Towards an Automated Deployment Planner for Composition of Web Services as Software Components

Abbas Heydarnoori [1]

School of Computer Science
University of Waterloo, Canada

Farhad Mavaddat [2]

School of Computer Science
University of Waterloo, Canada

Farhad Arbab [3]

Department of Software Engineering
Centrum voor Wiskunde en Informatica (CWI), The Netherlands

Abstract

In this paper, we present our work-in-progress on developing an automated deployment planner for the composition of Web services as software components using the Reo coordination middleware in a distributed environment. Web services refer to accessing services over the Web. Reo is an exogenous coordination model for compositional construction of component connectors based on a calculus of mobile channels that has been developed at CWI (the Netherlands). Reo has a strong theoretical underpinning which makes it a good candidate model for coordinating the work of Web services participating in a composition. Suppose a new Web application has been developed by composing a number of Web services with different requirements and constraints. To run the application, it is required to deploy it on a number of hosts with different computational capabilities available to the application in the distributed environment (e.g., Internet) so that all constraints and requirements are satisfied. Because of the many parameters and constraints in such a deployment problem, it is difficult to do it manually. Thus, an automated deployment planner is required for this purpose.

Key words: software components, software deployment, Reo coordination model.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

The Internet is rapidly changing from a set of wires and switches that carry packets into a sophisticated infrastructure that delivers a set of complex value-added services to end users [1]. The term "Web service" came into being to represent a unit of business logic that an organization exposes to other organizations on the World Wide Web. Web services are gradually becoming the most popular distributed computing paradigm for the Internet [2]. Advances in Internet infrastructure and rapid evolution of the WWW are major enablers of Web services.

Web services can be stand-alone or linked together to provide enhanced functionality. In other words, Web services are inter-operable building blocks for constructing applications. Therefore, the composition of Web services is an important issue and a general coordination model for composing Web services is required. The Reo coordination model is a good candidate for serving this purpose [3]. Reo presents a paradigm for composition and exogenous coordination of software components based on the notion of mobile channels. In the Reo model, complex coordinators, called connectors, are compositionally built out of simpler ones. Reo has a strong theoretical underpinning and its logic is mathematically modeled. In [4] one can find a coalgebric formal semantic for Reo. Reo promotes loose coupling, distribution, mobility, exogenous coordination, and dynamic reconfiguration. These properties make Reo a suitable candidate model for composing Web services in a distributed environment.

Suppose a distributed application has been developed which utilizes a number of Web services with different requirements and constraints. In addition, the Reo coordination middleware is used to coordinate the work of these Web services. In this method, Web services are viewed as black box software components, i.e., there are no concerns regarding their developments and internals. To run the application, it is required to instantiate different components of the application on different hosts with different computational capabilities available to the application in the distributed environment. Furthermore, this should be done in such a way that all requirements and constraints are met. This process is called *software deployment*.

For large applications which consist of many components with many constraints and should be distributed on a large number of hosts with different characteristics, manual deployment is impractical. Furthermore, users may have specific requirements in such a deployment. For example, they may want a specific component to be instantiated on a specific host. This complicates the problem even more. Thus, an automated deployment planner is required to effectively specify where different components should be instantiated in the distributed environment. In this paper, our work-in-progress on developing

[1] Email: aheydarnoori@uwaterloo.ca
[2] Email: fmavaddat@uwaterloo.ca
[3] Email: Farhad.Arbab@cwi.nl

such an automated deployment planner is described.

This paper is organized as follows. Section 2 provides the required background of this research. In this section, Web services and the Reo coordination model are briefly described. In Section 3, developing a deployment planner for Web services compositions using the Reo coordination middleware is discussed. Section 4 reviews some related works and finally in Section 5, concluding remarks are provided.

2 Background

The following topics form the background of our work: Web services, and the Reo coordination model. In this section, we describe them.

2.1 Web Services

As the term "Web service" shows, it refers to accessing services over the World Wide Web. According to the definition of Web services by IBM [5], "Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes".

As this definition shows, Web services can be seen as the building blocks for constructing distributed Web applications. A significant difference between the Web services model and other existing models such as CORBA/IIOP, COM/DCOM, and Java/RMI is that Web services can be written in any language and can be accessed using the HTTP protocol. In other words, in the Web services computing model, distributed software components are interfaced via non-object-specific protocols [6].

2.1.1 Viewing Web Services as Software Components

The aim of component-based software development is to make a new system by composing and integrating existing software components together. This method of software development has many advantages and it has been proposed as a paradigm for developing more reliable and higher-quality software systems within shorter development time, lower cost and effort [7]. Because of the many advantages of CBSD, there is a widespread belief that one day developers could easily assemble applications from prebuilt components instead of writing them from scratch.

As mentioned earlier, Web services are self-contained, self describing modular units providing location independent business or technical services that can be published, located, and invoked across the Web. Thus, one can view them as a natural extension of software component thinking. Web services, as software components, represent black box functionalities that can be reused without worrying about how those services are implemented, or where they

are situated [8]. In other words, they can be used as the building blocks for the development of complex distributed applications.

2.2 Reo Coordination Model

Reo is a channel-based coordination model that exogenously coordinates the cooperative behavior of component instances in a component-based application [3]. From the point of view of Reo, an application consists of a number of component instances communicating through connectors that coordinate their activities. The emphasis of Reo is on connectors, their composition and their behavior. Reo does not say much about the components whose activities it coordinates. In Reo, connectors are compositionally constructed out of a set of simple channels. Thus, channels represent atomic connectors. A channel is a communication medium which has exactly two channel ends. A channel end is either a *source* channel end or a *sink* channel end. A source channel end accepts data into its channel. A sink channel end dispenses data out of its channel. Although every channel has exactly two ends, these ends can be of the same or different types (two sources, two sinks, or one source and one sink). Reo assumes the availability of an arbitrary set of channel types, each with well-defined behavior provided by the user. However, a set of examples in [3] show that exogenous coordination protocols that can be expressed as regular expressions over I/O operations correspond to Reo connectors which are composed out of a small set of only five primitive channel types:

- *Sync:* It has a source and a sink. Writing a value succeeds on the source of a *Sync* channel if and only if taking of that value succeeds at the same time on its sink.
- *LossySync:* It has a source and a sink. The source always accepts all data items. If the sink does not have a pending read or take operation, the LossySync loses the data item; otherwise the channel behaves as a *Sync* channel.
- *SyncDrain:* It has two sources. Writing a value succeeds on one of the sources of a *SyncDrain* channel if and only if writing a value succeeds on the other source. All data items written to this channel are lost.
- *AsyncDrain:* This channel type is analogous to *SyncDrain* except that the two operations on its two source ends never succeed simultaneously. All data items written to this channel are lost.
- *FIFO1:* It has a source and a sink and a channel buffer capacity of one data item. If the buffer is empty, the source channel end accepts a data item and its write operation succeeds. The accepted data item is kept in the internal buffer. The appropriate operation on the sink channel end (read or take) obtains the content of the buffer.

In Reo, a connector is represented as a graph of nodes and edges such that: zero or more channel ends coincide on every node; every channel end

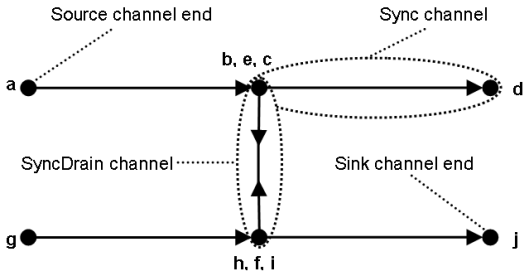

Fig. 1. Barrier synchronization connector in Reo

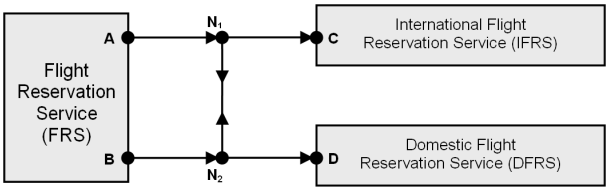

Fig. 2. Modeling the flight reservation system with Reo

coincides on exactly one node; and an edge exists between two (not necessarily distinct) nodes if and only if there exists a channel whose channel ends coincide on those nodes. As an example of Reo connectors, Fig. 1 shows a *barrier synchronization* connector in Reo. In this connector, a data item passes from a to d only simultaneously with the passing of a data item from g to j and vice versa. This is because of the *"replication on write"* property in Reo, and different characteristics of different channel types.

2.2.1 An Example of Composing Web Services Using Reo

In the following, we provide a simple example of how a Reo connector such as barrier synchronization can be used to compose a number of Web services together.

Suppose a travel agency wants to offer a Flight Reservation Service (FRS). For some destinations, a connection flight might be required. For example, if you want to travel from Toronto to Glasgow, you need to travel from Toronto to London. Then, you need to travel from London to Glasgow. Suppose some other agencies offer services for International Flight Reservation (IFRS) and Domestic Flight Reservation (DFRS). Thus, FRS commits successfully whenever both IFRS and DFRS services commit successfully. This behavior can be easily modeled by a barrier synchronization connector in Reo (Fig. 2). The FRS service makes commit requests on channel ends A and B. These commits will succeed if and only if the reservations at the IFRS and DFRS services succeed at the same time.

This example shows how Reo succeeds in modeling complex behaviors. In Reo, it is easily possible to construct different connectors by a set of simple composition rules out of a very small set of primitive channel types. One can find a more elaborate introduction to Reo in [9], and a detailed description of the language and its model in [3].

3 Developing A Deployment Planner

In the previous section, we described the Reo coordination model and introduced it as a good candidate model for composing and coordinating Web services. For more information on compositional construction of Web services using the Reo coordination model, you can refer to [10] where the requirements of Reo-enabled Web services are discussed. That paper shows the necessary layers between Web services and the Reo coordination middleware which are necessary for composing them together using the Reo connectors. In this section, another aspect of this problem is considered and our ongoing work on developing a deployment planner for the composition of Reo-enabled Web services is introduced. We begin with a description of the software deployment process.

3.1 Software Deployment Process

Software deployment is a sequence of related activities for placing a developed application into its target environment and making the application available for use. Though this definition of software deployment is reasonable and clear, for developing an automated deployment planner, the characteristics and nature of the deployment activities must be described more clearly.

Different sequences of activities are mentioned in literature for the software deployment process. Some of them are discussed in Section 4. However, in our view, the software deployment process should include at least the following activities: *Acquiring*, *Planning*, *Installation*, *Configuration*, and *Execution*. Below are brief descriptions of these activities:

- *Acquiring:* In this activity, the components of the application being deployed and the metadata specifying the application are acquired from the software producer and are put in a repository to be used by other activities of the deployment process.
- *Planning:* Given the specifications of the component-based application, a target environment, and user-defined constraints, this activity determines where different components of the application will be executed in the target environment, resulting in a deployment plan.
- *Installation:* This activity uses the deployment plan generated in the previous activity to install the application into the target environment. More specifically, this activity transfers the components of the application from the repository to the hosts in the target environment.
- *Configuration:* After installing the application components into the target environment, it might be necessary to modify its settings and configurations. For example, after installing an application, one may want to set different welcome messages for different users.
- *Execution:* following the installation and configuration of the software ap-

plication, it can be run. More specifically, the installed application components into the hosts are launched, the interconnections among them are instantiated, the components are connected to the interconnections, and the software application actually starts to work.

In this paper, our focus is on the planning stage of this process. For large, complex applications similar to Web applications being considered in this paper, users should not be required to manually deploy a large number of components with different properties on several hosts in a distributed environment. Therefore, this process should be as automated as possible. In the following section, we talk about this automated deployment planner in more detail.

3.2 Deployment Planner

In the previous section, we defined the software deployment process in general and motivated the need to develop an automated deployment planner for deploying large, complex, component-based applications into distributed environments. In this section, we describe an automated deployment planner in the context of Web services compositions using the Reo coordination middleware.

Suppose the specification of the Web application to be deployed is given. In this application, a number of Web services are composed together by using a Reo circuit. Thus, this specification specifies these Web services, their requirements and constraints, and the Reo circuit used among them. The implementations of these Web services and their internals are not important and they are viewed as black box software components. Furthermore, this specification describes the Reo circuit by specifying the nodes of the Reo circuit, channels among these nodes and their types, and each Web service is connected to which node.

In addition to this specification, the specification of the available resources in the distributed environment is given. This specification describes a number of hosts and their computational capabilities. The computational capabilities of these hosts are different implementations of Reo channels they can support. In this level of abstraction, low level hardware parameters as CPU speed, memory, disk, etc. are not important. The reason is that we wish to focus on software abstraction and not hardware abstraction. As an example of computational capabilities, suppose host A can support three different implementations of the Reo's *Sync* channel (e.g., simple message passing, encrypted message passing, and using the shared memory). Logically, they are all implementations of the *Sync* channel, but their requirements, costs, and speeds differ. Similarly, different channel types have different implementations on different hosts.

Furthermore, users should be able to specify their constraints and requirements regarding the deployment of the application. Some examples of these

requirements are certain Web services on certain hosts, certain quality of service (QoS) requirements like cost, speed, and so on.

The deployment planner uses these specifications as input and generates the specification of a deployment plan. In this deployment plan, different pieces of the application (Web services and Reo nodes) are mapped to the available resources subject to the given constraints. In other words, this deployment plan specifies each of the Web services and nodes of the Reo circuit should run where in the target environment. In the following section, different issues that should be considered in developing such a deployment planner are discussed.

3.3 Challenges in Developing a Deployment Planner

In the previous sections, we provided the problem definition of our ongoing work on developing an automated deployment planner for Web applications. In this section, we present different aspects of this problem and provide a list of the sub-problems we have to cope with in order to solve the whole problem.

One of the important sub-problems that should be considered is related to resource allocation. The deployment planner is supposed to optimally allocate resources available at different hosts to accommodate the requirements and constraints of the application. So, generating such a deployment plan becomes a constraint satisfaction problem and so, it should be possible to develop a mathematical representation of that problem and then solve it. Generally, finding the best solution for such problems that have many parameters is impossible. So, we should try to find the best possible solutions for them. For this purpose, a set of heuristics should be developed and applied to effectively solve such constraint satisfaction problems. In Section 3.4, an example of such heuristics is provided.

Another important issue in a deployment is its quality. For any large, complex Web application multiple deployments in a distributed environment are typically possible. Obviously, some of those deployments are more effective than others in terms of some quality of service (QoS) requirements such as cost, reliability, speed, efficiency, and so on. Maximizing the QoS of a given system may require the system to be redeployed [11]. Thus, considering the issues related to QoS represents another important aspect of this project.

Other issues relate to specification languages. As mentioned earlier, the specification of the Web application to be deployed and the specification of the distributed environment should be provided as inputs to the deployment planner. Thus, specification languages are required for this purpose. We name these languages *Application Specification Language (ASL)* and *Resource Specification Language (RSL)* respectively. ASL will be used to specify Web services utilized in the application, the Reo circuit used to compose them, and requirements of the application. RSL will be used to specify different hosts in the distributed environment available to the application, their computational

capabilities, and their constraints. Furthermore, for generating deployment plans, a *Deployment Specification Language (DSL)* should be devised. The deployment planner will use this language to generate deployment plans.

3.4 A Graph-based Approach for Deployment Planning

At the time of writing this paper, we have used a graph-based approach to solve the software deployment problem. For this purpose, two graphs are made in this approach: the *Application Graph*, and the *Target Environment Graph*. The application graph models a component-based application as a graph of components connected by different channel types. The target environment graph models the distributed environment as a graph of hosts connected by different channel types that can exist between every two hosts. In other words, before starting the deployment planning, the channel types that can exist between every two hosts in the target environment are specified. Then, the deployment planning of an application is defined as the mapping of its application graph to its target environment graph, subject to maximization of the desired QoS parameter. As an example of how such efficient algorithms and techniques can be applied to effectively solve the deployment problem, in the following, we talk about finding the most cost-effective deployment configuration.

Suppose different hosts in the target environment have different costs and whenever they are being used, their costs should be paid to their administrator(s). In this situation, the most cost-effective deployment configuration should be found. For this purpose, a collection of available hosts in the distributed environment must be selected so that the total cost of the deployment is minimal and all components are also assigned to a host. It is easily possible to prove that this problem is equivalent to the *Minimum Set Cover* problem in graph theory [12].

Definition 3.1 (Minimum Set Cover Problem) Given a finite set U of n elements, a collection of subsets of U, $S = \{s_1, s_2, ..., s_k\}$ such that every element of U belongs to at least one s_i, and a cost function $c : S \rightarrow R$, the problem is to find a minimum cost sub-collection of S that covers all elements of U.

The cost-effective deployment problem can be converted to the minimum set cover problem in the following way:

- Set $U = \{C_1, C_2, ..., C_n\}$, i.e., the components of the application are set as the elements of the universe;
- Set $S = \{CS_{H_1}, CS_{H_2}, ..., CS_{H_m}\}$ in which each CS_{H_i} corresponds to host H_i, and it represents the set of components of the application that can be run on host H_i.
- Define $c : S \longrightarrow R$ return the cost of each host.

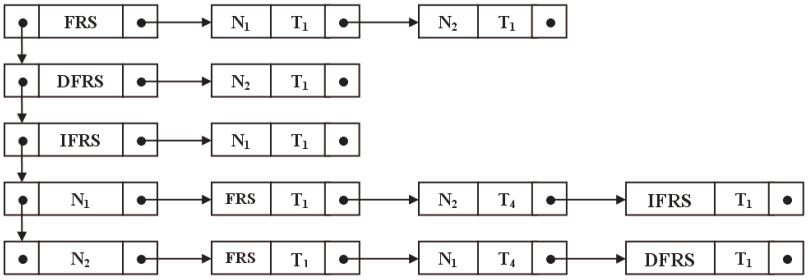

Fig. 3. Linked-list holding the application graph of the flight reservation system

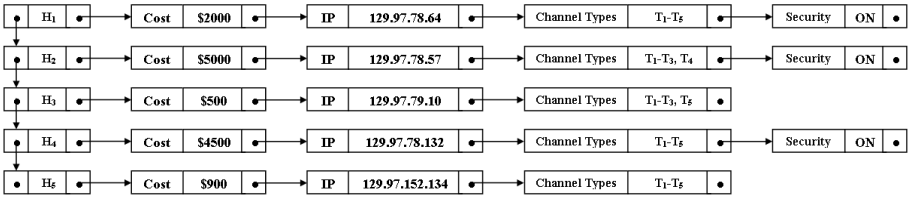

Fig. 4. A sample linked-list holding the properties of available hosts in the target environment

However, it is proved that the minimum set cover problem is a NP-hard problem and it can not be solved in polynomial time [12]. But, there exist some greedy approximation algorithms that can find reasonably good answers in polynomial time [12]. Thus, to solve the cost-effective deployment problem, first it can be converted to the minimum set cover problem as mentioned above. Then, by using existing algorithms for solving the minimum set cover problem, all components of the application will be assigned to at least one host and the cost of the deployment will be close minimal too.

3.5 Prototype Implementation

We have implemented a deployment planner tool by Java to represent how our approach works in practise. This tool can be used for planning the deployment of any kind of component-based applications, and it is not specific to Web-based applications.

The specifications of the application being deployed and the target environment are given to this tool by two input files. The data structure used to hold the information about the application and the target environment is *linked-list*. In this structure, the topology of the application is kept as a linked-list of components. Each component itself points to a linked-list containing the information about adjacent components and the channel types used to connect the current component to them (Fig. 3). Also, the information about the target environment is kept as a linked-list of hosts. Each host points to a linked-list holding the properties of that host (Fig. 4). These properties include different channel types they can support, their costs, their IPs, and so on. In Fig. 3 and Fig. 4, T_is represent different channel types or implementations. As we see, this linked-list data structure is flexible and gives

us the freedom to define as many properties as we want for different hosts. After processing the input files and generating these linked-lists, the deployment planner tool uses them and starts to generate the actual deployment plan. At the time of writing this paper, this tool can be used to find the most cost-effective deployment configuration in polynomial time ($O(log(n))$) by using the techniques introduced in the previous section and the greedy approximation algorithm presented in [12]. For future work, we plan to design a number of algorithms for maximizing other QoS properties (e.g., reliability, performance, security, etc.), and include them in this tool.

4 Related Work

Deployment of component-based applications into distributed environments is an open problem in both research and industry and it is possible to find a large number of tools and papers related to it. In this section, two of the most relevant ones are considered: *Software Dock* and *OMG D&C Specification*.

4.1 Software Dock

The University of Colorado Software Dock research project has created a distributed, agent-based deployment framework that supports cooperation among software producers themselves and between software producers and software consumers [13]. The Software Dock architecture is shown in Fig. 5. This architecture has the following components:

- *Release dock:* Its purpose is to serve as a release repository for the software systems provided by the software producer. In this repository, each software release is semantically described using a standard semantic schema: *Deployable Software Description* or *DSD*.
- *Field Dock:* It is a server residing at a consumer site and provides information about the resources and configuration of the consumer site.
- *Agents:* Each software release is accompanied by generic agents that perform software deployment processes with the help of interpreting the semantic description of the software release.
- *Wide-area event system:* The release dock generates events as changes are made to the software release that it hosts.

In Software Dock research project, software deployment is defined as a collection of interrelated activities that form the *software deployment life cycle* [14]. This cycle includes the following activities: *release*, *install*, *activate*, *update*, *adapt*, *reconfigure*, *deactivate*, *remove*, and *retire*. These activities can be divided into two groups:

- **Producer-side Activities:**
  - *Release:* This includes all the tasks required to package, prepare, provide, and advertise a system for deployment to consumer sites. This activity

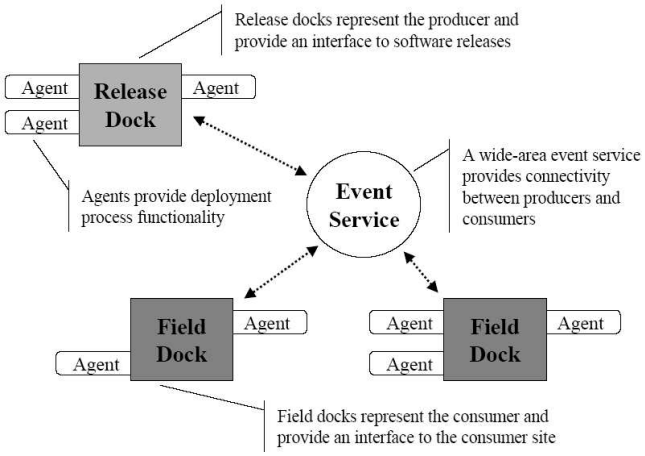

Fig. 5. Software Dock architecture (taken from [13])

acts as a bridge between development and deployment.

- *Retire:* When a software system or a given configuration of a software system is no longer supported by the software producer, this activity is done.

- **Consumer-side Activities:**
  - *Install:* This activity configures and assembles all of the necessary resources for using a given software system.
  - *Activate:* This is responsible for running or executing a deployed software system.
  - *Deactivate:* This is responsible for shutting down any executing components of an activated software system.
  - *Update:* This modifies a previously installed software system and deploys a new, previously unavailable configuration of a software system.
  - *Adapt:* This activity maintains the consistency of the currently selected configuration of a deployed software system.
  - *Reconfigure:* Its purpose is to select a different configuration of a previously deployed software system from its existing semantic description.
  - *Remove:* this activity is performed when a software system is no longer required at a consumer site.

A description of the actual deployment by the architecture presented in Fig. 5 follows. When a software system is to be installed on a given consumer site, initially an agent responsible for installing that software and the DSD description of that software are loaded onto the consumer site from the originating release dock. This agent docks at the local field dock and configures the software system using the DSD description of that software and the consumer site state information provided by the field dock. When this configuration is done, this agent asks the precise configuration that it requires from its release dock. It also may request other agents (such as update and adapt) from its release dock to come and dock at the local field dock and do other deployment activities. The wide-area event service provides a means of

connectivity between software producers and software consumers.

4.2 OMG Deployment and Configuration Specification

The "OMG Deployment and Configuration Specification" or "OMG D&C Specification" is an attempt towards a unified framework for the deployment of component-based applications [15]. The deployment process defined in this specification consists of five stages:

- *Installation:* During installation, the software package is put into a repository. This activity does not involve transfer of binary files to the hosts.
- *Configuration:* When the software is installed in the repository, its functionality can be configured.
- *Planning:* This planning involves selection of hosts the software will run on, the resources it will require to run, deciding which implementations will be used for component instances, and so on.
- *Preparation:* This activity prepares the target environment for the execution of the software.
- *Launch:* In this stage, the application is executed. As planned, component instances are created and configured on hosts in the target environment and the connections among the instances are established.

The OMG D&C Specification defines three platform independent models, the *component model*, the *target model*, and the *execution model*. Each of these models is also split into the data model and the runtime (management) model to reduce the complexity. The runtime models deal with runtime entities and they are outside the scope of this paper. Below are brief descriptions of data models:

- *Component Data Model:* This model provides information about installed and configured software packages in the repository. This information includes descriptions about interfaces, implementations, configurations, etc., of the package components.
- *Target Data Model:* This model describes the target environment in which the application will be deployed.
- *Execution Data Model:* This is the deployment plan and specifies the component-based application in terms of component instances, connections among them, and assignments of the instances to the computational hosts in the target environment.

As mentioned earlier, these models are platform independent. In order to use them with a specific component model, they should be transformed to platform dependent models, capturing the specifics of the concrete platform [16]. An example of this transformation to the CORBA Component Model (CCM) can be found in [15].

5 Conclusions

The aim of software deployment process is to bring a developed application into its target environment and make it available for use. For large, complex, component-based applications that include many components with different requirements and should be distributed in many hosts with different computational capabilities, manual deployment is not easy, and automated tools are required for this purpose. In this paper, we presented our ongoing work on developing an automated deployment planner for Web services applications using the Reo coordination middleware. The strong formal basis of Reo and its easy-to-use composition rules encouraged us to choose Reo as the coordination model for Web services compositions. In this method, Web services are treated as black box software components.

For the given specifications of a Web application to be deployed, available resources in the distributed environment, and user-defined constraints, a deployment plan specifies each of the components of the application should run on which of the hosts to instantiate a running Web application so that all requirements and constraints are met.

Acknowledgment

The authors would like to acknowledge Dr. Nikolay Diakov for his helpful comments in developing the ideas expressed in this paper.

References

[1] Chandra, P., Fisher, A., Kosak, C., Ng, T. S. E., Steenkiste, P., Takahashi E. and Zhang, H., "Darwin: Customizable resource management for value-added network services," In Proceedings of the 6th IEEE International Conference on Network Protocols, Oct. 1998.

[2] Gergic, J., Kleindienst, J., Despotopoulos, Y., Soldatos, J., Patikis, G., Anagnostou, A. and Polymenakos, L., "An Approach to lightweight deployment of web services," In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering 2002 (SEKE 2002), ACM Press, 635-640.

[3] Arbab, F. "Reo: A Channel-based Coordination Model for Component Composition," Mathematical Structures in Computer Science, 14, 3 (June 2004), 329-366.

[4] Arbab, F. and Rutten, J.J.M.M. "A Coinductive Calculus of Component Connectors," In Proceedings of 16th International Workshop on Algebraic Development Techniques (WADT 2002), LNCS 2755, Springer-Verlag, 2003, 35-56.

[5] "Web services: the Web's next revolution," https://www6.software.ibm.com/developerworks/education/wsbasics/wsbasics-a4.pdf, Last visited: Sep. 30, 2005.

[6] "Introduction to Web Services," http://www.embedded.com/story/OEG20020125S0103, Last visited: Sep. 30, 2005.

[7] Heydar Noori, A., and Mavaddat, F., "On Software Components Characterization and Specification," In Proceedings of the 9th Annual International CSI Computer Conference, Tehran, Iran, 2004.

[8] Stojanovic, Z., Dahanayake, A. and Sol, H., "Agile Modeling and Design of Service-Oriented Component Architecture," In Proceedings of the 1st European Workshop on Object-Orientation and Web Services at ECOOP 2003, Darmstadt, Germany, July 21-25, 2003.

[9] Arbab, F. and Mavaddat, F., "Coordination through channel composition," In Proceedings of the 5th International Conference on Coordination Models and Languages (Coordination 2002), LNCS 2315, Springer-Verlag, 21-38.

[10] Diakov, N. K. and Arbab, F., "Compositional Construction of Web Services Using Reo," In Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI'2004) (Porto, Portugal, April 13-14, 2004). INSTICC Press, 2004, 49-58.

[11] Mikic-Rakic, M., Malek, S., Beckman, N. and Medvidovic, N., "A Tailorable Environment for Assessing the Quality of Deployment Architectures in Highly Distributed Settings," In Proceedings of the 2nd International Working Conference on Component Deployment (CD 2004), Edinburgh, UK, May 2004.

[12] Cormen, T. H., Leiserson, C.E., Rivest, R.L., and Stein, C. "Introduction to Algorithms", Second edition, MIT Press, 2001.

[13] Hall, R.S., Heimbigner, D., and Wolf, A.L. A Cooperative Approach to Support Software Deployment Using the Software Dock. In *Proceedings of the 1999 International Conference on Software Engineering*, ACM Press, New York, May 1999, 174-183.

[14] Carzaniga, A., Fuggetta, A., Hall, R. S., Hoek, A. V. D., Heimbigner, D., Wolf, A. L., "A Characterization Framework for Software Deployment Technologies," Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.

[15] Object Management Group, "Deployment and Configuration of Component-based Distributed Applications Specification," http://www.omg.org/docs/ptc/04-05-15.pdf, Last visited: Sep. 30, 2005.

[16] Bulej, L. and Bures, T. Using Connectors for Deployment of Heterogeneous Applications in the Context of OMG D&C Specification. In *Proceedings of the INTEROP-ESA 2005 Conference*, Geneva, Switzerland, Feb 2005.