Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1071–1077

Concept of Platform for Hybrid Composition, Grounding and Execution of Web Services

Lev Belava
AGH University of Science and Technology, Mickiewicza 30, Cracow, Poland
Email: Lev.Belava@gmail.com

***Abstract*—This paper presents a concept of a software platform and a method of hybrid composition of web services and hybrid grounding of abstract composition plans. The paper also describes the architecture of the implemented platform and its modules.**

I. Introduction

RECENT scientific and industrial progress in the field of information technology clearly shows a tendency to switch data processing and computation from traditional models to network architectures. There is also another trend of switching over from large, complex monolithic software systems to groups of smaller, well-defined and interoperable applications. Due to this reason Service Oriented Architecture paradigm and its specific realization – Web Services – naturally fit these general trends. The practice of SOA services composition is a promising approach to developing new software systems with highly refined functionality that is achieved by using a combination of different services chosen from those available in a computer network. Very important parts of such software systems include service composition and grounding. Service composition focuses on creating complex plans from available services, while grounding is a process that transforms abstract composition plans into execution plans so that every abstract service used in an abstract composition plan is effectively associated with a real-world service instance and thus can be called during the execution process.

Hybrid service composition is a method that allows its users to combine different service composition techniques. It offers more flexibility and control of the composition process itself due to the ability to choose different composition techniques for different parts of the composed plan. Hybrid grounding is also a method that allows similar flexibility and control of a grounding process. It allows to mix and match different grounding techniques for different parts of an abstract composition plan.

A. Service Composition and Grounding in SOA

SOA is a software engineering paradigm which generally describes various aspects of software systems that utilize specific entities called services. On the other hand, SOA does not define services strictly – they just have to be relatively independent from each other and offer various functionalities. Such an engineering approach offers different advantages like easier integration of legacy software into new business processes, safer and more reliable upgrades of software components, etc. Web Services are possibly the most common SOA implementation nowadays.

Service composition is a concept of combining different services for data processing purposes. It enables engineers to create complex processes by combining various functionalities offered by available services. So far numerous service composition techniques have been developed.

Manual and semi-automatic service composition methods e.g. [1] and [2] are relatively popular in the scientific community. Such kinds of approaches are fairly easy to understand and implement. When creating service compositions all decisions are made by the user who is provided with some kind of advice or narrowing choice options at most. However, such methods do not offer service composition process automation.

Different automatic service composition approaches have been proposed. Variations of forward and backward chaining methods as in [3], [4], etc. were presented. Hierarchical Task Networks methods were proposed in such works as [5] and [6]. Ontological descriptions of services can be used by reasoners for composition creation [7]. Petri nets were used for service modeling and composition in [8].

Composition methods can, but do not have to, assume that service instances are available and reachable somewhere in a network. So, if a method is not concerned with the availability of service instances, it will produce abstract composition plans. On the other hand, grounding is a process of enriching composition plans with vital information that allows necessary service instances to be used during the execution process. Therefore, abstract composition plans have to be grounded prior to being ready for execution. Several service composition grounding methods have been proposed, some of them are based on brokers [9] while some others are matching-based [10], heuristic [11], agent-based [12] or even ontology-based [13] and [14]. Each one of these approaches uses different perspectives on the grounding process thus allowing their users to fit their needs in a very varied and not always interoperable ways.

B. Problem Statement

There are numerous methods for creating and grounding service compositions. However, every particular approach cannot be an ideal solution from all points of view. Imagine a situation when a user of an SOA software system wants to use some predefined service composition parts and combine

978-1-4673-4471-5/$25.00

them with an output of an automatic composition method. The concept of hybrid service composition was specifically proposed in order to solve such kind of problems [15]. This concept allows to use multiple service composition methods during the creation of a service composition plan.

A similar problem is observed in the grounding of abstract composition plans because grounding methods may vary a lot in terms of their work principles as well as optimization targets (QoS, cost, etc.). The concept of hybrid grounding tries to address this problem by utilizing different grounding methods during the grounding of one particular abstract service composition plan.

So far, several service composition platforms have been presented in scientific literature. The most important ones include SWORD [16], METEOR-S [17], MAESTRO [18], SPICE [19]. However, none of them tries to solve the problem of more flexible composition or picking and using a grounding method. SWORD uses first order logic, METEOR-S adopts the Constraint Satisfaction Problem engine for producing a composition, MAESTRO is based on a particular graph method with backward chaining and SPICE uses backward chaining with branching for optimization purposes.

A variety of concepts and methods for service composition and grounding methods, platforms and approaches has been proposed. However, none of them is perfect from every point of view, e.g. such perspectives as composition plan languages or optimization targets for grounding. In order to solve this issue a concept of a hybrid composition, grounding and execution platform was developed. It adopts approaches that enable users to have more flexibility during composition and grounding processes by allowing to use different composition and grounding methods together.

II. Proposed Platform Concept

The architecture of the hybrid composition, grounding and execution platform consists of five key modules that are cooperating together. The concept also incorporates external elements – web services. These services are used by various modules to produce and execute composition plans. Fig. 1 shows the architecture of platform and data flows between different modules.

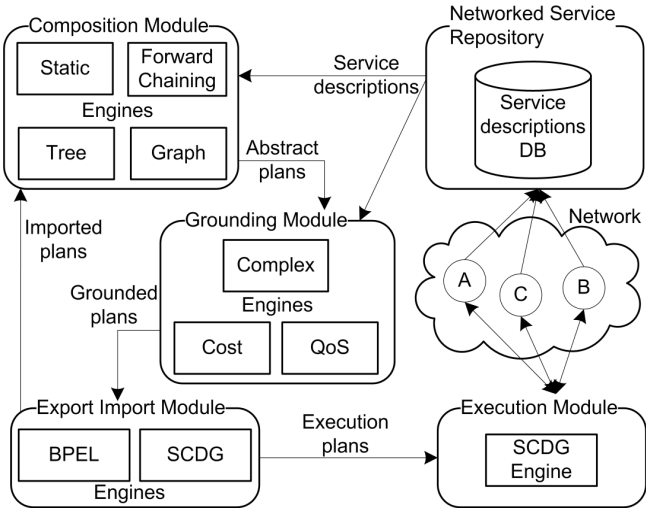

Fig 1. Architecture of Hybrid Composition, Grounding and Execution Platform

The Composition Module is a platform component that actually performs all hybrid service composition tasks and produces abstract composition plans. This module interacts closely with the Networked Service Repository from which it gets web service descriptions. Moreover, it might use the Export Import Module from which it obtains plans for the static composition engine. Abstract composition plans that are produced by this module are forwarded to the Grounding Module for further processing. The module consists of four different service composition engines which can work together to produce composition plans.

The static service composition engine provides necessary functionality to combine different pieces of the composition plan that can be imported or generated by other composition engines. The static engine uses two main operations to work on composition plans: DELETE and INSERT. The "Delete" operation cuts out a specified part of a plan and "Insert" pastes one plan into another. The INSERT (1, 2, plan1, plan2, 4) operation scheme is presented on Fig. 2. Plan1 and plan2 represent two input plans for the operation. Plan3 is the result of inserting plan2 from the first non-root node to "Service 4" node into plan1 between "Service1" and "Ser-

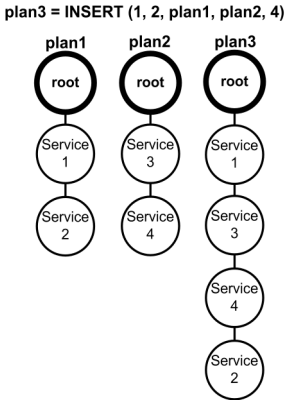

Fig 2. INSERT operation scheme

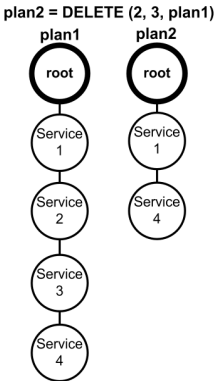

Fig 3. DELETE operation scheme

vice 2" nodes. The DELETE (2, 3, plan1) operation scheme is presented on Fig. 3. Plan2 is the result of cutting a chain of services from plan1 starting at "Service 2" and finishing at "Service 3".

To proceed further we need to provide a definition for a service input and output type. Input or output service types in the proposed approach consist of two parts: the first – a formal description of the data format that the service accepts as input or returns as output, the second – semantic information that describes the meaning of that data.

A forward chaining service composition engine creates service composition plans by using a simple chaining algorithm similar to the one proposed in [4]. Its simplified scheme of action is to successively add new elements to the end of the plan if their input types are consistent with the previous element's output type. The general idea is to create such a chain of elements that its last element will have the desired output type.

A tree-based service composition engine creates service compositions by using a method that creates not just a chain of elements, but a tree. This method is relatively similar to forward chaining but it allows to search the produced trees and because of that the results of its work are more optimal than the results of simple chaining techniques.

A graph-based service composition engine uses a composition method that is similar to the one proposed in [20]. Basically, at the beginning the composition algorithm produces a complete services dependency graph. This directed graph is created by treating abstract services as nodes in a graph and then connecting the nodes with directed arcs if one service's output type is identical to other service's input type. Then such a graph could be processed by Dijkstra or some other pathfinding algorithms. Fig. 4 presents a sample complete services dependency graph. Each node in that graph is described by its input type ("IN") and output type ("OUT").

The Grounding Module allows abstract composition plans, that were produced by the composition module, to be grounded. It cooperates closely with the services repository from which it gets full information profiles about service instances that are available on the network. Such a profile consists not only of the service address and input/output types but also includes additional parameters such as QoS and cost. The three grounding engines in the Grounding Module include QoS, cost and complex. QoS and cost grounding methods were chosen as sample approaches that can be successfully combined in a complex engine. There is a possibility to use and combine other grounding methods as well.

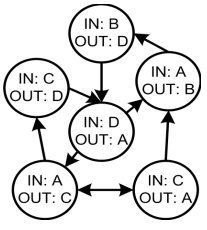

Fig 4. Example of a complete service dependency graph

The goal of the QoS optimization engine is optimizing QoS parameters of composition plans or their parts. For example, one can request that QoS parameters for some part of the abstract composition plan have to reside between some desired maximum and minimum values. In such case the QoS engine will look for service instances that fit the provided values best.

The cost optimization engine works similarly to the QoS engine, but it has a task to optimize the cost of composition plans or their respective parts.

The complex optimization engine allows to create a hierarchical structure of grounding preferences which let the user apply additional optimizations in cases where the engine on a higher level of hierarchy will find several equally fitted service instances. For example, we can imagine a situation in which the cost parameter is the most important target of the composition optimization, but we would like to choose a service with the best QoS in case there are several service candidates with the same cost value.

The Export Import Module provides functionality that allows the abstract and grounded composition plan to be imported or exported from or to files. There are two export-import engines that were implemented for the proposed platform – BPEL and SCDG.

The BPEL engine is able to import [22] and export [21] composition plans that are written in a BPEL language. Not all the BPEL functionality is currently implemented, but core elements like conditionals, loops and the parallel execution of services are fully supported.

The SCDG engine allows to work with composition plans that are presented as Service Composition Directed Graphs. The SCDG is a graph-based model of service composition representation that was proposed in [22].

An Execution Module executes grounded service composition plans. To-date only the SCDG execution engine has been implemented, although there is a possibility to include other engines. To do that one might also need to develop an appropriate import-export engine first.

A Networked Service Repository Module is a web service that on the one hand allows web services to be registered in it and on the other hand provides information about these services for composition and grounding modules. This module also employs a standalone database for service descriptions to be stored in it. A database engine could be either external or internal in relation to the Networked Service Repository. External database engines, however, are much faster and reliable with large data sets and thus more preferable.

III. Use Case Scenario

We can imagine an on-line trading system which allows its users to search for products, place orders and ultimately buy goods by entering financial and personal data into the system. There are all sorts of government regulations and industry standards for personal and financial data because of its sensitive nature. Therefore, we can be sure that some parts of the composition plans in such kind of software platforms will be predefined specifically to obey all sorts of reg-

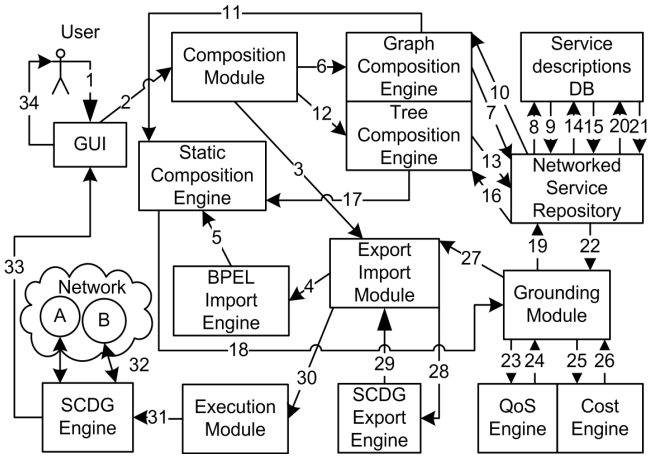

Fig 5. Service composition, grounding and execution diagram

ulations and standards. On the other hand, such systems may benefit from automatic or semi-automatic service composition techniques after all.

Hybrid service composition was proposed to solve exactly such kinds of problems by providing the necessary interoperability between different service composition methods.

A. System's Internal Operation - From Composition to Execution

Fig. 5 presents a diagram with an example of how a service composition plan is made, grounded and executed in a system which implements the platform concept proposed in this article.

1. The user provides necessary personal and financial data and the parameters of the desired products.

2. This data is delivered to a Composition Module.

3. The Composition Module sends a request to an Export Import Module to make an import of a standard-required part of the composition plan which will handle personal and financial data.

4. The Export Import Module transfers the request to a BPEL Import Engine which will actually perform the task of importing.

5. The BPEL Import Engine sends a part of the imported composition plan to a Static Composition Engine, so it can be merged with automatically composed parts later.

6. The Composition Module initiates a Graph Composition Engine and transfers composition parameters to it.

7. The Graph Composition Engine makes a request to a Networked Service Repository and asks for a list of available services types.

8. The Networked Service Repository makes an appropriate query in a Service Descriptions Database.

9. The Service Descriptions Database processes the query and sends back the results.

10. The Networked Service Repository provides the Graph Composition Engine with a list of all available services types (not instances).

11. The Graph Composition Engine sends the prepared part of the future service composition plan to the Static Composition Engine.

Steps 12..17 are similar to steps 6..11.

18. The Static Composition Engine merges all parts of the composition plan into one abstract service composition plan and delivers it to a Grounding Module for grounding.

19. The Grounding Module makes a request to the Networked Service Repository and asks it to provide a list of real-world service instances whose inputs and outputs correspond to the inputs and outputs of the services in the abstract composition plan.

Steps 20 and 21 are similar to steps 8 and 9.

22. The Networked Service Repository provides the Grounding Module with a list of required real-world service instances.

23. The Grounding Module initiates a QoS Engine and delivers the appropriate part of the plan plus the lists of service instances to it.

24. The QoS Engine grounds a part of the greater plan and sends it back to the Grounding Module.

Steps 25 and 26 are similar to steps 23 and 24.

27. The grounded composition plan is delivered to the Export Import Module.

28 The Export Import Module initiates a SCDG Export Engine and provides it with a grounded composition plan.

29. An exported composition plan is delivered back to the Export Import Module.

30. The Export Import Module sends the exported composition plan to the Execution Module for plan execution to be made.

31 The Execution Module initiates a SCDG Execution Engine and provides it with a composition plan.

32. The SCDG Execution Engine executes the composition plan.

33. Plan execution results are delivered to the interface.

34. The interface renders the acquired results and presents them to the user.

B. A Closer Look at Composition and Grounding

Fig. 6 presents a visualization of an abstract composition plan which deals with sensitive personal and financial data provided by the user. There are several service calls in it: “AssignUniqueID” – assigns a unique ID number to a user-provided data set, “Encrypt” – encrypts the data set, “Archive” – archives the previously encrypted data, “ValidateData” – makes appropriate validations of the user-provided data.

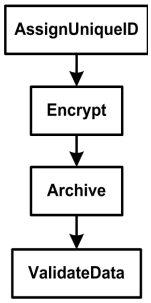

Fig 6. Abstract composition plan in a BPEL language with predefined service calls

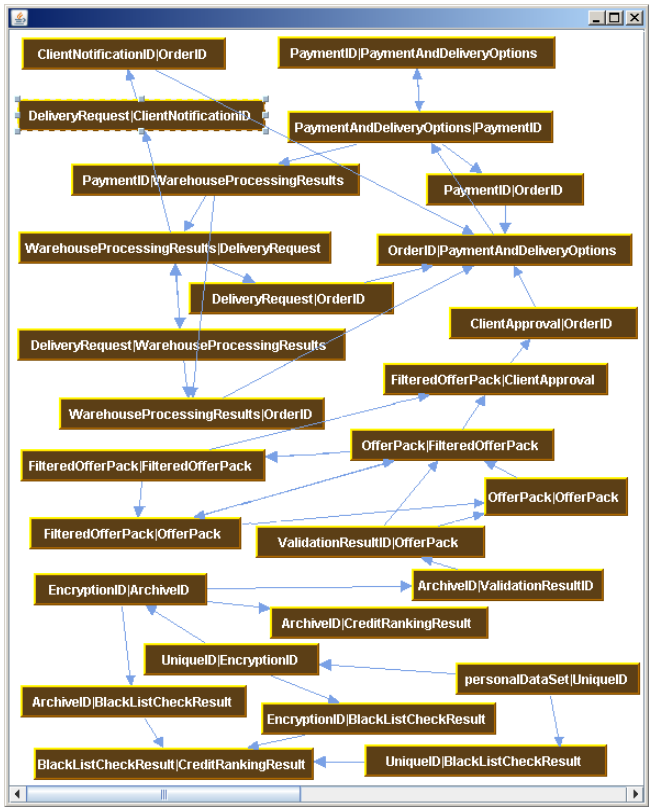

Fig 7. Complete service dependency graph for Service Repository

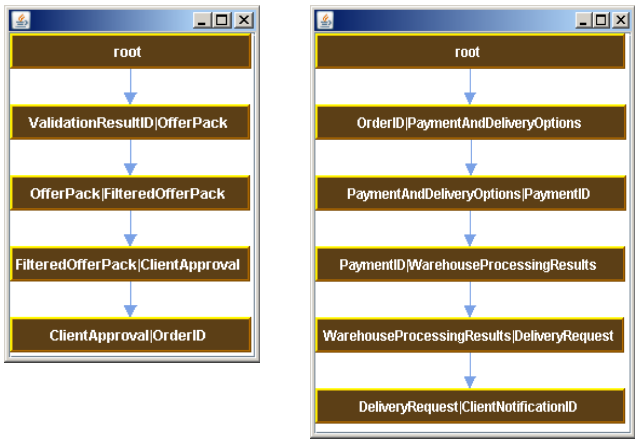

Fig 8. Graph (left) and Tree (right) Composition Engines work results

The automatic generation of the second and third parts of a composition plan was done by graph and tree composition engines. The graph-based engine composed the part of the plan responsible for product finding, selecting and placing an order in a system. The tree-based engine composed the part responsible for the processing of the order that had been placed earlier.

Fig. 7 presents a visualization of a complete services dependency graph of all registered types of web services that were registered in the Networked Service Repository. That exact graph was generated by a Graph Composition Engine during the composition process itself and visualized by the visualization functionality of the software platform. All service type IDs were automatically generated by the Networked Services Repository. Each of these IDs consisted of service input type name, "|" character and service's output type name.

The left part of Fig. 8. presents a visualization of an abstract plan that was generated by a Graph Composition Engine. "ValidationResultID" is an output type of the last service call in a predefined part of the service composition which was imported from a BPEL file, so it was passed to the Graph Composition Engine as a desired input type. "OrderID" type is a type which corresponds to the output type of the order creation service, so it was passed to the composition engine as a desired output type of the composition.

The subsequent steps of a plan generated by the Graph Composition Engine are as follows:

1. "ValidationResultID|OfferPack" represents an automatic wide search of possible products on the client's request.
2. "OfferPack|FilteredOfferPack" represents automatic filtering of the previously found products.
3. "FilteredOfferPack|ClientApproval" represents the client's acceptance of a product offer.
4. "ClientApproval|OrderID" represents generating an order for the offer that had already been accepted.

The right part of Fig. 8. presents a visualization of an abstract plan generated by a Tree Composition Engine. "OrderID" type was passed to the composition engine as a desired input type because it has to be the same as the output type of a Graph Composition Engine's work result. "ClientNotificationID" represents the result of client notification which always happens after an order is processed, so it was passed to the Tree Composition Engine as a desired output type.

The subsequent steps of a plan generated by the Tree Composition Engine are as follows:

1. "OrderID|PaymentAndDeliveryOptions" represents a user's process of choosing payment and delivery options for a created order.
2. "PaymentAndDeliveryOptions|PaymentID" represents the act of payment for the delivery by a client.
3. "PaymentID|WarehouseProcessingResults" represents all background warehousing processing such as searching for the warehouse nearest to the client, scheduling product pickup from the shelf, packing etc.
4. "WarehouseProcessingResults|DeliveryRequest" represents creating a delivery request to a logistic company which will actually deliver the products to the customer.
5. "DeliveryRequest|ClientNotificationID" represents the client notification process during which the client receives information about the delivery and other order related things.

All three parts of a complete abstract service composition plan were merged after they were created or imported by the corresponding engines. After that the complete plan was divided into three grounding areas and grounded in a hybrid mode.

The first grounding area consisted only of the steps from the first part of the plan which had been imported from the

Fig 9. Grounded composition plan

BPEL. Because this part is very important and regulated by government and industry standards, it was grounded only by a QoS Engine which was tuned to select the best available service instance no matter the cost.

The second grounding area was defined as a steps chain from "ValidationResultID|OfferPack" service till "OrderID|PaymentAndDeliveryOptions". The main grounding engine for that area was the Cost Engine and the second one was the QoS engine. The Cost Engine, however, was configured to choose not the absolutely best service from a variety of the available ones, but a range of acceptable services within a provided distance from the best one. The additional grounding engine for the second area was the QoS Engine which was able to choose the service instance with the best cost from the range of the previously selected ones by the Cost Engine.

The third grounding area was defined as a steps chain from "PaymentAndDeliveryOptions|PaymentID" till "DeliveryRequest|ClientNotificationID". It was grounded similarly to the second part but the difference was that the main grounding engine was the QoS Engine and the second one was the Cost Engine.

Fig. 9. presents a visualization of a grounded composition plan. The only difference between the visualizations of the abstract and grounded composition plans are URL addresses of the WSDL files in every step of the composition. These addresses unequivocally correspond to real-world service instances due to that fact that the data in each WSDL file describes a concrete service instance.

The execution of a service composition plan was made with the use of an Execution Module which was making service calls to appropriate instances by their URLs.

C. Implementation Details

The described platform was implemented in Java 6 programming language. Apache Tomcat 7 was used as the servlet container which hosted all the services and the Networked Service Repository as well. Spring Web Services 2 framework was used for the creation of the all web services including the Networked Service Repository. All the communication between the services, repository and Execution Module was carried out using a SOAP protocol over HTTP. Hibernate 4 was used as an object relation mapping framework for storing all service descriptions data in an in-memory H2 version 1.3 database. Such kind of database was used instead of a standalone one because it is easier to use and maintain in projects of the prototype nature. The SCDG was implemented upon JgraphT 0.8 library abstractions which also provided valuable algorithms for the Graph Composition Engine. Jdom 1.3 library for XML was used to handle all XML operations across all platform modules. JGraph 5 library was used to draw the visualizations of service composition plans in the SCDG format such as Fig. 7, 8 and 9.

IV. Conclusion and Future Work

This work presents an approach to creating a software platform that allows its users to combine different composition and grounding methods. Such features enable software users to control composition and grounding processes in a different and more powerful way thus allowing them to create better suited abstract and grounded composition plans.

The proposed approach was also verified during the implementation and execution tests of the described platform. The verification revealed that hybrid composition and hybrid grounding approaches are viable tools that can be used to create a better suited abstract and executable service composition plans.

The main implication of the presented work is the fact that users of software platforms that implement the proposed approach will have more flexibility and control over service composition and grounding processes. Many composition and grounding methods have been proposed, yet each of them is different and may not suit all the needs of the end-point customer. Furthermore, to satisfy all the upcoming and even hitherto unknown customer needs software systems must allow changes to be introduced in them. The ability to choose and combine different service composition and grounding methods addresses these problems by enabling users to select and merge optimal methods for their needs.

There are also three main directions of the upcoming work for the proposed concept. The first one is studying the desired properties of the service universe and service granularity in order to achieve high automation rates. The second one is a hybrid execution of grounded plans. The combined usage of different execution engines might bring some additional features since these engines might employ different approaches and thus be valuable from different points of view. The last direction of the studies has to be made in the field of dynamic composition, grounding and execution methods. Such methods can be very desirable e.g. in soft-

ware platforms where the fault-tolerance level of services is low or the environment itself may constantly be changing.

REFERENCES

[1] E. Sirin, J. Hendler, B. Parsia, "Semi-automatic composition of Web Services using semantic descriptions", in *Proc. Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS2003*, Apr. 2003, pp. 17–24.

[2] E. Sirin, J. Hendler, B. Parsia, "Filtering and selecting semantic Web Services with interactive composition techniques", *IEEE Intelligent Systems*, vol. 19, pp. 42–49, 2004.

[3] S. Thakkar, C. Knoblock, J. Ambite, C. Shahabi, "Dynamically composing Web Services from on-line sources", in *Proc. AAAI-2002 Workshop on Intelligent Service Integration*, July 2002.

[4] M. Sheshagiri, M. Desjardins, T. Finin, "A planner for composing services described in DAML-S", in *Proc. AAMAS Workshop on Web Services and Agent-based Engineering*, July 2003.

[5] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau, "HTN planning for Web Service composition using SHOP2", *Web Semantics: Science, Services and Agents Journal*, vol. 4, pp. 377–396, 2004.

[6] S. Sohrabi, J. Baier, S. McIlraith, "HTN planning with preferences", *Web Semantics: Science, Services and Agents Journal*, vol. 4, 2004, pp. 377–384.

[7] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, "DAML-S:Web Service description for the Semantic Web", in *Proc. International Semantic Web Conference (ISWC) 2002*, June 2002, pp. 348–363.

[8] R. Hamadi, B. Benatallah, "Petri Net-based model for Web Service composition", in *Proc. 14th Australasian database conference on Database technologies*, 2003, pp. 191–200.

[9] D. Chakraborty, Y. Yesha, A. Joshi, "A distributed service composition protocol for pervasive environments", in *Proc. 2004 IEEE Wireless Communications and Networking Conference*, Mar. 2004, pp. 2575–2581.

[10] S. Sun, X. Tang, X. Yan, D. Chen, "A symmetric matchmaking engine for Web Service composition", in *Proc. 15th International Conference on Parallel and Distributed Systems*, Dec. 2009, pp. 810–814.

[11] D. Liu, Z. Shao, C. Yu, D. Chen, G. Fan, "A heuristic QoS-aware service selection approach to Web Service composition", in *Proc. 8th IEEE/ACIS International Conference on Computer and Information Science*, June 2009, pp. 1184–1189.

[12] J. Tang, X. Xu, "An adaptive model of service composition based on policy driven and multi-agent negotiation", in *Proc. 5th International Conference on Machine Learning and Cybernetics*, Aug. 2006, pp. 113–118.

[13] H. Yan, W. Zhijian, L. Guiming, "A novel Semantic Web Service composition algorithm based on QoS ontology", in *Proc. 2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, June 2010, pp. 166–168.

[14] S. Bleul, T. Weise, "An ontology for quality-aware service discovery", in *Proc. First International Workshop on Engineering Service Compositions*, Dec. 2005, pp. 35–42.

[15] L. Belava, "Concept of hybrid service composition in SOA environment", *Automatyka*, vol. 13/2, pp. 189–197, 2009.

[16] S. Ponnekanti, A. Fox, "SWORD: A developer toolkit for Web Service composition", in *Proc. 11th International WWW Conference*, May 2002.

[17] R. Aggarwal, "Constraint driven Web Service composition in METEOR-S", in *Proc. IEEE International Conference on Services Computing*, Sep. 2004, pp. 22–30.

[18] V. Chifu, I. Salomie, A. Riger, V. Radoi, "A graph based backward chaining method for Web Service composition", in *Proc. IEEE 5th International Conference on Intelligent Computer Communication and Processing*, Aug. 2009, pp. 237–244.

[19] E. Silva, L.F. Pires, M. Sinderen, "An algorithm for automatic service composition", in *Proc. 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, July 2007, pp. 65–74.

[20] Y. Wang, H. Wang, X. Xu, "Web Services selection and composition based on the routing algorithm", in *Proc. 10th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pp. 69–73, Oct. 2006.

[21] L. Belava, "Algorithm for the conversion of service composition directed graph into BPEL service composition plans", *Automatyka*, vol 15/2, pp. 71–80, 2011.

[22] L. Belava, "Transforming BPEL service composition into a service composition directed graph for better composition plan management", in *Proc. 25th European Conference on Modelling and Simulation*, June 2011, pp. 424–429.