Deploying Loosely Coupled, Component-based Applications into Distributed Environments

Abbas Heydarnoori and Farhad Mavaddat
School of Computer Science,
University of Waterloo,
Waterloo, ON,
Canada, N2L 3G1
{aheydarnoori, fmavaddat}@cs.uwaterloo.ca

Farhad Arbab
Department of Software Engineering,
Centrum voor Wiskunde en Informatica,
P.O. Box 94079, NL-1090 GB,
Amsterdam, The Netherlands
Farhad.Arbab@cwi.nl

Abstract

With significant advances in software development technologies in recent years, it is now possible to have complex software applications, which include a large number of heterogeneous software components distributed over a large network of computers with different computational capabilities. To run such applications, their components must be instantiated on proper hardware resources in their target environments so that some requirements and constraints are met. This process is called software deployment. For large, distributed, component-based applications with many constraints and requirements, it is difficult to do the deployment process manually, and some automated tools and techniques are required. This paper presents a graph-based approach for this purpose that is not dependent on any specific component technology and does the deployment planning with respect to the communication resources required by application components and communication resources available on the hosts in the target environment. In our approach, component-based applications and distributed environments are modeled with the help of graphs. Deployment of an application is then defined as the mapping of the application graph to the target environment graph.

1 Introduction

In the past, software applications were stand-alone systems, without any connections to other software applications. In recent years, software applications have become more and more complex. They may consist of a large number of different components distributed over a large number of computers, and large networks have moved to the center of software applications. Furthermore, with the arrival of the Internet and new advances in Internet infrastructure, it is possible to have completely distributed applications that may consist of many heterogeneous components. In these applications, since different components provide their functionality with different constraints and requirements, they should be installed on proper hardware resources in the distributed environment so that their constraints are satisfied and they provide the desired quality of service (QoS). In addition, different resources have different computational capabilities, making it impossible to install any kind of software components on them. Thus, after the development of an application, a sequence of activities should be done to place that application into its target environment and bring that application into an executing state. This sequence of activities is referred to as the *software deployment process*, and includes the following activities: *acquiring* the developed application from its producer; *planning* where and how different components of the application should be installed in the target environment, resulting in a deployment plan; *installing* the application into its target environment according to its deployment plan; *configuring* it; and finally *executing* it.

For simple stand-alone software systems that should be deployed only to a single computer, deployment activities can be easily done manually. But, suppose a complex component-based application is being deployed into a large distributed environment so that some QoS parameters, such as performance or reliability, are also maximized. In this situation, the deployment process is not so straightforward, and automated tools and techniques are required for this purpose. Consequently, the software deployment process has been given special attention both in research and industry in recent years and it is possible to find many tools and papers addressing different activities of the software deployment process from different perspectives [1, 2, 3, 4, 5]. However, to our knowledge, few if any of these deployment approaches notices the characteristics (e.g., behavior, cost, speed, security, etc.) of interconnections among the components of the application. However, these characteristics have significant effects on application's QoS. This paper presents a graph-based approach that focuses on these properties for planning the deployment of loosely coupled, component-based applications into distributed environments. For this purpose, the concept of *channel* is used to model intercommunications among components. A channel is a point-to-point communication medium with well-defined behavior. A component-based application is then modeled as a graph of components connected by a number of channels, possibly with different characteristics. A distributed environment is also modeled as a graph of hosts connected by different channel types that can exist between every two hosts. Then, deployment planning is defined as the mapping of the application graph to the target environment graph so that the desired QoS parameter is maximized. As an example of this approach, we present how this mapping can be effectively done so that the cost of a deployment is minimized.

This paper is organized as follows: Section 2 talks about the Reo coordination model which is used as an example of channel-based coordination models throughout this paper. In Section 3, the inputs of the deployment planning process are discussed. In Section 4, our graph-based approach for deployment planning is described and finally in Section 5, concluding remarks are provided.

2 Case Study: Reo Coordination Model

Reo is a channel-based coordination model that exogenously coordinates the cooperative behavior of component instances in a component-based application [6]. 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 [6] 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 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 C only simultaneously with the passing of a data item from B to D and vice versa. This is because of the *"replication on write"* property in Reo, and different characteristics of

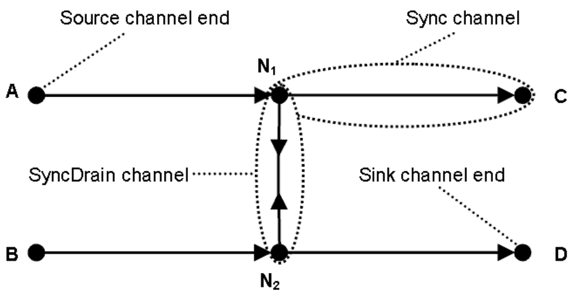


Figure 1. Barrier synchronization connector in Reo

different channel types. 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 [7].

2.1 Example: Modeling a Flight Reservation System with Reo

In this section, we provide a simple example of a flight reservation system which is used as the running example throughout this paper. In this example, the barrier synchronization connector in Reo is used to compose a number of Web services together. Web services refer to accessing services over the Web [8]. In this example, they are treated as black-box software components.

Suppose a travel agency wants to offer a Flight Reservation Service (FRS). For some destinations, a connection flight might be required. 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 behavior is because of the semantic of the barrier synchronization connector in Reo.

3 Deployment Planner Inputs

To generate deployment plans, the following inputs should be specified: (1) the component-based application being deployed, (2) the distributed environment in which the application will be deployed, and (3) the user-defined constraints regarding this deployment. In the following, these inputs are described in more detail.

3.1 Specification of the Application being Deployed

Any loosely coupled, component-based application consists of a number of *components* and *interconnections* that connect them. The nature of these components and interconnections are irrelevant to this specification. For example, components could be threads, processes, services, Java beans, CORBA components, and so on. In our model, a software component is viewed as a black-box software entity which reads data from its input port and writes data to its output port. How it manipulates the data, or its internal details are not important. The communication among these black-box entities is done via their interconnections. Again, these component interconnections could be anything connecting them; for example, glue code, middleware, connectors, and so on. Regardless of the type of these interconnections, different components send data/messages to other components and receive data/messages from other components of the application. Thus, it is possible to assume that the communication among the application components is done via a number of channels with different characteristics. Specially, it is proved that the primitives of other communication models (such as message passing, shared spaces, or remote procedure calls) can be easily modeled by the channel-based communication model [6].

In summary, the specification of the application should specify different components of the application and the channel types among them (e.g., Fig. 2).

3.2 Specification of the Target Environment

In this paper, the target environment for the deployment of the application is a distributed environment consisting of a number of hosts with computational capabilities (e.g., PCs, laptops, servers, etc.) connected by a network. Furthermore, the required software for the communication among the application components

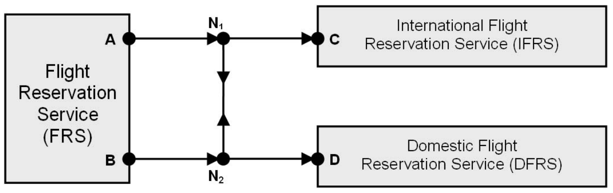

Flight Reservation Service (FRS)
A
B
N₁
N₂
C
D
International Flight Reservation Service (IFRS)
Domestic Flight Reservation Service (DFRS)

Figure 2. Modeling a flight reservation system with Reo

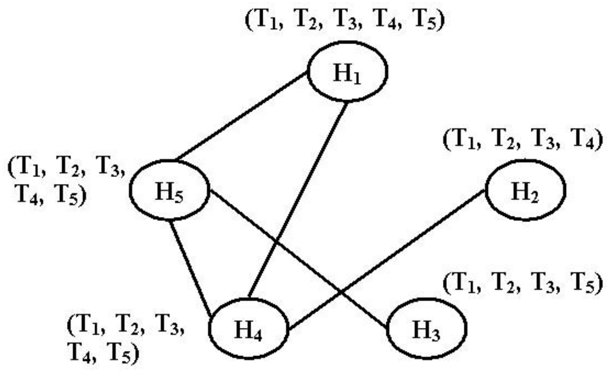


Figure 3. A sample target distributed environment for the deployment of the flight reservation system

(e.g., the Reo coordination middleware) has been already installed on them. However, since different hosts may have different hardware properties, it might be impossible to install some sorts of communication software on them, or they may not be able to support some features of the communication software installed on them. It is also possible that different features/versions of the communication software are installed on different hosts because of some reasons (e.g., cost, security, etc.). With respect to this discussion, available hosts in the target environment may provide different sorts of communication resources required to interconnect applications' components. In particular, since we are modeling the interconnections among the application components as a set of channels with different characteristics, different hosts might be able to support different sets of channel types (or implementations) with different behaviors and QoS characteristics. Thus, in this paper, communication resources available on different hosts are different channel types (or implementations) they can support. As an example, Fig. 3 shows a sample target environment for the flight reservation system consisting of five hosts $H_1 - H_5$, connected by a network (solid lines). In this figure, T_ds represent different channel types (or implementations) that different hosts can support. For example, in the case of using Reo coordination model, $T_1 - T_5$ could be defined as the following channel types (or implementations):

- T_1: *Sync* channel type implemented by shared memory;
- T_2: *Sync* channel type implemented by encrypted peer-to-peer connection;
- T_3: *Sync* channel type implemented by simple peer-to-peer connection;
- T_4: *SyncDrain* channel type;
- T_5: *SyncSpout* channel type.

Logically, $T_1 - T_3$ are all implementations of the same channel type (*Sync*). However, their hardware requirements and QoS characteristics differ.

3.3 Specification of the User-defined Constraints and Requirements

Users may have special requirements and constraints regarding the deployment of the application that should be taken into account during the deployment planning. For example, users may want a special component to be run on a certain host, or they may have certain QoS requirements such as security, cost, or reliability. The deployment planner needs this information to generate a plan that answers these requirements too.

For example, in the flight reservation system, suppose users require the transfer of data between FRS and IFRS to be encrypted. In addition, they want FRS to be run on H_1, IFRS to be run on either H_2 or H_3.

4 Deployment Planning

After specifying the deployment planner inputs, they can be used to generate the actual deployment plan. Fig. 4 shows one sample deployment for the flight reservation system. As can be seen in this figure, different components of the application and channels among them are mapped to different hosts in the target environment and network links among them for the purpose of this deployment. In this section, we show how graphs can be used to solve this mapping problem.

4.1 Modeling the Deployment Planner Inputs

The deployment planner inputs should be modeled with well-defined structures in order to be used for effective deployment planning purposes. In this section, we show that it is easily possible to develop graph representations of these inputs. This graph-based modeling can have several advantages. First, it is possible to have visual representation of the inputs. Second, graph theory algorithms can help us in designing deployment planning algorithms. Third, it is possible to use graph theory symbols to formally represent deployment planner inputs and to prove the correctness of designed deployment planning algorithms.

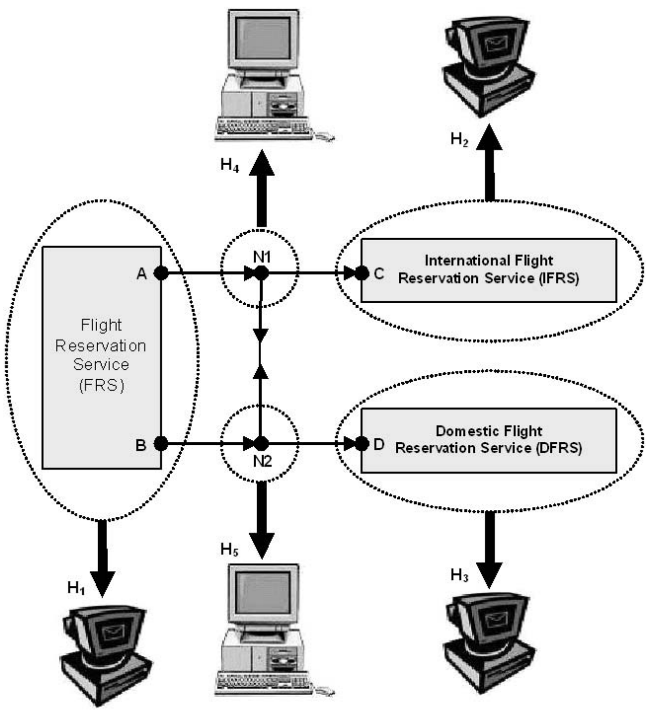

H4
H2
A
N1
C
International Flight Reservation Service (IFRS)
Flight Reservation Service (FRS)
B
N2
D
Domestic Flight Reservation Service (DFRS)
H5
H1
H3

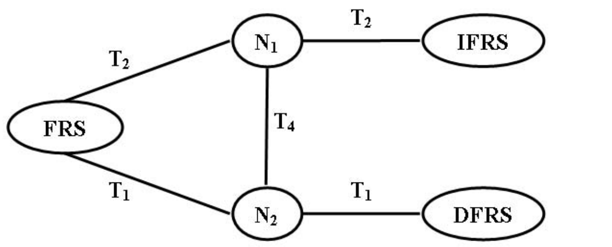
Figure 4. A sample deployment for the flight reservation system

4.1.1 Modeling the Application Being Deployed

In section 3.1, we mentioned that loosely coupled, component-based applications can be viewed as a number of components connected by a number of channels with different characteristics through which they communicate. With respect to this description of component-based applications, it is possible to model any loosely coupled, component-based application as a graph whose nodes are application components and its edges are channels among these components.

Definition 4.1 ***(Application Graph)*** *Suppose C_is represent different components of the application, and T_ds represent different channel types. Then, application graph* $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$ *is defined as a graph on* $V_{\mathrm{AG}} = \{C_1, C_2, ..., C_n\}$ *in which each edge* $e \in E_{\mathrm{AG}}$ *has a label* $l_e \in \{T_1, T_2, ..., T_k\}$.

For example, Fig. 5 shows the application graph for the flight reservation system. This graph is built with respect to both the specifications of the application being deployed, and user-defined constraints regarding this deployment. For example, in the specification of the application (Fig. 2), *Sync* channels are used to connect FRS and IFRS components. But, as mentioned



Channel Types: T_1=Sync, T_2=Encrypted Sync, T_4=SyncDrain

Figure 5. Application graph for the flight reservation system

in section 3.3, users want the transfer of data between FRS and IFRS to be encrypted. Thus, in the application graph presented in Fig. 5, *Encrypted Sync* channel type is used between FRS and IFRS components.

4.1.2 Modeling the Target Environment

As mentioned in section 3.2, in this paper the target environment for the deployment of the application is a number of hosts with different computational capabilities connected by a network in a distributed environment and each of them can support a set of channel types. With respect to this description of the target environment, it is possible to model the target environment with the help of a graph in which:

- Nodes represent available hosts in the distributed environment;
- Edges represent different channel types that can exist between every two hosts.

To generate such a graph, first it is required to notice to the following definitions.

Definition 4.2 ***(Adjacent Hosts)*** *Two distinct hosts H_x and H_y are adjacent if there is a direct physical link between them in the distributed environment.*

As an example, hosts H_1 and H_4 in Fig. 3 are adjacent.

Definition 4.3 ***(Virtually Connected)*** *Two distinct hosts H_x and H_y are virtually connected if there is not any direct physical link between them in the distributed environment. But, they are connected indirectly through intermediate hosts.*

As an example, hosts H_1 and H_2 in Fig. 3 are virtually connected.

Definition 4.4 ***(Transitive Channel Type)*** *Suppose two hosts H_x and H_y are virtually connected. A channel type T_d is transitive if it is possible to create a channel of type T_d between them when (1) both of them can support channel type T_d, and (2) all intermediate hosts between them can also support channel type T_d.*

For example, in the Reo coordination model, channel type *Sync* is a transitive channel type.

Definition 4.5 ***(Non-transitive Channel Type)*** *A channel type T_d is non-transitive if it is possible to create a channel of type T_d between two hosts H_x and H_y only when (1) both of them can support channel type T_d, and (2) they are adjacent.*

As an example, in the Reo coordination model, channel type *SyncDrain* is a non-transitive channel type.

With respect to the above definitions, target environment graph is defined in the following way:

Definition 4.6 ***(Target Environment Graph)*** *Suppose H_is represent different hosts in the target environment, T_ds represent different channel types, and e_{H_x,H_y,T_d} represents an edge from node H_x to node H_y with label T_d. Then, the target environment graph* $\mathrm{TG} = (V_{\mathrm{TG}}, E_{\mathrm{TG}})$ *is defined as a graph on $V_{\mathrm{TG}} = \{H_1, H_2, ..., H_m\}$ in which the set of edges $E_{\mathrm{TG}} = \bigcup\{e_{H_x,H_y,T_d}\}$ is determined in the following way:*

- *If T_d is a transitive channel type, then there exists an edge e_{H_x,H_y,T_d} between two distinct nodes H_x and H_y only if (1) both of them are adjacent or virtually connected, (2) both of them support channel type T_d, and (3) if they are virtually connected, all intermediate hosts support channel type T_d.*
- *If T_d is a non-transitive channel type, then there exists an edge e_{H_x,H_y,T_d} between two distinct nodes H_x and H_y only if (1) they are adjacent, (2) both of them support channel type T_d.*
- *If T_d can be supported by host H_x, then there is an edge e_{H_x,H_x,T_d} from H_x to H_x (loopback edge).*

As an example, Fig. 6 shows the target environment graph generated by this method for the distributed environment presented in Fig. 3. To make the figure simpler, loopback edges are not shown. For a more specific example, consider hosts H_1 and H_2 which are virtually connected (i.e., through host H_4). As mentioned in section 3.2, in this example, $T_1 - T_3$ are different implementations of the *Sync* channel type which

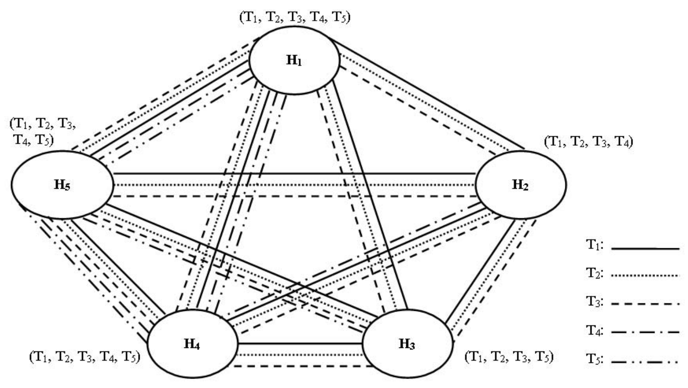


Figure 6. Target environment graph for the distributed environment presented in Fig. 3. $\mathbf{T_1 - T_3}$ are transitive channel types. $\mathbf{T_4 - T_5}$ are non-transitive channel types. For simplicity, loopback edges are not shown.

is a transitive channel type. Thus, it is possible to have channels of types $T_1 - T_3$ between H_1 and H_2. Furthermore, both H_1 and H_2 support channel type T_4 (i.e., *SyncDrain*) which is a non-transitive channel type. However, since H_1 and H_2 are not adjacent, it is impossible to have a channel of type T_4 between them.

4.1.3 Target Environment Graph for a Peer-to-Peer Distributed Environment

In a peer-to-peer (P2P) distributed environment (e.g., Internet), two or more computers (called nodes) can directly communicate with each other, without the need for any intermediary devices [9]. In this situation, it is not required to consider the issues related to the physical connectivity among hosts, i.e., transitive property of channel types. In this case, the definition of the target environment graph becomes much simpler.

Definition 4.7 *The target environment graph* $\mathrm{TG} = (V_{\mathrm{TG}}, E_{\mathrm{TG}})$ *for a P2P distributed environment is a graph on $V_{\mathrm{TG}} = \{H_1, H_2, ..., H_m\}$ in which there exists an edge e_{H_x,H_y,T_d} between two not necessarily distinct nodes H_x and H_y if and only if both of them can support channel type T_d.*

4.2 Deployment Planning Algorithms

As mentioned at the beginning of section 4, during the deployment planning, different application components and channels among them are mapped to different hosts in the target environment and network links

Component Name	Candidate Hosts
FRS	H_1
IFRS	H_2, H_3
DFRS	H_1, H_2, H_3, H_4, H_5
N_1	H_1, H_2, H_4, H_5
N_2	H_1, H_2, H_4, H_5

Table 1. Candidate hosts for the deployment of the flight reservation system components

among them so that all requirements and constraints are satisfied. If consider the sample deployment presented in Fig. 4 again, you may notice that in this deployment, different nodes and edges of the application graph AG shown in Fig. 5 are mapped to different nodes and edges of the target environment graph TG presented in Fig. 6. In this way, it is possible to see the deployment planning as a graph mapping problem from the application graph to the target environment graph. In this section, we talk about the required algorithms to solve this graph mapping problem. However, before everything, we begin with defining some general terms which are used in the rest of this paper.

Definition 4.8 ***(Candidate Host)*** *Let* $T_{C_i} = \{T_d | T_d \in T, \exists\{C_i, C_j\} \in E_{\mathrm{AG}} : l_{\{C_i,C_j\}} = T_d\}$ *represent all required channel types by component* C_i *in the application graph* $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$ *and let* $T_{H_x} = support(H_x)$ *represent the set of channel types that host* H_x *can support. Then, host* H_x *is a candidate host for the deployment of component* C_i*, only if (1)* $T_{C_i} \subseteq T_{H_x}$*, and (2) host* H_x *satisfies user-defined constraints regarding the deployment of component* C_i*.*

This definition implies that a host H_x is a candidate host for the deployment of component C_i if it supports all required channel types by component C_i in the application graph and also the deployment of component C_i on host H_x meets user-defined constraints. As an example, Table 1 shows the candidate hosts for the deployment of the flight reservation system components. For a more specific example, consider component IFRS. In the application graph presented in Fig. 5, IFRS just requires channel type T_2 and all of the hosts in the target environment presented in Fig. 3 support this channel type. But, as mentioned in section 3.3, users want IFRS to be deployed on either hosts H_2 or H_3. So, with respect to this constraint, candidate hosts for the deployment of component IFRS are H_2 and H_3.

Definition 4.9 ***(Candidate Deployment)*** *Suppose* CH_{C_i} *represents the set of candidate hosts for the deployment of component* C_i*. Then, a candidate deployment* D_c *is a set of pairs* (C_i, H_x) *in which every component* C_i *in the application graph* $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$ *is mapped to a host* H_x *in the target environment graph* $\mathrm{TG} = (V_{\mathrm{TG}}, E_{\mathrm{TG}})$ *so that host* H_x *is a candidate host for the deployment of component* C_i*, i.e.,* $D_c = \{(C_i, H_x) | C_i \in V_{\mathrm{AG}}, H_x \in V_{\mathrm{TG}}, H_x \in \mathrm{CH}_{C_i}\}$*.*

For example, $\{(\mathrm{FRS} \mapsto H_1), (\mathrm{IFRS} \mapsto H_2), (\mathrm{DFRS} \mapsto H_3), (\mathrm{N}_1 \mapsto H_4), (\mathrm{N}_2 \mapsto H_5)\}$ and $\{(\mathrm{FRS} \mapsto H_1), (\mathrm{IFRS} \mapsto H_3), (\mathrm{DFRS} \mapsto H_3), (\mathrm{N}_1 \mapsto H_4), (\mathrm{N}_2 \mapsto H_5)\}$ are two candidate deployments for the flight reservation system.

Definition 4.10 ***(Valid Deployment)*** *A candidate deployment* D_c *is a valid deployment, if for all edges* e_{C_i,C_j,T_d} *in the application graph* $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$ *if components* C_i *and* C_j *are mapped to two not necessarily distinct hosts* H_x *and* H_y *in the target environment, then it should be possible to create a channel of type* T_d *between hosts* H_x *and* H_y*, i.e., there should be an edge* e_{H_x,H_y,T_d} *in the target environment graph* $\mathrm{TG} = (V_{\mathrm{TG}}, E_{\mathrm{TG}})$*. Formally speaking,* $\forall e_{C_i,C_j,T_d} \in E_{\mathrm{AG}} \Rightarrow \exists e_{D_c(C_i),D_c(C_j),T_d} \in E_{\mathrm{TG}}$*.*

As an example, $D_c = \{(\mathrm{FRS} \mapsto H_1), (\mathrm{IFRS} \mapsto H_2), (\mathrm{DFRS} \mapsto H_1), (\mathrm{N}_1 \mapsto H_1), (\mathrm{N}_2 \mapsto H_2)\}$ is an invalid deployment for the flight reservation system. Because, there is an edge e_{N_1,N_2,T_4} in the application graph presented in Fig. 5. But, there is not an edge $e_{D_c(N_1),D_c(N_2),T_4} = e_{H_1,H_2,T_4}$ in the target environment graph presented in Fig. 6. In other words, with respect to the specification of the target environment presented in Fig. 3, it is impossible to create a channel of type T_4 between hosts H_1 and H_2.

With respect to above definitions, it is typically possible to deploy a complex component-based application into a large distributed environment in many different ways. As an example, consider again the candidate hosts for deploying each of the components of the flight reservation system shown in Table 1. As can be understood from this table, it is possible to deploy this application into the target environment in at most $160 = 1 \times 2 \times 5 \times 4 \times 4$ different ways (because some of them are invalid deployments). Obviously, this number is much bigger for complex applications deployments. However, when some QoS parameters, such as cost, performance, reliability, etc., are considered, some of these candidate deployments are equivalent, some are better than others and only a few of them may accommodate the constraints and requirements of the application. Thus, when QoS of the application is important, it should be tried to deploy the application so that its desired QoS parameter is maximized.

One naive solution to this problem is to generate all candidate deployments by permuting the sets of candidate hosts for different components of the application. Then, the desired QoS parameter of all valid candidate deployments is measured and the best one is selected. The complexity of this algorithm is $O(mn+m^n) = O(m^n)$, where m is the number of available hosts in the target environment and n is the number of components of the application. As we see, this is an exponentially complex solution to the deployment problem. Thus, when the number of candidate deployments is large, it is impractical to generate all of them and then select the best one. So, a set of algorithms and heuristics should be designed and applied to effectively solve such an exponentially complex problem. The following definition, provides a formal definition of the deployment problem we intend to solve.

Definition 4.11 ***(Deployment Problem)*** *Suppose deployment planner inputs are used to build the application graph and the target environment graph according to the methods presented in section 4.1.* CH_{C_i} *also represents the set of candidate hosts for the deployment of component C_i. Then, for the given application graph* $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$*, target environment graph* $\mathrm{TG} = (V_{\mathrm{TG}}, E_{\mathrm{TG}})$*, and QoS parameter Q, the problem is to find a polynomial time function $D : V_{\mathrm{AG}} \rightarrow V_{\mathrm{TG}}$ such that the following three conditions are satisfied:*

1. *Application's Q parameter is maximized;*
2. $D(C_i) = H_x \Rightarrow H_x \in \mathrm{CH}(C_i)$*. This means that all components of the application must be mapped to one of their respective candidate hosts for the deployment;*
3. $\forall e_{C_i,C_j,T_d} \in E_{\mathrm{AG}} \Rightarrow \exists e_{D(C_i),D(C_j),T_d} \in E_{\mathrm{TG}}$*. This means that the deployment D must be a valid deployment.*

This definition implies that during the deployment, it is possible to map several application components to a single host if that host is a candidate host for the deployment of those components. Furthermore, if there exists a channel of type T_d between two components in the application graph, then those components can be mapped to two different hosts only if there exists a channel of type T_d between them in the target environment graph.

As an example of how such efficient algorithms and techniques can be applied to effectively solve the deployment problem, in the following section, polynomial time algorithms for minimizing the cost of a deployment when the target environment is a P2P distributed environment are provided.

```
for each component $C_i$ in the application do
    Find the set of candidate hosts, $\mathrm{CH}_{C_i}$;
    if $\mathrm{CH}_{C_i}$ == null then
        return "No Answer!";
    end
    else
        $H_x$ = cheapest host in the set $\mathrm{CH}_{C_i}$;
        Output: $C_i \mapsto H_x$
    end
end
```

Figure 7: Cost-effective deployment algorithm when the cost should be paid for each component

4.2.1 Cost-effective Deployment

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, one QoS parameter of a deployment is its cost and should be minimized in the deployment plan. For this, two different cases can be considered:
Case 1: The cost should be paid for each component. In this case, for every component to be run on each host, its cost should be paid separately. For example, for each component to be run on host H_1, \$1000 should be paid to its administrator(s). Thus, if five components to be run on host H_1, $5\times\$1000 = \$5000$ should be paid. The required algorithm of this case is simple. In this case, in the set of candidate hosts for the deployment of each of the application components, the cheapest one is selected and that component is deployed on it. The pseudocode of this algorithm is shown in Fig. 7. This algorithm has the polynomial complexity $O(mn)$.
Case 2: The cost should be paid for each host, no matter how many components will be run on it. In this case, the number of components will be run on each host is not important; if the cost of one host is paid, it is possible to run as many components as you want on it. The complexity of this case is much more than the previous one. In this case, it should be tried to select a subset of available hosts in the target environment so that the total cost of the deployment is minimized and all the components of the application are also assigned to a host. It is easily possible to prove that this problem is equivalent to the *Minimum Set Cover* problem [10].

Definition 4.12 ***(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 $c : S \longrightarrow R$, the problem is to find a minimum cost subset of S that covers all elements of U.*

```
$X = \emptyset, \tau = \emptyset$;
while $X \neq U$ do
    Find the set $\omega \in S$ that minimizes
    $c(\omega)/|\omega \backslash X|$;
    $X = X \cup \omega, \tau = \tau \cup \{\omega\}$;
end
Output: $\tau$
```

Figure 8: Greedy approximation algorithm for the minimum set cover problem

This case of the cost-effective deployment problem can be converted to a 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_x} corresponds to host H_x and it represents the subset of application components that can be run on host H_x. In other words, each CS_{H_x} is a subset of application components which H_x is in their lists of candidate hosts for the deployment.
- Define $c : S \longrightarrow R$ so that $c(CS_{H_x}) = c'(H_x)$. Function $c' : H \longrightarrow R$ returns the cost of each host.

Theorem 4.1 *If we define the elements of the minimum set cover problem as mentioned earlier, then the solution of the minimum set cover problem satisfies all conditions of the deployment problem defined in definition 4.11.*

To save space, the proof of this theorem is not provided here. However, it is proved that minimum set cover problem is a NP-hard problem and it can not be solved in polynomial time [11]. But, there exist some greedy approximation algorithms that can find reasonably good answers in polynomial time. One of the key algorithms for solving this problem is provided in Fig. 8 [11]. The main idea in this algorithm is to iteratively select the most cost-effective $s_i \in S$ and remove the covered elements until all elements are covered. The complexity of this algorithm is $O(log(|U|))$ [11].

To solve this case of the cost-effective deployment problem, first it should be converted to the minimum set cover problem as mentioned earlier. Then, it is easily possible to use the greedy approximation algorithm presented in Fig. 8 to find a reasonably good solution for the problem. In other words, by using this algorithm, all components of the application will be assigned to at least one host and total cost of the deployment will be close to minimum too. As an example of using this greedy approximation algorithm, consider the flight reservation system example. With respect to Table 1, the elements of the minimum set cover problem are defined in the following way:

- $U = \{\text{FRS}, \text{IFRS}, \text{DFRS}, \text{N}_1, \text{N}_2\}$;
- $S = \{\{\text{FRS}, \text{DFRS}, \text{N}_1, \text{N}_2\}, \{\text{IFRS}, \text{DFRS}, \text{N}_1, \text{N}_2\}, \{\text{IFRS}, \text{DFRS}\}, \{\text{DFRS}, \text{N}_1, \text{N}_2\}, \{\text{DFRS}, \text{N}_1, \text{N}_2\}\}$;
- $c'(H_1) = \$1000$, $c'(H_2) = \$2500$, $c'(H_3) = \$2000$, $c'(H_4) = \$1500$, $c'(H_5) = \$1000$.

By applying the greedy approximation algorithm, we will have the following results and the minimum cost will be $3000:

- $\{(\text{FRS} \mapsto H_1), (\text{DFRS} \mapsto H_1), (\text{IFRS} \mapsto H_3), (\text{N}_1 \mapsto H_1), (\text{N}_2 \mapsto H_1)\}$;
- $\{(\text{FRS} \mapsto H_1), (\text{DFRS} \mapsto H_3), (\text{IFRS} \mapsto H_3), (\text{N}_1 \mapsto H_1), (\text{N}_2 \mapsto H_1)\}$.

Note that it is possible to use the algorithm presented here more generally for some other QoS parameters too, when you want to minimize the total usage of some resources of available hosts in the target environment. In this situation, it is possible to define the cost function c to return the amount of that resource for each host and then use the greedy approximation algorithm presented in Fig. 8 to find the solution.

5 Conclusions and Future Work

The software deployment process is defined as a sequence of related activities for placing a developed application into its target environment and making the application available for use. For simple stand-alone applications that should be installed only on a single computer, this process is easy. But, for complex component-based applications that should be deployed into a large distributed environment and some QoS parameters should also be maximized, the deployment process is not that straightforward. This paper presented a graph-based approach for this deployment planning which uses the concept of channels to capture the properties of interconnections among the components of the application. The approach presented in this paper is general and is not dependent on any specific component technology or model (e.g., COM, CORBA, EJB, etc.) and can be used for deploying any kind of loosely coupled, component-based applications into distributed environments.

This paper also presented the required algorithms for minimizing the cost of a deployment when some costs must be paid upon using the hosts in the target environment. For future work, we plan to design efficient algorithms for other QoS parameters such as reliability, performance, security, and so on. We also plan to devise some specification languages for specifying the application being deployed, the target environment, and user-defined constraints.

References

[1] Hnetynka, P. Making Deployment of Distributed Component-based Software Unified. In *Proceedings of CSSE 2004 (Part of ASE 2004)*, Austrian Computer Society, Linz, Austria, Sep. 2004, 157-161.

[2] Lestideau, V. and Belkhatir, N. Providing Highly Automated and Generic Means for Software Deployment Process. In *Proceedings of the 9th International Workshop on Software Process Technology (EWSPT 2003)*, Helsinki, Finland, September 1-2, 2003, 128-142.

[3] 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 Second International Working Conference on Component Deployment (CD 2004)*, Edinburgh, UK, May 20-21, 2004.

[4] 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.

[5] Object Management Group, *Deployment and Configuration of Component-based Distributed Applications Specification.* http://www.omg.org/docs/ptc/04-05-15.pdf.

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

[7] 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.

[8] Web Services Conceptual Architecture. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[9] Schollmeier, R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Proceedings of the IEEE 2001 International Conference on Peer-to-Peer Computing (P2P2001)*, Linkping, Sweden, August 27-29, 2001.

[10] Hassin, R. and Levin, A. A Better-Than-Greedy Approximation Algorithm For The Minimum Set Cover Problem. *SIAM Journal on Computing*, 35, 1 (2005), 189-200.

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