Extension of Compositional Information Systems Development for the Web Services Platform

Dmitry Briukhov, Leonid Kalinichenko, Iliya Tyurin

Institute of Informatics Problems RAS
{brd,leonidk,turin}@synth.ipi.ac.ru

Abstract. The use of Web services on the World Wide Web is expanding rapidly to make applications interoperable in information systems (IS). Web services providing interfaces to information and software components are convenient entities for producing their compositions having Web service appearances. At the same time, most large scale enterprise solutions that are deployed today are composed of a combination of different technologies that go together to compose many diverse applications. An approach for compositional information systems development in a multi-technological framework including Web service components is discussed. This paper proposes to extend the SYNTHESIS method for compositional information systems development (CISD) to the world of Web services. The CISD method is intended for correct composition of existing components semantically interoperable in the context of a specific application. Originally, the CISD method has been developed for the object-oriented platforms (like CORBA, RMI, J2EE). In the CISD, an ontological model and canonical object model (the SYNTHESIS language) are used for the unified representation of the new application (specification of requirements) and of the pre-existing components. Discovery of components relevant to the application and producing their compositions is provided in frame of the domain ontology and the canonical object model. To apply the CISD method for Web services, the mapping of WSDL specifications into the canonical model is required. The basic steps of the approach for the information system compositional development applying Web services are demonstrated.

1 Introduction

As XML becomes a standard for data representation spread among various platforms, it becomes easier to make information and software components to interoperate in information systems (IS) applying Web services. Web Services Description Language (WSDL [19]), Universal Description, Discovery and Integration (UDDI [17]) and Simple Object Access Protocol (SOAP [15]) provide basic facilities for technical interoperability in the world of Web services.

Web services can be treated as components that can be re-used to build information systems. While technical interoperability is supported by the Web service architecture, semantic interoperability is to be provided by specific organization of

compositional development. This paper[1] describes an approach to use the SYNTHESIS method for compositional information systems development (CISD) [2] over Web services. The CISD method is intended for correct composition of existing components semantically interoperable in the context of a specific application (defined by a specification of requirements). In contrast to technical interoperability (provided by middleware technologies, such as CORBA, or Java RMI) in CISD the interoperability is considered in a broader, semantic aspect.

Originally, the CISD method has been developed for the object-oriented middlewares (like CORBA, J2EE), but it can be extended also for other platforms. In this paper we present an extension of the compositional IS development method for the Web Services platform. To apply the CISD method for Web services we need to extend specifications of Web services with ontologies. Also, the mapping of WSDL specifications into the canonical model of the CISD (SYNTHESIS language) should be provided.

The paper is organized as follows. After brief characterization of the related work, a short introduction into the CISD method is given. The intention of this introduction is to make further examples in the text better readable. Section 4 presents the main principles of mapping the WSDL specification into the SYNTHESIS specification. In section 5 an extension of the CISD method for the Web services platform is demonstrated. The conclusion summarizes the results obtained.

2 Related Work

Web Services is an additional middleware solution that has emerged from efforts to perform distributed computing over the public Internet, particularly exchanges between enterprises. In spite of many of its attractive capabilities, Web Services is unsuitable for many important operations, including rapid multi-use services (e.g., database transactions), or as the internal mechanism linking parts of component-based applications. Therefore solutions are needed to enable applications in diverse technologies to interoperate using web services technology. A Web Services architecture should seamlessly integrate with other infrastructure facilities on an "integration broker" platform [21,20]. Components from different platforms are expected to be composable together with and into the web services.

The CISD [2] is a compositional multi-platform method that is in a good correspondence with the idea of the integration broker [21]. This paper shows how to extend this method to the Web Services. Web Services requires a conceptual modeling and architectural framework on top of existing Web Service standards and architectures. Current Web services standards enable publishing service descriptions and finding services on a match based criteria such as method signatures or service category. More promising directions include DAML-S [6] providing an ontology of services, the service profile for advertising and discovering services; the process model, which gives a detailed description of a service's operation.

[1] The work was supported by the Russian Foundation for Basic Research (grants 01-07-90376 and 03-01-00821).

Brief comparison of CISD with the known DAML-S based approaches follows. The METEOR-S work [18] states that current Web service discovery mechanism is inefficient, as it does not support discovery based on the capability of the services and thus leading to a lot of irrelevant matches. This project [18] adds an ontology-based approach to organize registries, enabling semantic classification of all Web services based on domains. An approach for semantic publication and discovery using WSDL descriptions map inputs and outputs of Web services to ontological concepts. The operations themselves should be mapped to concepts, all inputs and outputs in the WSDL description should not only be mapped to concepts in the domain specific ontology but also grouped according to operations. Similar mapping of operations and their parameters to ontologies has been originally provided by CISD [2].

In [14] the semantic web initiative at W3C (such as DAML-S) and tools that may help bridge the gap between the current standard solutions and the requirements for advanced web service discovery are analyzed. The analysis concludes that DAML-S is still in its infancy and a lot of work has to be done in order to overcome its limitations and problems. The paper [12] reports results of DAML-S usage to describe capabilities of Web services so that they can find each other on the basis of the information that they provide. An approach for integrated Semantic Web technology for automating customized, dynamic discovery of Web services together with interoperation through semantic translation is presented in [11]. Alongside with promising results, the paper warns that achieving automated Web service composition requires a fundamental shift in industrial frameworks from executing predefined process models to computing and adapting execution plans from abstract objectives.

Main differences of CISD [2] and DAML-S oriented approaches [14,12,11] consist in the following. CISD is based on a refinement technique [1] and on an extensible typed model for definition of services and workflows. Basic assumption of the method is that specifications of user needs and specifications of existing services may differ significantly. Ontological conformance is only one step in discovery of services potentially relevant to the user requirements. Further steps include an attempt to reconcile the value, structural and typing discrepancies between specifications applying transformation functions [2,3]. Adapting of behaviors is possible to some extent due to the features of the refinement technique that tolerates significant differences between abstract specification of requirements and pre-existing concrete implementation of a service. Reasoning reported in [11] applies another technique - proof in a description logic.

Another difference is that the CISD method [2], instead of using a specialized model (like DAML-S), applies quite general modeling framework suitable for equivalent mapping into it of various existing modeling facilities. Various ontological and process specification facilities have been experienced [10,16]. An approach for a specific model (DAML-S) mapping into the general modeling framework of CISD has been presented in [10].

3 Fundamentals of the CISD Method

The main distinguishing feature of the CISD method is a creation of compositions of component specification fragments refining specifications of requirements. Refining specifications obtained during the compositional development, according to the refinement theory, can be used anywhere instead of the refined specifications of requirements without noticing such substitutions by the users.

The 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 (patterns of reuse), composition of identified fragments into specifications refining the requirements, justification of substitutability of the results of such transformations instead of the specifications of requirements. The compositional specification calculus [7], intentionally designed for such manipulations uses the following concepts and operations.

A *type reduct* R_T is a subspecification of an abstract data type T specification. The specification of R_T should be formed so that R_T becomes a supertype of T.

For identification of a fragment of existing component type that may be reused in implementation of another type, a common reduct for these types should be constructed. *Common reduct* for types *T1* and *T2* is such reduct R_{T1} of *T1* that there exists a reduct R_{T2} of T2 such that R_{T2} is a *refinement* of R_{T1}. R_{T2} is called a *conjugate* of the common reduct. Establishing a refinement ensures that R_{T2} can be correctly substituted everywhere instead of R_{T1}.

For creation of composition of identified reusable fragments into specification refining the specification of requirements the type calculus operations (such as meet, join and product) are used. The *meet* operation T_1 & T_2 produces a type T as an 'intersection' of specifications of the operand types. The *join* operation T_1 | T_2 produces a type T as a 'join' of specifications of the operand types [7].

Briefly, the CISD method consists in the following. To make analysis of an information system being developed, an existing object analysis and design methods [13] extended to achieve the completeness of specifications is applied. In particular, such method is extended with a means for ontological definitions of the application domains, for definition of type invariants and predicative specifications of operations. Then, at the design phase, using a repository of component specifications, a search of the components relevant to the specifications of requirements is undertaken. This search is based on the integrated ontological context of requirements and components. Then, identification of reusable fragments of components, including reconciliation of various conflicts between specifications, and composition of such fragments into specifications concretizing the requirements should be made. And finally, a property of refinement of requirements by such compositions should be justified.

4 Representation of WSDL specifications in the SYNTHESIS language

Mapping the Web Services specifications into the SYNTHESIS language [8] is required to extend the CISD method to Web services.

In the CISD, the SYNTHESIS language intends to provide for uniform (canonical) representation of heterogeneous data, programs and processes for their use as interoperable entities. Strongly typed, object-oriented subset of the language (that is 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.

In the SYNTHESIS language the type specifications are syntactically represented by frames, their attributes - by slots of the frames. Additional information related to attributes can be included into metaslots. Syntactically frames are embraced by figure brackets { and }, slots are represented as pairs *<slot name> : <slot value>* (a frame can be used as a slot value), slots in a frame are separated by semi-colons.

4.1 Port Type

A port type describes a set of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a set of input and output messages, a port type is a set of operations.

A port type can optionally extend one or more other port types. In such cases the port type contains the operations of the port types it extends, along with any operations it defines.

The specification of port type is the following:

```
<wsdl:portType name="port_name">
    <wsdl:documentation .... /> ?
    <wsdl:operation name="operation_name"> *
    ...
    </wsdl:operation>
</wsdl:portType>
```

The port type is mapped into the SYNTHESYS type:

```
{port_name;
  in: type;
  <list_of_operations>;
}
```

4.2 Port Type Operation

A port type operation describes an operation that a given port type supports. An operation is a set of message references. Message references may be to messages this operation accepts, that is input messages, or messages this operation sends, that is output or fault messages.

```
<wsdl:operation name="operation_name">
    <wsdl:documentation .... /> ?
    <wsdl:input message="input_message_name"> ?
        <wsdl:documentation .... /> ?
    </wsdl:input>
    <wsdl:output message="output_message_name"> ?
        <wsdl:documentation .... /> ?
    </wsdl:output>
    <wsdl:fault name="ncname"
                message="fault_message_name"> *
        <wsdl:documentation .... /> ?
    </wsdl:fault>
</wsdl:operation>
```

The port type operation is mapped into the SYNTHESIS function:

```
operation_name: {
  in: function;
  params: {<list_of_parameters>};
  [{{<predicative_specification>}}]
};
```

4.3 Message

A message describes the abstract format of a particular message that a Web service sends or receives. The format of a message is typically described in terms of XML element and attribute information items.

A part constituent describes a portion of a particular message that a web service sends or receives. The format of a part is described by reference to type definition or element declaration constituents.

```
<wsdl:message name="message_name">
    <wsdl:documentation .... /> ?
    <part name="part_name" element="element_name"?
          type="type_name"?/> *
</wsdl:message>
```

The message is mapped into a list of SYNTHESIS function parameters. Each part constituent is mapped into separate parameter:

```
operation_name: {in: function;
  params: {<kind>part_name/type_name, ...};
};
```

or

```
operation_name: {in: function;
  params: {<kind>part_name/element_name, ...};
};
```

The <kind> of parameter ("+" - input, "-" - output, "" - input-output) corresponds to message exchange pattern in WSDL.

4.4 Service

A service describes the set of port types that a service provides and the ports they are provided over.

```
<wsdl:serviceType name="service_name">
    <wsdl:portType name="port_name"/> +
</wsdl:serviceType>
```

The service is mapped into SYNTHESIS module which contains types corresponding to WSDL port types:

```
{service_name;
  in: module;
  type: {
    {port_name;
      in: type;
      ...
    }
  }
}
```

4.5 XML Schema Types

At the abstract level, the Types Component is a container for imported and embedded schema components. By design, WSDL supports any schema language for which the syntax and semantics of import or embed have been defined. Support for the W3C XML Schema Description Language is required of all processors. Principles of mapping the XML Schema Datatypes into the SYNTHESIS types are described in [4].

4.6 Example

To demonstrate the mapping we consider the *HotelReservationService.* Its specification in WSDL looks as follows:

```
<wsdl:types>
  <xsd:simpleType name="SetOfHotels">
    <list itemType="HotelAndPrice" />
```

```
  </xsd:simpleType>
  <xsd:complexType name="HotelAndPrice">
    <xsd:element name="hotel" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="roomType" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
  </xsd:complexType>
</wsdl:types>

<message name="InputAsk">
  <part name="city" type="xsd:string"/>
  <part name="roomType" type="xsd:string"/>
  <part name="fromDate" type="xsd:date"/>
  <part name="toDate" type="xsd:date"/>
</message>
<message name="OutputAsk">
  <part name="return" type="SetOfHotels"/>
</message>
<message name="InputComments">
  <part name="comment" type="xsd:string"/>
</message>

<portType name="HotelReservation">
  <operation name="askForRoom">
    <input message="tns:InputAsk"/>
    <output message="tns:OutputAsk"/>
  </operation>
  <operation name="sendComments">
    <input message="tns:InputComments"/>
  </operation>
</portType>

<serviceType name="HotelReservationService">
  <portType name="HotelReservation" />
</serviceType>
```

The specification of this Web service in SYNTHESIS is:

```
{HotelReservationService;
  in: module;
  type: {
    {HotelReservation;
      in: type;
      askForRoom: {in:function;
        params: {+city/string, +roomType/string,
          +fromDate/time, +toDate/time,
          -return/SetOfHotels;};
      };
      sendComments: {in:function;
```

```
        params: {+comment/string};
      };
    };
    {HotelAndPrice;
      in: type;
      hotel: string;
      city: string;
      roomType: string;
      price: integer;
    };
    {SetOfHotels: {set; type_of_element:
                          HotelAndPrice;};
    };
  }
}
```

5 Compositional Development over the Web Services Platform

The approach for compositional IS development over the Web services platform consists in an extension of the CISD [2]. Compositional design according to such extended approach includes the following steps:

- mapping the Web services specifications into the SYNTHESIS language;
- searching for relevant Web services;
- identifying fragments of existing Web services, that may be reused in IS being developed;
- resolve conflicts between specifications;
- composition of such fragments into specification refining the specification of IS;
- implementation of IS.

To illustrate this approach the following example will be used. Consider the development of IS type *TravelService* over two existing components (implemented as Web services): *AirTicketReservationService* and *HotelReservationService.*

TravelService as a specification of requirements contains operations: *getHotelInfo*, *getAirTicketInfo* and *sendComments*. The specification of *TravelService* type in SYNTHESIS looks as follows:

```
{TravelService;
  in: type;
  getHotelInfo: {in:function;
    params: {+info/Travel,
             -returns/{set; type_of_element: Hotel;}
    };
  };
  getAirTicketInfo: {in:function;
    params: {+info/Travel,
             -returns/{set; type_of_element: Ticket;}
    };
```

```
    sendComments: {in:function;
      params: {+comment/string}
    };
  };
  {Travel;
    in: type;
    fromCity: string;
    toCity: string;
    fromDate: time;
    toDate: time;
    roomType: string;
    flightClass: string;
    person: Person;
  };
```

AirTicketReservationService contains the *AirTicketReservation* port type, which contains *getAvailableTickets* and *sendComments* operations. *HotelReservationService* contains the *HotelReservation* port type, which contains *askForRoom* and *sendComments* operations. The mapping of WSDL specification of *HotelReservationService* Web service into the SYNTHESIS language is given in section 4.6.

5.1 Searching for Relevant Web Services

Specifications of requirements and pre-existing components must be associated with ontological contexts defining concepts of the respective subject areas. Ontological concepts are described with their verbal definitions similar to definitions of words in an explanatory 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 approach [2].

For each constituent (service type, port type, message) defined in WSDL, the corresponding ontological concept is defined. Its verbal definition is taken from the *documentation* element that should be defined inside each WSDL constituent. Due to the correlation established, ontologically relevant service components can be found.

5.2 Identifying Relevant Fragments

This step consists in identifying among probably relevant types (corresponding to existing Web services) those that may be used for the concretization of the types of specification of requirements.

For identification of a fragment of existing component type that may be reused in implementation of another type, the common reduct for these types should be constructed. The specification of common reduct for types *TravelService* and *HotelReservation* is the following:

```
{R_TS_HR;
  in: reduct;
    metaslot
      of: TravelService;
      taking: {getHotelInfo, sendComments};
      c_reduct: CR_TS_HR;
    end;
};
```

A slot *of* refers to the reduced type *TravelService*. A list of attributes of the reduced type in the slot *taking* contains names of its attributes and functions that are to be included into the reduct. In our case the reduct includes functions *getHotelInfo* and *sendComments*.

Its conjugate is specified as a *concretizing reduct* that incorporates the correspondence between attributes and functions of a reduct being refined and its refinement, as well as the required functions reconciling conflicts between specifications of these reducts:

```
{CR_TS_HR;
  in: c_reduct;
    metaslot
      of: HotelReservation;
      taking: {askForRoom};
      reduct: CR_TS_HR;
    end;
  simulating: {
    R_TS_HR.getHotelInfo(info) ~
        CR_TS_HR.askForRoom(info.toCity, info.roomType,
                            info.fromDate,info.toDate),
    R_TS_HR.sendComments(comment) ~
        CR_TS_HR.sendComments(comment)
  };
};
```

Slot *reduct* refers to the respective common reduct. A slot *of* refers to the reduced type *HotelReservation*. A list of attributes of the reduced type in the slot *taking* contains names of its attributes and functions that are to be included into the concretizing reduct. In our case it includes functions *askForRoom* and *sendComments*. Predicate *simulating* shows how the common reduct state is interpreted by an instance of the collection. E.g., it defines that function *getHotelInfo* of reduct *R_TS_HR* is refined by function *askForRoom* of concretizing reduct *CR_TS_HR* and shows the correspondence of their parameters. The concretizing reduct may also contain functions resolving various conflicts between specifications of the considered types.

In the same way the common reduct *R_TS_ATR* and concretizing reduct *CR_TS_ATR* for types *TravelService* and *AirTicketReservation* are specified. Reduct *R_TS_ATR* contains operations: *getAirTicketInfo* and *sendComments*.

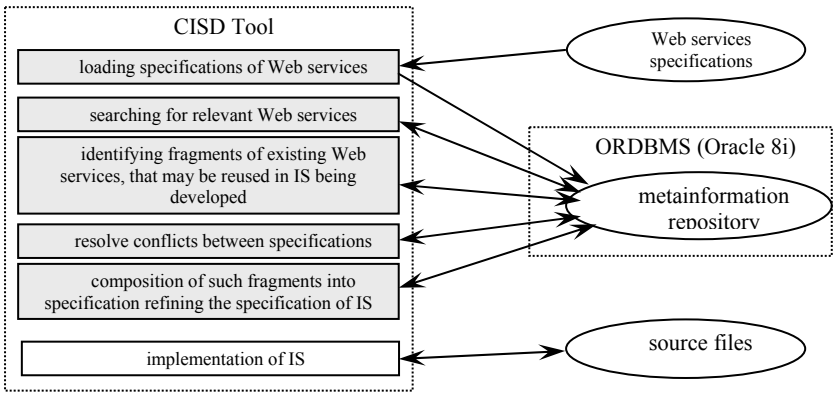

Fig. 1. CISD Tool Structure

5.3 Composition of Relevant Fragments

For creation of composition of identified fragments into specification refining the specification of requirements the type calculus operations (such as meet, join and product) [7] are used.

In our example for implementing the *TravelService* type the join of reducts of *AirTicketReservation* and *HotelReservation* types is used. The specification of concretizing type *CT_TS_HR_ATR* is constructed as join of reducts *R_TS_HR* and *R_TS_ATR*:

```
CT_TS_HR_ATR = R_TS_HR[getHotelInfo, sendComments] |
               R_TS_ATR[getAirTicketInfo, sendComments]
```

Resulting type *CT_TS_HR_ATR* contains functions: *getHotelInfo*, *getAirTicketInfo* and *sendComments*. The concretizing type refines the *TravelService* type.

5.4 Implementation of the Developing IS

During implementation the results obtained at previous steps are transformed into the form of source files in the programming language. Implementation of the IS being developed includes the code generation for constructed reducts and concretizing types. Each reduct and concretizing type is implemented as a separate class. Code generation is performed using special macro language. Programs written in the macro language may be customized to meet particular needs of specific implementation platforms.

5.5 CISD Tool

Figure 1 shows a general structure of the tool supporting compositional IS development. The tool is being developed reusing the SYNTHESIS compositional IS

design method prototype [2]. The tool is based on Oracle 8i and Java 2 under Windows environment.

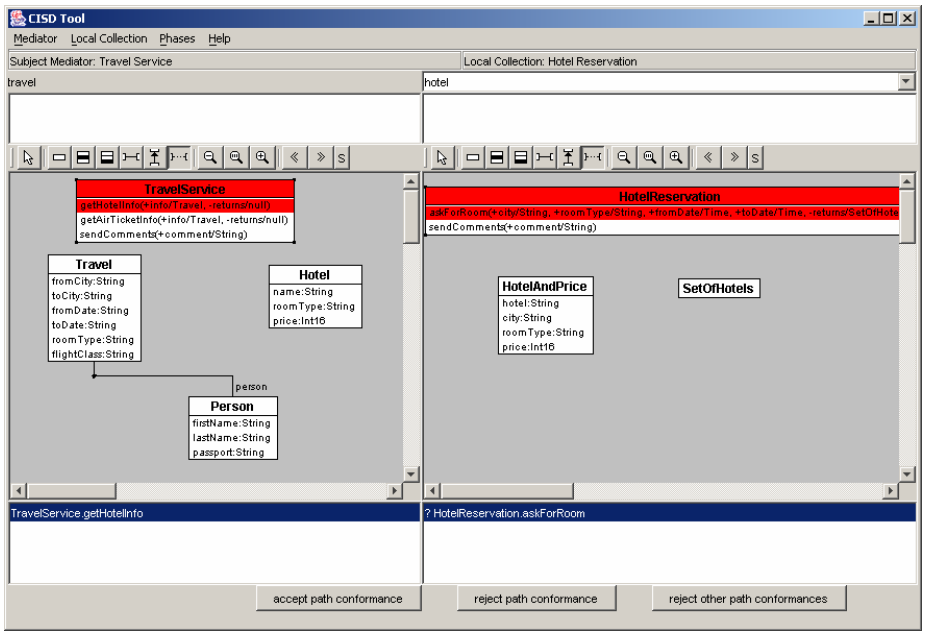

Fig. 2. An Example of CISD Tool GUI

In figure 2 an example of CISD Tool GUI is presented. It shows the GUI for confirmation of relevance between the specification of requirements element (function *getHotelInfo*) and the component element (function *askForRoom*).

6 Conclusion

The paper presents an approach for information systems development as a process of composition of existing Web services. It is based on the compositional information systems development method described in [2]. To extend this method to Web services, the mapping of WSDL specifications into SYNTHESIS specifications was introduced. The basic steps of constructing Web services composition refining a specification of requirements were demonstrated. The approach proposed makes possible composition of components developed in frame of different technologies (Web Services, CORBA, J2EE) that work together to compose many diverse applications. Specific integration architecture [20,21] is assumed for the implementation.

In the future we plan to extend presented approach with process specifications of Web services (in spirit of process reuse [9] represented similarly to WS-BPEL [5] or DAML-S [6], but based on the refinement and bisimulation technique).

References

[1] Abrial J.-R. The B Book: assigning programs to meaning. Cambridge University Press, 1996

[2] Briukhov D.O., Kalinichenko L.A. Component-Based Information Systems Development Tool Supporting the SYNTHESIS Design Method. In *Proc. of the East European Symposium on "Advances in Databases and Information Systems"*, Poland, Springer, LNCS No.1475, 1998

[3] Briukhov D.O., Kalinichenko L.A., Skvortsov N.A., Stupnikov S.A. Value Reconciliation. Mediators of Heterogeneous Information Collections Applying Well-Structured Context Specifications In Proceedings of the Fifth International Baltic Conference on Databases and Information Systems BalticDB&IS'2002, Tallinn, Estonia, June 3-6, 2002

[4] Briukhov D.O., Tyurin I.N. Mapping XML Schema Data Types into SYNTHESIS Types (in Russian). In *Proc. of the Russian Conference "Digital Libraries: Advanced Methods And Technologies, Digital Collections" (RCDL'2002)*, Dubna, 2002

[5] Business Process Execution Language for Web Services, Version 1.0. http://www-106.ibm.com/developerworks/library/ws-bpel/

[6] *DAML-S 0.7 Draft Release*. http://www.daml.org/services/daml-s/0.7/

[7] Kalinichenko L. A. Compositional Specification Calculus for Information Systems Development. In *Proc. of the East-West Symposium on Advances in Databases and Information Systems (ADBIS'99)*, Maribor, Slovenia, September 1999, Springer Verlag, LNCS, 1999

[8] Kalinichenko L. A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of informatics, Russian Academy of Sciences, Moscow, 1995

[9] Kalinichenko L.A. *Workflow Reuse and Semantic Interoperation Issues. Advances in workflow management systems and interoperability.* A.Dogac, L.Kalinichenko, M.T. Ozsu, A.Sheth (Eds.),. NATO Advanced Study Institute, Istanbul, August 1997

[10] L.A. Kalinichenko, N.A. Skvortsov Extensible Ontological Modeling Framework for Subject Mediation . Proceedings of the Fourth All-Russian Conference on Digital Libraries, RCDL'2002, Dubna, October 15 – 17, 2002

[11] Daniel J. Mandell, Sheila A. McIlraith. A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[12] Massimo Paolucci, Katia Sycara, Takuya Nishimura, Naveen Srinivasan. Toward Semantic Web Services. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[13] *Paradigm Plus Reference Manual*. Protosoft, 1997

[14] Thomi Pilioura, Aphrodite Tsalgatidou , Alexandros Batsakis. Using WSDL/UDDI and DAML-S in Web Service Discovery. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003

[15] *Simple Object Access Protocol (SOAP) 1.1.* W3C Note 08 May 2000. http://www.w3.org/TR/SOAP/

[16] Stupnikov S.A., Kalinichenko L.A., Jin Song DONG Applying CSP-like Workflow Process Specifications for their Refinement in AMN by Pre-existing Workflows In

Proceedings of the Sixth East-European Conference on Advances in Databases and Information Systems ADBIS'2002, September 8-11, 2002, Bratislava, Slovakia
[17] *UDDI Version 3.0 Specification.* http://uddi.org/pubs/uddi_v3.htm
[18] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. and Miller, J. METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (to be published)
[19] *Web Services Description Language (WSDL) Version 1.2*. http://www.w3.org/TR/wsdl12
[20] *Web Services Software Architecture (WSSA)*, RFI, OMG Document # bei/2003-01-04, January 6, 2003
[21] *White Paper on Web Services Integration Architecture*, October 28, 2002, OMG Document Numbers:bei/2002-10-02