3
Reference Scenario

The execution infrastructure for SCS components allows deployment actors to manage tasks such as instantiation, configuration, interception, execution, etc. However, this responsibility involes a high number of management tasks. Thus, they are susceptible to human errors, even on basic applications. Distributed scenarios need an advanced deployment process to deal with several challenges: heterogeneity, scalability, monitoring, management, etc. Therefore, to deploy software systems a Deployment Infrastructure is required to manage the lifecycle of components, and reduce the complexity of typical deployment tasks. This infrastructure allows deployment actors to plan a remote and decentralized deployment of distributed, multi-language, and multi-platform component-based applications. In this chapter we describe three main points of SCS, organized as follows: Section 3.1 describes SCS component model; Section 3.2 shows the execution infrastructure for SCS components; Section 3.3 details the SCS deployment infrastructure focused on planning activity.

3.1 SCS Component Model

Software Component System(SCS) is a component-based middleware designed for CORBA architectures. This provides a middleware infrastructure suitable for distribution, installation, configuration, execution, and deployment of components [Cerqueira09]. SCS component is the logic unit ready for composition and reuse, it also encapsulates interaction, configuration and introspection features. SCS was inspired by Microsoft COM(Component Object Model) and OMG CCM(CORBA Component Model), however SCS components avoid COM and CCM complexity. SCS tries to maintain flexibility, simplicity, and usability using a small set of API. SCS components are composed of facets and receptacles, both are service ports that belong to its interaction model. Facets are ports for providing services offering a specific interface, thus available services for each component are accessed by facets. Receptacles are ports to request services, in other words, they are access points to compon-

ent services. Through the use of its interaction model a configuration model is enabled, this allows component connections to be replaced at the time of execution. The introspection model defines a set of functions that allows components to be inspected in execution time, for example users could check the available facets. Figure 3.1 shows a SCS component with its facets and receptacles ports. Also, it displays three specific facets that support its configuration and introspection model: IComponent, IReceptacles, and IMetaInterface. IComponent permits identification, activation, and deactivation of components. IReceptacles manage inter-connections between remote and local components. IMetaInterface provides functions for introspection.

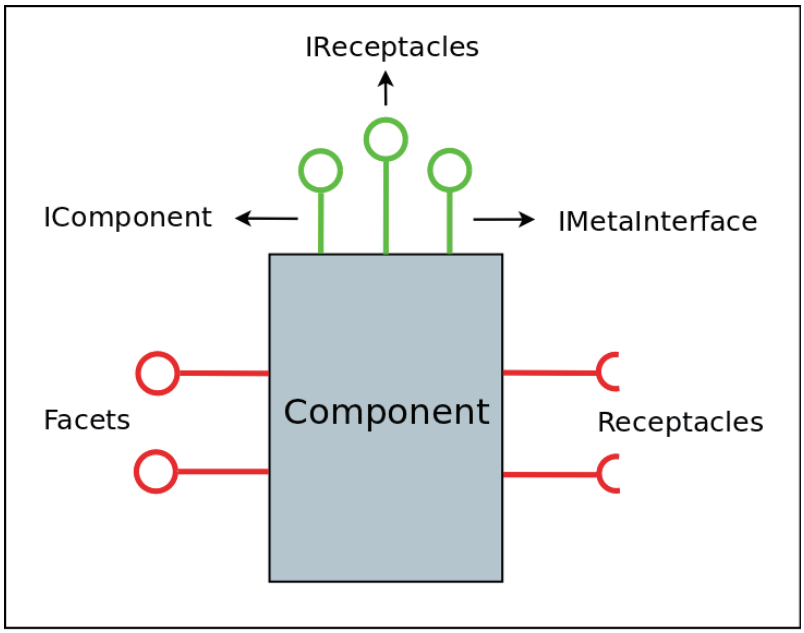

Figure 3.1: SCS Component Model

3.2 Execution Infrastructure

The execution infrastructure for SCS component-based applications allows users to instantiate, configure, suspend, intercept, and execute SCS components [Augusto08]. The execution infrastructure could be remotely managed because all facets are CORBA objects. This infrastructure is displayed in figure 3.2, consisting of a Container, ExecutionNode, and Repository.

- **Container:** The container is responsible for managing a common memory space for SCS components. It provides facilities to load, intercept, suspend, resume, and monitor components. The container is flexible because it permits the loading of different components.

- **Execution Node:** The ExecutionNode represents a node, and operates as a gateway for physical machines. Also, the ExecutionNode manages the building of containers, and controls their access to physical resources.
- **Repository:** The Repository stores published components that could be accessed remotely. The Repository is used by the Container to access component implementations.

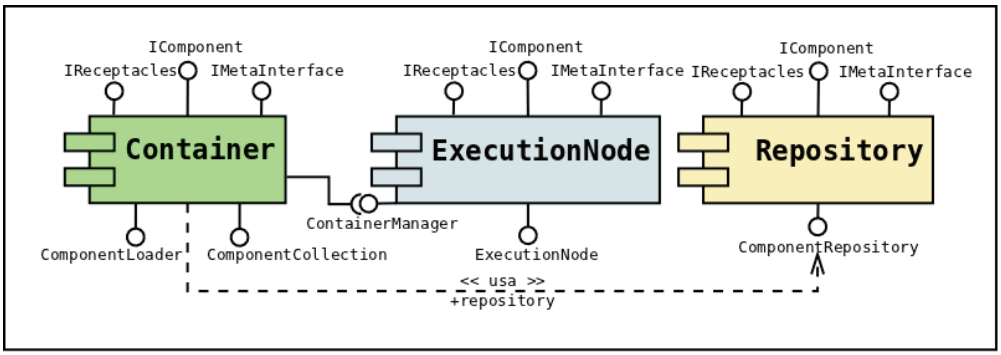

Figure 3.2: SCS Execution Infrastructure

3.3 Deployment Infrastructure

A deployment infrastructure for SCS components was developed [Junior09] to address the deployment process, and automate the management tasks of the execution infrastructure. Deployment infrastructure has two main services: Packager and DeployManager. The Packager service is responsible for the packaging process and solving static dependencies. DeployManager orchestrates the creation of deployment plans. Additionally, a modified version of ExecutionNode and the Repository components were developed for supporting both system and component packages, respectively.

Figure 3.3 shows the deployment process for SCS components, composed of eight activities, detailed as follows: First, the application's components are packaged(1) using the packager service, it automates the generation of packages and assures compliance of the packaging system. Second, the publishing activity(2) consists of moving the packages and file descriptors into a previously known repository. Third, the planning activity(3) elaborates a plan to deploy all components. The DeployManager service involves the execution infrastructure components that are responsible for deciding a high level plan configuration. Fourth, the deployment activity(4) executes the previous plan, that is, components are installed on a target environment based on physical machines. Fifth, all components are activated(5). Sixth, once the execution has

been completed all components are deactivated(6). Seventh, all components are removed(7). Eighth activity, components are retired(8).

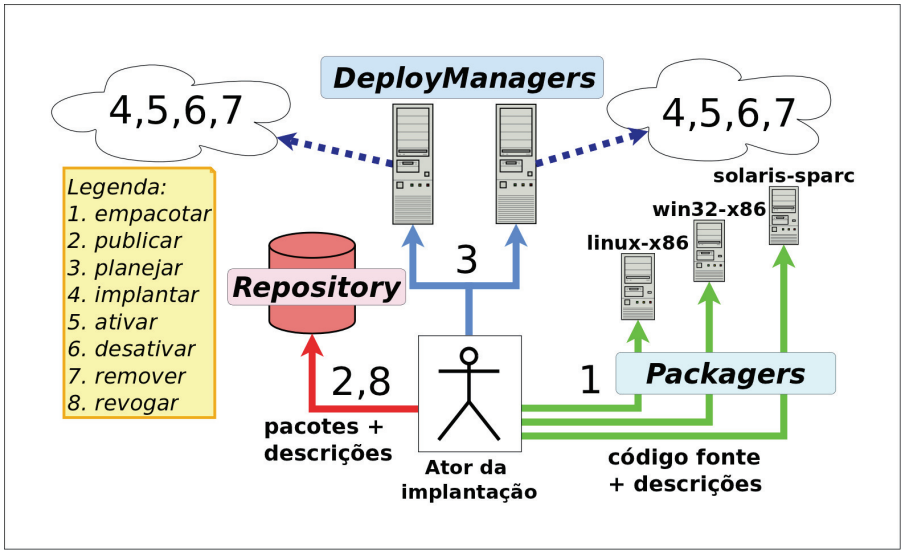

Figure 3.3: Overview of SCS Deployment Infrastructure

3.3.1 Deployment Planning

When deployment actors need to validate architectures and QoS parameters of applications, they need deployment plans to test with different configurations of distributed target environments. Therefore, deployment planning constitutes a critical part within the deployment process. To deal with these issues, an entity needs to be responsible to build a consistent deployment plan. DeployManager allows the construction of a deployment plan, creates an initial configuration, sets an incremental level of details, and makes calls to the function deploy. DeployManager involves the execution infrastructure providing several capabilities to deploy SCS components. DeployManager uses virtual entities to represent user components and the SCS execution infrastructure. DeployManager was designed combining virtual entities, a plan, and a deployment service. Deployment planning allows users to specify a deployment plan based on incremental level of details. Users can refer to an automatic or manual mapping of the execution infrastructure and physical resources. Automatic mapping is the highest level to write a deployment specification. To deploy a component is necessary to create a plan and a virtual entity. Then we need to specify the type of component, connections between components, and call the deploy function. DeployManager implements a round-robin algorithm for automatic mapping. On the other hand, manual mapping has three incremental levels: grouping user components in the same container, grouping

containers in the same execution node, and setting a specific physical machine for the execution node. A list of physical machines is previously loaded by DeployManager.

The packaging process is bundled into a packager service, offering facilities to package components on distributed architectures. This service includes a solution to solve static dependencies issues, considering the language and platform supported by SCS components. The packager service has chosen the package system "LuaRocks" to allow developers to describe metadata, dependencies, compilation, and installation procedures. DeployManager permits subscriptions to public repositories, which supports being shared by many DeployManagers. Therefore, developers need to search a list of public repositories, and publish their component implementations.