Towards an Adaptive Deployment and Configuration Framework for Component-based Distributed Systems

William R. Otte, Douglas C. Schmidt and Aniruddha Gokhale
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203
{wotte,schmidt}@dre.vanderbilt.edu, a.gokhale@vanderbilt.edu

ABSTRACT

Software frameworks that support the deployment and configuration (D&C) of large-scale component-based distributed real-time and embedded (DRE) systems must often adapt their behavior at runtime to accommodate changing requirements and environments. For example, a D&C framework may need to interact with new deployment targets to interact and integrate with either new or legacy systems that use different communication or deployment mechanisms. Moreover, the D&C framework itself may require customization to satisfy domain requirements, *e.g.*, to change the behavior of error handling or event logging. This paper describes the shortcomings of the OMG D&C standard in terms of its ability to support heterogeneity and adapt its behavior in response to changing requirements. We also show how our *Locality-Enabled Deployment and Configuration Engine* (LE-DAnCE) provides novel approaches for addressing these limitations by enabling heterogeneous deployments, customizable behavior, and runtime adaptation of the deployment and configuration frameworks for component-based DRE systems.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications Component Middleware*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable Libraries*

1. INTRODUCTION

Large-scale distributed real-time and embedded (DRE) computing systems, such as shipboard computing environments [6] and air-traffic management [2] systems, are increasingly being developed with the use of component-based software technologies. Component-based development not only offers useful abstractions for developing large systems [4] by encouraging systematic reuse and composition, they also simplify the deployment and configuration process at run-time.

The CORBA Component Model (CCM) [8] along with the Deployment and Configuration Specification (D&C) from the Object Management Group [9], and the SOFA component model [1] assist in the deployment and configuration of component-based applications.

Production large-scale distributed computing systems often cannot be limited to a single component model, particularly if they must integrate and interface with legacy systems. While it is possible to use multiple individual deployment frameworks to deploy and configure applications, this approach can complicate the planning process (*i.e.*, assigning instances to nodes, ensuring that sufficient resources exist, performing static verification, etc.), thereby leading to problems during system integration. These problems stem from potentially incompatible tooling, metadata formats, and problems coordinating the activity of disparate deployment infrastructures.

The original *Deployment and Configuration Engine* (DAnCE) framework provides an offline deployment and configuration for the *C*omponent Integrated ACE ORB (CIAO) [10] CCM implementation. The Locality-Enhanced (LE-DAnCE) version described in this paper provides a deployment toolchain that can handle heterogeneous deployments and adapt its behavior dynamically to meet changes in the requirements of the applications it deploys.

The remainder of this paper is organized as follows: Section 2 provides an overview of the OMG D&C Specification and the DAnCE framework; Section 3 summarizes the challenges motivating the work reported in this paper; Section 4 describes our extensions to the OMG D&C standard and implementation of LE-DAnCE to address these challenges; Section 5 compares DAnCE to other deployment and configuration frameworks; and Section 6 presents concluding remarks.

2. OVERVIEW OF THE OMG D&C SPECIFICATION

The Deployment and Configuration Specification[9] (D&C) is a standard created by the Object Management Group (OMG) intended to provide both a comprehensive data model and a run-time model to manage the development, packaging, deployment, and configuration of component-based applications. The runtime interfaces and metadata are defined as a platform-independent model (PIM) through a set

of UML models and associated semantic rules to create a specification that is entirely agnostic to any particular component model.

To use the D&C specification with a particular component model, a platform-specific mapping (PSM) must be created. This PSM consists of a set of rules that transform the UML models in the PIM into appropriate concrete language artifacts most appropriate for the target component model. Currently, the only PSM transformation standardized by the OMG is for the CORBA Component Model (CCM) [8], where the D&C PSM translates the data model into two formats: XML schema for on-disk storage and interchange between tools, and IDL for runtime representation and communication between deployment entities. The runtime models are translated into CORBA 2.x interfaces.

The runtime deployment infrastructure, shown in Figure 1 consists of a three-tier architecture that exists at two levels, *i.e.*, each node contains a *NodeManager* entity that acts as the front-end for management deployments on a single node, while a global *ExecutionManager* entity coordinates the actions of a set of *NodeManagers* to accomplish a distributed deployment.

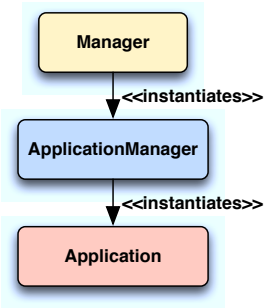

Figure 1: Architecture of the OMG D&C Specification

During the first phase of the deployment process, a user agent provides a *Deployment Plan* to the ExecutionManager, which splits the plan into *locality-constrained* plans (which contains deployment metadata for a single node only) and provides these locality-constrained plans to individual NodeManagers. Each NodeManager creates two *ApplicationManager* entities: a *DomainApplicationManager* at the global level and a *NodeApplicationManager* at the node level.

These ApplicationManager entities coordinates the second phase of deployment during which the *application* entity is created and component instances are loaded into memory. In the third phase of deployment the application entity coordinates the configuration and connection management, which provides any configuration present in the deployment metadata to deployed instances and establishes connections specified in the plan, respectively.

3. ADAPTIVE D&C CHALLENGES IN COMPONENT-BASED DRE SYSTEMS

The LE-DAnCE deployment framework is motivated by a desire to have a deployment framework that is able to both deploy heterogeneous applications (consisting of potentially multiple component frameworks in DRE systems) and adapt its behavior to meet changing requirement and expectations.

This section describes the key challenges of creating a heterogeneous and adaptive D&C tool that have motivated the key features of LE-DAnCE that are described in Section 4.

3.1 Challenge 1: Support for Heterogeneous Deployments

The process of applying the PSM to the OMG D&C specification specializes the PIM using a particular component model (such as CCM or EJB) as the target for deployment. Transforming the model with the deployment target, *i*.e. the component model we wish to deploy, as the object of the transformation has the following two categories of important categories specialization of the UML model and semantics found in the PIM:

(1) Data Model and runtime model transformation. The data and runtime model that results from the PIM to PSM transformation is mapped to a format suited to the deployment of the target component model. Transforming an OMG D&C-based model to CCM, for example, results in the creation of a data model and runtime interfaces that are specified in the OMG interface definition language (IDL). This transformation itself does not pose an inherent problem for supporting heterogeneous deployments (*i*.e., deployments consisting of more than one deployment target). This is due to the fact that almost all of the the IDL data structures that are created are agnostic to the deployment target in that they can easily represent non-CCM entities. However, some of the IDL data structures contain concrete data elements that are specific to CCM. For example, the data structures used to communicate connection metadata contains CORBA Object references. If an attempt was made to reuse the same transformation (including the IDL and the data structures) for other non-CCM component models these data structures might not be semantically meaningful.

(2) Configuration property language. The transformation defines a particular *property language* that communicates target-specific metadata (such as shared library names, entry points, and component model specific configuration data) in the D&C deployment plan. This property language consists of standard-defined name/value pairs that are encoded in property fields that decorate most entries in a deployment plan. These fields are used by a D&C framework to describe metadata specific to a component model that is needed to deploy and configure instances.

Section 4.1 describes how LE-DAnCE addresses the challenge of supporting heterogeneous deployments by introducing an Installation Handler, a well-defined interface used by D&C infrastructure to manage instance life-cycles.

3.2 Challenge 2: Customized Behavior During Deployment

Our experience with the DRE system domains described in Section 1 has demonstrated that applications may have different expectations of the behavior of the D&C infrastructure based on (1) the domain requirements (*e.g.*, safety-

criticality, QoS requirements), or (2) the stage in the development process (*e.g.*, development/testing vs. deployment/-operation). These differences in behavior include the following:

Customized error handling semantics. The deployment process for an application may result in many types errors, ranging from incorrect configuration data that may cause components to initialize improperly to application faults that cause runtime entities to crash. While some applications (*e.g.*, in safety-critical domains) should only be activated if and only if they have error-free deployments, other applications (*i.e.*, applications that are fault tolerant) may want their applications activated with "best-effort" deployment semantics, whereby deployment errors may be suppressed so as to not inhibit successful deployment and activation. Moreover, some end-users may want to ignore certain classes of errors (*e.g.*, an invalid CPU affinity setting) or errors from individual instances in a deployment.

Application liveness/status monitoring. End-users may want to leverage customized mechanisms to monitor the liveness/status of particular instances in their applications, particularly in "best-effort" deployment scenarios. Such mechanisms may be constrained by the types of information end-users want to capture, or the format and/or transport used to deliver system events. This information may be useful at the application layer (*e.g.*, to ensure that certain services are available and to glean information about their configuration) or to a runtime planner/system management service *e.g.*, to enable automatic failure detection/recovery).

Customized discovery. Proper deployment and functioning of applications often depends on discovery services that can locate elements of the deployment infrastructure or to accomplish connections between instances in a deployment plan. Certain domains (*e.*g., security critical) many have stringent requirements as to how these discovery services must secure and manage access to these services that cannot be foreseen by the D&C implementer

Section 4.2 describes how LE-DAnCE addresses the challenge of providing easily customized behavior during deployment by creating a well-defined interfaces users can leverage to provide customizations invoked during deployment.

3.3 Challenge 3: Customization of Behavior at Runtime

D&C infrastructure intended for long-running systems—or intended to provide deployment services to a variety of applications in DRE systems—must often adapt to changing conditions and requirements at runtime.

One use-case for adaptive behavior in D&C framework is the ability to select deployment-time behavior customizations (see Section 3.2) since not all customizations may be appropriate for a particular deployment. Moreover, it may not be possible to know *a priori*, *i.*e. before the deployment tools are distributed to a target computing environment, which component models a D&C infrastructure may need to deploy. For example, an application may be assembled from components implemented with several different CCM implementations, which while compatible at run-time, have differing interfaces for deployment. Ideally, the D&C infrastructure should be able to upgrade at run-time its capability to deploy different versions of component models without requiring recompiling or restarting the infrastructure.

Section 4.3 describes how LE-DAnCE addresses the challenge customizing behavior at run-time by providing a facility to deploy installation handlers and interceptors at runtime.

4. DECOUPLING THE D&C SPECIFICATION FROM TARGET COMPONENT MODEL

To address the challenges described in Section 3, we enhanced our existing DAnCE D&C framework with a novel infrastructure entity called the *LocalityManager*. The LocalityManager represents a key change in how the OMG D&C specification transforms platform-independent D&C models to target specific component models. Rather than mapping the entire specification to a particular *component model*, we map the data and runtime model to a particular *distribution middleware* that is used only to represent and communicate deployment metadata and deployment directives at runtime. Using such an approach for mapping the D&C PIM to concrete language elements allows us to reuse much of the data model which, as outlined in Section 3.1 is largely agnostic to the deployment target.

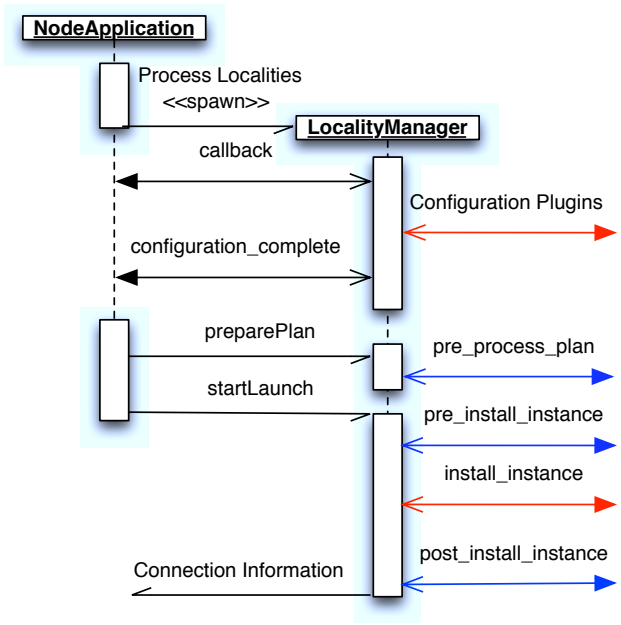

Figure 2: Locality Manager

The LocalityManager, a key feature in our Locality-Enhanced DAnCE (LE-DAnCE), is an entity spawned by the NodeApplication entity described in Section 2 The LocalityManager entity is intended to be a generic application server, maintaining a strict separation of concerns between *generic* deployment logic and the *specific* run-time logic necessary to deploy a particular deployment target. To provide a well-defined interface between the NodeApplication and the LocalityManager, we have reused elements from the D&C spec-

ification by including operations from the *Manager* interface and inheriting from the *ApplicationManager* and *Application* interfaces.

Figure 2 shows the initial start up sequence of the LocalityManager.

The remainder of this section describes the structure and functionality of the LocalityManager.

4.1 Instance Installation Handlers

To address the challenge described in Section 3.1, the implementation of the LocalityManager is entirely agnostic to the particular component model it is attempting to deploy, delegating all *component model specific* life-cycle management operations to pluggable Instance Installation Handlers, which we describe below.

Instance installation handlers represent a well-defined interface that is used by the LocalityManager to manage the life-cycle of all entities that are installed during deployment. The operations that are included in this interface were heavily influenced by the typical CCM Component life-cycle, which is shown in Figure 3. The included operations allow the LocalityManager to install/remove an instance, create/remove a connection, indicate that configuration is complete, and to activate/passivate an instance. The operations in this interface are currently used by the locality manager to perform all initial deployment actions, and can be used in the future to provide for application re-deployment and re-configuration in the future.

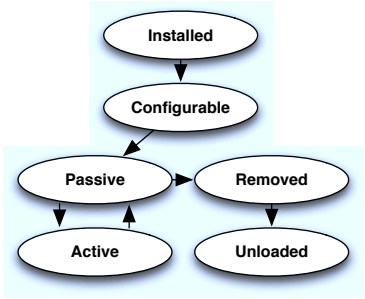

Figure 3: Typical CCM Component Lifecycle

It is important to note that not all instance types will require every lifecycle operation in the installation handlers to be implemented. For example, a total of four installation handlers were created to support the installation and management of CIAO components. First, it is necessary to instantiate a CIAO container to host any components hence an installation handler was created that initializes the CIAO runtime and is capable of instantiating containers.

Second, an installation handler was created to support the installation of CCM Homes. Neither of these first two entities have the same number of lifecycle states as a CCM component. For example, neither have connections nor distinct active/passive states, so the relevant operations in the handler remain unimplemented. Finally, handlers were created that are able to load components directly from a dynamically loaded shared library or from an appropriate factory operation on a CCM Home. These handlers implement all of the lifecycle operations in the installation handler.

Despite the differences in how each of these entities is installed and behaves at runtime, the common interface for managing their lifecycle allows the LocalityManager to treat each as an abstract instance. More importantly, it allows the LocalityManager to easily be configured to deploy entirely new instance types provided appropriate installation handlers are loaded.

For example, assume an application is made of only CIAO components. To accomplish this deployment, the LocalityManager would require two installed installation handlers - one for the containers that will host the components, a second installation handler that manages the life-cycle of the components. In this case, the deployment plan would contain a instance that represents a container, and other instances that represent the components to be installed. The LocalityManager would first select (based on metadata in the plan) the installation handler for the container and invoke the "install" operation, which causes the container handler to bootstrap the CIAO infrastructure. Next, the LocalityManager will select the handler for CIAO Components, and invoke "install" operations for each component instance, which will cause the handler to interact with the already installed container to create a component.

As a further example, lets assume that we now wish to introduce heterogeneity into this deployment example by also including non-CCM component instances. This can be accomplished by annotating the instances with an appropriate identification string and providing appropriate installation handlers for the new component model.

4.2 Deployment Portable Interceptors

Addressing the challenge described in Section 3.2, by providing a mechanism for end-users to customize the behavior of the middleware, the LocalityManager also implements a mechanism which can be used to modify the elements of the deployment plan both before and after invocation of each life-cycle management operation. This mechanism, which we call "Deployment Portable Interceptors", was inspired by CORBA Portable Interceptors [7], and is described below.

The Deployment Portable Interceptor (DPI) facility in the LocalityManager allows end-users to supplement or modify behavior during deployment. The operations in the DPI interface derived from the operations present in the Installation Handler interface. Each operation in the Installation Handler interface resulted in two operations added to the DPI interface – one which is invoked *before* the lifecycle operation, and another which is invoked after.

In Figure 2, for example, the LocalityManager invokes a DPI hook before (a *pre* hook) and after (a *post* hook) the *install_instance* lifecycle operation. All of the "pre" interceptors receive the same parameters as their associated Instance Handler operation, and are allowed to manipulate those parameters to change the behavior of the operation. For example, an alternative discovery service for connections may be implemented by overriding the *pre_connect* interception point with logic that would retrieve the appropriate con-

nection reference and modify the parameters passed to the *connect_instance* operation.

The "post" interceptors generally receive the same parameters of the lifecycle operation that preceded them, in addition to an additional parameter that contains any error result (*i.e.*, exception) that may have arisen during execution. Unlike the "pre" interceptor, the "post" event is only allowed to manipulate the error parameter, if present. This parameter allows the interceptor to, for example, log success or failure of the event (*i.e.*, for a system health and status service), or to clear the error status, causing that error to be overlooked by the LocalityManager implementation (*i.e.*, for implementation of best-effort deployment semantics).

4.3 Configuration of Handlers and Interceptors

Finally, to address the challenge outlined in Section 3.3 and provide a mechanism to provision both Installation Handlers and Deployment Interceptors at runtime, the LocalityManager is capable of installing these entities during deployment as they would any other instance as described below.

Allowing runtime adaptation of the deployment framework requires the ability to dynamically add or remove instance installation handlers and deployment interceptors on a per-deployment basis. In the LocalityManager, we have added a facility which invokes user-supplied configuration plug-ins during start-up through a well-defined interface. In Figure 2, this process takes place after the LocalityManager initially calls back to the NodeApplication to receive configuration metadata that is present in the deployment plan.

metadata provided to the LocalityManager consists of a series of name/value pairs. The name of each property is used to select an appropriate configuration plug-in to which the value is provided. By including both a property on the LocalityManager instance in the plan that describes the desired Instance Handlers and Deployment Interceptors for the plan, and a configuration plug-in that is able to interpret that property, it is possible to load them before the LocalityManager attempts to install any instances in the plan.

This facility has utility outside the configuration of Handlers and Interceptors. For example, we used these plug-ins to change QoS parameters (such as priority or CPU affinity) of the LocalityManager instance at deployment time without introducing platform-specific code into the LocalityManager.

5. RELATED WORK

DeployWare [3] is a framework for managing heterogeneous software deployments in grid environments. Deployments in this system are described using a domain-specific modeling language that captures deployment metadata in a manner agnostic to the eventual deployment target. Heterogeneous deployments are then accomplished by using appropriate "personalities", which are hierarchies of Fractal components that implement parts of the deployment process. Unlike the OMG D&C specification, DeployWare does not provide a well-defined set of metadata that can be used throughout the application development lifecycle nor does it provide a way to model hardware resources in the computational domain. Such metadata is desirable for fostering both reuse and a library of COTS component applications. As a result, DeployWare can be harder to use in larger projects in which multiple, independent teams must collaborate.

ADAGE [5] is another grid deployment tool that is capable of heterogeneous deployment capable of deploying both CCM and MPI applications. In this system, applications are described in a middleware-specific description language which is provided to a "translator" that converts that description into a middleware agnostic format called the Generic Application Description (GADe) model. Like the OMG D&C specification, it provides a description language for hardware resources, but does not provide an expressive vehicle for component metadata. For example, it is not possible to capture specific component/node pairings, which are decided by the deployment tool, or to capture QoS attributes, such as Processor/Core affinity or process priority. While this automatic planner included in ADAGE makes the planning process easier for the grid environments for which this tool is intended, it is not desirable for DRE systems in which specific control over the application topology may be required to provide sufficient quality of service for the application.

SOFA [1] is a component model with its own D&C framework that provides many advanced features for component-based software, including behavior specification and verification, software connectors for supporting many communication middleware platforms, and a robust redeployment mechanism. While SOFA's component model and D&C framework have many advanced and interesting features, it supports neither heterogeneous deployment nor adaptation of the behavior of the D&C framework found in DAnCE.

6. CONCLUDING REMARKS

This paper described the LocalityManager, which is an extension to the OMG D&C specification and key feature of LE-DAnCE, that adds three important capabilities to the original standardized deployment framework to support heterogeneous deployment and adaptation. First, the LocalityManager uses *Instance Installation Handlers* to deploy applications that use heterogeneous component models by encapsulating middleware-specific deployment logic in a well-defined interface that handles all lifecycle events. Second, it can adapt the behavior of the deployment tool-chain at runtime through the use of *Deployment Portable Interceptors*. Third, the D&C tool-chain can adapt more readily to changing requirements by having the ability to load both installation handlers and interceptors at runtime.

The implementation of heterogeneous deployment and interceptors found in the LocalityManager described in this paper is complicated by the fact that the deployment plan metadata defined by the D&C specification is poorly suited to capture deployment ordering or dependencies. It is therefore hard to determine the order in which instances should be installed when there are implicit dependencies, *e.g.*, CIAO containers must be installed prior to the components they host. We addressed this challenge in the LocalityManager by following a FIFO approach to select the order of installing instance types. While sufficient for our current end-users,

this approach will not scale as the number of installed interceptors and/or installation handlers increase.

We plan to address this issue by adapting the hierarchical deployment specification techniques in the DeployWare and SOFA component models. In particular, as outlined in Section 4.1, we will leverage this prior work to build robust redeployment and reconfiguration capabilities into DAnCE to support adaptive deployment behavior in applications managed by this framework. Moreover, as we gain a complete understanding of the shortcomings of the OMG D&C specification and the associated PIM to PSM mapping process, we will work within the OMG to produce an updated specification.

All the software described in this paper is available in open-source form in the 0.8.2 release of CIAO and DAnCE available at `download.dre.vanderbilt.edu`.

7. REFERENCES

[1] T. Bures, P. Hnetynka, and F. Plasil. Sofa 2.0: Balancing advanced features in a hierarchical component model. *Software Engineering Research, Management and Applications, ACIS International Conference on*, 0:40–48, 2006.

[2] C. Esposito and D. Cotroneo. Resilient and timely event dissemination in publish/subscribe middleware. *International Journal of Adaptive, Resilient and Autonomic Systems*, 1:1 – 20, 2010.

[3] A. Flissi, J. Dubus, N. Dolet, and P. Merle. Deploying on the grid with deployware. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 177–184, Washington, DC, USA, 2008. IEEE Computer Society.

[4] G. T. Heineman and B. T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Reading, Massachusetts, 2001.

[5] S. Lacour, C. PÃl'rez, and T. Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *In 6th IEEE/ACM International Workshop on Grid Computing (Grid2005*. Springer, 2005.

[6] P. Lardieri, J. Balasubramanian, D. C. Schmidt, G. Thaker, A. Gokhale, and T. Damiano. A Multi-layered Resource Management Framework for Dynamic Resource Management in Enterprise DRE Systems. *Journal of Systems and Software: Special Issue on Dynamic Resource Management in Distributed Real-time Systems*, 80(7):984–996, July 2007.

[7] Object Management Group. *Interceptors FTF Final Published Draft*, OMG Document ptc/00-04-05 edition, Apr. 2000.

[8] Object Management Group. *CORBA Components v4.0*, OMG Document formal/2006-04-01 edition, Apr. 2006.

[9] OMG. *Deployment and Configuration of Component-based Distributed Applications, v4.0*, Document formal/2006-04-02 edition, Apr. 2006.

[10] N. Wang, D. C. Schmidt, A. Gokhale, C. Rodrigues, B. Natarajan, J. P. Loyall, R. E. Schantz, and C. D. Gill. QoS-enabled Middleware. In Q. Mahmoud, editor, *Middleware for Communications*, pages 131–162. Wiley and Sons, New York, 2004.