A Graph-based Approach for Deploying Component-based Applications into Channel-based Distributed Environments

Abbas Heydarnoori and Walter Binder
Faculty of Informatics, University of Lugano, Lugano, Switzerland
Email: firstname.lastname@usi.ch

Abstract— **With significant advances in software development technologies, 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 user requirements and constraints are also met. This process is called *software deployment*. However, this process is often challenging for large, distributed, component-based applications with many constraints and requirements. This article presents a graph-based deployment approach that does the deployment 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. This article further discusses how this mapping could be done to minimize the cost and to maximize the reliability of deployments.**

Index Terms— **Component-based applications, software deployment, distributed environments, communication channels, reliability.**

I. 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 many computers, and large networks have moved to the center of software applications. In these applications, since different components provide their functionality only when their constraints and requirements are met, they should be installed on proper hardware resources in the distributed environment in order for them to provide the expected quality of service (QoS). In addition, different resources have different computational capabilities, making it impossible to install any kind of software components on them. The goal of the *software deployment process* [3], [4] is to place an already developed application into its target environment and bring it into an executing state. For simple stand-alone software systems that should be deployed only to a single computer, deployment activities can be easily done. However, 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.

This article presents a graph-based approach that does the deployment with respect to the communication resources required by application components and communication resources available on the hosts in the target environment. For this purpose, the concept of *channel* is used to model the 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. Deployment planning is then defined as the mapping of the application graph to the target environment graph so that the desired QoS parameter is maximized. As examples of this approach, we show how this mapping can be effectively done for *minimum cost* and *maximum reliability* of deployments.

This article is organized as follows: Section II presents the Reo coordination model as an example of channel-based coordination models and provides a running example which is used throughout this article. The required inputs for our deployment planning process are discussed in Section III. These inputs are then modeled as graphs in Section IV. Next, Section V introduces the problem of software deployment as a graph mapping problem. Section VI presents algorithms for solving this mapping problem for the QoS parameters cost and reliability. Finally, Section VII provides an overview of related work and Section VIII concludes.

II. Reo Coordination Model

Reo is a channel-based coordination model that exogenously coordinates the cooperative behavior of component instances in a component-based application [5]. From

This article is based on the articles "Deploying Loosely Coupled, Component-based Applications into Distributed Environments" presented in ECBS'06 [1], and "Reliable Deployment of Component-based Applications into Distributed Environments" presented in ITNG'06 [2].

doi:10.4304/jsw.6.8.1381-1394

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 [5] 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, Figure 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 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 [6].

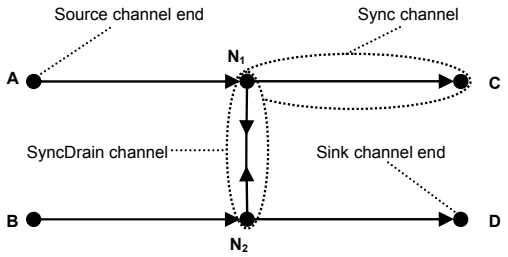

Figure 1. Barrier synchronization connector in Reo

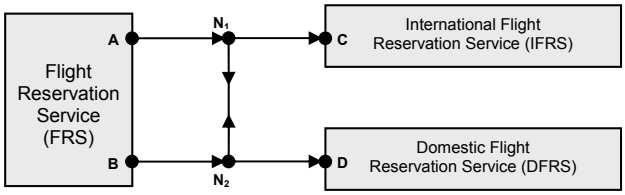

Figure 2. Modeling a flight reservation system with Reo

A. 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 article. In this example, the barrier synchronization connector in Reo is used to compose a number of Web services. Web services refer to accessing services over the Web [7]. 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 (Figure 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.

III. 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.

A. 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 [5].

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

B. Specification of the Target Environment

In this article, 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 (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 article, communication resources available on different hosts are different channel types (or implementations) they can support. As an example, Figure 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):

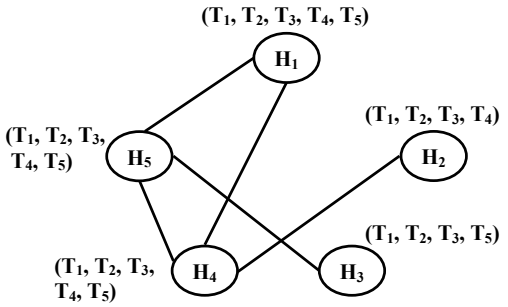

Figure 3. A sample target distributed environment for the deployment of the flight reservation system

- 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.

C. 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.

IV. 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 possible to develop graph representations of these inputs. This graph-based modeling makes it feasible to apply graph theory algorithms in designing deployment planning algorithms.

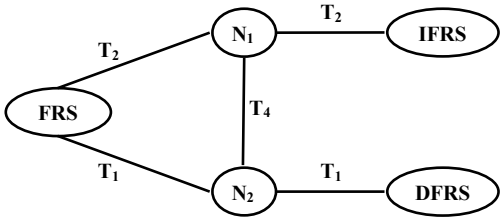

Figure 4. Application graph for the flight reservation system

A. Modeling the Application

In Section III-A, 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 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, Figure 4 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 (Figure 2), $Sync$ channels are used to connect FRS and IFRS components. Nonetheless, as mentioned in Section III-C, users want the transfer of data between FRS and IFRS to be encrypted. Thus, in the application graph presented in Figure 4, *Encrypted Sync* channel type is used between FRS and IFRS components.

B. Modeling the Target Environment

As mentioned in Section III-B, in this article 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 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 Figure 3 are adjacent.

Definition 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 Figure 3 are virtually connected.

Definition 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 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 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, Figure 5 shows the target environment graph generated by this method for the distributed environment presented in Figure 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 III-B, in this example, T_1–T_3 are different implementations of the $Sync$ channel type which 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., $SyncDrain$)

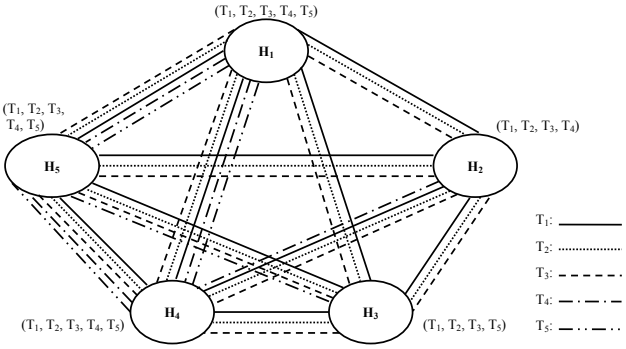

Figure 5. Target environment graph for the distributed environment presented in Figure 3. T_1–T_3 are transitive channel types. T_4–T_5 are non-transitive channel types. For simplicity, loopback edges are not shown.

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.

C. 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 [8]. 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 7 (P2P Target Environment Graph): 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.

V. Deployment Problem

After specifying the deployment planner inputs, they can be used to generate the actual deployment plan. Figure 6 shows a sample deployment for the flight reservation system. As can be seen in this figure, for the purpose of this deployment, different components of the application and channels among them are mapped to different hosts in the target environment and network links among them in such a way that all requirements and constraints are satisfied. More specifically, it could be observed that different nodes and edges of the application graph AG shown in Figure 4 are mapped to different nodes and edges of the target environment graph TG presented in Figure 5. In this way, we can define the deployment problem as a graph mapping problem from the application graph to the target environment graph. To this end, we begin with defining some terms.

Definition 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.

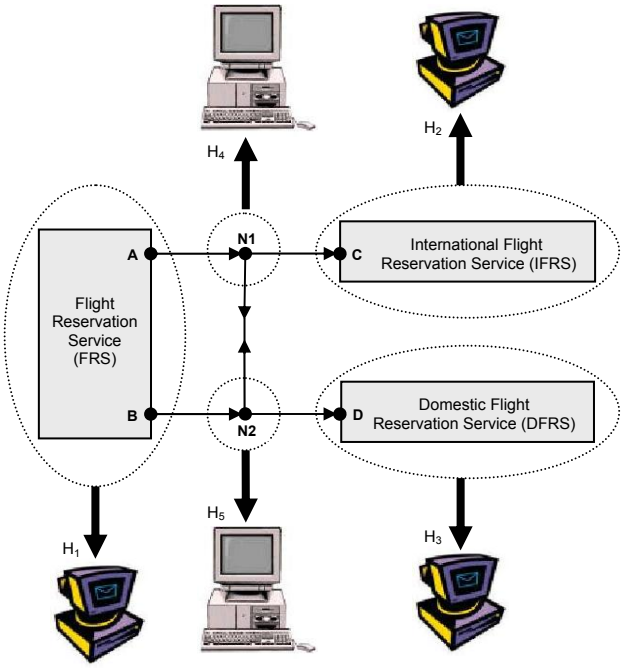

Figure 6. A sample deployment for the flight reservation system

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 I 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 Figure 4, IFRS just requires channel type T_2 and all of the hosts in the target environment presented in Figure 3 support this channel type. However, as mentioned in Section III-C, 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 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 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 Figure 4. Nonetheless, 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 Figure 5. In other words, with respect to the specification of the target environment presented in Figure 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 I. 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 11 (Deployment Problem): Suppose

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 I.
CANDIDATE HOSTS FOR THE DEPLOYMENT OF THE FLIGHT RESERVATION SYSTEM COMPONENTS

deployment planner inputs are used to build the application graph and the target environment graph according to the methods presented in Section IV. 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.

VI. DEPLOYMENT ALGORITHMS

Section V introduced the problem of component deployment as a graph mapping problem in which the application graph is mapped to the target environment graph. This section presents polynomial time algorithms for solving this mapping problem when the target environment is a P2P distributed environment and the desired QoS parameters are minimum cost [1] and maximum reliability [2] of deployments.

A. 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. To this end, 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

```
for each component C_i in the application do
    Find the set of candidate hosts, CH_{C_i}
    if CH_{C_i} == null then
        return "No Answer!"
    end
    else
        H_x = cheapest host in the set CH_{C_i}
        Output: C_i ↦ H_x
    end
end
```

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

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 Figure 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 see that this problem is equivalent to the *Minimum Set Cover* [9] problem in graph theory.

Definition 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.

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 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 11.

```
X = ∅, τ = ∅
while X ≠ U do
    Find the set ω ∈ S that minimizes c(ω)/|ω\X|
    X = X ∪ ω, τ = τ ∪ {ω}
end
Output: τ
```

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

It is proved that minimum set cover problem is a NP-hard problem and it can not be solved in polynomial time [10]. Nevertheless, 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 Figure 8 [10]. 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 $O(log(|U|))$ [10].

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 Figure 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 I, the elements of the minimum set cover problem are defined in the following way:

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

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

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

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 Figure 8 to find the solution.

B. Reliable Deployment

One of the most important QoS parameters of a software system is its reliability, defined as the probability of failure-free software operation for a specified period of

time in a specified environment [11]. In the context of distributed environments, one potential problem is network failures. In these environments, connectivity losses can lead to disastrous effects on the system's reliability, and the software application may not provide its desired functionality. To reduce the risks of this problem, one solution is to make the communications among the application components as local as possible. In this way, components located in the same host can communicate regardless of the network's status. Thus, from this perspective, the most reliable deployment configuration can be defined as one with the least amount of communications among the hosts in the distributed environment. From another point of view, this can be also seen as the increased performance. This is due to the fact that in distributed environments, network communications have some overheads on software applications. Consequently, reduced communication among hosts can result in a higher performance.

One impractical way for finding the most reliable deployment configuration is to generate all possible deployment configurations by permuting the sets of candidate hosts for different components of the application. Then, the deployment configuration with the greatest number of local channels among the application components (or the least number of channels among the hosts) is selected. However, when the number of possible deployment configurations is large, this approach will not work effectively. To solve this problem in polynomial time, we show that the reliable deployment problem corresponds to the multiway cut problem in graph theory [12].

Definition 13 (Multiway Cut Problem): Let $G = (V, E)$ be an undirected graph on $V = \{v_1, v_2, ..., v_n\}$ in which each edge $e \in E$ has a non-negative weight $w(e)$, and let $T = \{t_1, t_2, ..., t_m\} \subseteq V$ be a set of terminals. Multiway cut is the problem of finding a set of edges $E' \subseteq E$ such that the removal of E' from E disconnects each terminal from all other terminals, and solution cost $\mathrm{MC} = \sum_{e \in E'} w(e)$ is also minimized.

Suppose $\mathrm{AG} = (V_{\mathrm{AG}}, E_{\mathrm{AG}})$ is the application graph of the software application being deployed, $V_{\mathrm{TG}} = \{H_1, H_2, ..., H_m\}$ represents the set of available hosts in the target environment, and CH_{C_i} represents the set of candidate hosts for the deployment of component C_i. To solve the reliable deployment problem, a graph $G = (V, E)$ is made in the following way:

- $V = V_{\mathrm{AG}} \cup V_{\mathrm{TG}}$.
- $E = E_{\mathrm{AG}} \cup E_H$, where $E_H = \{\{C_i, H_x\} | C_i \in V_{\mathrm{AG}}, H_x \in \mathrm{CH}_{C_i}\}$.
- $w(e) = \begin{cases} 1 & e \in E_{\mathrm{AG}} \\ n^2 & e \in E_H \end{cases}$. Here n^2 shows a large number.

Figure 9 shows an example of a graph developed in this way for the application graph presented in Figure 4, and the target environment presented in Figure 3. In this graph, if we set hosts as the terminals of the multiway cut problem, we prove in the following theorem that the solution of the multiway cut problem is the solution of the reliable deployment problem we intend to solve.

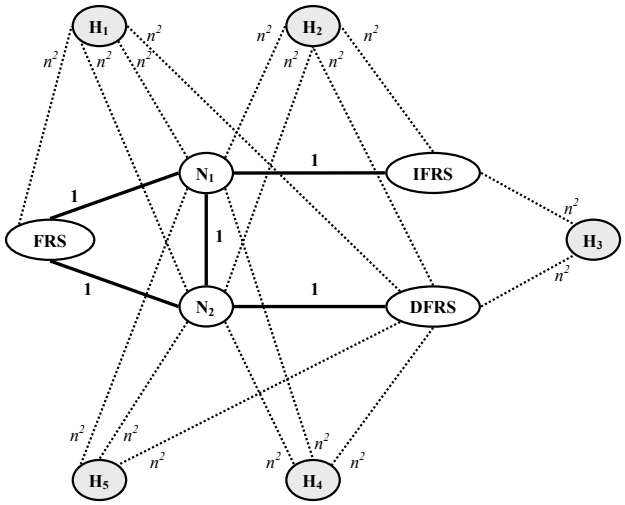

Figure 9. A graph built for finding the most reliable deployment configuration of the application presented in Figure 4 into the target environment shown in Figure 3.

Theorem 2: Suppose graph $G = (V, E)$ is built in the way mentioned earlier, and hosts of the target environment are set as the terminals. Then, the multiway cut solution of this graph is the solution of the reliable deployment problem we are looking for. This means that the application components that lie in the same subgraph with a host should be deployed on that host, and this deployment configuration has the least number of channels among hosts.

Proof: Suppose n represents the number of components of the application, and k represents the size of the E_H, i.e., $k = |E_H|$. In the multiway cut solution we are looking for, each component must be assigned to exactly one host. Thus, $(k - n)$ edges whose total weight is $n^2(k - n)$ will be removed from the E_H in the cut. Also, suppose L_{OPT} represents the solution of the reliable deployment problem, i.e., the least number of channels among hosts after the deployment of the application. Actually, these channels are those application graph edges that lie in the cut, and their total weight is $L_{\mathrm{OPT}} \times 1 = L_{\mathrm{OPT}}$. Thus, our goal is to prove that $\mathrm{MC} = n^2(k - n) + L_{\mathrm{OPT}}$.

Case A: $\mathbf{MC \leq n^2(k - n) + L_{OPT}}$. Suppose a deployment $D : V_{\mathrm{AG}} \longrightarrow V_{\mathrm{TG}}$ whose cost is optimum is done, i.e., it has L_{OPT} number of channels among hosts. Now, assume that $\mathcal{C}$ is its corresponding cut in the graph $G = (V, E)$:

$$\mathcal{C} = \underbrace{\{\{C_i, H_x\} | D(C_i) \neq H_x, \{C_i, H_x\} \in E_H\}}_{\mathcal{M}} \bigcup \underbrace{\{\{C_i, C_j\} | D(C_i) \neq D(C_j), \{C_i, C_j\} \in E_{\mathrm{AG}}\}}_{\mathcal{N}}$$

$\mathcal{M}$ represents the set of edges of E_H that lie in the cut, and $\mathcal{N}$ represents the set of edges of E_{AG} that lie in the cut. The size of $\mathcal{M}$ is $(k - n)$ and the size of $\mathcal{N}$ is L_{OPT}. Furthermore, the weight of the edges in $\mathcal{M}$ is n^2

and the weight of the edges in $\mathcal{N}$ is 1. With respect to this description:

$$
\begin{array}{l}
w(\mathcal{C}) = \\
w(\{\{C_i, H_x\} | D(C_i) \neq H_x, \{C_i, H_x\} \in E_H\}) + \\
w(\{\{C_i, C_j\} | D(C_i) \neq D(C_j), \{C_i, C_j\} \in E_{\mathrm{AG}}\}) = \\
n^2 \times |\{\{C_i, H_x\} | D(C_i) \neq H_x, \{C_i, H_x\} \in E_H\}| + \\
1 \times |\{\{C_i, C_j\} | D(C_i) \neq D(C_j), \{C_i, C_j\} \in E_{\mathrm{AG}}\}| = \\
n^2(k-n) + L_{\mathrm{OPT}}
\end{array}
$$

Since MC is the cost of the optimum multiway cut, for sure, $\mathrm{MC} \leq w(\mathcal{C})$. Therefore, $\mathrm{MC} \leq n^2(k-n) + L_{\mathrm{OPT}}$.

Case B: $\mathbf{MC \geq n^2(k-n) + L_{OPT}}$. Suppose $\mathcal{C}$ is the optimum multiway cut for graph $G = (V, E)$ whose cost is MC. Now, we want to use this cut to generate its corresponding deployment D. For this purpose, we prove the following subcases:

B.1: Cut $\mathcal{C}$ includes at most $(\mathbf{k-n})$ edges of $\mathbf{E_H}$. Suppose we want to find a cut whose cost is the heaviest. In the deployment configuration we are looking for, each component should be assigned to exactly one host. For this purpose, for each component C_i in graph G, we keep an arbitrary edge connecting that component to an arbitrary host, and we cut the rest of the edges in E_H and E_{AG}. Since the maximum number of edges in the application graph is $\binom{n}{2}$, the cost of this cut is at most $\binom{n}{2} + n^2(k-n)$. Thus, the cost of the multiway cut $\mathcal{C}$ can not be more than $\frac{n^2}{2} + n^2(k-n)$. This means that the cut $\mathcal{C}$ includes at most $(k-n)$ edges of E_H. Because, for example, if it includes $(k-n+1)$ edges of E_H, then the cost of the cut would be $\binom{n}{2} + n^2(k-n+1)$ which is more than the maximum cost we found here.

B.2: Each component $\mathbf{C_i}$ is connected to at most one host in the cut $\mathcal{C}$. Suppose a component C_i is connected to two different hosts H_x and H_y in the cut. This means that H_x and H_y are connected together in the cut. However, since H_x and H_y belong to the set of terminals, this is impossible. Therefore, C_i is connected to at most one host in the cut.

B.3: Each component $\mathbf{C_i}$ is connected to exactly one host in the cut $\mathcal{C}$. From subcases B.1 and B.2 together, it can be easily understood that each component C_i is connected to exactly one host in the cut $\mathcal{C}$. $D(C_i)$ represents the host on which component C_i is mapped.

By using the subcase B.3, cut $\mathcal{C}$'s corresponding deployment configuration D can be made. Suppose $L_D = |\{\{C_i, C_j\} | D(C_i) \neq D(C_j), \{C_i, C_j\} \in E_{\mathrm{AG}}\}|$ represents the cost of the deployment configuration D, i.e., the number of channels among the hosts in the deployment configuration D. In the following, we prove the correctness of case B:

1) For each terminal $t_i \in T$, find a minimum-cost set of edges $\mathcal{C}_{t_i}$ whose removal disconnects t_i from the rest of the terminals
2) Discard cut $\mathcal{C}_{t_x}$ whose cost $w(\mathcal{C}_{t_x})$ is the heaviest
3) Output the union of the rest, call it $\mathcal{C}$.

Figure 10. Approximation algorithm for solving the multiway cut problem.

$$
\begin{array}{rcl}
\mathrm{MC} & = & n^2(k-n) + \\
 & & |\{\{C_i, C_j\} | \{C_i, C_j\} \in \mathcal{C}, \{C_i, C_j\} \in E_{\mathrm{AG}}\}| \\
 & \geq & n^2(k-n) + \\
 & & |\{\{C_i, C_j\} | D(C_i) \neq D(C_j), \{C_i, C_j\} \in E_{\mathrm{AG}}\}| \\
 & & = n^2(k-n) + L_D \\
\Longrightarrow & & \mathrm{MC} \geq n^2(k-n) + L_D \geq n^2(k-n) + L_{\mathrm{OPT}}
\end{array}
$$

Cases A and B together imply that $\mathrm{MC} = n^2(k-n) + L_{\mathrm{OPT}}$. Therefore, the correctness of Theorem 2 is proved. ■

In Theorem 2, we showed that the solution of the reliable deployment problem can be found by solving the multiway cut problem in graph theory. However, it is proved that the multiway cut problem is an NP-hard problem when the number of terminals is greater than two. Thus, unless P=NP, it does not have a polynomial time solution [12]. However, it is possible to find many approximation algorithms for the multiway cut problem in literature [12], [13], [14]. One of the well-known and simple approximation algorithms developed by Dahlhaus et al. is provided in Figure 10 [12]. As an example, Figure 11 shows an example of applying this algorithm on the graph presented in Figure 9. As we see in this figure, one of the main problems of these approximation algorithms is that some components may not be assigned to any hosts (e.g., FRS). To solve this problem, after applying the multiway cut approximation algorithm on the graph, we check whether or not all components are assigned to a host. If there are some components which are not assigned to any hosts, we connect those components to one of their candidate hosts for the deployment, and we cut all the application graph edges that are connected to those components. Since, we are actually removing from the multiway cut approximation some heavy edges that connect the components to the hosts, this approach not only will solve the problem, but also will improve the approximation of the multiway cut. Consequently, the result is closer to the optimum solution we are looking for. After applying this improvement on the multiway cut approximation presented in Figure 9, one possible solution for the reliable deployment problem is $\{(FRS \mapsto H_1), (IFRS \mapsto H_2), (DFRS \mapsto H_2), (N_1 \mapsto H_2), (N_2 \mapsto H_2)\}$.

VII. Related Work

An application can provide its expected functionality only when it is deployed and configured correctly in its

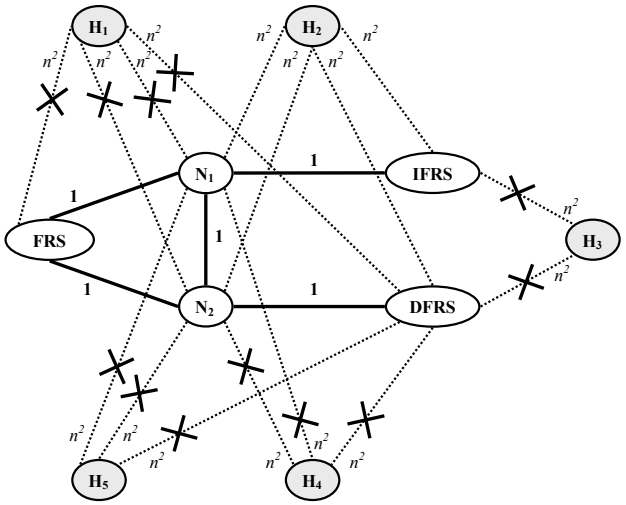

Figure 11. An approximation for the multiway cut of the graph presented in Figure 9.

operational environment. Consequently, The process of software deployment has been the subject of extensive research in recent years and a variety of commercial as well as research-based tools and techniques for the purpose of software deployment have been introduced. This section provides an overview of these tools and techniques and characterizes them in terms of their support for different activities of the deployment process as introduced in [3]: (i) *Release:* packages, prepares, and advertises a system for deployment into operating environments; (ii) *Acquire:* acquires the components of the application from the software producer and puts them in a repository; (iii) *Plan:* given the specifications of the component-based application, the target environment, and user-defined constraints regarding this deployment, this activity determines where and how different components of the application will be executed in the target environment, resulting in a deployment plan; (iv) *Install:* uses the deployment plan generated in the previous activity to install the application into its operating environment; (v) *Configure:* changes the configuration of an already installed software system; (vi) *Activate:* actually launches the application; (vii) *Update:* modifies a previously installed system and deploys a new, previously unavailable configuration of that system or updates the components of the system with newer releases of those components; (viii) *Deactivate:* shuts down executing components of an activated system; (ix) *Uninstall:* completely removes the software system from its operating environment; and (x) *Retire:* makes a system release unavailable.

A. Commercial Software Deployment Technologies

A variety of commercial technologies support different activities of the software deployment process. We classify these technologies into six main categories as shown in Table II and characterize them in terms of their support for the activities of the software deployment process. A '•' in Table II illustrates full support, a '○' indicates partial support, and no circle means no support.

User-driven installers are used to install and uninstall software systems from a single machine. Many installers may also support some sort of configuration by which users can add or remove some functionalities from the installed software systems. The limitations of these tools include: they are targeted to a single machine and it is typically impossible to use them for distributed platforms. Also, users themselves have to administer their software systems.

Package managers often come with modern operating systems to assist in installing, uninstalling, and updating software systems. The main goal of all package managers is to install software packages in such a way that the correct dependencies among them are preserved. However, they are also targeted to a single machine and do not support distributed systems or large scale deployments. Additionally, they are also user-driven.

Systems management tools provide a set of capabilities (e.g., tools, procedures, policies, etc.) that enable organizations to more easily support their hardware and software resources. Systems management tools often have a centralized architecture in which the IT administrator performs operations from a centralized location and they are automatically applied to many systems in the organization. Systems management tools support many of the software deployment activities in a distributed environment. However, the issues associated with them are that they are often heavy and complicated systems, they all require reliable networks, they are all based on complete administration control, and they are more suitable for medium to large organizations.

In *remote sessions*, software systems are deployed to a single server machine. Then, each client initiates a session on that server and invokes desired software systems on it. Therefore, these tools only support the execution activity of the deployment process. The advantages of these tools are that they reduce the inconsistencies in deployed clients when the functionality is extended and it is not required to deploy the same application to several machines. The disadvantages are server load, under-utilized client resources, and consumption of network bandwidth.

In *publish/subscribe tools*, users express their interests ("subscribe") in certain kinds of events on a server, such as installing a new application or updating installed applications. Then, whenever these events happen on that server, they will be applied ("publish") automatically to the subscribed machines. This method is an efficient approach for the distribution of a software system from a source machine to a large number of target machines over a network. Nevertheless, the limitation is that the users themselves have to subscribe for the deployment of applications. Furthermore, these tools might not be efficient in costly and low-bandwidth networks.

Web-based deployment tools try to use the connectivity and popularity characteristics of the Internet during the process of software deployment. In these tools, it is not required to install or update the software system on every

TABLE II.
COMPARISON OF DIFFERENT COMMERCIAL DEPLOYMENT TECHNOLOGIES IN TERMS OF THEIR SUPPORT FOR THE ACTIVITIES OF THE SOFTWARE DEPLOYMENT PROCESS

Deployment Technology	Representative Tools	Software Deployment Process									
		Release	Acquire	Plan	Install	Configure	Activate	Update	Deactivate	Uninstall	Retire
User-Driven Installers	InstallShield [15], InstallAnywhere [16], Setup Factory [17], Setup Builder [18]	•			•	○		○		•	
Package Managers	Linux RPM [19], Fedora yum [20], Debian Dpkg [21], Solaris pkg [22], Gentoo Portage [23]	•		○	•	○		•		•	
Systems Management Tools	Microsoft Systems Management Server [24], IBM Tivoli Management Environment [25], Altiris Deployment Solution [26], HP Open-View [27]			•	•	•	•	•	•	•	
Remote Sessions	Citrix [28], PowerTCP [29], SSh [30]						•				
Publish/Subscribe Tools	TIBCO Rendezvous [31], IBM Gryphon [32], Oracle Java Message Service [33]	•	•		•			•			
Web-based Deployment Tools	Java Web Start [34], Microsoft Windows Update [35], Microsoft ClickOnce [36]	•	•		•	○	•	•			

single host separately. Instead, the software application is deployed only to a single Web server. Then, client machines (users) connect to this server to download the application files or updates automatically. However, one of the major limitations of these tools is that they are useless when there is no Internet connectivity.

B. Research-based Software Deployment Techniques

The process of software deployment has been the subject of extensive research in the past few years. This section provides an overview of research-based deployment techniques and classifies them into eight major categories as illustrated in Table III. This table also compares these techniques in terms of their support for the activities of the software deployment process. Note that this categorization represents different directions of interest in research-based deployment techniques and the same technique may fall in two or more deployment categories.

As mentioned earlier in this article, it is often possible to deploy a component-based application into its operational environment in various ways, specifically when its operational environment is a distributed environment. Nevertheless, it is clear that some of these deployment configurations are better than others in terms of some QoS attributes such as efficiency, availability, reliability, and fault tolerance. *QoS-aware deployment* approaches [37], [54], [38], [4] aim to address this aspect of software deployment. This article also presented a QoS-aware deployment approach for improving the reliability and the cost of the deployments.

The software architecture research community has also addressed configuration and deployment issues for component-based applications [39], [40], [55], [56], [57]. A software architecture represents high-level abstractions for structure, behavior, and key properties of a software system. Software architectures are usually specified in terms of *components* that define computational units, *connectors* that define types of interactions among components, and the *configuration* that describes the topologies of components and connectors. For this purpose, *Architecture Description Languages* or *ADLs* have been developed to describe software systems in terms of their architectural elements. In deployment techniques that employ the concepts of software architecture, ADLs are used to specify valid deployment configurations. Then, during the process of deployment, candidate deployment configurations are checked against those valid deployment configurations.

Model-driven architecture (*MDA*) [58] is an approach for software development based on models in which systems are built via transformations of models. MDA defines two levels of models: *Platform-Independent Model* (*PIM*) and *Platform-Specific Model* (*PSM*). Developers begin with creating a PIM which is then transformed step by step to a more platform-specific model until the desired level of specificity is approached. In the case of software deployment, the MDA approach starts with a platform-independent model of the target environment and the transformations finish with specific deployment configurations for the considered component-based applications [42].

Agent-based deployment approaches, like *software dock* [44] and *TACOMA* [45], use mobile agents for the purpose of software deployment. A *mobile agent* is defined as an object that migrates through many hosts in a heterogeneous network, under its own control, to perform tasks using resources of those hosts [59].

Deployment of component-based applications into *computational grids* has been the subject of extensive research [46], [60], [47], [61], [62], [63], [63]. A computational grid is defined as a set of efficient computing resources that are managed by a middleware that gives transparent access to resources wherever they are located on the network [46]. A computational grid can include many heterogeneous resources (e.g., computational nodes with various architectures and operating systems), networks with different performance properties, storage resources of different sizes, and so on. To take advantage of the computational power of grids, the application

TABLE III.
COMPARISON OF DIFFERENT RESEARCH-BASED DEPLOYMENT TECHNIQUES IN TERMS OF THEIR SUPPORT FOR THE ACTIVITIES OF THE SOFTWARE DEPLOYMENT PROCESS

Deployment Approach	Representative Techniques	Software Deployment Process									
		Release	Acquire	Plan	Install	Configure	Activate	Update	Deactivate	Uninstall	Retire
QoS-Aware Deployment	DeSi [37]			●							
	MAL [38]			●	○	○		○			
	Caspian [4]			●							
Architecture-Driven Deployment	Prism-DE [39]		○	●	●		●	●			
	Olan [40]	●	●		●	●	●				
Model-Driven Deployment	OMG D&C [41]		●	●	●	●	●	○			
	Deployment Factory [42]		●	●	●	●	●	○			
	DAnCE [43]		●	●	●	●	●	○	●	○	
Agent-based Deployment	Software Dock [44]	●	●		●	●		●			
	TACOMA [45]	●	●		●			●			
Grid Deployment	Globus Toolkit [46]			●	●	●	●				
	ORYA [47]		●	○	●					●	
Hot Deployment	OpenRec [48]		●	○	●	●	●	●	●	●	
	MagicBeans [49]		●	○	●	●	●	●	●	●	
AI Planning-based Deployment	Sekitei [50]			●							
	CANS [51]		●	●	●	○		○		○	
Formal Frameworks	LTS [52]	●	●	○	●	○	●	●		●	
	Conceptual [53]				●						

deployment must be as automated as possible while taking into account application constraints (e.g., CPU, Memory, etc.) and/or user constraints to prevent the user from directly dealing with a large number of hosts and their heterogeneity within a grid.

Hot deployment approaches [64], [48], [49], [65], [66], [67], [68], [69], [70], [71], [72] address the ability of dynamically deploying software components during the program runtime in autonomic environments. In autonomic environments, a software system can automatically adapt its runtime behavior with respect to the configuration of the drastically changing execution environment and user requirements [73]. In this context, it is required to dynamically install, update, configure, uninstall, and replace software components without affecting the reliable behavior of the system or other constituent components.

Planning the deployment of component-based applications into network resources has also gained attention in the artificial intelligence research community [50], [51], [74], [75], [76]. Two reasons have been mentioned for this [77]: (1) the requirement to satisfy the qualitative (e.g., reliability) and quantitative (e.g., disk quota) constraints; and (2) the fact that software deployment may involve selecting among compatible components as well as insertion of auxiliary components.

Finally, there are also some work that provide platform-independent formal frameworks for the deployment of component-based applications [52], [53]. In these frameworks, different activities of the software deployment process are defined formally in a platform-independent manner that are suitable for derivation of theoretical results. For example, they can give deployment tool developers a theoretical basis to implement systems with well-defined behavior.

VIII. CONCLUSIONS

The software deployment process is defined as a sequence of related activities for placing a developed application into its target environment and making it available for use. However, this process is often challenging for complex component-based applications that should be deployed into a large distributed environment and some QoS parameters should also be maximized. This article presented a graph-based approach for deploying the component-based applications into distributed environments. This approach uses the concept of channels to capture the properties of interconnections among the components of the application. In this 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. This article also presented the required algorithms for minimizing the cost and maximizing the reliability of a deployment.

REFERENCES

[1] A. Heydarnoori, F. Mavaddat, and F. Arbab, "Deploying loosely coupled, component-based applications into distributed environments," in *Proceedings of the 13th IEEE International Symposium and Workshop on Engineering of Computer Based Systems*. IEEE Computer Society, 2006.

[2] A. Heydarnoori and F. Mavaddat, "Reliable deployment of component-based applications into distributed environments," in *Proceedings of the 3rd International Conference on Information Technology: New Generations*. IEEE Computer Society, 2006, pp. 52–57.

[3] A. Heydarnoori, *Deploying Component-based Applications: Tools and Techniques*, ser. Studies in Computational Intelligence. Springer, 2008, vol. 253, pp. 29–42.

[4] ——, "Caspian: A QoS-aware deployment approach for channel-based component-based applications," David R. Cheriton School of Computer Science, University of Waterloo, Tech. Rep. CS-2006-39, 2006.
[5] F. Arbab, "Reo: A channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2004.
[6] F. Arbab and F. Mavaddat, "Coordination through channel composition," in *Coordination Models and Languages*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2315, pp. 275–297.
[7] C.-a. Sun, R. Rossing, M. Sinnema, P. Bulanov, and M. Aiello, "Modeling and managing the variability of Web service-based systems," *Journal of Systems and Software*, vol. 83, no. 3, pp. 502–516, 2010.
[8] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proceedings of the First International Conference on Peer-to-Peer Computing*, Aug. 2001, pp. 101–102.
[9] R. Hassin and A. Levin, "A better-than-greedy approximation algorithm for the minimum set cover problem," *SIAM Journalon on Computing*, vol. 35, pp. 189–200, July 2005.
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.
[11] M. R. Lyu, *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.
[12] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," *SIAM Journalon on Computing*, vol. 23, pp. 864–894, August 1994.
[13] G. Călinescu, H. Karloff, and Y. Rabani, "An improved approximation algorithm for multiway cut," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 48–52.
[14] V. V. Vazirani, *Approximation Algorithms*, 2nd ed. Springer, 2002.
[15] "InstallShield Developer," http://www.installshield.com/isd/.
[16] "Zero G software deployment and lifecycle management solutions," http://www.zerog.com/.
[17] "Setup factory," http://www.indigorose.com/suf/.
[18] "Setup builder," http://www.gppsoftware.com/setupb/.
[19] "RPM package manager," http://www.rpm.org/.
[20] "Yum: Yellow dog updater," http://linux.duke.edu/projects/yum/.
[21] "Package maintenance system for Debian," http://packages.debian.org/dpkg/.
[22] "Oracle solaris specification," http://www.oracle.com/us/products/servers-storage/solaris/.
[23] "Portage," http://www.gentoo.org/.
[24] "Systems management server home," http://www.microsoft.com/smserver/.
[25] "IBM Tivoli software," http://www.tivoli.com/.
[26] "Altiris deployment solution," http://www.altiris.com/.
[27] "HP OpenView," http://www.hp.com/openview/.
[28] "Citrix," http://www.citrix.com/.
[29] "PowerTCP," http://www.dart.com/powertcp/.
[30] "Secure shell (SSH)," http://www.ssh.com/.
[31] "TIBCO Rendezvous," http://www.tibco.com/software/messaging/.
[32] "IBM Gryphon," http://www.research.ibm.com/distributedmessaging/.
[33] "Java message service (JMS)," http://java.sun.com/products/jms/.
[34] "Java web start technology," http://java.sun.com/products/javawebstart.
[35] "Microsoft windows update," http://update.microsoft.com.
[36] "ClickOnce: Deploy and update your smart client projects using a central server," http://msdn.microsoft.com/msdnmag/issues/04/05/clickonce/.
[37] M. Mikic-Rakic, S. Malek, N. Beckman, and N. Medvidovic, "A tailorable environment for assessing the quality of deployment architectures in highly distributed settings," in *Proceedings of the 2nd IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 3083, 2004, pp. 1–17.
[38] D. Wichadakul and K. Nahrstedt, "A translation system for enabling flexible and efficient deployment of QoS-aware applications in ubiquitous environments," in *Proceedings of the 1st IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 2370. Springer, 2002, pp. 287–310.
[39] M. Mikic-Rakic and N. Medvidovic, "Architecture-level support for software component deployment in resource constrained environments," in *Proceedings of the 1st IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 2370. Springer, 2002, pp. 31–50.
[40] R. Balter, L. Bellissard, F. Boyer, M. Riveill, and J.-Y. Vion-Dury, "Architecturing and configuring distributed application with Olan," in *Proceedings of the 1st International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 1998, pp. 241–256.
[41] "Deployment and configuration of component-based distributed applications specification," http://www.omg.org/.
[42] P. Hnetynka, "A model-driven environment for component deployment," in *Proceedings of the 3rd ACIS/IEEE International Conference on Software Engineering Research, Management and Applications*. IEEE Computer Society, 2005, pp. 6–13.
[43] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. Gokhale, "DAnCE: A QoS-enabled component deployment and configuration engine," in *Proceedings of the 3rd IFIP/ACM Working Conference on Component Deployment*, 2005, pp. 67–82.
[44] R. S. Hall, D. Heimbigner, and A. L. Wolf, "A cooperative approach to support software deployment using the software dock," in *Proceedings of the 21st International Conference on Software Engineering*. ACM Press, 1999, pp. 174–183.
[45] N. P. Sudmann and D. Johansen, "Software deployment using mobile agents," in *Proceedings of the 1st IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 2370. Springer, 2002, pp. 217–237.
[46] S. Lacour, C. Pérez, and T. Priol, "A software architecture for automatic deployment of CORBA components using grid technologies," in *Proceedings of the 1st Conference On Software Deployment and (Re)Configuration*, 2004, pp. 187–192.
[47] V. Lestideau and N. Belkhatir, "Providing highly automated and generic means for software deployment process," in *Proceedings of the 9th European Workshop on Software Process Technology*, ser. Lecture Notes in Computer Science, vol. 2786. Springer, 2003, pp. 128–142.
[48] J. Hillman and I. Warren, "An open framework for dynamic reconfiguration," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 594–603.
[49] R. Chatley, S. Eisenbach, and J. Magee, "MagicBeans: A platform for deploying plugin components," in *Proceedings of the 2nd IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 3083, 2004, pp. 97–112.
[50] T. Kichkaylo, A. Ivan, and V. Karamcheti, "Constrained component deployment in wide-area networks using AI

planning techniques," in *Proceedings of the 17th IEEE International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2003, pp. 1–10.

[51] X. Fu and V. Karamcheti, "Planning for network-aware paths," in *Proceedings of the 4th International Conference on Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, vol. 2893. Springer, 2003, pp. 187–199.

[52] Y. D. Liu and S. F. Smith, "A formal framework for component deployment," in *Proceedings of the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*. ACM Press, 2006, pp. 325–344.

[53] A. Parrish, B. Dixon, and D. Cordes, "A conceptual foundation for component-based software deployment," *The Journal of Systems and Software*, vol. 57, no. 3, pp. 193–200, 2001.

[54] M. Mikic-rakic, S. Malek, and N. Medvidovic, "Improving availability in large, distributed component-based systems via redeployment," in *Proceedings of the 3rd IFIP/ACM Working Conference on Component Deployment*, 2005, pp. 83–98.

[55] V. Quéma and E. Cecchet, "The role of software architecture in configuring middleware: The ScalAgent experience," in *Proceedings of the 7th International Conference on Principles of Distributed Systems*, ser. Lecture Notes in Computer Science, vol. 3144, 2003, pp. 120–131.

[56] J. Matevska-Meyer, W. Hasselbring, and R. H. Reussner, "Software architecture description supporting component deployment and system runtime reconfiguration," in *Proceedings of the 9th Workshop on Component-Oriented Programming*, 2004.

[57] D. Hoareau and Y. Mahéo, "Middleware support for the deployment of ubiquitous software components," *Personal and Ubiquitous Computing*, vol. 12, no. 2, pp. 167–178, 2008.

[58] "OMG model driven architecture," http://www.omg.org/mda/.

[59] D. Rus, R. Gray, and D. Kotz, "Transportable information agents," *Journal of Intelligent Information Systems*, vol. 9, no. 3, pp. 215–238, 1997.

[60] S. Lacour, C. Pérez, and T. Priol, "Deploying CORBA components on a computational grid: General principles and early experiments using the Globus Toolkit," in *Proceedings of the 2nd IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, vol. 3083, 2004, pp. 35–49.

[61] P. Brebner and W. Emmerich, "Deployment of infrastructure and services in the open grid services architecture (OGSA)," in *Proceedings of the 3rd IFIP/ACM Working Conference on Component Deployment*, 2005, pp. 181–195.

[62] H. L. Bouziane, C. Pérez, and T. Priol, "Extending software component models with the master-worker paradigm," *Parallel Computing*, vol. 36, no. 2-3, pp. 86–103, 2010.

[63] E. Mathias, V. Cavé, S. Lanteri, and F. Baude, "Grid-enabling SPMD applications through hierarchical partitioning and a component-based runtime," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*. Springer, 2009, pp. 691–703.

[64] E. Patouni and N. Alonistioti, "A framework for the deployment of self-managing and self-configuring components in autonomic environments," in *Proceedings of the 7th International Symposium on World of Wireless, Mobile and Multimedia Networks*. IEEE Computer Society, 2006, pp. 480–484.

[65] A. Akkerman, A. Totok, and V. Karamcheti, "Infrastructure for automatic dynamic deployment of J2EE applications in distributed environments," in *Proceedings of the 3rd IFIP/ACM Working Conference on Component Deployment*, 2005, pp. 17–32.

[66] H. Cervantes and R. S. Hall, "Autonomous adaptation to dynamic availability using a service-oriented component model," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 614–623.

[67] M. Hicks and S. Nettles, "Dynamic software updating," *ACM Transactions on Programming Languages and Systems*, vol. 27, no. 6, pp. 1049–1096, 2005.

[68] H. Liu, "A component-based programming model for autonomic applications," in *Proceedings of the 1st International Conference on Autonomic Computing*. IEEE Computer Society, 2004, pp. 10–17.

[69] S. R. Mitchell, "Dynamic configuration of distributed multimedia components," Ph.D. dissertation, University of London, 2000.

[70] J. a. P. A. Almeida, M. Van Sinderen, and L. Nieuwenhuis, "Transparent dynamic reconfiguration for CORBA," in *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*. IEEE Computer Society, 2001, pp. 197–207.

[71] D. Hagimont, P. Stolf, L. Broto, and N. Palma, "Component-based autonomic management for legacy software," Y. Zhang, L. T. Yang, and M. K. Denko, Eds. Springer, 2009, pp. 83–104.

[72] Y. Li, M. Zhou, C. You, G. Yang, and H. Mei, "Enabling on demand deployment of middleware services in componentized middleware," in *Proceedings of the 13th International Symposium on Component-Based Software Engineering*, ser. Lecture Notes in Computer Science, vol. 6092. Springer, 2010, pp. 113–129.

[73] R. Murch, *Autonomic Computing*. Prentice Hall, 2004.

[74] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi, "The role of planning in grid computing," in *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, 2003, pp. 9–13.

[75] S. Gribble, "The Ninja architecture for robust internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[76] P. Reiher, R. Guy, M. Yarvis, and A. Rudenko, "Automated planning for open architectures," in *Proceedings of the 3rd IEEE International Conference on Open Architectures and Network Programming*. IEEE Computer Society, 2000, pp. 17–20.

[77] T. Kichkaylo and V. Karamcheti, "Optimal resource-aware deployment planning for component-based distributed applications," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 2004, pp. 150–159.

Abbas Heydarnoori is a post-doctoral fellow at the University of Lugano, Switzerland. His research interests include software deployment, dynamic program analysis, and program comprehension. Abbas received his PhD from the University of Waterloo, Canada. Contact him at abbas.heydarnoori@usi.ch.

Walter Binder is an assistant professor at the University of Lugano, Switzerland. His research interests focus on dynamic program analysis, virtual execution environments, and aspect-oriented programming. Walter holds a PhD from the Vienna University of Technology, Austria. Contact him at walter.binder@usi.ch.