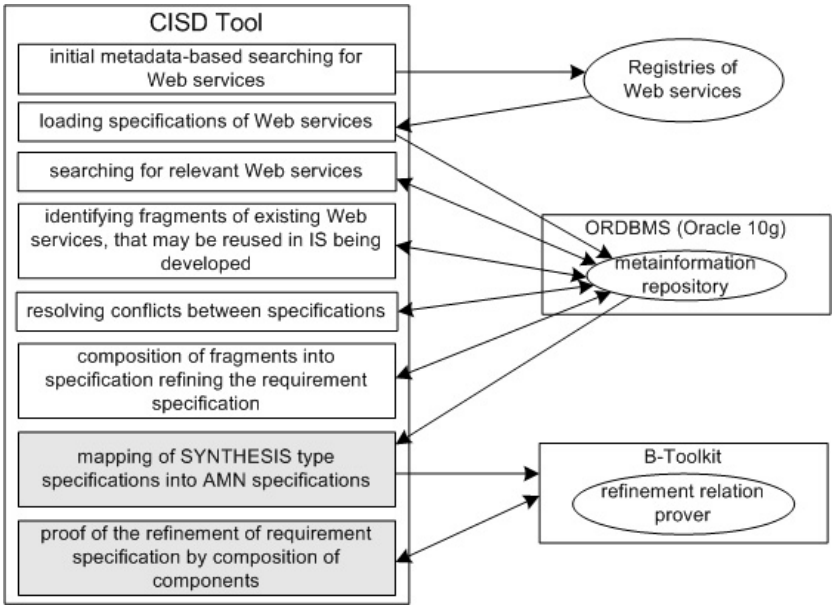

Fig. 1. CISD Tool structure

We shall consider formal verification of compositional development of a part of *Funding Agency* system managing finances aimed at support of research projects. A group of researchers prepares a proposal and registers it at the Funding Agency to obtain a grant. The funding Agency nominates experts to review proposals and decides whether to support a proposal or not.

As a example of a requirement specification consider the type `Secretary` of a Funding Agency system. A secretary of the agency identifies a set of available experts (`obtainExperts` method), searches for relevant experts and reports the number of relevant experts (`searchForExperts` method). An expert is considered to be relevant if her or his research area is `computer science`. The secretary selects some relevant experts to review a proposal (`dispatch` method). An expert can be selected to review a proposal if and only if her or his area of expertise is the same as the research area of the proposal. The area of expertise of an expert can be more specific than the research area of the expert, for example data mining can be an area of expertise.

`Secretary` type specification in the SYNTHESIS language is as follows.

```
{ Secretary; in: type;
  availableExperts: {set; type_of_element: Expert;};
  obtainExperts:  { in: function;
   params: {+exprts/{set; type_of_element: Expert;}};
   {{ this.availableExperts' = exprts }};
  };
  searchForExperts: { in: function; params: {-numberOfExperts/integer};
```

```
   {{ this.availableExperts' = {exp/Expert | in(exp, availableExperts) &
                                exp.researchArea = "computer sci"} &
      numberOfExperts = card(this.availableExperts')  }};
  };
  dispatch: { in: function; params: { +revi/Review };
    {{  ^isempty(this.availableExperts) &
         ex exp/Expert (in(exp, this.availableExperts) &
            exp.area_of_expertise = revi.forProposal.area &
            revi.byExpert' = exp)  }};
  };
}
```

Input and output parameters of methods are marked by `+` and `-` respectively, term `this` denotes a reference to an instance of a type for which a method is called. Terms marked by apostrophe refer to the post-state of the system, built-in boolean function `in` checks whether an item belongs to a set, `card` function returns a cardinality of a set, `isempty` function checks whether a set is empty, `^` denotes logical not, `ex` denotes existential quantifier.

We can assume that, as a result of the primary stages of compositional development, two Web services have been chosen to be relevant to `Secretary` type (requirement).

A first service is `Dispatcher` service specified in OWL-S with grounding in WSDL. Pre-conditions and effects in this service are defined using SWRL [12]. To save space the `Dispatcher` service XML code is omitted:

```
<wsdl:portType name="Dispatcher">
  <wsdl:operation name="getExperts" ... </wsdl:operation>
  <wsdl:operation name="checkExpert" ... </wsdl:operation>
  <wsdl:operation name="countExperts"  </wsdl:operation>
</wsdl:portType>
```

`Dispatcher` looks for all the specialists which are potential experts (operation `getExperts`). After that `Dispatcher` considers potential experts one by one and chooses relevant experts such that their research field is `computer science` (`checkExpert` method) and reports the number of relevant experts (`countExperts` method). Semantics of `Dispatcher` service is provided by the OWL-S service profile specification. To save space this XML code is omitted.

A second service is the `Executive` service specified in the SYNTHESIS language and uses for pre-conditions and effects the formulae of the canonical model that is not tractable (as it uses full first order predicate logic). A specialist can be appointed to review a submission if and only if his/her field of expertise is the same as research field of submission, specialists with PhD degree are preferred (`appoint` method).

```
{ Executive; in: type;
  relevantExperts: {set; type_of_element: Specialist;};
  expertsChecked: boolean;
  appoint: {in: function; params: { +revi/Evaluation };
```

```
    {{ this.expertsChecked = true & ^isempty(this.relevantExperts) &
        ( ex exp/Specialist(in(exp, this.relevantExperts) &
             exp.fieldOfExpertise = ev.submRef.field &
             exp.degree = "PhD" & revi.bySpecialist' = exp)  |
        ( all exp/(Specialist)((in(exp, this.relevantExperts) &
              exp.fieldOfExpertise = revi.submRef.field) ->
                   exp.degree <> "PhD") &
              ex exp/(Specialist)(in(exp, this.relevantExperts) &
                 exp.fieldOfExpertise = revi.submRef.field &
                 revi.bySpecialist' = exp) ) ) }};
    };
}
```

To satisfy the requirement specified by type `Secretary` it is required to create an appropriate composition of `Dispatcher` and `Executive` components. For this purpose WSDL+OWL-S specification of `Dispatcher` should be mapped into the `Dispatcher` type of the canonical model:

```
{ Dispatcher; in: type;
  experts: {set; type_of_element: Specialist;};
  expertsGot: boolean;
  relevantExperts: {set; type_of_element: Specialist;};
  expertsChecked: boolean;

  getExperts: { in: function; ... };
  checkExpert: { in: function; ... };
  countExperts: {in: function; params: {-numberOfExperts/integer};
   {{ isempty(this.experts) & this.expertsGot = true &
      numberOfExperts = card(this.relevantExperts) &
      this.expertsChecked' = true
   }};
  };
}
```

The required composition intended to refine `Secretary` type is the join of `Dispatcher` and `Executive` types that is `DispatcherJOINExecutive` type:

```
{ DispatcherJOINExecutive; in: type;
  experts: {set; type_of_element: Specialist;};
  expertsGot: boolean;
  relevantExperts: {set; type_of_element: Specialist;};
  expertsChecked: boolean;

  getExperts: { in: function; ... };
  checkExpert: { in: function; ... };
  countExperts: {in: function; ... };
  appoint: {in: function; ... };
}
```

Discovery of relevant services includes also the establishment of ontological relevance of structure and methods of requirement and component types.

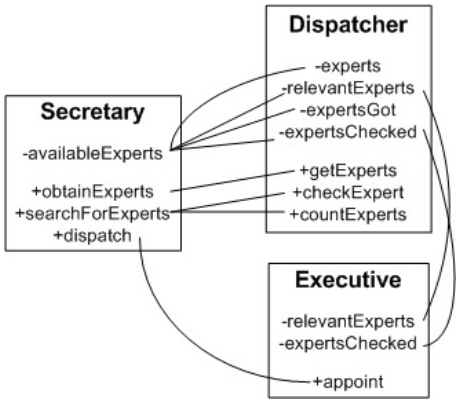

Fig. 2. Ontological relevance diagram

Specifications of requirement and pre-existing components must be associated with ontological contexts defining concepts of the respective subject areas. Here we assume that ontological concepts are described with their verbal definitions similarly to definitions of words in a dictionary. Fuzzy relationships between concepts of different contexts are established by calculating correlation coefficients between concepts on the basis of their verbal definitions. The correlation coefficients are calculated using the vector-space model [4]. This can be done with the help of CISD tool [4][5]. In this paper only the final picture of ontological relevance is presented on Fig. 2. Arcs on the figure denotes a relation of ontological relevance. For example, method `appoint` of type `Executive`, a pair of methods `checkExpert` and `countExpert`, four state attributes of type `Dispatcher` were established to be relevant to method `dispatch` of type `Secretary`, method `searchForExperts`, attribute `availableExperts` of type `Secretaty` respectively. Note that ontological relevance should be established also for types used in specification of requirement type `Secretary` (`Proposal`, `Expert`, `Review` types) and types used in composition of components type `Dispatcher` ⊔ `Executive` (`Submission`, `Specialist`, `Evaluation` types). In the example considered type `Proposal` was established to be relevant to the type `Submission`, type `Expert` – to the type `Specialist`, type `Review` – to the type `Evaluation`.

4 Formal Verification of Web Service Discovery

To show that the discovery of Web services is correct, we need to prove that a relevant type of specification of requirements `Secretary` is refined by the composition of Web services obtained (`Dispatcher` ⊔ `Executive`). To use automatic/interactive tools of B-technology for proving the refinement it is required to map `Secretary` and `Dispatcher` ⊔ `Executive` types into AMN.

In this section general principles underlying the mapping of the SYNTHESIS language into AMN is demonstrated by the example of mapping the `Secretary`

type into AMN. Mapping of abstract types into AMN is based on extensional principle: every type is represented by its extent – a constant set of possible instances of the type. For a set of interrelated types a special AMN construction is provided, so-called *context machine*.

MACHINE *FundingAgency_structureContext*
SETS *AVAL*; ...
CONSTANTS $Obj, \ldots, ext\_Review, ext\_Secretary, ext\_Expert, \ldots$
PROPERTIES $Obj \in POW(AVAL) \wedge ext\_Review \in POW(Obj) \wedge$
$ext\_Secretary \in POW(Obj) \wedge ext\_Expert \in POW(Obj) \ldots$
END

It contains a definition of the extent of all object types – *Obj* which is a subset of the set of all abstract values (*AVAL*) expressible in the language and definitions of extents of all types required (*ext_Secretary*, *ext_Expert*, etc.). An extent of every type (for example, `Secretary`) is a subset of *Obj*: $ext\_Secretary \in POW(Obj)$. $POW(Obj)$ denotes a set of all subsets of the set *Obj*.

An abstract type is represented in AMN by a separate construction:

REFINEMENT *Secretary*
INCLUDES *Review*
SEES *String_TYPE*, *Bool_TYPE*, *Expert*, *FundingAgency_structureContext*, ...
VARIABLES *available_experts*
INVARIANT $available\_experts \in ext\_Secretary \rightarrow POW(ext\_Expert)$
OPERATIONS ...

REFINEMENT is the most universal AMN construction, since it can be used both as refined and as refining machine in the hierarchy of refinement. Therefore this construction is most preferable for homogeneous representation of abstract types in AMN. This construction is composed with context machine, with machines corresponding to other types used (`Expert`, `Review`, etc.) and to auxiliary constructions (*String_TYPE*, *Bool_TYPE*) with the help of AMN composition facilities (SEES, INCLUDES [2]).

State attributes of a type is represented in AMN by variables. Variables are appropriately typed as total functions from extent of the type into a type of the attribute in the INVARIANT section. Methods of a type are represented in AMN as operations defined as generalized substitutions [2]. Every operation is of sort

$$op = S$$

Here *op* is an operation signature, S is a substitution, defining the effect of the operation on the state space. The Generalized Substitution Language (GSL [2]) provides for description of transitions between system states. Each generalized substitution S defines a predicate transformer, linking some post-condition R with its *weakest* pre-condition $[S]R$. This guarantees preservation of R after the operation execution. In such case we say that S *establishes* R. We shall use substitutions given in the table 1.

Here S, T stand for substitutions, x, y, t are variables, E, F denote expressions, G, P are predicates, $P\{x \rightarrow E\}$ denotes predicate P having all free occurrences of variable x replaced by E.

Table 1. The Generalized substitutions and their semantics

The generalized substitution S	$[S]P$
$x := E$	$P\{x \rightarrow E\}$
$x := E \mid\mid y := F$	$[x, y := E, F]P$
ANY t WHERE G THEN T END	$\forall t \bullet (G \Rightarrow [T]P)$
$S\,;\ T$	$[S][T]P$

To save space full representation of a method by AMN operation is provided only for the method `searchForExperts` of the type `Secretary`.

$receiveExperts(av, expts) = \ldots$
$dispatch(av, revi) = \ldots$
$numberOfExperts \leftarrow searchForExperts(av) =$
PRE $av \in ext\_Secretary$ THEN
  ANY $v1, numberOfExperts1$ WHERE
    $v1 \in POW(ext\_Expert) \wedge numberOfExperts1 \in NAT \wedge$
    $\exists(exprts).(exprts \in POW(ext\_Expert) \wedge$
      $(exprts = \{exp \mid exp \in ext\_Expert \wedge exp \in availableExperts(av) \wedge$
        $researchArea(exp) = "computer\ sci"\} \wedge$
        $v1 = exprts \wedge numberOfExperts1 = \mathrm{card}\,(exprts)))$
  THEN
    $availableExperts(av) := v1;$
    $numberOfExperts := numberOfExperts1$
  END
END

This example demonstrates some principles of mapping SYNTHESIS specifications of mixed pre and post-conditions of methods (formulae of typed first order logic) into AMN generalized substitutions. The general idea is to extract terms referring to the system post-state from a formula and replace them by auxiliary variables. In case of `searchForExperts` method these terms are `this.availableExperts'` and output parameter `numberOfExperts`. Auxilary variables are $v1$ and $numberOfExperts1$. After that the formula should be transformed into AMN predicate: every term, built-in operation or logical operation of the formula should be transformed into AMN expression, built-in operation or logical operation. Note that every operation has an obligatory input parameter av that denotes an instance of the type for which a method is called (parameter av is a representation of `this` reference).

In the same way the composition type `Dispatcher` $\sqcup$ `Executive` is mapped into AMN construction *DispatcherJOINExecutive*.

REFINEMENT *DispatcherJOINExecutive*
INCLUDES *Evaluation*
SEES *String_TYPE*, *Bool_TYPE*, *Submission*, *Expert*, *Specialist*,
FundingAgency_structureContext
VARIABLES *dispatcher*, *relevantExperts*, *expertsGot*, *experts*
INVARIANT
$dispatcher \in POW(ext\_Dispatcher) \wedge$
$relevantExperts \in ext\_Dispatcher \rightarrow POW(ext\_Specialist) \wedge$
$expertsGot \in ext\_Dispatcher \rightarrow BOOL \wedge$
$experts \in ext\_Dispatcher \rightarrow POW(ext\_Specialist) \wedge \ldots$
OPERATIONS
$getExperts(av) = \ldots$
$numberOfExperts \leftarrow countExperts(av) =$
PRE $av \in ext\_Dispatcher$
THEN
ANY $v1, numberOfExperts1$ WHERE
$v1 \in BOOL \wedge numberOfExperts1 \in NAT \wedge experts(av) = \varnothing \wedge$
$v1 = TRUE \wedge numberOfExperts1 = \mathrm{card}\,(relevantExperts(av))$
THEN
$expertsGot(av) := v1;\ numberOfExperts := numberOfExperts1$
END
END ;
$checkExpert(av) =$
PRE $av \in ext\_Dispatcher$
THEN
ANY $v2, v1$ WHERE
$v2 \in POW(ext\_Specialist) \wedge v1 \in POW(ext\_Specialist) \wedge \neg\,(experts(av) = \varnothing) \wedge$
$\exists(exp).(exp \in ext\_Specialist \wedge exp \in experts(av) \wedge$
$(researchField(exp) = "computer\ sci" \wedge v1 = \{exp\} \cup relevantExperts(av) \vee$
$researchField(exp) \neq "computer\ sci" \wedge v1 = relevantExperts(av)) \wedge$
$v2 = experts(av) \setminus \{exp\})$
THEN
$experts(av) := v2;\ relevantExperts(av) := v1$
END
END ;
$appoint(av, revi) = \ldots$
END .

Mapping of services presented in the SYNTHESIS language into AMN is done automatically by means of mapping facilities developed as a part of CISD tool. These facilities implement principles introduced above.

The last stage of the proof consists in applying the automation facilities of B-Technology to prove that construction *Secretary* is refined by construction *DispatcherJOINExecutive*. Complex proofs are carried out with human expert intervention. We use B-Toolkit 5.4.1. In the example at hand, it automatically

formulated 20 theorems, expressing the fact that construction *Secretary* is refined by construction *DispatcherJOINExecutive*. Large number of theorems is explained by automatically subdividing complex theorems by the tool into simpler ones to prove them independently. In the table 2 total number of theorems formulated and number of theorems automatically proved are shown.

Table 2. The number of theorems

	Number of theorems	Number of automatically proved theorems
Theorems of the initialisation refinement	6	1
Theorems of refinement for operation *getExperts*	3	2
Theorems of refinement for operation *checkExpert*	4	3
Theorems of refinement for operation *searchForExperts*	2	1
Theorems of refinement for operation *dispatch*	5	3
Total number of theorems	20	10

Complete proof of all the refinement theorems make us sure that discovered services and their proper composition implement the requirement specification.

5 Related Work

In order to enhance Web service descriptions, existing approaches extend UDDI or WSDL. For instance, [23] combines OWL-S and UDDI by embedding an OWL-S profile description into a UDDI data structure. The UDDI registry is augmented with an OWL-S matchmaking component. [8] uses OWL-S profile elements with no corresponding UDDI. It defines specialized UDDI T-Models for each unmapped elements in the OWL-S Profile. Mechanisms for augmenting WSDL to provide semantic descriptions and for enhancing UDDI to provide semantic discovery are defined in [20]. Extensions to Web service description are presented as annotated WSDL 1.1 files. The internal organization of UDDI data structures are modified to act as place holders of semantic information [18]. In the SYNTHESIS CISD we combine the canonical model and UDDI similarly to [23].

A number of capability matching algorithms have been proposed for OWL-S. They use the service descriptions in the Service Profiles and the ontologies that are available to decide whether there is a match between service requests and advertisements. A first family of approaches relies on an extensive ontology where OWL ontological classes in request and advertisements are compared. The matching process [13] is reduced to subsumption between the classes in the ontology. Different degrees of matching can be detected.

A second family represents capabilities in terms of the state transformation. The respective matchmakers [14][19][3][22] compare the state transformation described in each advertisement to the one described in the request. They compare

both outputs and inputs of the IOPEs. If the output required by the requester subsumes that of the advertisement, then the inputs are checked. If the inputs requested are subsumed by the input acceptable to the service, then the service is a candidate.

Distinctively, our approach uses a full first-order predicate language, which is more powerful than description logic as used in OWL-S. In our approach, it is possible to interactively prove a refinement relation between type specifications. Type specifications are used as ontological concept definitions as well as abstract specifications of services.

A Service aggregation matchmaking (SAM) [7] can be used to match queries with service registries enriched with OWL-S ontologies. SAM provides more flexible matching with respect to matchmakers of the entire services. It performs a fine-grained matching at the level of atomic processes and sub-services. It can return (when no full match is possible) a list of partial matches.

The service discovery approach proposed in our work is a significant Generalization of the SAM capabilities. According to the compositional calculus used, we discover the most common reduct (fragment) of request and advertisement services and try to construct a composition of such common reducts developed for existing advertisements that should refine the request [4].

6 Conclusion

In this paper we reported the latest results that we obtained extending the CISD method for compositional information systems development to the semantic composition of Web services. The CISD method has been developed for correct composition of software components. It was originally designed [4] for object-oriented platforms (like CORBA, RMI, J2EE). In CISD, an ontological model and a canonical object model (both based on the SYNTHESIS language) are used for the unified representation of the new application (specification of requirement) and of the pre-existing components. Discovering of components relevant to the application and producing their compositions are provided in frame of the domain ontology and the canonical object model. In 2003, we started to investigate the application of the CISD method to the composition of Web services. We studied the mapping of WSDL specifications into the canonical model and we defined the basic steps in the composition of Web services [5].

Meanwhile, the lack of semantic interoperability of Web service infrastructure motivated researchers to develop rich specifications catering for semantically well-founded reasoning about services. Focusing on the realization of complex processes and considering an interactive active approach that allows harnessing the intractability of full first order logic, we now present a novel approach extending and adapting CISD to web service composition. The approach leverages a mapping of the SYNTHESIS language into the Abstract Machine Notation (AMN). AMN is a formal method providing for interactive proof of a refinement relation between type specifications.

The paper shows by example how the CISD method extended with such mapping can be applied for interactive provable discovery of the application relevant

complex Web services to develop their composition refining a specification of requirement. We are convinced that such approach can co-exist with approaches based on OWL-S or similar ideas for applications where automatic service discovery does not constitute the absolute requirement and can be done interactively (semi-automatic) with human expert intervention. We are currently applying the approach that we have presented to the composition of services in the e-science framework of the Virtual Observatory in astronomy [6] project.

References

1. Abrial J.-R. B-Technology. Technical overview. – BP International Ltd., 1992.
2. Abrial. J.-R. The B-Book. – Cambridge University Press, 1996.
3. T. Andrews, et. al. Business Process Execution Language for Web Services // http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ – 2003.
4. Briukhov D.O., Kalinichenko L.A. Component-based information systems development tool supporting the SYNTHESIS design method // Advances in Databases and Information Systems: Proc. of the Second East European Conference. – Berlin-Heidelberg: Springer-Verlag, 1998. – P. 305-327.
5. Briukhov D.O., Kalinichenko L.A., Tyurin I.N. Extension of Compositional Information Systems Development for the Web Services Platform // Advances in Databases and Information Systems: Proc. of the Second East European Conference. – Berlin-Heidelberg: Springer-Verlag, 2003. – P. 16-29.
6. Briukhov D.O., Kalinichenko L.A. et. al. Information Infrastructure of the Russian Virtual Observatory (RVO). – http://synthesis.ipi.ac.ru/synthesis/publications/rvoii/rvoii.pdf – Moscow: IPI RAN, 2005. – 173 p.
7. A. Brogi, S. Corfini, R. Popescu. Composition-oriented Service Discovery // Department of Computer Science, University of Pisa.
8. Colgrave et. al. Using WSDL in a UDDI Registry // UDDI TC Note. – 2003.
9. D.Fensel, C. Bussler. Web Services Modeling Framework // Electronic Commerce: Research and Applications. – http://www.wsmo.org/papers/publications/wsmf.paper.pdf – 2002.
10. Kalinichenko L.A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. – Moscow, 1995.
11. Kalinichenko L.A. Compositional Specification Calculus for Information Systems Development // Advances in Databases and Information Systems: Proc. of the 3rd East European Conference. – Berlin-Heidelberg: Springer-Verlag, 1999. – P. 317-331.
12. Kalinichenko L.A., Skvortsov N.A. Extensible ontological modeling framework for subject mediation // Proc. of the Fourth Russian Scientific Conference "DIGITAL LIBRARIES: Advanced Methods and Technologies, Digital Collections. – Dubna, 2002.
13. L. Li, I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology // Proc. 12th Internationall World Wide Web Conf. – 2003.
14. D. Martin, et. al. Bringing Semantics to Web Services: The OWL-S Approach // J. Cardoso and A. Sheth (Eds.): Proc. SWSWPC 2004, LNCS 3387. – Springer, 2005.
15. Kalinichenko L.A., Skvortsov N.A. Ontology reconciliation in terms of type refinement // Proc. of the Sixth Russian Conference on Digital Libraries RCDL2004. – Pushchino, 2004.

16. OWL Web Ontology Language Reference // http://www.w3.org/TR/owl-ref/
17. OWL-S Coalition. OWL-S 1.0 Release // http://www.daml.org/services/owl-s/1.0/
18. M. Paolucci, T. Kawamura, T. Payne, K. Sycara. Importing the Semantic Web in UDDI // Proc. of Web Services, E-Business and Semantic Web Workshop, CAiSE. – 2002.
19. M. Paolucci et al. Semantic Matching of Web Services Capabilities // The Semantic WebISWC 2002: First International Semantic Web Conf., LNCS 2342. - - Springer-Verlag, 2002.
20. P. Rajasekaran, J. Miller, K. Verma, A. Sheth. Enhancing Web Services Description and Discovery to Facilitate Composition // LSDIS Lab, Computer Science Department, University of Georgia. – Athens, 2004.
21. Simple Object Access Protocol (SOAP) 1.1 // W3C Note 08 May 2000. – http://www.w3.org/TR/SOAP/
22. E. Sirin, B. Parsia, J. Hendler. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques // IEEE Intelligent Systems. – July/August 2004.
23. N. Srinivasan, M. Paolucci, K. Sycara. Adding OWL-S to UDDI, implementation and throughput // Robotics Institute, Carnegie Mellon University. – 2003.
24. S.A. Stupnikov. Automation of refinement verification in information systems compositional design// The Systems and Means of Informatics: Special Issue *Formal Methods and Models for Compositional Infrastructures of Distributed Information Systems.*— Moscow: IPI RAN, 2005. – P. 96-119. (In Russian)
25. SWRL: A Semantic Web Rule Language: Combining OWL and RuleML // W3C Member Submission 21 May 2004. – http://www.w3.org/Submission/SWRL/
26. UDDI Version 3.0 Specification // http://uddi.org/pubs/uddi_v3.htm
27. Web services description language (wsdl) 1.1 // W3C note 15 March 2001. – http://www.w3.org/tr/wsdl/
28. http://www.b-core.com/ONLINEDOC/BToolkit.html