A COMPONENT-BASED SOFTWARE ARCHITECTURE
Reconfigurable Software for Ambient Intelligent Networked Services Environments

Michael Berger, Lars Dittmann
COM·DTU, Technical University of Denmark, Oersteds Plads Bldg 343, DK-2800, Lyngby, Denmark

Michael Caragiozidis, Nikos Mouratidis
APEX AG, Bundesgasse 16, CH-3011, Bern, Switzerland

Christoforos Kavadias
TELETEL SA, 124, Kifissias Avenue, 115 26 Athens, Greece

Michael Loupis
SOLINET GmbH, Mittlerer Pfad 26, 70499 Stuttgart, Germany

Keywords: Component-based Software, Ambient Intelligence, Reconfigurable Software, SDL, UML, OSGi.

Abstract: This paper describes a component-based software architecture and design methodology, which will enable efficient engineering, deployment, and run-time management of reconfigurable ambient intelligent services. A specific application of a media player is taken as an example to show the development of software bundles according to the proposed methodology. Furthermore, a software tool has been developed to facilitate composition and graphical representation of component based services. The tool will provide a model of a generic reusable component, and the user of the tool will be able to instantiate reusable components using this model implementation. The work has been carried out within the European project COMANCHE that will utilize component models to support Software Configuration Management.

1 INTRODUCTION

The main objective of COMANCHE (COMANCHE, 2008) is to develop and validate a generic framework for Software Configuration Management (SCM), which will pave the way to the realization of technically and commercially viable private spaces incorporating ambient intelligence features. For this purpose the project will specify and develop the COMANCHE modular and scalable architecture targeting the provision of consistent, secure, low-cost (low-effort) SCM services across today's heterogeneous, and multi-vendor environments. The realization of the COMANCHE SCM services framework will be built on an adequate software engineering and knowledge management infrastructure that the project will deliver. The main components of this infrastructure will be the following:

i) The COMANCHE Knowledge Management Framework, which will provide the means for effectively conceptualizing, organizing, discovering, and exploiting the tremendous amounts of attribute information, pertaining to SCM

ii) A modular component-based software architecture and an adequate design methodology and tool, which will effectively address the engineering and run-time management of reconfigurable software for ambient intelligent networked services environments.

iii) A formal modelling methodology and a consistency validation framework for capturing and analyzing the structure and run-time behavior of distributed software systems.

Berger M., Dittmann L., Caragiozidis M., Mouratidis N., Kavadias C. and Loupis M. (2008).
A COMPONENT-BASED SOFTWARE ARCHITECTURE - Reconfigurable Software for Ambient Intelligent Networked Services Environments.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 174-179
DOI: 10.5220/0001883901740179

This paper will mainly focus on ii) Component based software architecture and furthermore it will present a methodology for a Model driven design tool that will provide the means for efficiently mapping target applications on the component-based architecture. In this paper we adopt and apply in the field of pervasive (ambient intelligent) services techniques and practices on software reconfigurability presently applied in the field of industrial automation applications (Endsley, 2000), (Wang, 2002), (Whisnant, 2003). The main contribution here is the mapping to adopted standards (OSGi, 2008), (SDL, 2008), (UML, 2008) and the related tool. The organization of the paper is as follows: In the following section 2, a sample COMANCHE application is provided as an example of an OSGi based Implementation of a composite SW Component. Following this, section 3 presents the component based architecture and section 4 describes the design of OSGi compatible SW components on an OSGi Service Platform including properties of the COMANCHE design tool. Finally, section 5 is the conclusion.

2 SAMPLE COMANCHE APPLICATION

We assume the existence of a media player device equipped with the appropriate drives and interfaces so that multimedia content can be acquired from storage media as well as a network interface through which all the communications are taking place. On the device an OSGi Service Platform is deployed. There is need for having the multimedia contents, which are accessible by the device, streamed over the network under various schemes (encoding, broadcast or unicast, scaling, etc). Towards this direction a special bundle that will become the central part of the streaming service is installed and activated.

1) By default, the bundle, when activated, searches for services that provide access to multimedia content (access to the disc drive(s), USB stick or internal hard-disk). These services are offered by other bundles already activated on the platform. While this location of services is attempted, filtering aspects can be enforced to lead to the most appropriate selection. For example, there must be a requirement for locating a service that allows for concurrent sessions over the medium under their control and/or on demand navigation in the content.

2) Once the available multimedia sources have been enumerated there is need for obtaining information regarding the format of the content. This might not be performed by the orchestrating code but by services contained either in the central bundle or being active on the platform.

3) Once the multimedia sources and the corresponding information have been initialised, media encoding/decoding services are located. Again filtering parameters are enforced according to the setup of the component.

4) Finally, services for providing the final encapsulation and streaming of the encoded media are “integrated”.

5) Thereafter, the SW component that has been synthesized from the services deployed on the platform under the control of the orchestrator is ready to operate and provide the application for which it is designed.

6) Throughout its operation, service modifications may occur. These may regard additions or removal of bundles that offer functionality that can be or already utilised.

7) Similarly, the SW component may receive configuration updates that can trigger the reestablishment of its structure.

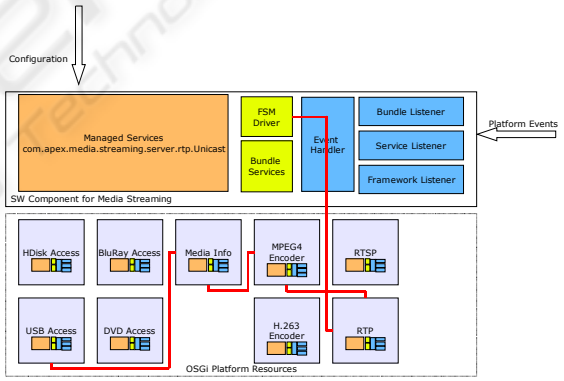

Figure 1: Example of an OSGi based Implementation of a composite SW Component.

A graphical representation of such a synthesis can be seen in Figure 1. The rectangle at the top encloses the components of a management (orchestrator) OSGi bundle (SW component) that is capable of making use of the bundles (service objects) enclosed by the rectangle at the bottom of the Figure. The red line presents a setup that provides an application that streams, via unicast RTP, media stored on a USB stick encoded by an MPEG4 encoder. In the chain presented below additional decoders may be used to produce content that can be used by the selected encoder.

3 GENERIC COMPONENT-BASED ARCHITECTURE

Components are pre-implemented software modules (or already instantiated but customizable objects within an object-oriented model) and are used as building blocks to construct the controller software.

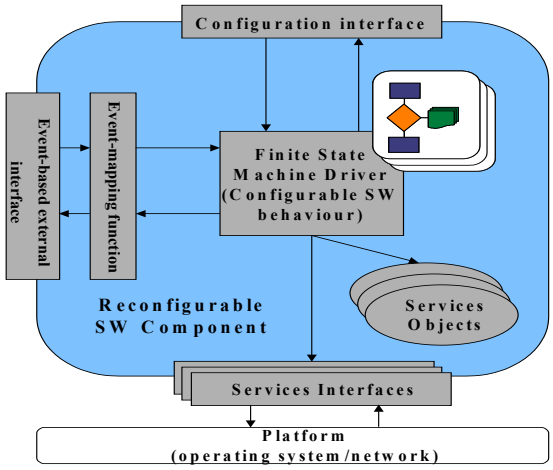

Figure 2: Component structure.

A component defines the functionality of a device or subsystem, which can be as simple as an I/O device like a position sensor, or a control algorithm like proportional-integral-derivative (PID) control, or as complex as a composed subsystem like coordinated axes. The structure of a software component includes a set of event-based external interfaces with registration and mapping mechanisms, communication ports, a control logic driver, and service protocols, as shown in Figure 2.

3.1 Finite State Machine Driver & Configuration Interface

The finite-state machine (FSM) driver, is designed to separate function definitions from control logic specifications and to support control logic reconfiguration at executable code level. The finite-state machine driver can be viewed as an interface to access and modify the control logic inside a component, which is traditionally hard coded in the component implementation. Every component that executes some control logic should have such a driver inside itself. The control logic of a component can then be fully specified as a state table for execution. A finite-state machine driver will generate commands to invoke operations of the services objects at run time according to its state table and received events. State tables can also be packed as data and passed to another component to reconfigure the receiver component's behavior remotely.

The finite-state machine driver of a component is a center piece to enable post-implementation reconfiguration of component behaviour. It invokes the service object functions based on current component state and incoming events to communication ports, which are specified by state table entries. Function calls are statically bound to internal events at implementation time, and the finite-state machine driver invokes the service object functions by generating internal events for the corresponding services objects. For each component, multiple state tables can be designed to specify different desired behaviors in different system modes. However, only one state table for a component can be active at a time.

Although the finite-state machine driver introduces additional overhead to the system as a component has to go through more steps to invoke an operation, such processing and related bindings are statically configured before normal execution. Therefore, the overhead introduced by the finite-state machine driver should be negligible to the application-level performance; it may be significant at low level due to frequent long jumps and pipeline flushes caused by the sequence of function calls to process state transitions. On the other hand, efforts and costs for software reconfiguration when application requirements change, overweigh performance for resource-rich systems such as PC-based controllers.

The component state table is configured (changed) through the configuration interface. The mechanism is detailed in section 4. If we consider the case of the sample application described in section 2, we may consider a system comprising the following four components:

- An **Orchestrator** component that is able to make use of the OSGi platform services, and its FSM driver realises the media streaming application logic. The Orchestrator software component contains the following three software components each of which makes use of the service objects enclosed by the rectangle at the bottom of Figure 3.
- A **GetMedia** component that is capable of retrieving the media content from a variety of interfaces (USB, HDrive, BlueRay, DVD, etc).
- An **Encoding** component that is capable of encoding the retrieved media in different

formats (e.g. MPEG-4, H.263, etc) in consistence with the application needs.

- A **Streaming** component that is responsible for handling encoded media streaming.

The logic of the above-mentioned SW components is illustrated in Figure 3. In this Figure, the rectangles represent SW components, and the ellipses represent service objects. For instance the **Orchestrator** SW Component makes use of the **GetMedia**, **Encoding**, and **Streaming** SW Components, as well as of the **Init** and **Stop** Service Objects. The arrows in Figure 3 represent **function calls** invoked by the FSM Driver of the Orchestrator SW component.

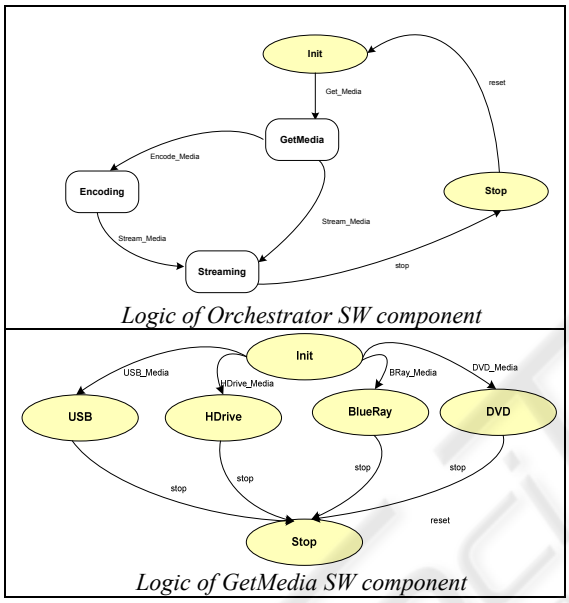

Figure 3: Logic of Orchestrator and GetMedia, SW components.

3.2 Event-based External Interfaces & Event Mapping Function

Event-Based External Interfaces are designed to expose component functionalities to the external world, i.e., define operations that can be invoked from other components.

External interfaces in the COMANCHE architecture are represented as a set of acceptable global (external) events with designated parameters. Event-based interfaces enable operations to be scheduled and ordered adaptively in distributed and parallel environments and allow components to be integrated, at executable code level, into the system.

A customizable event-mapping mechanism is present in each component to achieve the translation between global events and the component's internal representations. Such a mapping separates a component implementation from its interfaces, thus making multiple implementations of one operation possible. And so, it allows flexibility and code reuse during component development. Since the mapping is internal, it can be customized without having to know interactions with other components.

The control software constructed with the COMANCHE component model consumes less computation resources and supports more predictable execution.

3.3 Services Interfaces

Service Interfaces define execution environments of a component and so make components adaptive to different platforms. The Service Interfaces are used to communicate with the service platform and network (rather than other components). Thus, COMANCHE components implementing network drivers, user interfaces, etc, will use the service interfaces. Examples of Service Interfaces include scheduling policies, inter-process communication mechanisms, and network protocols.

Service Interfaces are implemented as a set of attributes of a component. Selection of services is implemented by assigning the desired values for the service attributes. Such selection is based on the mechanisms available on a given platform and performance (such as timing and resource) constraints of a system. The selected services will be bound to the corresponding function calls provided by the infrastructures either statically or during the software initialization.

4 DESIGN OF OSGI-BASED CONFIGURABLE SW FRAMEWORK

The implementation of a component based software architecture compliant with OSGi implies the deployment of a *Service Platform* (a Java VM in which an OSGi Framework operates) on a *Service Platform Server* (a device). In this configuration, bundles can be installed and activated as well as deactivated and removed within the execution context of the Service Platform. From the OSGi perspective the bundles and the services they provide within one running Service Platform should realise the software components in the COMANCHE architecture by implementing the following interfaces:

1. An interface (configuration) through which configuration can be achieved regarding both the behaviour and structure
2. An interface (external) that makes available the services implemented by the component towards other entities inside the execution environment
3. An interface (platform/device) through which all the transactions with respect to utilisation of the device hardware resources are performed

4.1 Definition of Structural Configuration and Generic Design Principles

The structural definition of the component consists of a sufficient orchestration of the available resources so that the outcome of the assembly of the software modules, when operating, provides the envisaged functionality that is described in Figure 2. This pattern of organisation should be maintained from the simplest to the more complex form of resource orchestration so that it can be inherently supported at any level of component integration.

According to the above, the borders of the elementary component can be defined as the minimum integration of resources that can operate as an entity that:

1. its behaviour can be configured,
2. provides an external interface through which it can serve requests from other resources, and
3. if possible interacts with physical resources of the hosting platform.

This structural model may be applied to all Java classes that are contained in a bundle, installed on a Service Platform, so that each of them constitutes a clearly defined software component or cumulatively to any number of Java packages contained in a bundle so that all together can be used as a single software component. In the latter case the borders of that component enclose all the required resources and each of the characteristics of the software component may be provided by different Java classes. Increasing the complexity, a software component may span the limits of a single bundle by accumulating resources, through OSGi compliant service registration and utilisation operations, contained in several ones. In this case the need for structural configuration arises, since the selection of the proper services for usage should follow the corresponding rules that in our case can constitute the structural configuration of the software component. Mapping, therefore, the COMANCHE concepts on the OSGi principles we conclude that a COMANCHE software component can be provided as an OSGi *Service Application* that is "a set of bundles that collectively implement a specific function used within a Service Platform".

According to the principles described above, a Software Component being able to be reconfigured with respect to both structure and behaviour and function in the context of COMANCHE architecture can be instantiated as a proper orchestration of OSGi resources.

4.2 Control Structure of a Software Component

For every synthesis of OSGi resources aiming at the composition of a COMANCHE Software Component a special bundle in the role of the orchestrator is activated. This bundle contains a package of Java Classes that will be responsible for organising the service resources on the same OSGi platform. The bundle registers a number of Managed Services, following the above described PID convention, with the Framework so that the proper configuration can be submitted to the Software Component. Upon receiving configuration dictionaries, these services perform the proper translation of the content in the following ways:
(a) assign values to parameters that regard functionality and services contained in the same bundle, (b) maintain filters to be used during service resolution, (c) modify their own registered parameters so that other services may utilise these according to the matching achieved between registered and filtering parameters.

Since the operation of a Software Component is based on the loose coupling among available services there is a constant need for maintaining the references to the available service providers so that new registrations of services that fit better specific needs or de-registration of previously located ones can be imminently detected and handled accordingly. For this purpose the bundle in the role of the orchestrator registers the corresponding Service, Bundle and Framework Listeners so that all the relevant events can be monitored and handled according to the needs of the Software Component that is synthesized under the control of this bundle. Handling of events is related mainly with updating the service references in case an event indicates any changes in the synthesis of the Software Component.

The overall structure with respect to the controlling part of a Software Component that has been presented above is presented in Figure 4.

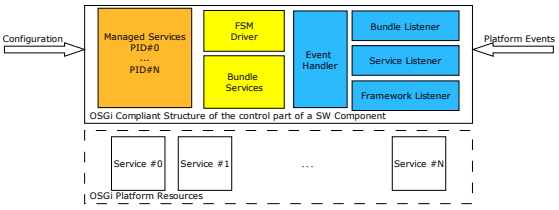

Figure 4: SW Component Control Structure.

Once the orchestrator bundle has registered and activated the listeners and the managed services, the SW component can be configured with respect to both its behaviour as this is reflected by the FSM adjustments and its structure with respect to the loose coupling with the potential service resources residing on the same platform.

4.3 COMANCHE Design Tool

The COMANCHE Design tool has the following properties:

1) Support the graphical representation of software systems making use of widely adopted standards as SDL (Simple Declarative Language), UML (Unified Modeling Language)
2) Support the composition of systems through the use and customisation of available software components.
3) provide a model (a template with no implemented functionality) of the generic reusable SW component.
4) Allow the creation and maintenance of a repository of reusable SW components to be used for the synthesis of services software. The user of the tool is able to instantiate reusable SW components using the above mentioned model implementation
5) Enable the mapping of component-based UML/SDL specifications to reconfigurable object-oriented services implementations.
6) Enable the mapping of component-based UML/SDL specifications to OWL instantiations.

The development of the COMANCHE design tool will not start from scratch, as it will exploit the implementation mapping capabilities (generation of C++/JAVA implementation code from UML/SDL specifications) of the existing development environment from the IST REMUNE project. The existing features has been complemented with innovative work on the development of techniques for (a) mapping abstract UML/SDL application specifications on component-based UML/SDL specifications, (b) mapping component-based UML/SDL specifications to efficient (in terms of performance and security), reconfigurable object-oriented (JAVA, C++) services implementations.

5 CONCLUSIONS

Based on the COMANCHE specification of a reconfigurable component-based architecture, an OSGi based configurable SW framework has been designed. The FSM Driver is a centrepiece to enable post-implementation reconfiguration of component behaviour. It invokes the object functions based on current component state and incoming events to external interfaces. This is achieved through the definition of customisable State Table entries that will be used for controlling the logic of the services exposed by the component. Development of the COMANCHE tools included techniques for mapping abstract UML/SDL application specifications on component-based UML/SDL specifications, and mapping component-based UML/SDL specifications to efficient, reconfigurable object-oriented services implementations.

ACKNOWLEDGEMENTS

This work has been performed in the framework of the IST-034909 project COMANCHE, which is partly funded by the European Commission.

REFERENCES

The COMANCHE project, www.ist-comanche.eu, 2008

E.W. Endsley, M. R. Lucas, and D. M. Tilbury. Software tools for verification of modular FSM based logic control for use in reconfigurable machining systems. *in Proc. 2000 Japan-USA Symp. Flexible Automation*, Ann Arbor, MI, July 23–26, 2000, pp. 565–568.

S. Wang, K.G Shin. Constructing reconfigurable software for machine control systems. *Robotics and Automation, IEEE Transactions on Volume 18, Issue 4,* Aug 2002 Page(s): 475 – 486

K. Whisnant, Z. T. Kalbarczyk, R. K. Iyer. A system model for dynamically reconfigurable software. *IBM Systems Journal Volume 42 , Issue 1* (January 2003) table of contents Pages: 45 - 59

The OSGi Alliance, www.osgi.org, 2008

Simple Declarative Language (SDL), sdl.ikayzo.org, 2008

Unified Modeling Language (UML), www.uml.org, 2008