OSAMI COMMONS - An open dynamic services platform for ambient intelligence

Naci Dai
Eteration A.S.
Istanbul Turkey
naci.dai@eteration.com

Alejandra Ruiz Lopez
Tecnalia
Bilbao, Spain
Alejandra.Ruiz@tecnalia.com

Elmar Zeeb
Universität Rostock
Rostock, Germany
elmar.zeeb@uni-rostock.de

Jan Krueger, Oliver Dohndorf
TU Dortmund
Dortmund, Germany
krueger@ls4.cs.uni-dortmund.de

Jesus Bermejo
Telvent
Sevilla, Spain
jesus.bermejo@telvent.com

Wolfgang Thronicke
Siemens
Paderborn, Germany
wolfgang.thronicke@siemens.com

Felix Cuadrado Latasa
Universidad Politecnica de Madrid
Madrid, Spain
fcuadrado@dit.upm.es

Christoph Fiehe, Anna Litvina
Materna Information & Communications
Dortmund Germany
cfiehe@materna.de

Isaac Agudo
Universidad de Malaga
Malaga, Spain
isaac@lcc.uma.es

Abstract

Today we live in an environment surrounded with networked converging devices. Human computer interactions are becoming personalized and a new concept of a global and cross-domain platform is emerging to exploit the full potential of the network in all business areas. In this convergence process, the software platform should be able to personalize itself dynamically in devices according to the context. OSAmI-Commons, an ITEA2 project for developing an open-source common approach to such a dynamic service-based platform, allows any type of device to connect and exchange information and services. OSAMI consortium is contributing to defining the foundations of a cross-platform open-services ecosystem. The sustainability of this platform is an objective beyond the project duration.

1. Introduction

The world is currently moving to a one-to-many social relationship between humans and computers. These are represented, among many others, by fixed and mobile phones, Wi-Fi routers, gaming consoles, MP3 players, TVs, set-top boxes and infrastructures with impressive computing and storage capabilities. A new concept of a global and cross-domain platform is emerging to exploit the full potential of the network in all business areas. In this convergence process, the software platform should be able to personalize itself dynamically in devices according to the context.

The aim of OSAmI-Commons is the design of a basic, widely applicable SOA-oriented component platform, its development, test and its provision as open source. The project consists of a number of national subprojects, each focusing on a certain field of application. The main objectives are interoperability,

maintainability, reliability, as well as automated configuration and management of devices and services across domains.

The software component platform specified by the OSGi Alliance forms the technical basis of the OSAmI platform. It provides lifecycle management for software components as well as local service interactions as defined in service-oriented architectures and will be combined with the Web Services approach in order to implement distributed, dynamically configurable, vendor-neutral and device-independent solutions

OSAmI-Commons is therefore developing an open-source common approach to such a dynamic service-based platform that allows any type of device to connect and exchange information and services. It allows service retrieval from centralized and distributed resources, enable connection between various vertical markets and allow the development of new business solutions. Services and service delivery are the driving forces in the software industry currently.

The approach proposed by OSAmI-Commons will establish an ecosystem built on an European understanding of the relationship between the citizens and the service infrastructures based on open-source and customer friendly ambient services. The project set out to address new markets through the development and application of its technology in vertical domains such as environmental sustainability, health, education, city services and tools for software development. In parallel, the consortium is contributing to defining the foundations of a cross-platform open-services ecosystem.

The sustainability of this platform is an objective beyond the project duration. OSAmI is developing open technologies for all types of device – allowing even very small devices to be perceived as services. It will enable sensor networks to be mapped to aggregated services and dynamic containers to be linked with open repositories of service implementations. Profiling of computing nodes and systems will allow reuse of aggregated service implementations. Finally, digital identity federations and security issues are being addressed for building service systems within the 'web of objects'.

2. Principles and Architecture

The subject of SOA has been studied extensively elsewhere [2,3]. SOA is also the basis for designing the OSAMI Commons architecture; the embedded space could benefit even more from using SOA since the underlying platform (Processors and Operating Systems) changes more frequently. Furthermore, the OSGi Alliance specifies a universal middleware that follows the SOA design paradigm, which can be used for the development of services and applications in embedded systems.

The integration of devices, sensors and actuators is not just an increasingly important requirement for home networks, it is also required for building more effective business applications, e.g. real-time monitoring of the location of freight in logistics applications, or in building a cost effective health care infrastructure to e.g. remotely monitor vital parameters from patients. The diversity of used protocols is huge and still increasing, which makes the integration of devices time consuming and expensive.

In this paper we outline the guidelines and general principles for defining the architecture of the OSAMI Commons system [1].

.

2.1. Motivation and Objectives

OSGi technology supports the deployment of software components to devices and the device-internal interaction between components.. Web Service technology supports the efficient and flexible construction of distributed applications and service systems.

OSAmI integrates both technologies in order to support flexibly adaptable distributed applications. The applications are based on sets of cooperating devices where the set of devices or the portfolio of provided services may dynamically change.

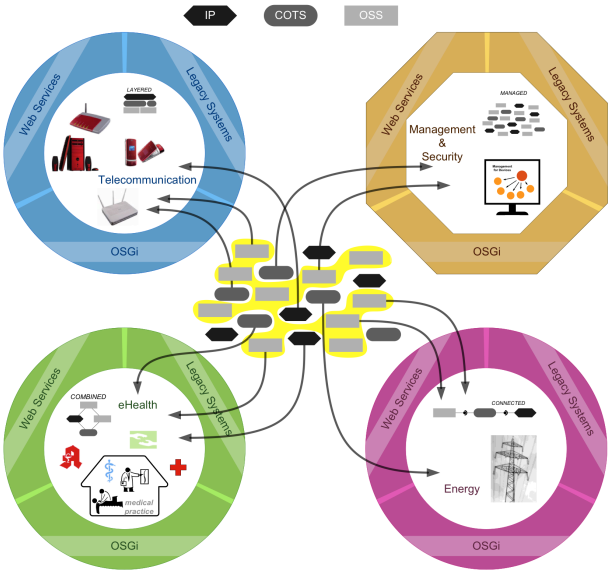


Figure 1. OSAMI-Commons

Figure 1 represents a pool of emerging software products, software services, libraries, and frameworks available as open source, commercial-off-the-shelf, and intellectual property solutions. In various domains (as examples eHealth domain, telecommunication domain, and energy domain) these software building blocks are created and can be used cross-border. All project partners support and use open source software building blocks (OSAmI Commons) and combine with own in-

house intellectual property solutions to construct new products and establish software product lines.

This is achieved by layering, connecting and combining through defined interfaces and technologies used in the projects, namely OSGi and Web Services. Management, development and security strategies control the software construction process.

The objectives of the OSAmI -Commons architecture is to supply open source software and software engineering foundations for forming an adaptable and flexible software platform for distributed service systems which:

- are based on cooperating devices and embedded systems,
- can explore and adapt to dynamically changing environments and service offers,
- can detect and adapt to dynamically changing requirements and user needs,
- can support evolutionary applications,
- can flexibly share resources, cooperate and interact with coexisting applications,
- an integrate comfortable functions for automated fault, configuration, accounting, performance and security management,
- can recognize, respect and automatically enforce user and system policies in order to perform automated system and application control functions

2.2. OSAmI Architecture

An OSAmI system carries a defined physical and logical architecture. The physical architecture defines the actual hardware and software components, their assembly and their interfaces. The logical architecture defines the application and service components, their interactions and interfaces..

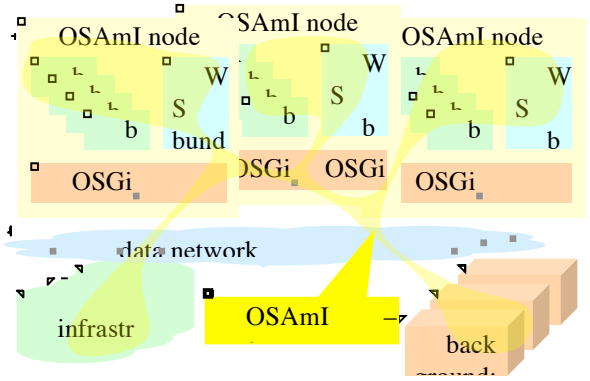
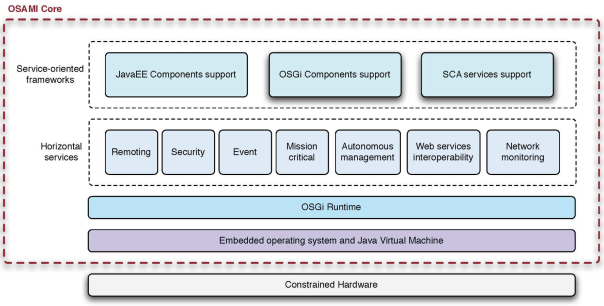


Figure 2. OSAmI System Structure

Figure 2 shows the physical architecture of an OSAmI system given by three networked nodes that are linked to background systems. The OSGi bundles implement OSAmI application components and services. The amoeba-shaped entity represents an OSAmI application.

The physical architecture of a general OSAmI system is structured into OSAmI nodes, OSAmI systems, OSAmI support infrastructure elements and OSAmI background systems:

- A node is an embedded device or (small) general-purpose computer where a local operating system, a network communication interface, a Java virtual machine implementation and the OSGi framework support the configuration, life cycle management and execution of OSGi software bundles. Different bundles may interoperate with each other via local service interfaces.
- A system is a (dynamically changing) set of nodes, OSGi bundles of which implement services and provide for corresponding Web Service interfaces. Some OSGi bundles can implement applications, which utilize a dynamically changing set of local and remote services and may cooperate (via commonly used services) with coexisting applications.
- An infrastructure element is based on a node or set of nodes and supplies general system building services, particularly software bundle repositories, service description / service type directories, service implementation directories, topologies, policy directories and configuration data bases.
- A background system is typically based on powerful and networked services provided by independent



back-office and legacy systems.

Figure 3. OSAmI Core

Figure 3 shows the OSAmI node core which is considered as the OSAmI services runtime. OSAmI Core provides management capabilities allowing to dynamically build (start, configure, stop, manage) minimal application server platforms.

Logically, an OSAmI system is a set of applications and services that can be distributed across a set of nodes. Applications provide the desired functionality of a system. Services in turn provide their functionality for the applications or other services.

Due to the device orientation there are two major forms of applications:

- Distributed applications may consist of cooperating components that are hosted by different nodes. They typically utilize distributed services and focus on an integrated support of user and / or device groups.

- Local device based applications mainly utilize local services and are oriented at the scope and personal owner of a device.

Moreover, due to the device orientation, the services can be structured into four layers:

- Distributed services take advantage of the data network. They can explore the current environment and utilize remote services.
- System-based services contain inherent know-how about the system itself to perform their task.
- Device based monitoring and control services use local sensor and actor services in order to perform enhanced, application-oriented monitoring and control functions.
- Basic sensor and actor services are hosted on (small) sensor and actor devices providing for real-time interfaces to the physical world. They are not necessarily web services but may introduce an arbitrary protocol which has to be dealt with.

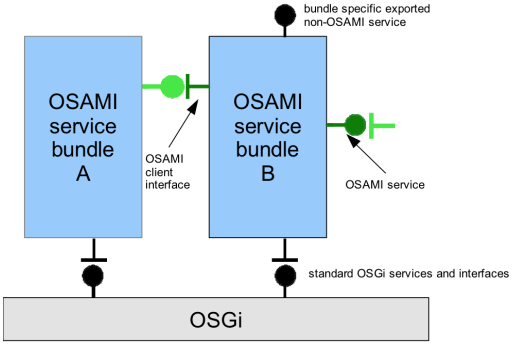


Figure 4. An OSAmI Compliant OSGi Bundle

The OSAmI services integrate seamlessly in the OSGi service architecture. They rely on the OSGi technology to define specific OSAmI service interfaces.

OSAmI bundle B in Figure 4 implements the client interface for services provided by bundle A, in turn it implements and provides OSAmI services B.

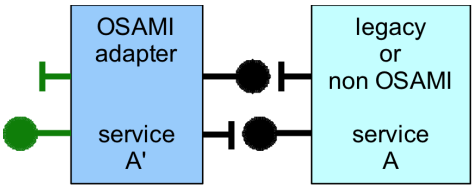


Figure 5. OSAmI-"fication" of Service A Using an OSAmI Adapter

Creating OSAmI-aware OSGi services requires implementing the OSAmI service interfaces. This can be done from scratch, delivering a pure OSAmI service, or use non-OSAmI services which are accessed through an OSAmI compliant adapter as depicted in Figure 5.

3. Platform and Tools

OSAmI platform comprises the design and development of the run-time components of the OSAMI architecture as well as those of supporting tools. The components utilize the existing basic OSGi platform and provide building blocks for applications. The tools support the development, application configuration and management.

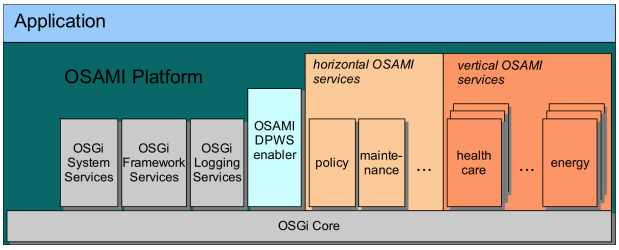


Figure 6. OSAmI Commons in Context

Figure 6 shows the OSAmI platform in the context of an OSGi/OSAmI application. OSAmI commons consist of the following new components and services:

- The DPWS [] enabler is a generic web-service adapter for OSAmI services. Using DPWS remote devices and services can be discovered, integrated and used.
- Horizontal OSAmI services provide domain-neutral features like serviceability, remote management extensions and ambient services (localization, context-awareness) based on standardized OSAmI-APIs.
- Vertical OSAmI services utilize these components to provide building blocks for dedicated applications domains.

The platform supports inter domain interoperability, i.e. based on the common horizontal architecture, applications are able to perform device and service discovery, communication and interactions even across different domains, such as a car system that is able to access home content or infotainment devices can be used to interact with patients after leaving the hospital. The following selection of Horizontal OSAmI Services form describe some of the primary capabilities of the platform

3.1. Context Store

The Context Store is a service, which manages arbitrary events persistently. In contrast to the standard OSGi EventAdmin specification, the context store thus addresses the complex processing of context events, which denote semantic information for ambient applications. The service supports automatic registration and dependency injection to interface with context sources and context processing services according to the standard OSGi mechanisms. In order to evaluate complex context conditions which may arise the developer can easily access previous context events from

the past due their time-ordered nature. A convenient SQL-like query mechanism allows the formulation of complex queries on the context store.

The context store contains additionally a two-way bridge between OSGi eventing thus allowing persistent management and access of these events as well as interfacing with services that are not context store aware by design. The underlying modular plug-in architecture also provides support for building distributed context store applications with additional pluggable communication support (currently XMPP is supported).

The context store is the key information hub in the Siemens mobile demonstrator and is the basis for the incident management component.

3.2. OSAmI Device Integration

The OSAmI Device Integration [5] is the central component in the OSAmI platform to cope with the large number of standards and protocols to integrate devices into modern ambient applications by minimizing required implementation and maintenance efforts. OSGi itself provides a mechanism that addresses device integration called OSGi Device Access. This mechanism is specified in the OSGi compendium specification [9]. OSGi Device Access solves the problem of automatic matching and loading of drivers at runtime for hot plugging technologies like USB. This mechanism can be compared to traditional dynamic device driver registries as present in operating systems like Windows, Mac OSX or Linux. From the perspective of the OSAmI project, this mechanism does lack one important feature. It does not decouple applications from device communication technologies and standards. As this is a main requirement for the OSAmI Platform to exchange devices and communication technologies there is a need for a more abstract device access mechanism that and a specification that defines how to model interfaces between drivers and applications.

The OSAmI Device Integration basically defines an abstract level of device access on top of the OSGi Device Access. Applications are decoupled from underlying device communication technologies, by applying service-oriented principles like loose coupling to the interfaces between applications and drivers. This can be achieved by using mechanisms available in OSGi like the service registry and the service mechanism. Applications often need access to high level functionality like a temperature value and can use different devices with the same interface given that the interface is technology independent. Depending on the underlying technology, the interface can be provided by the driver or directly mapped to high-level device communication technologies.

The OSAmI Device Integration consists of an implementation part and a specification part that defines four main parts: an extensions to the OSGi Device Access API, the concept of the abstract function level device access, design guides how to model interfaces of services on devices and design guides, how to map these interfaces to SOAP Web Services.

The technology independence is the main advantage of the OSAmI Device Integration and enables the development of ambient applications that are not coupled to specific device communication technologies but can work with several generations of technologies.

3.3. OSAmI System Management

3.3.1. Incident Management Service

One important task for system management is to detect system malfunctions or impending problems during operation. The incident management service is based on the context store and uses a two-step approach: Incident detectors monitor critical system variables and behavior. Since an arbitrary number of such "probes" can be realized, a fine tuned detection can be tailored to each specific system. Upon incident detection the incident manager service kicks in and uses a rule-based approach to automatically "heal" the system. For example if the battery is low the frequency of data updates from sensors can be reduced or components with high energy demands can be disabled temporarily. The key of the incident management system is that is modular built on OSGi service principles thus allowing dynamic deployment and configuration during runtime. So as new "recipes" for autonomous management are available they can easily be installed locally during operation.

Because the incident management service relies on the context store for operation it can easily be configured to work distributed. System management can thus be separated into a local and a remote work task, allowing complex analysis and incident management be handled remotely and only the corrective actions being deployed on the local system.

3.3.2. Policy-Based Management

Device-based service systems comprise devices with only limited resources. Within these systems, the technical management is faced with critical exceptions as well as spontaneously changing conditions and environments that require a rapid and adequate response. Moreover, the users are typically not familiar with the methods and procedures of technical management. Therefore, a comprehensive automated and light-weight management system is mandatory that ensures the proper runtime operation of the system in a fast, but also predictable and reliable manner.

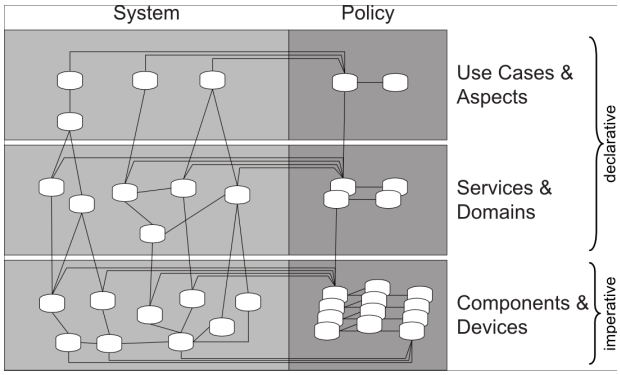


Figure 7. Model Layers and Policy Refinement

Policy-based management has been acknowledged as a promising way for management automation [6]. It enhances the approach of model-based management (MBM) [7] and separates policy definition and refinement at design time from efficient policy enforcement at runtime. We make use of declarative policies to impose abstract requirements to the system and imperative ones that ensure their fulfillment at runtime. The management objectives are defined by concise and easily understandable abstract high-level policies from which the detailed low-level policies are derived according to valid refinement relations. The managed system and its high-level policies are planned at design time by means of an object-oriented hierarchically organized system model. It comprises three layers, each describing the system with a different level of abstraction: the top layer reflects the use cases, the middle layer the service infrastructure, and the bottom layer the implementing software components. The abstract high-level policies describe requirements allocated to the use cases and functions, the middle-level policies represent service-level objectives assigned to the corresponding services, the low-level policies specify technical ECA rules applied to the software components at runtime. Rule conditions and actions are defined over the management variables provided by the software components. These variables form the Management Information Base (MIB) [8] of the system and are subdivided into status and configuration variables. Status variables describe the management-relevant state of the component; they are set from the application logic and read by the management. Configuration variables are means of configuring the component; they are set by the management and read from the application logic.

Model-Based Management

In the design phase, the system is modeled on three different layers, each comprising a self-contained and independent system model. The models of adjacent layers are connected through refinement relations. A refinement relation associates system elements with all those elements representing them on the next lower model layer. We distinguish between the following layers:

- **Use Cases & Aspects (U&A) -** The top layer provides an abstract view of the system. It is modeled in terms of use cases, assets, actors, and functions. The system behavior is specified by means of concise and easily understandable requirements according to various aspects like security, reliability, and performance.
- **Services & Domains (S&D) -** The middle layer represents the system from a service-oriented point of view. The system is described in terms of clients, services, and applications relations providing the functions defined on the upper layer.
- **Components & Devices (C&D) -** The bottom layer contains the technical view of the system. It contains the actual software components, resources, and devices which compose the system at runtime. The software components provide the services defined on the middle layer.

The automated refinement process exploits the model and its refinement relations introduced by the model designer. The higher-level policies are refined to the lower-level ones by traversing the model from top to bottom. This process is governed by an aspect-specific set of pattern instances. We distinguish between the following three pattern types:

- **Refinement Pattern (RP) -** A refinement pattern instance specifies the refinement relation between a model element of an upper layer and all those model elements of the next lower layer which contribute to its implementation or realization.
- **Evaluation Pattern (EP) -** An evaluation pattern instance supports the introduction of an abstract status variable within a model layer. An abstract status variable aggregates the values of a set of status variables according to a specific point of view. The evaluation pattern defines the corresponding arithmetic expression.
- **Control Pattern (CP) -** A control pattern specifies how higher-level policies are to be enforced on the more technical level. It represents the strategy which has to be applied for the refinement of higher-level policies to lower-level ones.

Policy-Based Runtime Management

The runtime management system [9] includes component managers enforcing the derived low-level policies. A component manager monitors, controls, and configures its assigned software components. Each software component is assigned to exactly one component manager, whereas a component manager might be responsible for more than one component. A component manager provides the following three management services: the configuration service for accessing management variables, the policy service for requesting evaluations of policy expressions and decisions in the context of the current system state, and

the binding service for the establishment of quality-assured service bindings.

Low-level policies are defined over management variables, event parameters, operations, and constants and have the form of conditions, expressions, and variable assignments. At runtime, they are described by efficiently executable byte code. The following policy types can be distinguished:

- **Policy Expression -** A Policy Expression is an expression over management variables, constants, and operations. It is evaluated on demand and allows to assess the current system state. Based on the result, a component can react accordingly by adapting its program flow.
- **Policy Condition -** A Policy Condition is a special case of a Policy Expression. The return data type is restricted to Boolean. Policy decisions are realized by means of Policy Conditions.
- **Policy Rule -** A Policy Rule represents an ECA rule specifying the actions to be performed when a certain event occurs. The actions are restricted to modifications of configuration variable assignments. The reactive management is performed by means of Policy Rules in order to achieve a reliable and adaptive system behavior.

3.4. Security

The security threats that concern the deployment of the bundles can be of three types: 1) The presence of malicious repositories for publishing the bundles. 2)"Man in the Middle" attacks where an attacker modifies a bundle or substitutes it completely with another when uploading or downloading. 3) It is possible that an attacker accesses the bundle repository or the client platform in order to modify the components stored there.

Based on an analysis of these scenarios, we have identified the main requirements are as follows:

- Simplicity
- Mobility/Portability
- Authentication of programmer
- Integrity of the bundles
- Generating secure signatures

Our aim was to achieve a simple and robust platform for the signatures of software components. The impact on the programmer (in terms of time and complexity) who uploads a bundle to a OSGi repository has to be of the absolute minimum, without this complicating the verification process for the users or the devices.

Limited mobility is provided for the programmers by the fact that the distribution of bundles does not require the use of personal certificates because they can upload the bundles to the OSGi repositories simply by accessing it with their usual username and password.

It is important to consider that in the programming model of free software, all the programmers are dynamic and the involvement of each one is variable. So not only do the habitual programmers have to be accommodated for but also any others that only contribute a small part to the project. For this reason an authentication mechanism for the repository based on identity federation, in such a way that the repository doesn't have to directly authenticate the programmer but rather the identity provider is what provides the necessary information.

The proposed solution to use temporary certificates (short-lived) generated from the attribute set that defines the programmer's identity provided by the identity provider for the code signature and thereby providing the integrity of the same. Furthermore because they are temporary certificates they can be used from any device irrespective of its geographical situation thereby avoiding the security requirement of always carrying a pair of keys, public and private for the signature of the components.

The use of a federation identity means that a programmer only has to be authenticated in the same way (and with the same credentials) as for other services in order to store a signed bundle. Naturally, the OSGi has authorization policies which determine the visibility of these bundles to other users, but we consider this process beyond the scope of this article.

For generating secure signatures by the user, this platform allows them to generate a pair of keys locally and the OSGi repository takes on the role of trusted third party certifying these keys, and certifying that the legitimate user is who they say they are. This scheme is different to the distribution system of Debian where the repositories sign its packets permitting their posterior verification, because in our scheme the verification of a bundle, the trust filter is done at programmer level and not by the repository.

4. OSAmI Tools

OSAmI Tools are part of an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing OSAmI Applications. Tools are strongly linked to productivity. An open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle is critical for the wide spread adoption of the architecture, and or building applications based on the OSAmI artefacts. OSAmI Tools are constructed on existing tools platforms such Eclipse. They extend the existing tools by adding support for service platforms needed by OSAmI applications, such as the Micro-JOnAS, Eclipse Equinox, Apache Felix OSGi frameworks, and provide integrated management agents for OSGi service platforms. OSAmI Tools include tools for building Web Application bundles, Service Creation (OSGi), service implementation using Java, Web Services and SOA.

OSAmI Tools consider the following complementary scenarios for developing ambient systems:

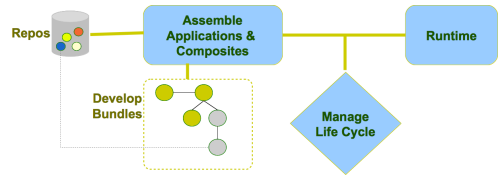


Figure 8. Tool Interactions in the life cycle of an OSAmI Application

Figure 8 describes the positioning of the tools in the life-cycle of an OSAmI application. OSAmI promotes SOA; this means building frameworks and exemplary extensible tools that enable the design, configuration, assembly, deployment, monitoring, and management of software designed around a Service Oriented Architecture (SOA).

4.1. Development Roles and Tool Categories

OSAmI Commons defines the following taxonomy to describe and categorize tools and runtimes available for building, deploying and managing OSAmI Applications:

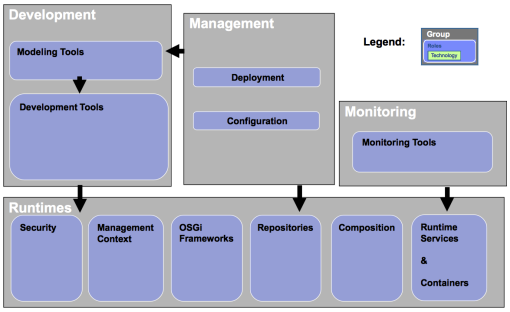


Figure 9. OSAmI Tools

- **Development -** These tools are used for creating new OSAmI components, services and applications. OSGi is the primary underlying Java technology for OSAmI. Development tools address different parts of OSGi development life cycle. Development tools are further subdivided into categories such as modelling and traditional programmer development tools. "***Building blocks centric***" tools that are focussed on accelerating applications development though supporting the reuse of OSAmI building blocks:
  - Bundle Development
  - Service Development
  - Application Development
- **Modelling** - Modelling centric tools, including, optionally, OSAmI methodology support The functionality includes Graphical editing of models, persistence support, teamwork and versioning of models support, model transformation, traceability and synchronization, documentation and ddl dode generation using models as input and development process support.
- **Management -** These tools are used for inspecting and managing OSGi framework (runtime) instances, and for controlling and inspecting bundles, services and applications that are deployed on these instances. These tools can be in many forms such as tools that run as Web applications, Rich desktop applications, TTY consoles, remote protocols, etc. Some management tools are specific to application and service vertical and they are used for controlling the runtime behaviours of these systems. Management tools are relevant both during development and runtime.
- **Monitoring -** Monitoring tools can be considered a special subset of management tools, but they are important enough to be considered in their own category. OSGi framework instances (runtimes) are fairly dynamic environments, new bundle maybe installed, started, old ones maybe stopped and removed. There maybe errors and messages that are logged during any of these stages. Monitoring tools provide feedback and diagnostics that help identify issues, progress, resolve them and find solutions.
- **Runtime -** Tools that are part of the Runtime that provide the environment and infrastructure to run OSAmI applications, such as OSGi frameworks and Repositories.

OSAmI Tools are also categorized based on the following development approaches as described in Figure 10:

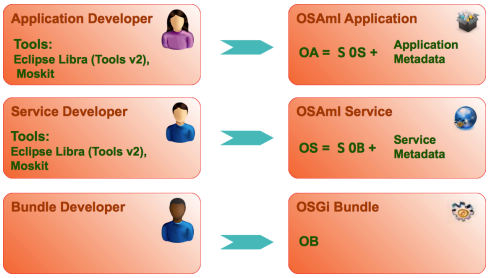


Figure 10. Development Roles and Approaches

- **Bundle-Oriented** - This approach is the most common one. Bundles are the conceptual units of modularity. Development units are one-to-one mapping of the bundles that are used by the runtimes. The developer typically creates a project for each one of the bundles under development. One or more development workspaces are setup for each peer in a distributed system. Bundles are deployed to OSGi frameworks based on deployment plans (derived from the bundle compositions, dependencies and target platforms [3].

- **Service-Oriented -** In this approach service-creation processes are followed (i.e. SOA). Typically this starts with the definition of a contract (top-down approach), or exposing some operations of Java

components as services (bottom-up). Services are the conceptual units of modularity. Deployment plans are prepared to deploy and run services as unit. All the bundles (generated or otherwise) must be added to the deployment plan to make a service available.

- **Application-Oriented -** This is the legacy approach for building systems, such as the Web applications in Java EE. Applications are the conceptual units of modularity. As it is the case in services, a set of bundles makes the applications. This "meta-data" is captured as an assembly and used for generating different styles of application configurations. A Deployment plan computes what bundles are needed from the application metadata.

4.2. Development Tools

4.2.1. Eclipse Libra

The results of OSAmI Development tooling works are contributed to open source community with the Eclipse Libra project [10]. Eclipse Libra is an open source project under the Eclipse Web Tools Platform Container Project. It has been created by the participation of OSAmI partners. Eclipse Libra provides extensive list of standard tools for OSGi Enterprise application development and in particular tools that integrate the existing WTP [11] and PDE [12] tooling so that OSGi Enterprise applications can be developed with both tooling at the same time.

4.2.2. License Management

A difficulty associated with assembling services and components in creating software systems is the large number of different licenses with particular characteristics that are not always compatible [13, 14]. It is very hard for developers to understand the implications of reusing and integrating different licences, which is a barrier for innovation.

The License Manager Tool's aim is providing developers a tool to guide them in the implications of integrating different developments. The License manager Tool defines 3 key concepts that explain how the compatibility is studied:

- **Topic -** This can be equivalent to the rights that the author wants to grant users when the software is released. It can easily be mapped with the criteria used by the Open Source Initiative to certify a license to be an open source license.
- **License -** A License is defined by its name (just for presentation purposes) and a link where it is published. The link is used in order to identify the license. The tool stores a list with different links where licenses are being published.
- **Compatibility -** The compatibility is defined by the relation between two licenses and a topic. That way it can be distinguished if a license is compatible with another around one topic. Imagine a library "L1" which is released under LGPL and the developer wants to develop a project that links dynamically L1, under Apache version 2 license. Then the compatibility is define as "License origin: LGPL, License destiny: Apache version 2 and Topic: dynamic link".

The License Manager Tool manages the information about the project license using the license specified on the OSGi Bundles. The tool requires the use of the use of the header, "Bundle-License", existing in OSGi Specification version 4.3.

OSAmI Commons have also created a license wiki that works similarly to the Wikipedia, letting anyone inform about incompatibilities among licenses and references to the sources.

Figure 11. License Manager

Moreover there is an API so the license manager can be used in other environments, as an example, it is being used on a repository client in order to assure that all the bundles being downloaded are compatible. The tool does not replace the lawyers but makes developers life a bit easier.

4.3. OSAmI Deployment Repository

The OSAmI project aims to enable seamless creation of bundle and service ecosystems. The OSAmI Deployment Repository contributes to that goal by registering the available functionality from OSAmI bundles. This enables the creation of OSAmI applications through service discovery and automated deployment.

4.3.1. Resource Deployment Model

The diversity among service communication models (web services, SCA, Declarative Services, OSGi services) for OSAmI applications, as well as the wide range of physical platforms where they are deployed (including embedded and mobile devices, as well as enterprise application servers) demands the use of a common set of abstractions for managing OSAmI components at deployment time. The repository uses an internal resource model, based on the OMG D&C Deployment and Configuration and CIM (Common Information Model) management model specifications. The model defines the base concept of resources, and allows each deployment unit (a bundle) to declare what resources provides to the platform, what logical requirements must be solved (including locality restrictions), and what constraints must be fulfilled by the underlying hardware for it to function correctly. A typing mechanism is adopted to categorize bundles,

resources and services developed with specific technologies and languages.

4.3.2. Development integration

The repository integrates into the OSAmI development lifecycle through multiple access mechanisms. First, it provides Eclipse and Maven plugins to easily upload newly developed bundles to the repository. Additionally, it presents a rich web interface that allows users to browse, discover and upload new elements. For each imported artefact the repository analyses its configuration metadata files and generates automatically an associated deployment unit model instance, with all the information relevant to its deployment.

4.3.3. Deployment capabilities

The repository enables deployment operations for OSAmI deployment and configuration agents. These elements can access the repository functionality through multiple means. The repository provides a complete REST remote interface, as well as a OBR (OSGi Bundle Repository) [15] service implementing the standard specification. Deployment requests trigger the execution of dependency resolution activities in order to find a feasible software combination that satisfies the requirements imposed by the bundles. The resolution process adopts a multi-faceted approach [16], integrating into the calculations aspects such as software logical dependencies, open source license compatibility, or hardware constraint satisfaction. Over this process the repository performs auxiliary requests to the external repositories from the main open source development communities, in order to satisfy bundle requirements.

5. Conclusion

In the long term, OSAmI partners are working on the foundations for an open-service ecosystem. The project will make possible to define the role that OSAmI technology can play in the context of the global roadmap of the 'web of objects'. An important element of OSAmI is a series of 'personality' demonstrators that show how users perceive the platform. The key ambient intelligence demonstrators cover:

- **Efficient energy -** This will contribute to a better understanding of energy loss in new generation buildings within a broader effort to move from energy-consuming to energy-producing buildings. OSAmI can contribute to a sustainable approach by simplifying the sharing and reuse of knowledge;
- **Assisted living -**This will contribute to rehabilitation for patients recovering from cardiac problems. OSAmI technology has a strong potential for cutting the cost of advanced solutions in healthcare increasing the quality of life and even saving lives;
- **Edutainment -** This will help improve education reinforcing the relationship with academia by recognising its importance in the innovation process when disruptive changes are happening;
- **City services -** An enabler for this market to take-off OSAmI technology is expected to provide higher quality services to citizens.

References

[1] OSAMI Consortium, "Principles, Architecture and Horizontal Services" v1.0, *ITEA2 Project ip0701 Deliverable,* ITEA2, 2010.

[2] Thomas Erl, "SOA Principles of Service Design", Prentice Hall/PearsonPTR, 2005.

[3] OSGi Alliance: OSGi Service Platform, Service Compendium, Release 4, Version 4.2., 2009.

[4] OASIS Consortium, Devices Profile for Web Services Specification v1.1, 2009

[5] OSAMI Consortium, "OSAmI Device Integration" v1.0, *ITEA2 Project ip0701 Deliverable,* , ITEA2, 2011.

[6] M. Sloman, "Policy Driven Management for Distributed Systems," Journal of Network and Systems Management, vol. 2, pp. 333–360, 1994

[7] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F.-J. Stewing, "Policy-Based Management for Resource-Constrained Devices and Systems," in Proc. of the 11th IEEE Int. Symposium on Policies for Distributed Systems and Networks (POLICY 2010), Fairfax, VA, USA, 2010.

[8] ISO, "Iso/iec 7498-4: Information processing systems – open systems interconnection – basic reference model – part 4: Management framework," 1989

[9] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F.-J. Stewing, "Lightweight Policy-Based Management of Quality-Assured, Device-Based Service Systems," in Proc. of the IEEE 24th Int. Conf. on Advanced Information Networking and Applications (AINA 2010). Perth, Australia: IEEE Computer Society, 2010, pp. 526–531.

[10] Eclipse Libra Project, http://www.eclipse.org/libra

[11] Eclipse WTP Project, http://www.eclipse.org/webtools

[12] Eclipse PDE Project, http://www.eclipse.org/pde

[13] Stallman, Richard M. The Free Software Definition. Free Software Foundation. 2005.

[14] Lerner, J. & Tirole, J.: "The Scope of Open Source Licensing", The Journal of Law, Economics, and Organization, vol. 21, p. 20–56, 2005

[15] OSGi Alliance and R. S. Hall. Bundle repository. OSGi Alliance RFC 112, 2006

[16] R. García, F. Cuadrado, and Juan C. Dueñas "A Model-Based Repository for Open Source Service and Component Integration". 6th International Conference on Software and Data Technologies, ICSOFT Seville, Spain. 2011