Dynamically reconfigurable resource-aware component framework: architecture and concepts

Dynamically Reconfigurable Resource-Aware Component Framework: Architecture and Concepts

Bojan Orlic, Ionut David, Rudolf H. Mak, and Johan J. Lukkien

Eindhoven University of Technology,
Eindhoven, The Netherlands
{b.orlic,i.david,r.mak,j.j.lukkien}@tue.nl

Abstract. Applications executed on a shared distributed platform compete for resources provided by the platform. In case these applications have highly fluctuating resource demands, a software architecture is required that provides support for runtime resource management. In position paper [1], we have proposed such architecture and have introduced its key concepts and entities. In this paper, we introduce a metamodel that captures the key concepts and we identify lifecycle models for both applications and individual components. A set of dynamic reconfiguration strategies is introduced and their relationship to the stages of the application lifecycle is given.

Keywords. Component framework, networked services, resource management, dynamic reconfiguration, application lifecycle, component lifecycle.

1 Introduction

The key idea behind the component-based software engineering (CBSE) approach to software development is to enable 3rd party composition in the process of system engineering and in that way facilitate building complex systems from predefined building blocks [2]. Composition of components is supported by component frameworks which address the component model, as well static (design time) and dynamic (run time) aspects of composition. We are interested in CBSE for distributed systems with limited resources, where resources concern processing power, memory capacity and network bandwidth. Resource management, however, is rarely addressed by a software component framework, and if it is, it is limited to static resource reservations. We target applications with highly fluctuating resource demands (e.g. video processing applications) in which static reservations are not suitable as they result in inefficient resource usage. For the combination of efficient runtime resource management and dynamic component composition a special architecture is needed. In this paper we further develop the architecture proposed in [1]. We choose to realize applications using the SOA style by connecting networked services, which allows us to include structural reconfiguration in our resource management strategies.

2 Architecture Description

Figure 1 gives a metamodel that specifies key abstractions of our architecture. The system layer contains a single resource manager that is in charge of all resources

I. Crnkovic, V. Gruhn, and M. Book (Eds.): ECSA 2011, LNCS 6903, pp. 212–215, 2011.

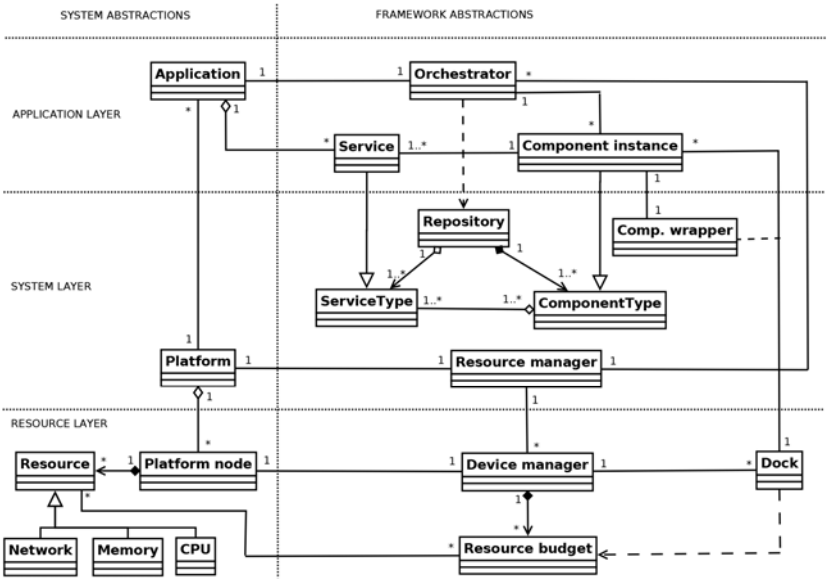

Fig. 1. Metamodel defining basic concepts, entities and interconnection mechanisms

provided by the nodes of the distributed platform. To perform its management tasks it deploys the services of local device managers, as shown in the resource layer. These services comprise installation and instantiation of components, monitoring their resource usage, and enforcement of resource reservations (budgets). In the application layer, orchestrator entities are responsible for the composition, deployment and operation of an application. Each application in the system will have a dedicated orchestrator. Since our applications are built from reusable and discoverable components whose instantiation, composition, and deployment can be performed in the SOA style, the system layer of our architecture also contains a repository of services and corresponding components that are available in the system. Applications are composed from services through binding provided and required interfaces. Docks are units of the deployment of the system that can host one or more components.

The lifecycles of applications and components, depicted in Figure 2, involve two main stakeholders – application designer and application operator. A designer can create applications either ad-hoc at runtime or he can define and save an application as a set of possible configurations. In the latter case, the operator chooses when to start the application, and the actual configuration is selected at runtime through negotiation between the resource manager, the orchestrator and the operator. The application lifecycle consists of composition, deployment and operation phases. In general, resource management decisions taken in the operation phase (marked 1 to 5) require revision of activities in the other two phases (steps A to E). In step A, the application designer composes an application from services available in the repository. In step B, he selects components that implement those services. During the composition phase, the application designer is also allowed to enter constraints for steps of the deployment phase, such as mapping of components to nodes (C), setting of QoS levels (D)

and allocation of budgets to components (E). Thus, prior to runtime, it is possible to specify a number of different configurations as well as the reconfiguration strategy (the logic that stipulates when to select which configuration).

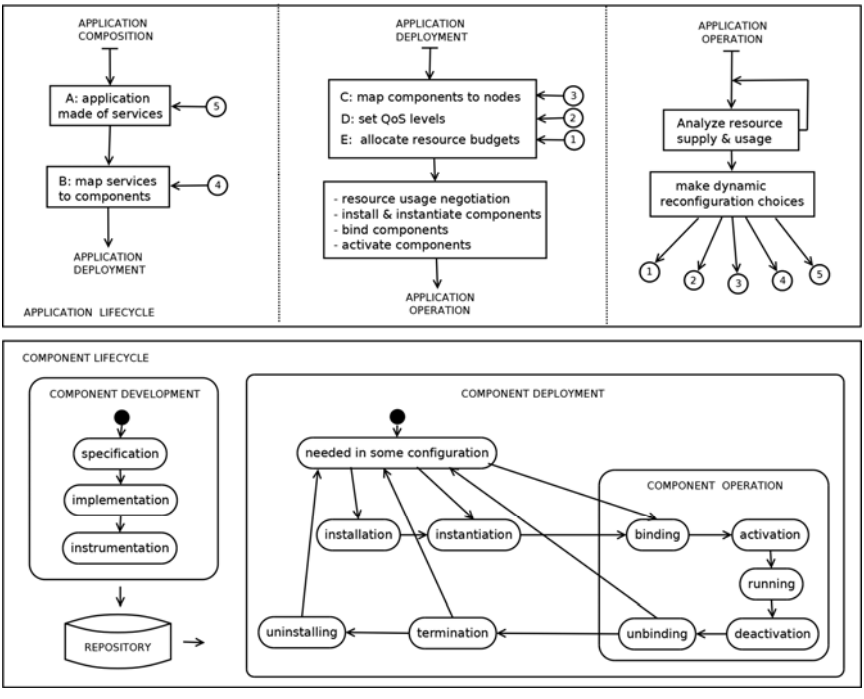

Fig. 2. Application and component lifecycles

In the operation phase, resource supply and usage is monitored and analyzed. If needed, dynamic reconfiguration is applied. In order of increasing severity we distinguish: 1) Realigning resource budgets. 2) QoS adaptation, in which service levels are adjusted to resource availability. It includes realigning local resource reservations and resolving dependencies between QoS levels of an application's components, e.g., the frame rate. 3) Reallocation, where components are migrated to nodes that better fit their resource demands. 4) Replacement of components by others that offer the same service but at a different resource demand. 5) Reconstruction of application structure, in which services are added, replicated, replaced, or removed.

The component lifecycle is specific to our framework and in accordance with the application lifecycle. The component development phase includes specification, implementation and instrumentation. The outcome of the instrumentation is a component deployable on a specific OS and compatible with our framework. Such a wrapped component is stored in the repository. The component deployment phase starts when a component is needed in some application and includes activities such as installation, instantiation, termination and uninstalling. In the component operation phase, the instantiated component can undergo binding, activation, deactivation and unbinding.

3 Conclusions and Future Work

In this paper, we have described a resource-aware component based architecture. Similar to others we identify three layers [3], and distinguish between local and global resource management [4], [5] and [6]. As in [7] and [8] resource management is combined with plugin framework and SOA technologies. In contrast to all frameworks cited, our framework allows structural changes as part of resource management strategies, and defines application and component lifecycles.

Current work involves implementing the framework and definition of models for resource usage prediction. Future work includes introducing resource management policies and assigning those activities to framework entities.

References

1. David, I., Orlic, B., Mak, R., Lukkien, J.J.: Towards Resource-Aware Runtime Reconfigurable Component-Based Systems. In: Proceedings of the 6th World Congress on Services, SERVICES 2010, Miami FL, USA, pp. 465–466 (2010)
2. Szyperski, C.: Component Software: Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley, Reading (2002)
3. Koulamas, C., Prayati, A., Papadopoulos, G.: A Framework for the implementation of Adaptive Streaming Systems. In: Proceedings of the 3rd ACM Workshop on Wireless Multimedia Networking and Performance Modeling (WMuNeP 2007), pp. 23–26 (2007)
4. Sachs, D.G.: A New Framework for Hierarchical Cross-layer Adaptation, Ph.D. dissertation, University of Illinois at Urbana-Champaign (2006)
5. Rizvanovic, L., Fohler, G.: The MATRIX: A Framework for Streaming in Heterogeneous Systems. In: RTMM - International Workshop on Real-Time for Multimedia, Italy (2004)
6. Kersten, B., van Rens, K., Mak, R.: ViFramework: A framework for networked video streaming components. In: Proceedings of the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2011 (2011)
7. Korostelev, A., Lukkien, J.J., Nesvadba J., Qian Y.: QoS Management in Distributed Service Oriented Systems. In: Parallel and Distributed Computing and Networks (2007)
8. Chen, S., Lukkien, J.J., Verhoeven, R., Vullers, P., Petrovic, G.: Context- Aware Resource Management for End-to-End QoS Provision in Service Oriented Applications. In: Proceedings of the Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments (SUPE 2008), pp. 1–6 (2008)