Architecture for Distributed Component Management in Heterogeneous Software Environments

E. Stoyanov, A. MacWilliams, Dieter Roller

***Abstract*—This paper proposes an alternative approach to software management in heterogeneous environments. It targets encapsulation and dependency management of component systems using Stanford Beer's Viable System Model (VSM) as requirements and organizational model. VSM is expressed with meanings of Common Information Model (CIM) extensions that serve as basis for an object-oriented representation of managed components. A control-loop architecture is proposed to facilitate monitoring of heterogeneous component environments using the developed model.**

I. Introduction

Software systems are becoming more and more complex every day. They involve different technologies for component management and communication. Software systems are developed and supported by various vendors, often especially tuned for the specific needs of the customer. In this way customers have the opportunity to select the product and the framework that suites them best for the different specific tasks and problems they might experience.

However, there is a side effect of this development, namely the sometimes difficult management of the deployed software. Reasons for it are the lack of common management instrumentation for tracking of problems, such as functional regression, configuration and deep dependencies between components of different nature. In production environment, where development process depends strongly on the availability of IT assets, stability is factor with a higher priority. This paper proposes elements of modern software engineering that affects the design of software support systems in a way to make them flexible and robust. We introduce a software architecture that adapts Stanfrod Beer's Viable System Model (VSM) with the help of which the platform supports inter-framework dependency tracking and related problems.

II. Heterogeneous Systems

Vendors of software support systems are realizing that extensible functionality is an important aspect of the growing demand of the customer companies utilizing the software. This is visible with the increasing usage of Rich Client Platforms (RCP) based on plug-in frameworks [1] and distributed component systems (DCS) [2]. While RCP and DCS are alone very rich environments, deployment of only one of them is not enough for full satisfaction of the user demand. For example RCP contributes to visual arrangement of graphics, tooling interfaces, and overall functionality, but is not designed for distributed tasks. DCS systems solve this problem with their dedicated design for distributed computing, abstraction of resources (such as databases) and standard approach to inter-operable communication and information exchange. It becomes more evident that development of distributed computing inside an organization demands deployment of additional technologies, such as multi-agent systems (MAS) for knowledge communication[3] and web services (WS) for inter-operable cooperation[4]. These interacting components become dependent in heterogeneous manner, for example a component residing in Enterprise Java Bean(EJB) container may be dependent on another remote component located inside .NET framework. Even if some frameworks provide local version and dependency chains management, the missing formal relations between the frameworks on the management level increases the management effort and respectively support costs.

III. Component Frameworks

Before we propose a model for management of heterogeneous systems we will take a short look at what software components are and how they are organized.

The common definition for a component is that it is a manageable and reusable software element that is portable between containers designed to manage its life-cycle using interfaces specified in a component specification and supported by the container. If we take a look at how currently the modern component technologies wrap the system and functional components it can be concluded that there are very general similarities when it comes to encapsulation and communication principles. For a more practical understanding of the problem we will discuss three modern component technologies, used in RCP and DCS in production-ready systems: Enterprise JavaBeans (EJB), OSGi Framework (OSGi), and Microsoft .NET platform (.NET). Deployment of all three of them serves as an usual setup of client side development tools (.NET and OSGi), server-side transactions and database interaction (EJB) and intra- and inter-organization interaction (EJB and .NET). In order to come closer to the idea of heterogeneous management of components we will introduce the common communication principles used in those component technologies, as well as their organizational hierarchy of containment.

A. Component Communication

Assuming the Architecture Description Language (ADL) vocabulary [5], the components communicate with their supporting infrastructure and with other components using

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 167–172.

interfaces with different *types* of *ports*. A *component interface* may consist of the following ports: *attributes*, *methods*, *event source*, *event sync*. Generally, those can be used in those three cases: when the component communicates with the management framework, when it communicates with other components or by communication with traditional (non-component) software entities, such as external functions and interfaces of classes outside the components.

1) *Communication with the Framework:* Communication with framework is usually performed by the components in the case where they need resources or information about certain states of the system. In the case of OSGi, the components (called bundles) communicate with the framework when there is a need to locate other services, as well as for bundle and services state-change notifications. In the EJB framework, the components communicate when they need to locate other objects, or they need special services as timing. The framework on its own communicates with the components with the help of context objects. Information within context is selectively used by the components. In .NET components are communicating with the framework when they need to retrieve component interfaces.

2) *Communication between components:* Communication between components happens on two levels: semantic (functional) and sequential (execution). Semantic communication is related to the way client interacts functionally with the component on the level of their functional interfaces. The sequential communication is usually handled with the help of the framework and is related to the concrete steps with which the actual information and variety transfer is achieved between the components. This includes marshaling/unmarshaling of data-types, data transfer protocol implementation and type mapping. Both aspects are important, because if one of the communicating sides fails to conform to both aspects, the communication between components is either wrong and may lead to undesirable behavior or is completely broken and possibly leading to subsystem malfunction. Figure 1 expresses semantic communication and execution communication.

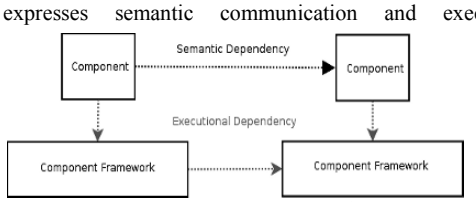

Figure 1. Component Dependencies

To achieve semantic communication the successful executional one has to be present. Another important aspect of component communication is that the component itself is dependent on the underlying framework and essential resources, such as network and database access, are utilized through communication with the upper management levels. Figure 1 illustrates this by the vertical arrows, which can represent queries for resource or communication adapter to achieve the actual communication with other components. That is why an essential point of heterogeneous management is the understanding of the hierarchical grouping and encapsulation of elements.

B. Encapsulation and Hierarchy

While the different component frameworks have different component life-cycle management techniques, there are some very basic common characteristics that deserve attention. Encapsulation is a key principle where components are providing interception points through which the hosting container uses for manipulation of the internal component state and routines for initialization, suspending or destruction. Let us see how the three discussed frameworks define encapsulation. In the J2EE framework the *Enterprise Java Bean* stays on the last level of component management hierarchy inside an J2EE-based system. The upper levels of containment and management are *EJB-Container*, *J2EE Server* and the J2SE *Virtual Machine* as a higher application manager and container. Similar hierarchy is observed in the other component frameworks. For example OSGi defines a component as a set of classes and services in the form of *Bundle*. The OSGi *Framework* acts as a management container for the bundles and stands higher on the hierarchy, followed by the execution environment, typically a JVM. The last level of containment in a *.NET Framework* is the module (DLLs). Assemblies aggregate those *modules* into components that are consumed by applications. Management, such as isolation and automated version handling, are performed by the *CLR* and its dynamic loader. A generalized diagram of encapsulation hierarchy for the three frameworks is shown on Figure 2.

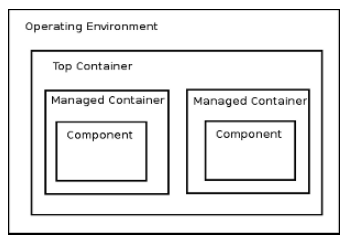

Figure 2. Component Hierarchy

IV. Management of Heterogeneity

There have been many attempts to provide agile software and IT environment in general but few target the problem of evolution support in heterogeneous systems with the accent of management and problem tracking. Our approach to heterogeneous management is based on adaptation of the *Viable System Model* (VSM) [6], a management model that was successfully deployed during the last two decades in different fields of knowledge and organizational management. We re-use the requirements of VSM in software communication and implement a VSM-like model for mapping of component entities. The main advantage of VSM as a template of constructing management systems is the clean definition of communication flow requirements and the task distribution between management entities. One of the main