Engineering Virtual Domain-Specific Service Platforms

Specific Targeted Research Project: FP7-ICT-2009-5 / 257483

Implementation of a Family of Service Platforms and Applications (Final)

Abstract

This document describes the implementation of a family of service platforms as outlined in the case studies of the project. This includes the integration among those platforms.

Document ID:	INDENICA – D5.3.2
Deliverable Number:	D5.3.2
Work Package:	5
Type:	Deliverable
Dissemination Level:	PU
Status:	final
Version:	1.0
Date:	2013-09-30
Author(s):	SIE, SAP, NDL

Project Start Date: October 1st2010, Duration: 36months

Version History

0.1	5 Sep 2013	Document Structure
0.2	26 Sep 2013	Draft Version
1.0	11. Oct 2013	Final Version

Document Properties

The spell checking language for this document is set to UK English.

Table of Contents

1 Introduction

This document describes the implementation of the service platform from the partners SAP, Siemens and NextDayLab (NDL). As outlined in previous documents, SAP is responsible for a Yard Management Platform, Siemens provides the Warehouse Management Platform and NDL provides Remote Maintenance Platform.

The document is structured as follows:

- The first section illustrates the integration scenario for both platforms. This forms the background for the technical implementation of the platforms itself as well as the integration among them (Virtual Service Platform, VSP).

- The next two sections provide details on the technical implementation of the two base platforms (YMS, WMS and RMS).

- The integration is shortly outlined in the following section.

2 Integration Scenarios

The integration of the Yard Management System (YMS), the Warehouse Management System (WMS) and the Remote Maintenance System (RMS) is done in a scenario taken from the real world: There is a warehouse with a yard attached. In the yard trucks are managed, meaning they are checked in and out. Additional, yard jockeys, who handle processes in the yard itself, need to be managed. Responsibility of the RMS is to provide the voice and video connectivity between the components systems. The warehouse takes care of loading, unloading and storing the goods from the trailers. More detailed information can be found in D5.1.

For illustration purpose the "unloading an error condition" process will be explained in detail. In this scenario a truck arrives at an appointed time and date, the truck arrives and the call between truck driver and security guard is established, the warehouse is busy, call between the system operator and the WMS support team is made and text notification is sent to the driver, trailer is taken to a free waiting bay, when the warehouse is free again the trailer is brought to the dock, unloaded, taken from the dock and checks out again. The whole process can be split into four main processes: check in, error handling, unloading, check out. These will be described in more detail in the following.

Some excerpts of the process' sequence diagram are used. Following abbreviations are used in the description:

ERP: Enterprise Resource Planning system, no actual part of the scenario but necessary for modelling a real world process. The system provides data on trailer's goods.

VSP: The Virtual Service Platform.

OP: Operator panel, an application on top of the VSP with which a person supervises and, if necessary, manages the whole process.

YUI/WUI: YMS/WMS user interface for the users of the systems, e.g. drivers and jockeys.

2.1 Check In

The check in-phase (see Figure 1) handles newly arrived trucks as shown in Figure 1. The first step is to create a new appointment, which informs the YMS that at some specific time a truck will arrive at the yard for unloading. Information provided is time and date of arrival, the driver driving the truck, the ID of the order and the expected loading-time. This is done manually by the operator.

As soon as the truck arrives at the Yard the "truckArrived" event is sent (automatically or by the gatekeeper) and the YMS informs the VSP that a truck (including its booking/order ID) has arrived. The VSP automatically establishes the call between the truck driver and the security guard in order to ensure his or her identity. Next, the VSP asks the ERP for necessary actions for the booking, which in

this case is just “unloading”. Then it checks, whether the WMS is able to handle the order. However, the WMS denies this for some reason.

After receiving the “no!” answer from the WMS, the VSP creates the phone call between the system operator and the WMS support team to force them to negotiate the approximate time when the WMS will become ready. The YMS then creates a new jockey task, so one of the free yard jockeys can take care of fetching the trailer and getting it to a free waiting bay. The jockey picks the task and notifies the YMS when it is done.

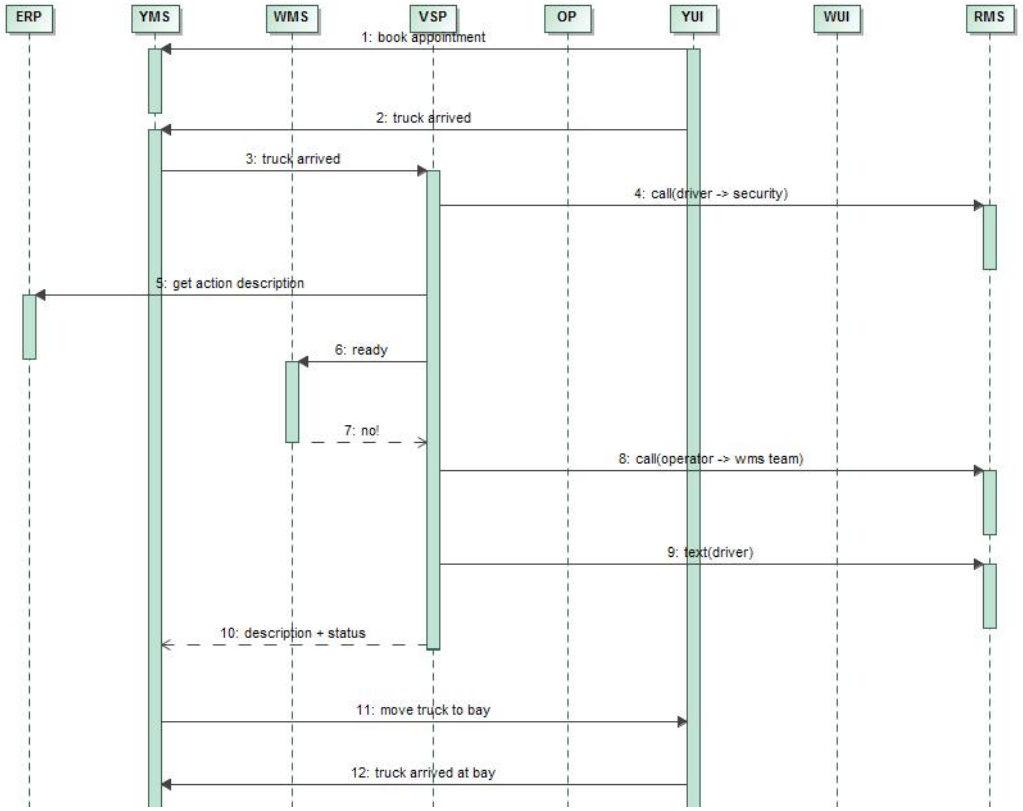

Figure 1Check-in Process

2.2 Error Handling

When an error occurs (see Figure 2), as happened upon check in, the operator, some person supervising the system, has to manually resolve the problem. In this case the operator is to be informed by the WMS support team when the approached problem could be approximately resolved and after that the RMS is dispatched by the VSP to notify the truck driver.

The VSP will inform the YMS, that the trailer can now be brought to the dock, for which the YMS creates a new jockey task. After a jockey has picked the task and marked it as done, the YMS notifies the VSP that unloading can be started.

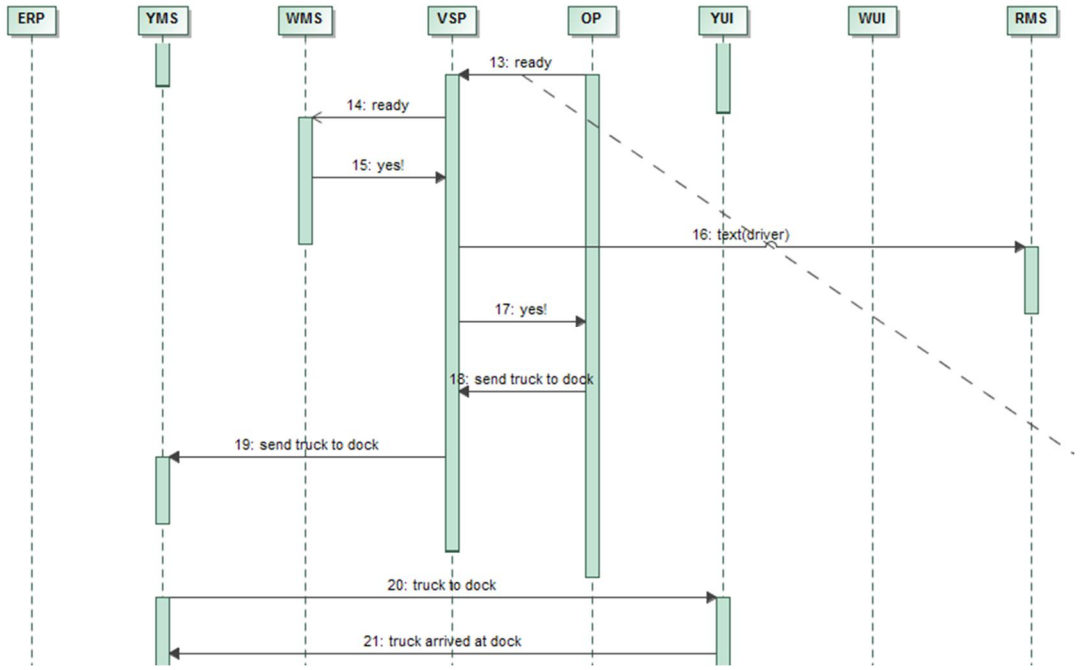

Figure 2 Error Handling Process

2.3 Unloading

When being notified that unloading of the track can be started, the VSP connects the truck driver to the camera at the unloading bay and creates a new unloading order (see Figure 3). Thanks to the camera connection during the unloading process the truck driver is continuously informed about the unloading progress, so he/she can be ready to take the truck right at the moment when the unloading ends. Processing of the unloading order is performed by the operator using the graphical user interface of the WMS – in the sequence diagram called WUI. First, the operator gets an empty box and inserts one or several from the truck unloaded products into it. Then he triggers registration of this box in the system using WUI. Next, the operator triggers storage request of the registered box into a suitable location in the warehouse. These two steps are done in a loop, till all products from the unloading order are processed. In the end, the operator uses the WUI to let the WMS know that the unloading was finished. The WMS in its turn informs the VSP that the unloading order has been successfully processed.

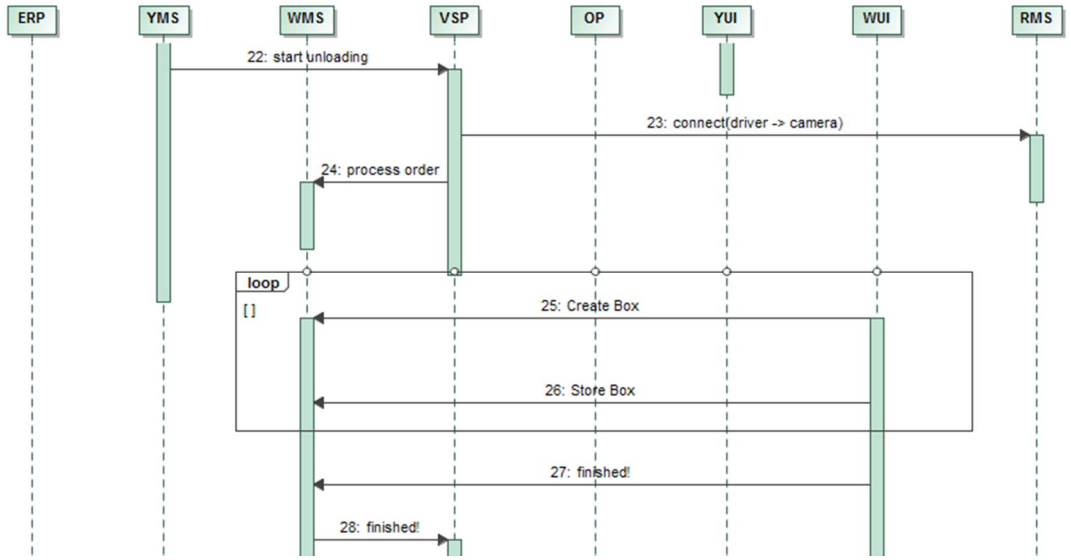

Figure 3 Unloading Process

2.4 Check Out

After unloading is finished (see Figure 4) the VSP establishes again the call between truck driver and the security guard and informs the operator that the error is resolved and everything is back to normal again. At the same time it notifies the YMS that the trailer can be fetched from the dock and the YMS creates a new jockey task for fetching the trailer from the dock. A jockey picks the task and marks it as finished as soon as he is done.

The ultimate step is the driver leaving the yard along with the trailer. This event is fired automatically or by the gatekeeper and confirms that the truck has left.

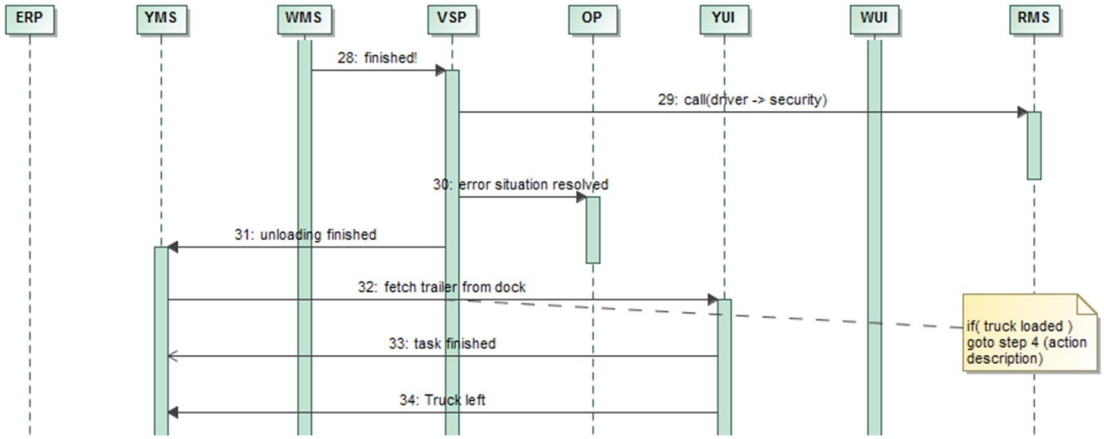

Figure 4 Check-Out Process

3 Warehouse Management System

This section includes a description of Warehouse Management System (WMS) its services, technologies and implementation details. And it describes integration with the virtual service platform of INDENICA.

3.1 Overall Architecture

WMS deals with storage pick up and flow of products in a warehouse. To achieve this, the system contains of tree main components: Warehouse Management System (WMS Core), Conveyor Control System (CCS) and Simulation (see Figure 5)

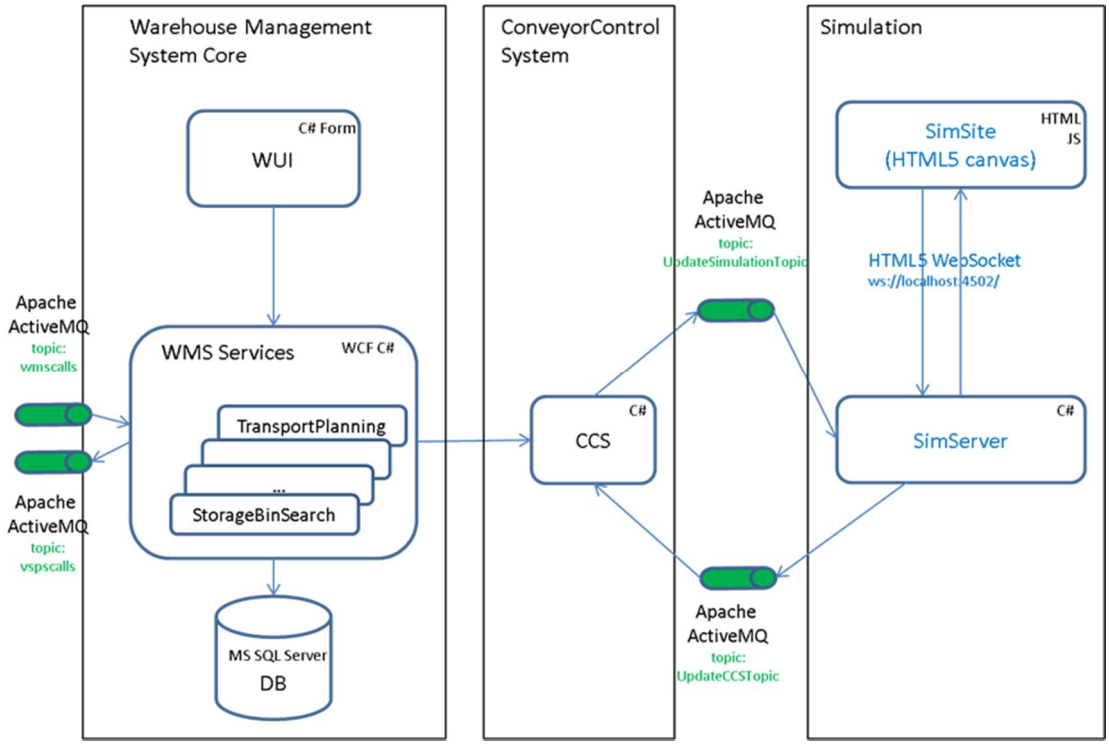

Figure 5 Warehouse Management System (WMS) Architecture

WMS Core is a top level system. It contains of several services. For example *StorageBinSearch* service searches for a suitable location for storing a box with products in the warehouse. The WMS services complete different mostly simple tasks and can be invoked separately or in a composite way to complete more complicated tasks. These services can be accessed by the operator of the system using the Warehouse User Interface (WUI, see Figure 6). Information about what products is in the warehouse and in which location is stored in a database.

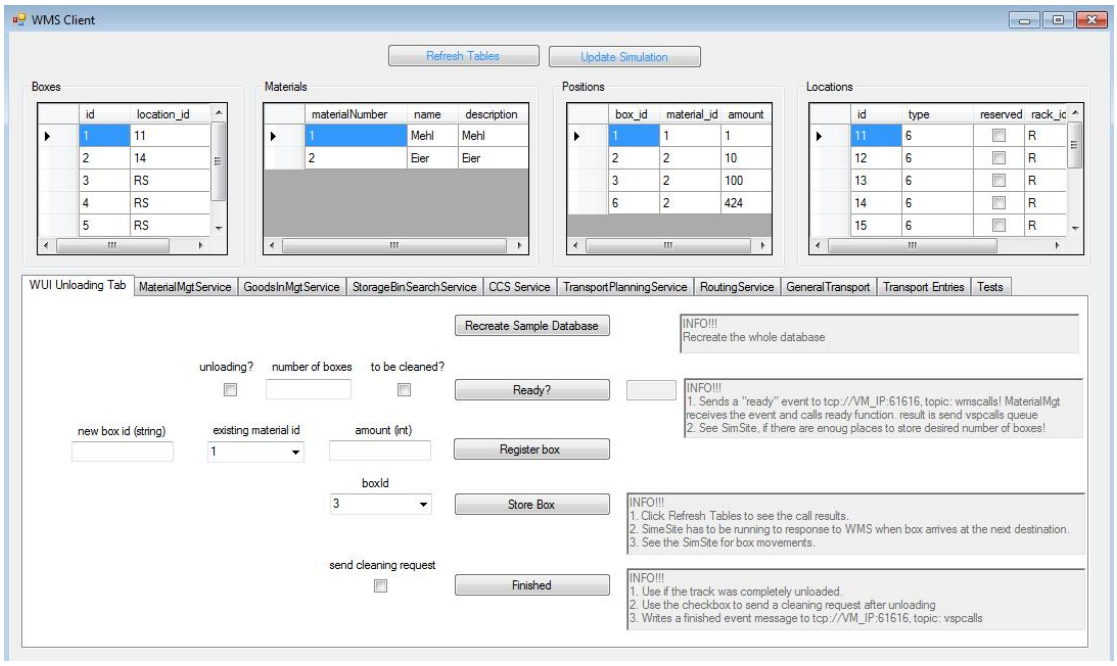

Figure 6 Warehouse User Interface (WUI)

CCS is serving as lower level transportation system. It is an abstraction layer to motors, conveyors and vertical lift modules and is in charge of transport jobs within the warehouse.

Simulation acts as the low level hardware system and visualizes the warehouse and stored products. It is implemented as a web site and contains of two components: SimSite (See Figure 7) as the actual simulation site and the hosting SimServer.

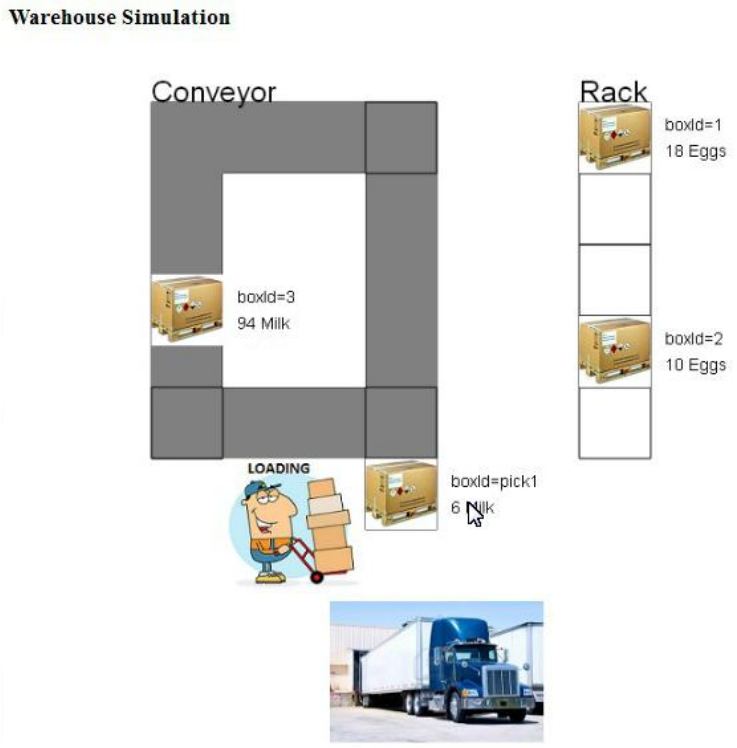

Figure 7 Warehouse Simulation Site (SimSite)

3.2 Used Technologies

WMS is currently realized as a solution in Visual Studio 2008 using .Net Framework 4.More specific, WMS and CCS Services are implemented in C# using Windows Communication Foundation API (WCF).

WCF Services are configured via Spring.NET dependency injection. So that internally requested service instances are retrieved dynamically from the Spring Container. Communication between CCS and Simulation is realized asynchronously via ActiveQM (.Net Message Service API -JMS for .NET). Communication between Simulation Site and Simulation Server is based on HTML5 WebSocket which enables bi-directional, full-duplex channels over a single TCP connection. Therefore the prerequisite for running simulation is WebSocket capable browser.

Communication with the external INDENICA virtual service platform is also managed using ActiveMQ. Currently there are two queues to handle it: one queue for incoming calls and one queue for outgoing calls.

The Warehouse User Interface (WUI) is a Windows Form, invoking WMS services via Spring.NET dependency injection.

WMS manipulates objects (boxes, materials, orders, etc.). Their mapping to relational database in the MS SQL Server is done using ADO.NET Entity Framework.

3.3 Available Services

The Warehouse Management System provides several services to accomplish warehouse tasks, for example creating a new box, inserting materials in the box, searching for the next free bin location, etc. One service can offer several methods. These services/methods are .NET WCF services and are internally accessed using WCF Spring Container.

The Table 1gives an overview of existing domain-specific services in the Warehouse Management System and a short description of them.

Service, Method Name	Description
Material Mgt Service bool CreateBox(string boxId) / bool DeleteBox(string boxId)	Creates / deletes a box
Material Mgt Service bool CreateMaterial(string materialNumber, string name, string description)	Creates a new material with the given name and description
Material Mgt Service bool InsertMaterial(string boxId, string content, int quantity)	Inserts the given amount of material to the box
Material Mgt Service bool CreateLocation(string locId, string typeId, string rackId, bool reserved) / bool DeleteLocation(string locId)	Creates / deletes a location
Material Mgt Service bool CreateTransportentry(string boxId, string binId) / bool DeleteTransportentry(string boxId)	Creates / deletes a transport entry
Material Mgt Service bool ReserveBin(string binId) / UnReserveBin(string binId)	Reserves / unreserves a bin

Material Mgt Service bool SetLocation(string boxId, string binId)	Sets location of the box to the *binId*
Material Mgt Service bool ReadyForUnloading(string orderId, string dockId, int numberOfBoxes, bool toBeCleaned)	Checks if WMS can store the given number of boxes of the order (unloading the truck)
Material Mgt Service bool ReadyForLloading(string orderId, string dockId, Dictionary<string, int> list)	Checks if WMS can load the list of the goods in the order on the truck at the certain dock (loading the truck)
Order Mgt Service bool CreateOrder(string orderId, string dockId, bool isUnloading, bool toBeCleaned, Dictionary<string, int> list) / bool DeleteOrder(string orderId)	Creates / deletes an unloading or loading order
Order Mgt Service bool ReserveOrder(string orderId) / bool UnReserveOrder(string ordered)	Reserves / unreserves the order
Order Mgt Service bool ProcessOrder(string orderId)	Sets the order status to *inProcess* and stars processing the order
Picking Service bool StartNextPick(string orderId)	Finds the next pick of the order with type *loading* and with the status *inProcess* and retrieves the box for loading
Picking Service bool EndLastPick(string orderId)	Finds the pick which was started and finishes it (either deletes the box if it's empty or stores the box back to WMS)
Material Mgt Service void Finished(bool orderId)	Informs the VSP that order processing was finished and deletes the finished order
GoodsIn Service bool RegisterBox(string boxId, string materialNumber, int amount)	Creates a new box and inserts the given amount of the material into this box
StorageBinSearch Service string SearchNextFreeBin()	Gets the id of the next free storage bin
TransportPlanning Service void StoreBox(string boxId)	Initiate storing of the given box in the next free bin
Routing Service string GetNextDestination(string boxId, string binId)	Gets the id of the next destination
Routing Service string GetNextTransportMedium(string boxId)	Gets the id of the next transport medium
GeneralTransport Service void InitiateTransport(string boxId, string binId)	Initiate the transport of the given box to the given bin
CCS Service void Move(string boxId, string transportMedium, string startLocId, string entLocId)	Moves the box using the given transport medium from the start location to the end location

Table 1 Domain-specific services in WMS

4 Yard Management System

The following sections should present an overview of the yard management from a more technical point of view. Aspects of architecture and used technologies will be highlighted as well as used techniques for achieving integration with the virtual service platform of INDENICA.

4.1 Overall Architecture

The yard management system is a web application that is intended to run in a cloud-based environment, e.g. SAP NetWeaverCloud. It is developed using standard techniques of web application development and resembles therefore a client-server-based architecture.

The server part runs in an OSGi-Environment and was designed to be modular to serve separation of concerns and thus improve flexibility and maintainability. The high level architecture is shown in Figure 8. It consists of multiple components that provide different services. Some of these bundles are optional and can be switched off, if not needed.

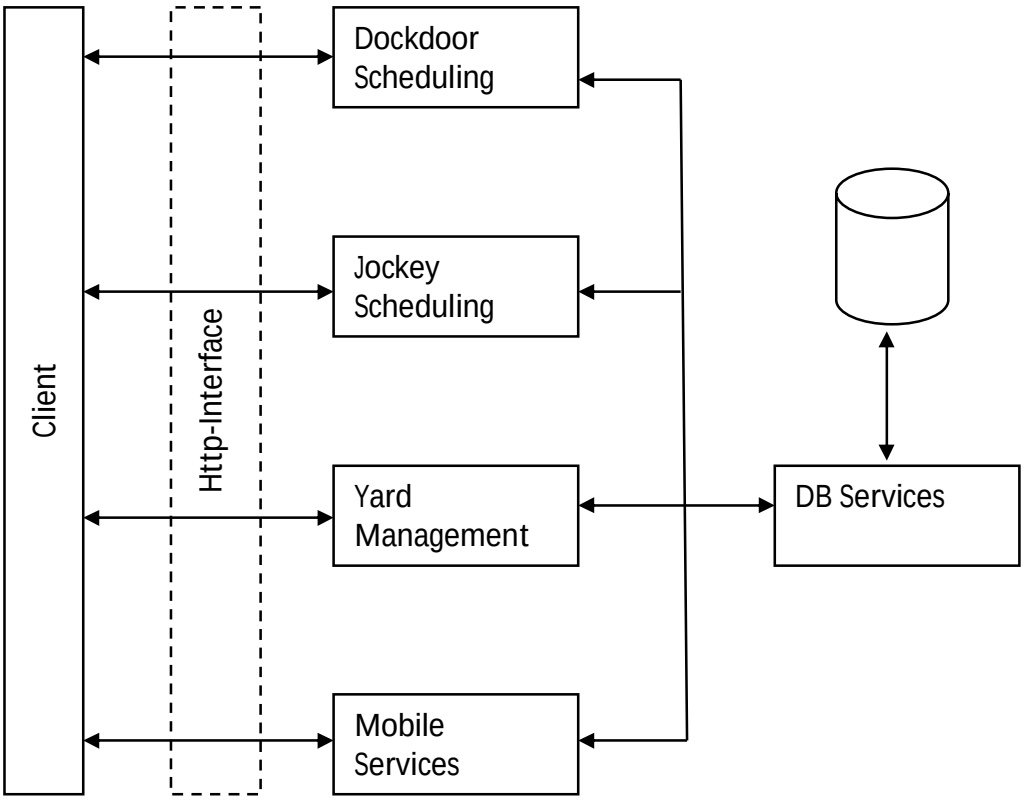

Figure 8 Architecture of Yard Management Server

The most basic component is the *DB*component providing fundamental facilities to access and alter domain-specific data in the database. It contains all business data objects and makes them available for other bundles as well as basic services to find, persist, change or delete data from the connected data base.

The *Dockdoor-Scheduling* component provides the inherent functionality of a yard management system. It coordinates the scheduling of appointments, drivers and docks. New appointments (an estimated time of arrival for a driver) are added, arriving drivers are assigned to free docks or waiting areas in case of no available dock. Appointments can be rescheduled in case of a delay or speedup e.g. in the loading process.

The *Jockey-Scheduling* component is in charge of coordinating the jockeys in the yard. Jockeys move full or empty trucks around the area, manoeuvre trucks in (un)loading position or complete similar tasks. These tasks are scheduled via this component. Different scheduling algorithms exist, e.g. location-based scheduling, where tasks are scheduled based on their location and the current locations of the jockeys.

The *Yard Management* component plays the role of a controller among all other components and orchestrates the higher-level business processes of a yard management system. It schedules actions based on events that are feed into the system e.g. an arriving truck. It creates jockey tasks for newly checked-in trailers to be brought to a waiting bay or similar.

Finally, the *Mobile* component contains services, which are specific for mobile devices. It manages the communication from and to mobile devices using the Comet web application model[1]. This allows specific jockeys to be notified on newly created tasks.

Every component publishes its own set of services that can be consumed over Http by a client, be it the management interface, a mobile client or the INDENICA virtual service platform.

On the client side, there exist three user interfaces, depending on the user. An interface for the yard operator (Figure 9) is provided that can be used to manage and oversee the activities on the yard. Second, two interfaces (Figure 10), one for the drivers and one for the jockeys, exists that are designed for usage on mobile devices.

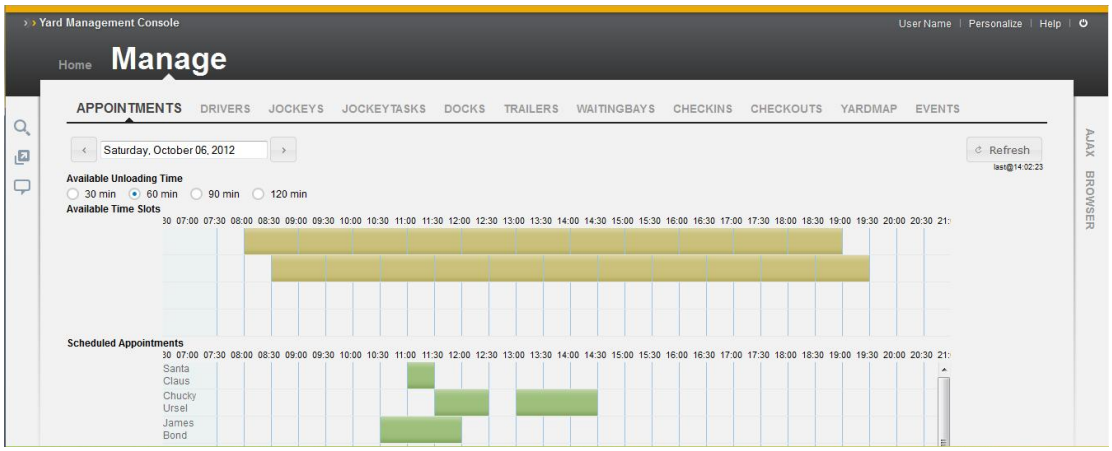

Figure 9 Yard Management Interface

[1]http://en.wikipedia.org/wiki/Comet_(programming)

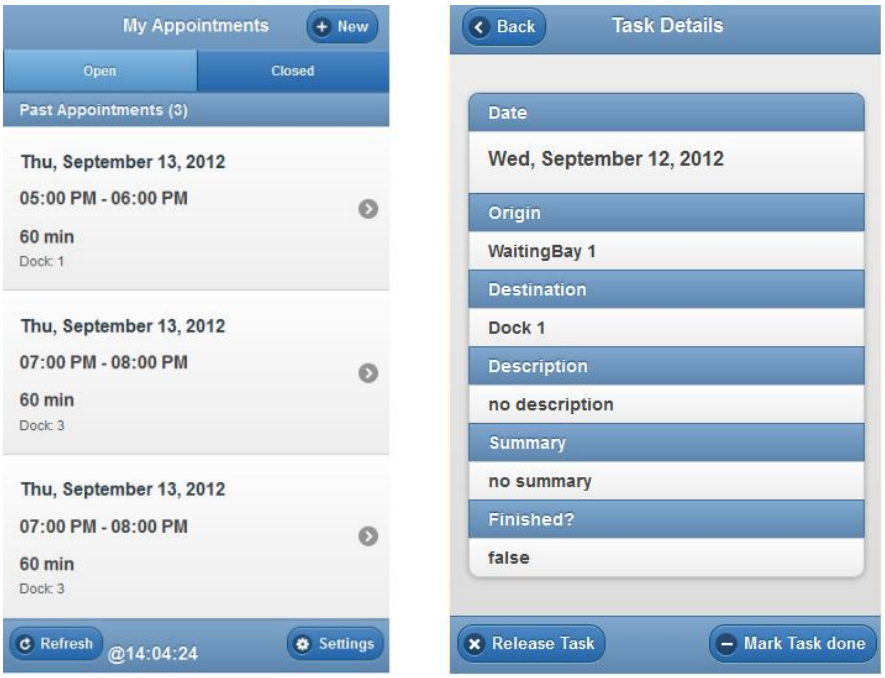

Figure 10 Driver Interface (l), Jockey Interface (r)

4.2 Used Technologies

The current use case application runs on SAP NetWeaverCloud, which uses an OSGi-based JavaEE6 application server. Every component in the yard management system is therefore an OSGi-bundle. The Spring web-framework is used internally for wiring and Http request handling. As no UI-rendering happens on server-side, the data will be transferred to the clients in JSON- or XML-format.

Additionally, other communication channels exist. Platform-internal events like an arriving truck or similar are propagated via JMS [2] in XML-format, so that asynchronous, decoupled communication with the INDENICA virtual service platform is possible. The use case application uses ActiveMQ as a JMS implementation.

Mobile devices connect to the yard management service using the Comet pattern. More specifically they use the Cometd framework, an implementation of this pattern. This enables push-messages from the server to connected clients.

The client code is written in JavaScript. The management interface uses the SapUi5 library[3], an HTML5-conform library for creating rich Internet applications.

Both mobile interfaces utilises the jQuery Mobile[4] library to generate a UI for mobile devices

[2] Java Message Service (http://de.wikipedia.org/wiki/Java_Message_Service)

[3] http://www.spyvee.com/SAPHTML5_DemoKit/

[4] http://jquerymobile.com/

4.3 Available Services

The Yard Management Service (YMS) publishes several services for communication with other components such as the VSP or external user interfaces. Currently all of these are REST services allowing access from anywhere. Detailed service description including input and output format can be found in the UML Documentation.

The following table (Table 2) gives an overview of all existing domain-specific services in the yard management system and a short description of them.

Service URL	Description
/dds/loadingFinished	Notify YMS that loading of a specific trailer is done
/dds/unloadingFinished	Notify YMS that unloading of a specific trailer is done
/dds/freeAppointment	Provides timeslots for new appointments with the given parameters (duration, time and date)
/dds/delayAppointment	Notify YMS that appointment will delay, returns new appointment date and time
/dds/preponeAppointment	Notify YMS that appointment can start earlier, returns new appointment date and time
/dds/dockAppointments	Send or get appointment to/fromdock door scheduler (DDS)
/yms/actionDescription	Gives description to YMS about action required for a specific trailer (booking)
/yms/truckArrived	Notify YMS that the truck (with a trailer) has arrived
/yms/truckLeft	Notify YMS that the truck (driver) has left
/jockey/jockeyTasks	Send or retrieve jockey tasks to/from jockey service

Table 2 Domain-specific Services

Additionally, there also exist services that provide meta-information or exist for debugging purposes. These services are explained in Table 3.

Service URL	Description
/db/schema	Provides the XSD schema for the database entities
/db/dummy	Clears and initialises the Database with test data
/dds/wadl	Provides the WADL schema of the DDS REST services
/jockey/wadl	Provides the WADL schema of the jockey REST services
/yms/wadl	Provides the WADL schema of the yms REST services
/yms/jms	HTTP tunnel for the JMS broker
/mobile/cometd	Interface for the CometD Service

Table 3 Technical Services

4.4 Variability Management

Besides the actual YMS, there are technologies and services, which enable variability management. As mentioned in D6.4.4 Cocktail was used to handle runtime variability of YMS for server side (Java) as well as client side (JavaScript). To achieve this, the cocktail variability runtime has to be deployed along with the rest of the YMS, which results in more services to be use.

Service URL	Description
/cocktail/resolutions	Service endpoint to manage variability (variation points)
/cocktail	Web interface used for easy variability management

Table 4 Variability Management

To allow users or administrators an easy configuration of variability the Web Interface of Cocktail is also deployed. It allows quick management of variation points at runtime and is also useful for seeing the current setup of the YMS. In Figure 11 the Cocktail Web Interface for the Dock Door Scheduling Service (DDS) is shown.

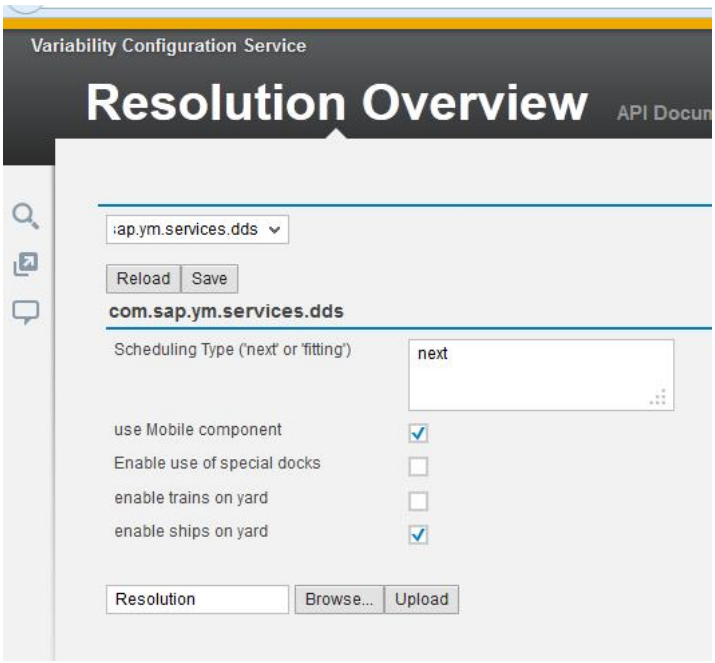

Figure 11 Variability Management Web Interface of DDS

With the help of the Variability management services and the web interface it is possible to modify the application at run time. As a proof of concept several basic variation and configuration points have been added to show the capabilities of the variability management. Below, in Table 5, a list of variation points (configurable elements) and their description can be found.

Variation Point	Description
DDS.schedulingType	Either “fitting” or “next”. Fitting tries to optimise the Dock-usage, so that as few idle-times as possible occur while minimising the number of used docks. Next just uses the next time available, even if it means, that there’s a idle time of only 30 minutes in-between, which is improbable to be filled by another appointment, as most are longer.
DDS.useExtendedDockTypes	Disables or Enables the use of special docks. Special docks can, for instance, have a “cooling” function for food or specific measures for hazardous materials.
DDS.useMobile	Indicates whether Jockeys use mobile devices or fixed devices at the locations. Mobile devices allow dynamic assignment of Jockeys according to their position. (i.e. the operator knows which jockey is closest to the first location of a given task and thus assign the task to this jockey)
DDS.trainsEnabled	If true the operator can see trains and railroads on the Yard and coordinate them. Usually not something to be changed at runtime, but still worth making variable to run the same application on different Yards independent of their needs.
DDS.shipsEnabled	Ship-support, see *trainsEnabled*.
Jockey.useGps	If this is set to true the locations will be treated using their GPS-coordinates.
Jockey.showGpsText	Only allowed with *useGps* true. Depending on this setting Locations on WebUI are displayed as GPS coordinates or the actual names. (e.g. Dock2 instead of X: 3.2 / Y 5.1)
Jockey.showGpsMap	Only allowed with *useGps* true. If set to true the Map shows the locations of the Jockeys on the map.
Jockey.gpsMapIsSatellite	Only allowed with *useGps* true. Determines whether the WebUI should show a satellite map of the Yard or a normal map with only streets and buildings in it.

Table 5 Variation Points

Some of the variation points have constraints such as *useGps*, which has to be set to true in order to enable *showGpsText*, *showGpsMap* or *gpsMapIsSatellite*. The selection of variation points was made to show the impact variability management can have on an application. It is in question whether all of the chosen variation

points make sense in this use case. However, some are merely added as a proof of concept (train and ship support) and not to add real value or logic to the YMS.

Having listed the variation points it is important to mention, how they are used and processed. There are application logic only variation points, such as *schedulingType*. Others are hybrid and influence application logic as well as user interfaces or interaction, such as *useMobile*, *useExtendedDockTypes* or *useGps*. Of course there are variation points solely influencing user interfaces and displayed information as well. Examples are *showGpsText*" and *gpsMapIsSatellite*.

This variety of the variation points implies the necessity of having server side variation management, in the Java code, and client side variation management, the JavaScript code in the browser. How Cocktails solves these issues is described in D6.4.4.

5 Remote Maintenance System

This section presents an overview of the Remote Maintenance System. Its objective is to describe the RMS Platform from the technical point of view with focus on system architecture, technologies used by the platform and web services it provides.

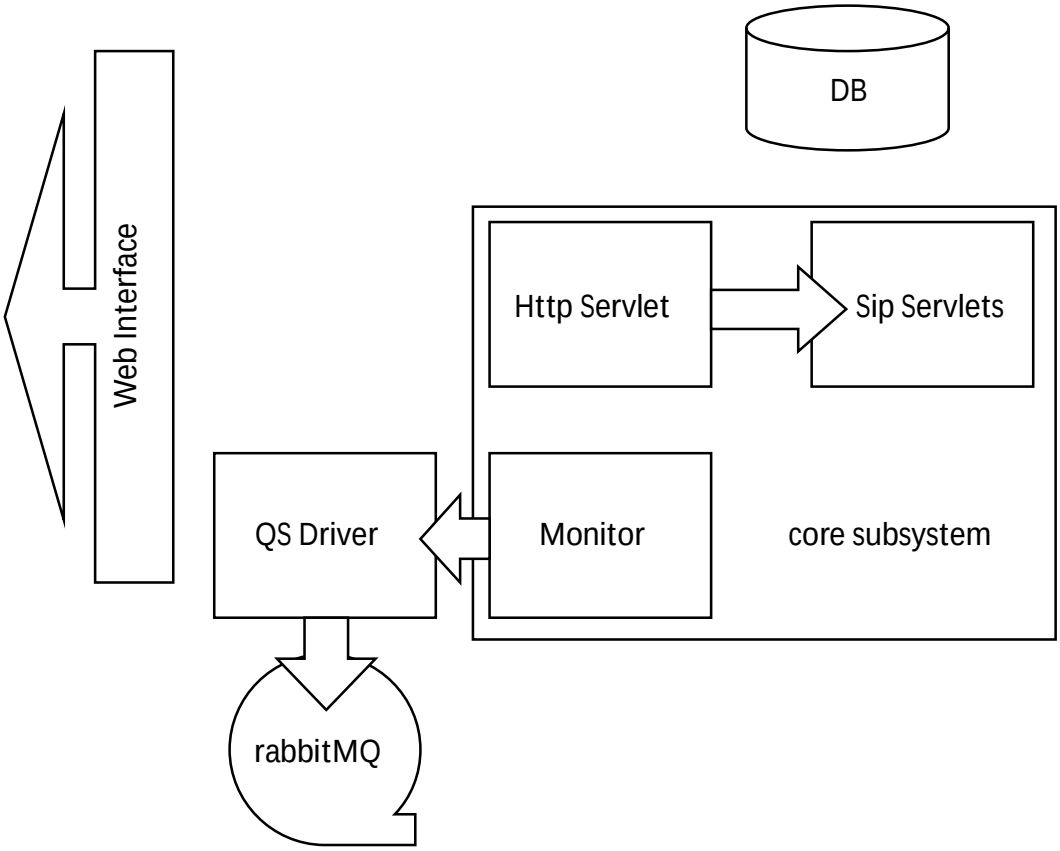

Figure 12 Architecture of Remote Maintenance System

5.1 Overall Architecture

The RMS system is divided into two distinct parts. The first one is the core RMS system and the latter is platform's pluggable web interface. The core part of the system is composed of Sip Servlets, Http Server and Monitoring. The Database component is optional, but it is highly recommended to be used, otherwise the platform will lack persistence capabilities.

Web Interface

RMS Platform exposes to the client a simple and coherent SOAP-based web interface, achieving at the same time the programming language agnostic design. Applications that can exploit the RMS web interface may be written in any language that supports SOAP based web services. Interface itself is functionally divided into three different service subsets, which are: call, supervision and adaptation services. Behind the web interface is hidden thin translator layer which converts the incoming SOAP data into JSON based lightweight messages.

Http Servlets

Http Servlets are used as a communication adapter between asynchronous sip servlets and front-end web interface. This component was implemented using standard javax.servlet.http package. Main module responsibility is to provide the synchronous access to the asynchronous sip servlets. Communication points between sip and http are obtained using *ConvergedHttpSession* interface.

Sip Servlets

The core logic of the RMS platform is implemented by the Sip Servlets component. Whole sip messaging is sent and processed by that part of the system. This component is composed of distinct classes each implementing the logic responsible for processing different sip messages. Sip related data that is received by the platform is sent to the application router according to the:

```
INVITE=("rms-indenica-core", "DAR\:From", "TERMINATING", "",
"NO_ROUTE", "0")
MESSAGE=("rms-indenica-core", "DAR\:From", "TERMINATING", "",
"NO_ROUTE", "0")
REGISTER=("rms-indenica-core", "DAR\:From", "TERMINATING", "",
"NO_ROUTE", "0")
OPTIONS=("rms-indenica-core", "DAR\:From", "TERMINATING", "",
"NO_ROUTE", "0")
```

Such solution allows easy substitution of older components by new ones without additional redeployments; therefore all required applications can be deployed once and used only when necessary.

Monitor

Monitor module is responsible for collecting various system characteristics, such as: used memory, CPU usage or storage related information. Monitoring data are collected using hyperic sigar which is platform agnostic library, written in Java.

5.2 Used Technologies

This section describes the main components of the RMS Platform, discussing some of their implementation details, used technologies and justifying the conceived approach with market impacts. The RMS Platform has been completely written in Java EE (version 6) in order to achieve high OS interoperability.

Core subsystem

Remote Maintenance System is based on the Mobicents platform, which is built on top of Red Hat's web server – JBoss version 7.1.,both of which are consecutive open source implementation of the JSR-289 specification (namely SIP Servlets v. 1.1) and JSR-316 (Java Enterprise Edition v. 1.6).

Mobicents Sip Servlets provides access to SIP protocol features via servlet's mechanisms without going into details of SIP protocol stack. Functionalities built on top of the MSS are combined with other web applications through the http

converged context, thus enabling faster and easier development of the SIP-based software solutions.

JBoss Application Server is, a robust and efficient web application server which supports distributed server systems, hot deployments, load balancing and many others. Nevertheless, the complex internals of JBoss AS do not affect the speed and effectiveness of the applications running on top of it due to the lightweight thread and memory management. Furthermore, using not proprietary and freely available software solutions has considerably positive impact on the growth potential of the small companies with the special attention to various IT industry start-ups.

Database

To increase interoperability even further the platform itself is not dependent on any specific database system. Instead of which the Hibernate framework is used to allow easy, almost pluggable (with a little or no additional configuration) database system interchange. Being not bounded to any specific database vendor has the main advantage of allowing the platform deployment in both: corporate environment, where proprietary solutions are expected to be used, as well as in the small company's backbone networks, which are very often exploiting free and open source solutions in order to minimise the overall costs of the running system. Undertaken approach seems to have considerable impact on substantial improvements in the market competitiveness of subjects involved in using and developing the RMS Platform.

5.3 Available Services

Services provided by the Remote Maintenance System are divided into two distinct parts, namely functional services and adaptation services. The former include call and supervision services, which cover various system functionalities, e.g. call and text message management, user registration, notification, etc. The adaptation services are used mainly for adapting the platform behaviour to the constantly changing network environment.

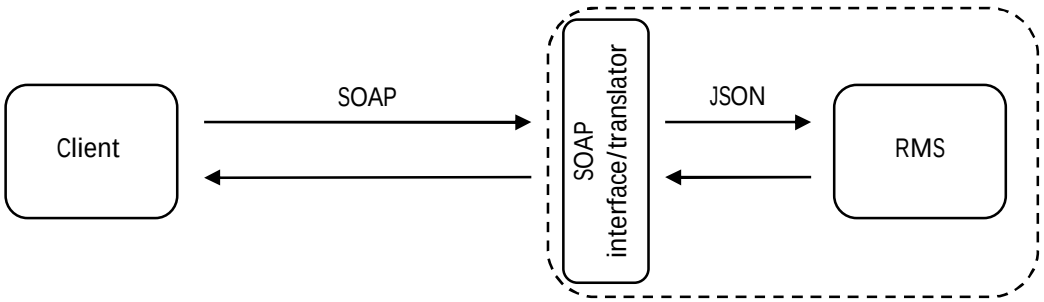

Figure 13 Communication with RMS

All services are available through the SOAP web interface which results in a programming language agnostic interface design, i.e. the client can be written in any programming language that supports SOAP communications with WSDL-defined web services.

```
<xs:complexType name="updateCameraInformation">
<xs:sequence>
<xs:element minOccurs="0"name="adminId"type="xs:string"/>
<xs:element minOccurs="0"name="cameraSipId"type="xs:string"/>
<xs:element minOccurs="0"name="cameraIpAddress"type="xs:string"/>
<xs:element name="xPosition"type="xs:double"/>
<xs:element name="yPosition"type="xs:double"/>
<xs:element name="horizontalAngle"type="xs:double"/>
<xs:element name="verticalAngle"type="xs:double"/>
<xs:element name="zoom"type="xs:double"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="updateCameraInformationResponse">
<xs:sequence>
<xs:element
minOccurs="0"name="return"type="tns:updateCameraInformationRESP"/>
</xs:sequence>
</xs:complexType>
```

Communication to and from the platform is realised by the SOAP protocol; messages are parsed to the XML then wrapped in the SOAP envelope and next they are sent through the network. Despite the outside platform communication is realised by SOAP, the whole inner platform communication (inside the RMS) is done by using the JSON lightweight data format.

Call Service,web methods	Description
initiateVoiceAndVideoSession(adminId, userSipId, [receiverSipIds], callParameters)	Creates voice and video session between two endpoints
initiateVoiceSession(adminId, userSipId, [receiverSipIds], callParameters)	Creates voice only session between two endpoints
terminateSession(adminId, userSipId, sessionId)	Ends the ongoing session
unregisterUser(adminId, userSipId)	Unregisters the user from the service platform
registerNewUser(adminId, userSipId, userIp, userName, userPassword, userGroup, userStatusId, userParameters)	Registers the user to the service platform
updateUserInformation(adminId, userSipId, userIp, userName, userPassword, userGroup, userStatusId, userParameters)	Updates information of the previously registered user
changeUserStatus(adminId, userSipId, userStatusId)	Changes user's status
getUserStatus(adminId, userSipId)	Receives user's status
getUserList(adminId)	Receives list of users
getActiveSessionIds(adminId)	Receives list of open sessions

Table 6 Technical Services – call management

One type of usage environments of services that implements the call service interface are various communication systems, which are to be used in order to provide reliable communication within the networks that must be separated from the Internet, e.g. corporate intranets, military or government intelligence facilities, etc. Another use case can be providing the voice and video services for immediate calls in case of some unexpected circumstances in facilities for public use (e.g. in case loosing luggage at the airport, the fast call to the authorised persons can advance the luggage recovery).

Supervision Service, web methods	Description
connectToCameraWithVoiceAndVideo(adminId, userSipId, cameraSipId, callParameters)	Connects the user endpoint to the camera endpoint with voice and video capabilities
connectToCameraWithVideoOnly(adminId, userSipId, cameraSipId, callParameters)	Connects the user endpoint to the camera endpoint with voice only capabilities
changeCodecOrResolution(adminId, sessionId, callParameters)	Changes used codec and resolution. (Will terminate the session first)
registerCamera(adminId, cameraSipId, cameraIpAddress, xPosition, yPosition, verticalAngle, horizontalAngle, zoom)	Registers the camera to the service platform
updateCameraInformation(adminId, cameraSipId, cameraIpAddress, xPosition, yPosition, verticalAngle, horizontalAngle, zoom)	Updates information of the previously registered camera
unregisterCamera(adminId, cameraSipId)	Unregisters camera from the service platform
getCamera(adminId, cameraSipId)	Receives camera related information
getCameraList(adminId)	Receives list of registered cameras
sendTextMessageToUser(adminId, textMessage, targetUserSipId)	Sends text message to the specific user
sendTextMessageToUserGroup(adminId, textMessage, [targetUserSipIds])	Sends text message to the specific user group
sendTextMessageToAllRegisteredUsers(adminId, textMessage)	Sends text message to all registered users
sendTextAlertWithVideo(adminId, textMessage, targetUserSipId, videoSourceSipId)	Sends text alert and establishes required video connection

Table 7 Technical Services - supervision

Main purpose of the supervision services is to provide unified interface to managing camera system, alarms and security staff. Obtaining camera information (such as position, vision angle or zoom) can be used to manage and control the complex camera systems easily and efficiently. Along with

Adaptation Service, web methods	Description
setTextMessageHistoryPermission(adminId, enabled)	Enables/disables text message history
getTextMessageHistoryPermission(adminId)	Retrieves enable/disable state of text message history permission
setDirectVoiceAndVideoSessionInitiationPermission(adminId, enabled)	Enables/disables direct voice and video calls
getDirectVoiceAndVideoSessionInitiationPermission(adminId)	Retrieves enable/disable state of direct voice and video calls permission
setRemoteVoiceAndVideoSessionInitiationPermission(adminId, enabled)	Enables/disables remote voice and video calls
getRemoteVoiceAndVideoSessionInitiationPermission(adminId)	Retrieves enable/disable state of remote voice and video calls permission
setDirectVoiceSessionInitiationPermission(adminId, enabled)	Enables/disables direct voice calls
getDirectVoiceSessionInitiationPermission(adminId)	Retrieves enable/disable state of direct voice calls permission
setRemoteVoiceSessionInitiationPermission(adminId, enabled)	Enables/disables remote voice calls
getRemoteVoiceSessionInitiationPermission(adminId)	Retrieves enable/disable state of remote voice calls permission
setDirectVideoSessionInitiationPermission(adminId, enabled)	Enables/disables direct voice calls
getDirectVideoSessionInitiationPermission(adminId)	Retrieves enable/disable state of direct voice calls permission
setRemoteVideoSessionInitiationPermission(adminId, enabled)	Enables/disables remote video calls
getRemoteVideoSessionInitiationPermission(adminId)	Retrieves enable/disable state of remote video calls permission
setDirectTextMessageSessionPermission(adminId, enabled)	Enables/disables direct text messaging
getDirectTextMessageSessionPermission(adminId)	Retrieves enable/disable state of direct text messaging permission
setRemoteTextMessageSessionPermission(adminId, enabled)	Enables/disables remote text messaging
getRemoteTextMessageSessionPermission(adminId)	Retrieves enable/disable state of remote text messaging permission
setMachineMonitoringPermission(adminId,	Enables/disables machine monitoring

enabled)	
getMachineMonitoringPermission(adminId)	Retrieves enable/disable state of machine monitoring permission
terminateSessionsInitiatedDirectly(adminId)	Terminates all remotely instantiated calls
terminateSessionsInitiatedRemotely(adminId)	Terminate all directly instantiated calls
terminateSessionsByPercent(adminId, percent)	Terminates specific percent of ongoing calls

Table 8RMS Adaptation services

Adaptation is realised by an on/off mechanism, which is used to enabling (disabling) various platform functionalities. To decrease complexity of the sip servlets, the adaptation functionality was built on top of the servlets as a filtering layer, allowing at the same time better separation of concerns. Adaptation of the platform is to be used when it is under a heavy load. In such scenario a various actions c

5.4 Installation Process

The whole installation process is automated in order to accelerate the configuration of the platform. Installation scripts are written in Maven, so the installing process looks very similar in Linux and Windows platforms.

Installation and configuration

Complete installation and configuration process can be performed just by invoking in the top code directory the following command:

```
mvn clean post-clean
-Djboss.server.ip=your.jboss.ip.address
-Drabbitmq.server.ip=rabbits.mq.ip.address
-Dcomponents.directory=/path/to/install/rms/platform
-Dserver.name=rms.platorm
-Dserver.domain=indenica.nextdaylab.com
```

It assures that all needed software will be downloaded (decent Internet connection and preinstalled Maven are required) and installed on the running machine. Installation directory can be chosen as well as some basic networking configuration in order to allow initial pre-configuration of the platform on different than the target machine.

Starting the platform

The platform can be started just by running the following command (on Linux):

```
cd /path/to/install/rms/platform/hsqldb.../hsqldb
java –cp lib/hsqldb.jar org.hsqldb.Server
cd /path/to/install/rms/platform/mss.../bin
standalone.sh –c standalone-sip.xml
```

On Windows double clicking on the main.bat script in the directory where the platform has been installed.

First run

When the platform is started the first time it is just bare Mobicents Sip Servlets platform preconfigured to serve as the Remote Maintenance System. To complete installation all required web application has to be compiled and deployed on the running platform. In order to achieve that the platform must be started first and then the following command must be invoked (in the top directory which contains the code):

```
cd rms-indenica
mvn clean jboss-as:deploy
-Djboss-as.username=admin
-Djboss-as.password=zylia
-Djboss.server.ip=your.jboss.ip.address
```

After that, the RMS platform is ready to work.

Testing

To be sure that the platform is installed correctly the set of tests is prepared. Tests can be started by invoking in the directory where the RMS code is placed:

```
cd rms-indenica/soap-interface-test
mvn test -Djboss.server.ip=your.jboss.ip.address
```

5.5 Real Time Communication

RMS Platform allows real time video and voice communication between SIP phones. This subsection explains the steps needed to configure a softphone client to work with the RMS Platform. For that particular purpose the Linphone freely available client is used.

After installation the Linphone client has to be configured. Example configuration consists of the steps, which are shown below.

Step 1

First start the client and go to the Options -> Preferences menu (Figure 14).

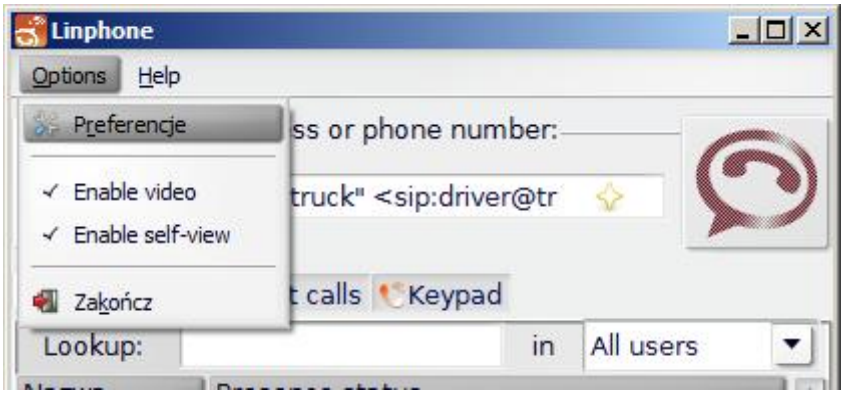

Figure 14 Linphone client menu

Step 2

Next ensure that the network settings are similar to those shown in Figure 15.

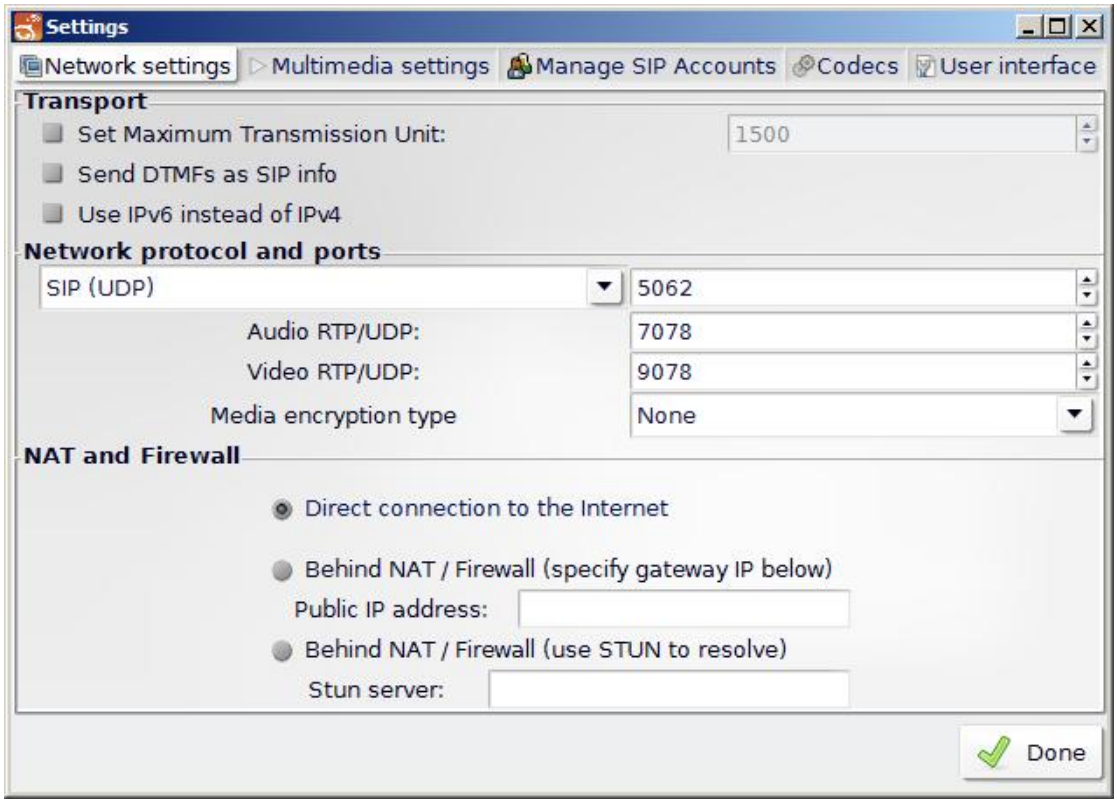

Figure 15 Network settings

Step 3

Then go to the Manage SIP Accounts tab and add new account. Fill the newly opened window (Figure 16) in following way.

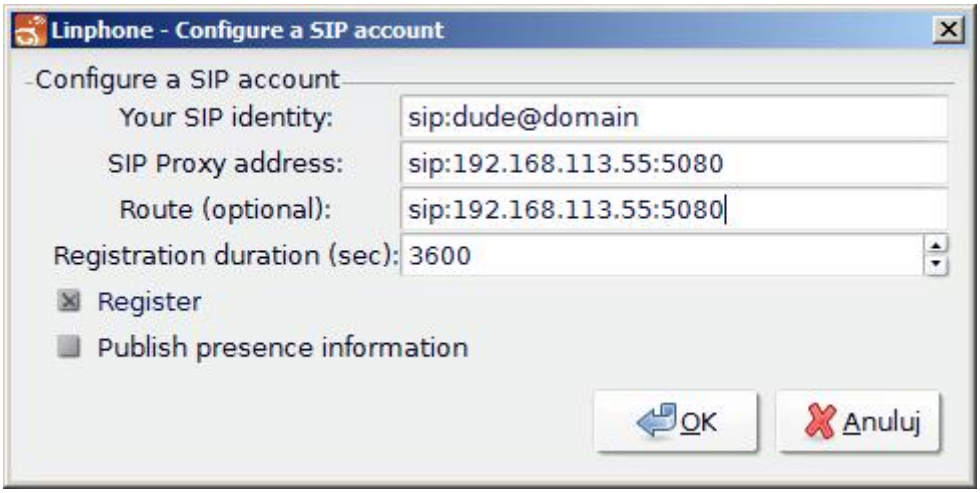

Figure 16 SIP account configuration

Please to be sure that the Register box is selected, to ensure automatic registration on the RMS Platform. Ensure that the SIP Proxy address and Route contains a valid IP address of machine on which the RMS Platform is running. Default port number should be 5080 unless it was changed during the installation or configuration process of the RMS Platform.

Step 4

After that, just click OK and the newly configured user should be registered by the RMS Platform.

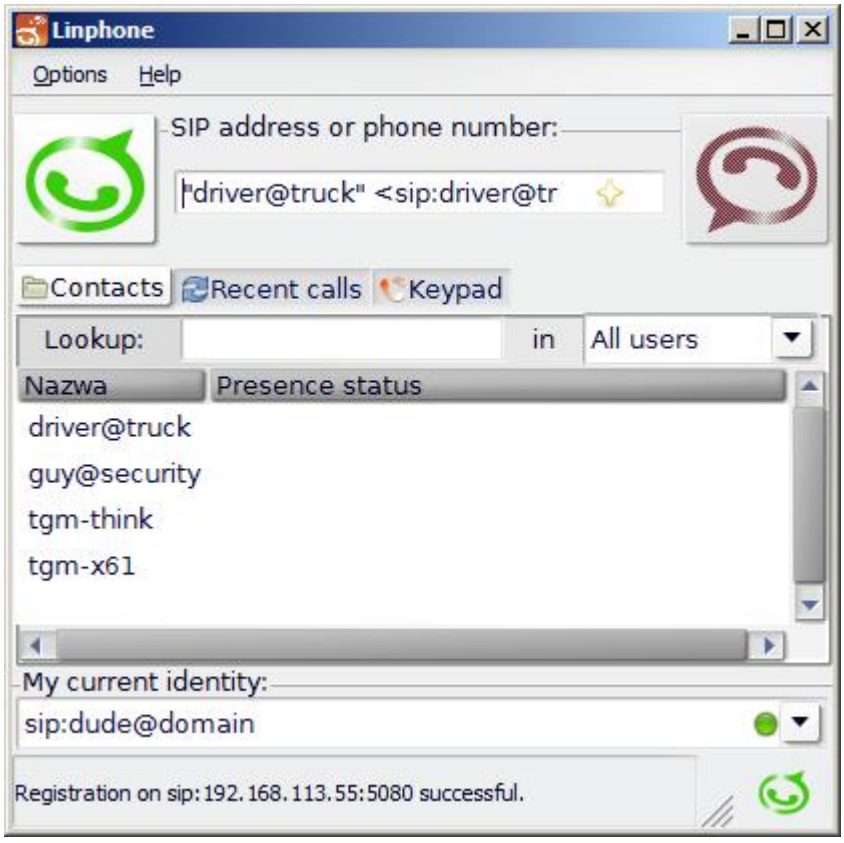

Figure 17 Properly configured new SIP account

If SIP account has been properly configured, the light at the right bottom corner of the program (Figure 17) should be green. Otherwise, its colour and shape will indicate origin of problem.

After configuring the client on another machine the connection can be easily established by calling the one client from the other.

6 Technical Integration

As discussed in the previous sections, the scenario for integration was taken from the real world and the process has already been explained. The technical integration, which results a virtual service platform, includes several different communication mechanisms and techniques.

The interim version of the virtual service platform comprises the integration of two service-based platforms, which are Yard Management System (YMS) and Warehouse Management System (WMS). In the final version, the virtual service platform will encapsulate a complete integration of three service-based platforms including the aforementioned platforms and a Remote Management System (RMS) provided by NDL.

6.1 Overview

In Figure 18, we present an overview of the INDENICA integration scenario. This is the accumulated result of an iterative analysis, design and development process. The starting point is the deliverable D5.1 where we describe the INDENICA case studies in which different domain service platforms need to be integrated. After that, several interviews are conducted with the domain experts from industrial partners in order to distil important high-level and platform- and technology-specific information. The outcomes are related requirements, variability, architectural decisions, and the list of services provided by the platforms along with the underlying technologies.

As we mentioned before, the interim version of the integration scenario embraces two service-based platforms, YMS and WMS, whilst the final version also include the third platform, the RMS. The final version is an incremental modelling and development based on the interim version.

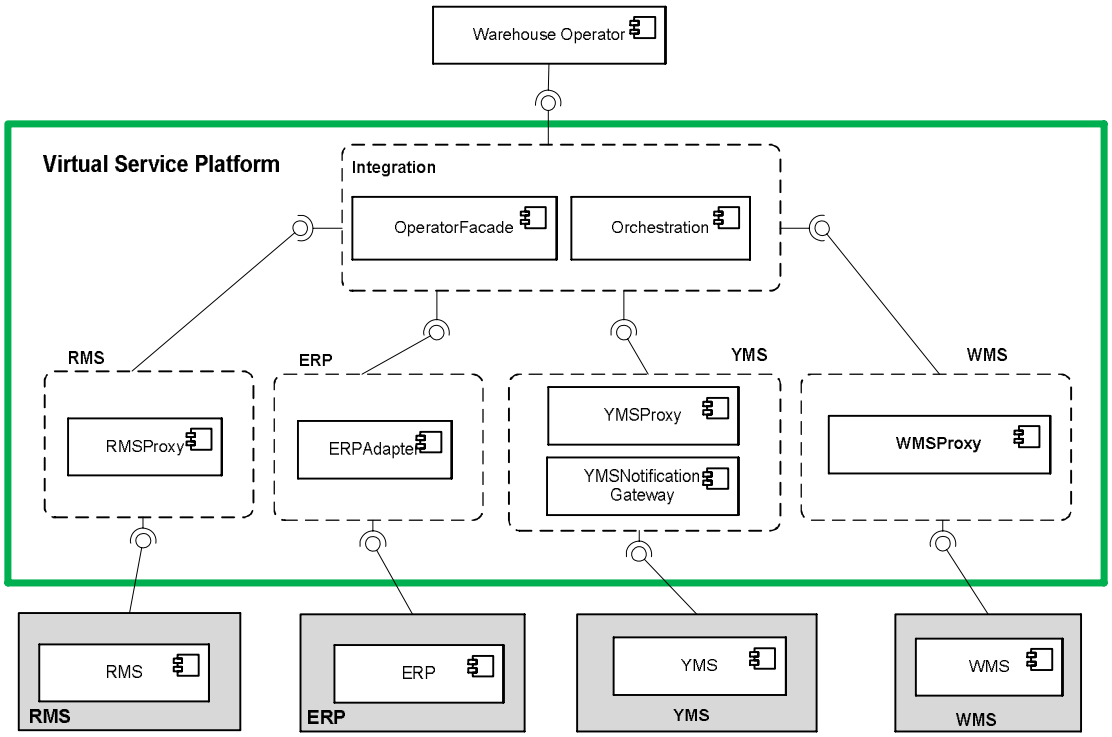

Figure 18 Overview of the INDENICA integration scenario

6.2 Architectural Design and Implementation

6.2.1 High-level view-based modelling

The requirements are described using the IRET tool (developed by PDM, see D1.2.2), the platform variability are captured using the EASy-Producer (developed by SUH, c.f. D2.2.1, D2.2.2, D2.4.1, and D2.4.2), and the architectural design decisions are captured by the ADvISE tool (developed by UNIVIE, c.f. D1.3.2).

ADvISE can help to generate a high-level Service Component View, which is useful for a *greenfield scenario*. Nevertheless, we can reuse the high-level Service Component View developed in the interim version and modify it according to the new variability and architectural decisions that occur due to the integration of the third platform RMS.

The Service Component View is created or modified by using the INDENICA view-based Tool Suite (c.f. D3.1, D3.3.1, D3.3.2). It models the high-level architecture of the integrated VSP (i.e., the green box in Figure 18). Therefore, the high-level architecture of the VSP described using INDENICA Tool Suite—as shown in Figure 19—is mainly used by software architects and/or domain experts and independent from the underlying runtime and communication technologies.

The major advantage of introducing the integrated VSP in the integration scenario is that to seal the application built on top (e.g., in this case, the Warehouse Operator Application) from the complexity of the underlying technologies of various service platforms and provide unified development interfaces via the *OperatorFacade* component. It also helps to avoid platform vendor lock-in because the substitutions

of any underlying service platforms by the others that provides similar or more functions mainly affect and require changes in the VSP. Thus, the application built on top still works as far as the interfaces of the façade remain stable. We note that the necessary changes in the VSP in case the service platforms are altered mostly happen in the corresponding adapters and proxies, for instance, the YMSProxy, WMSProxy, ERPAdapter, YMSNotificationGateway, WMSNotificationGateway, and RMSProxy.

Indeed, during the course of the interim VSP design and development, we experienced a major refactoring in the VSP when, due to some technical updates, the YMS and WMS platforms offered JMS communications instead of Web services.

Fortunately, we do not have to change the Service Component View but only need to change the Mapping DSL (c.f. D3.3.2) to annotate the previous «Web service» proxy components as «JMS» proxy components in the high-level view model and leverage the templates described in INDENICA Tool Suite to generate the corresponding skeletons and interfaces for communicating via JMS. The orchestration and integration logics developed in other unrelated components as well as the Warehouse Operator Application remain unchanged.

However, in the final version, due to the addition of the third platform, RMS, we need to change the high-level Service Component View accordingly. This can be simply achieved by added a new component, namely, RMSProxy, for representing the interaction with the RMS platform via a proxy as shown in Figure 1914.

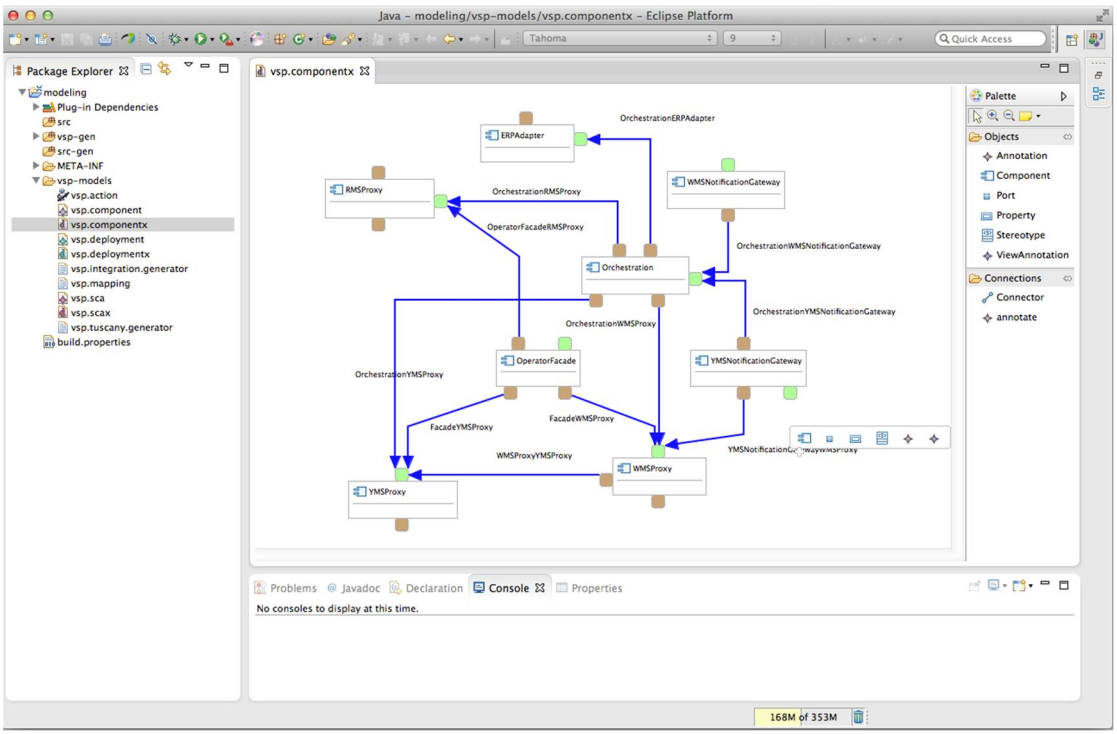

Figure 19 High-level modeling of the integrated VSP using INDENICA Tool Suite

6.2.2 Low-level view-based modelling and code generation

Once the high-level service component view is modelled, we can carry out formal validations using the OCL-like languages supported by INDENICA Tool Suite (see §3.3 in D3.3.1) or using the live constraint checking and validation (see §3.7 in D3.3.2).

Then, we use the Mapping DSL (see §3.5 and §5.2 in D3.3.2) to define a refinement of the high-level view model to the low-level Service Component View for SCA, which is specific for the Apache Tuscany SCA technology that we use to deploy and run the VSP. The low-level view model is shown in Figure 20.

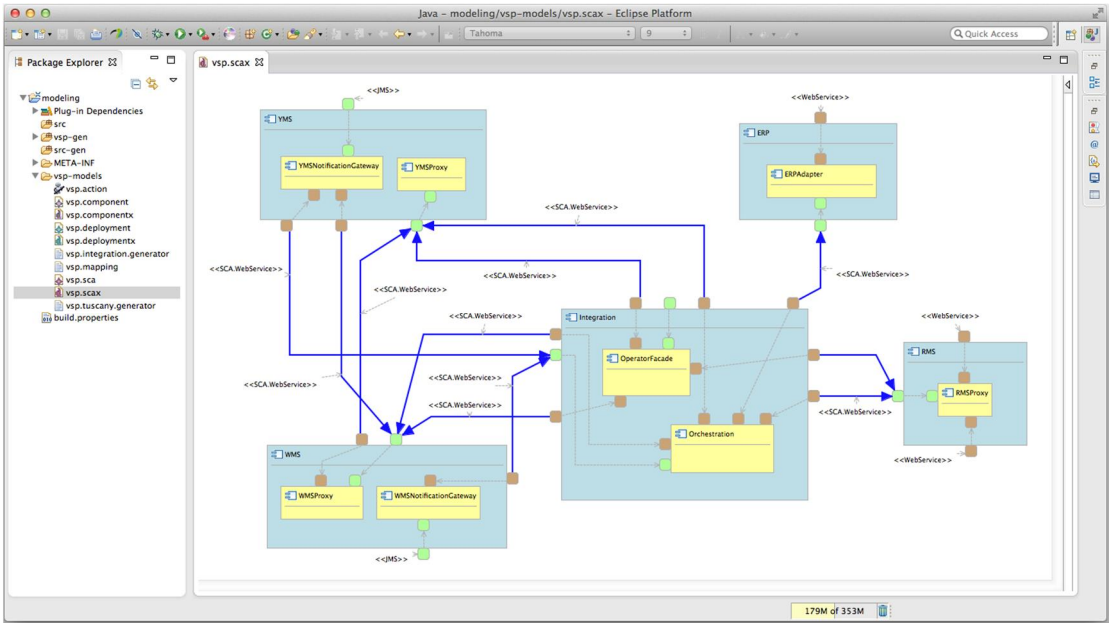

Figure 20 Low-level modelling of the integrated VSP for SCA using INDENICA Tool Suite

We note that the components are refined with more specific details and grouped into “composites”, which are the notion of component containers in SCA.

We also note that, we do not have to directly change the low-level Service Component View comparing to the interim version because the modifications of the Mapping DSL are enough to produce a corresponding low-level view.

Furthermore, we use the Generator DSL in order to enrich the refined low-level view models with further details such as the interface descriptions of the SCA components in Web service or Java, the implementation libraries, the concrete binding addresses, hosts, and ports, etc. (see §3.8 and §5.4 in D3.3.2).

The deployment strategy for the VSP is described using the Deployment View provided by the INDENICA Tool Suite. We show in Figure 21 the deployment strategy of the VSP.

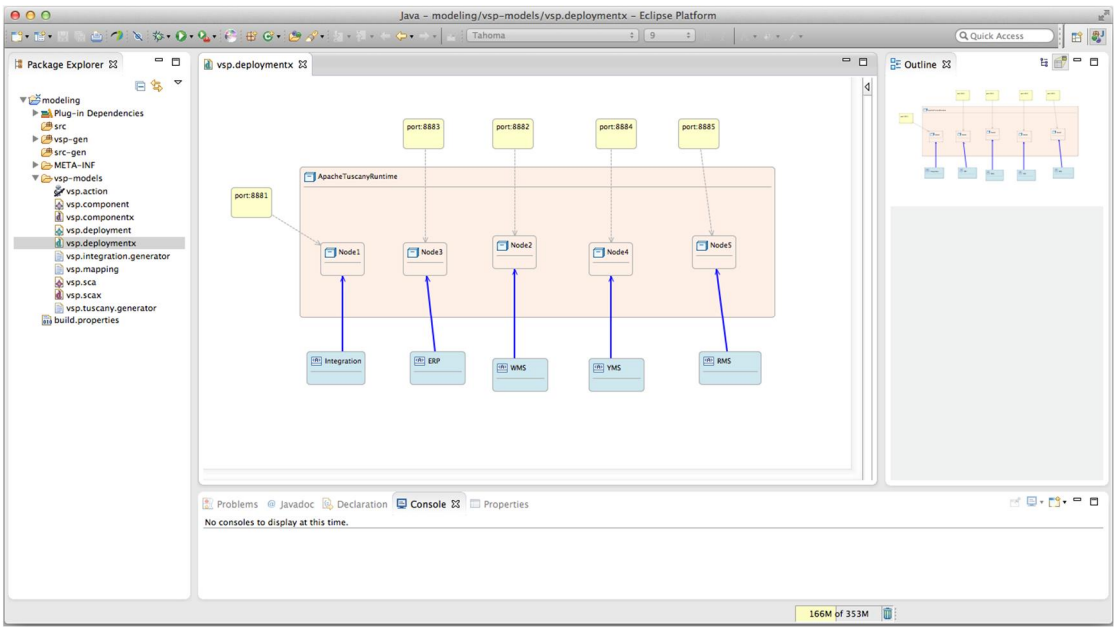

Figure 21 Modeling the deployment of the integrated VSP into Apache Tuscany Runtime

Based on the specification in the low-level Service Component View, the Deployment view, and the Generator DSL presented above, the implementation and configuration of the VSP will be generated automatically (see §3.8, §3.9, §5.5, §5.6 in D3.3.2).

In Figure 22, we show the generated Java code of the VSP components described above. SCA configurations such as SCA composite descriptions, bindings, and deployments are also generated.

INDENICA Tool Suite utilizes the separation of generated and non-generated code in order to keep a clean separation between the generated code and the particular orchestration and integration code written by the developers. That is, the developers mainly program the "-impl" parts shown in Figure 22 that actually implement the generated interfaces. These "-impl" parts will be untouched in the future iterations unless the developers explicitly command the code generators to override the existing code.

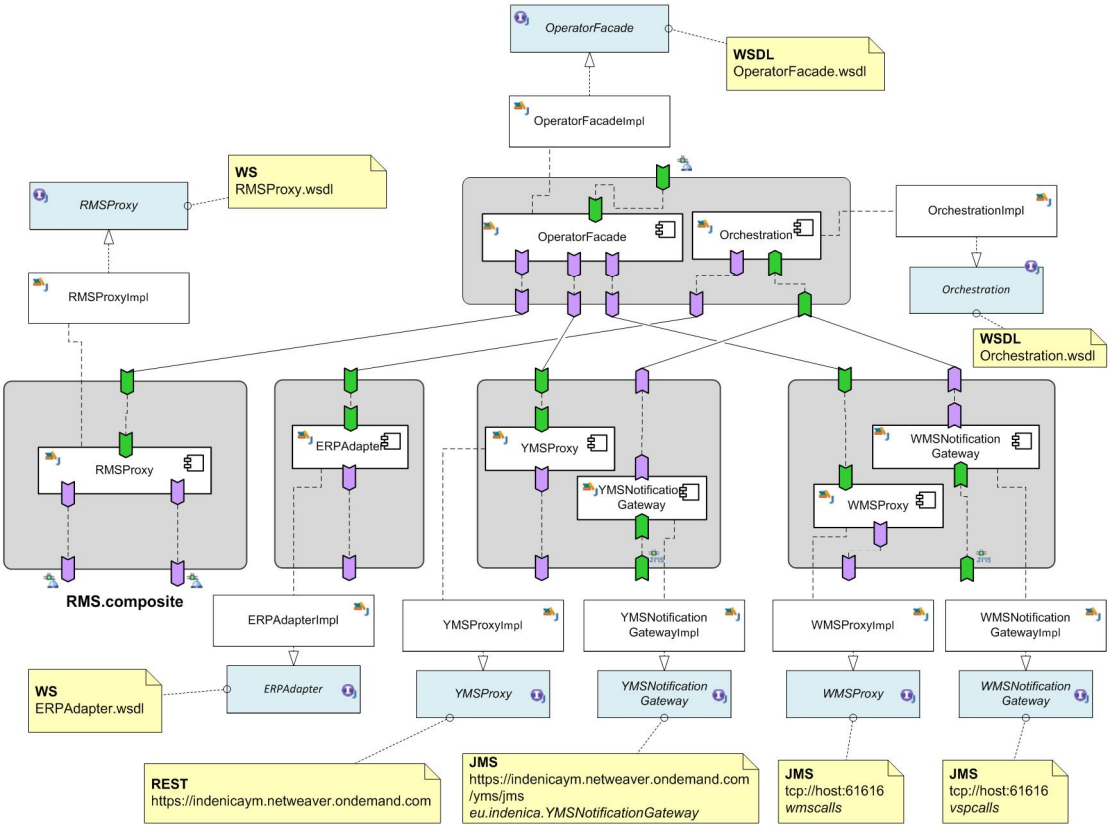

Figure 22 Overview of the generated VSP implementation

6.3 Deployment and Enactment

Essentially, for better load balancing and distribution, we decide to deploy each SCA composite described in the low-level view model in Figure 20 into a "*virtual*" node of the Apache Tuscany Runtime. One or multiple nodes can be hosted and executed in an Apache Tuscany logic domain that can correspond to an individual or a group of physical system. In our testing and demonstration environment, we host and run all four nodes in the same Apache Tuscany domain in a single PC.

The actual deployment of the VSP into a remote Apache Tuscany Runtime is performed by a Maven-based deployment tool developed by SAP as part of the Deployment component of the INDENICA Runtime provided in WP4 (see §3 in D4.2.1).

In Figure 24 we present the final deployment and execution of the INDENICA integration scenario that we demonstrated at the Second Year Review Meeting in Munich, Germany. We summarize the main aspects as following:

- Warehouse Operator: this is a standalone application running in a Java VM. The communication between Warehouse Operator and the VSP is done via Web service invocations. The VSP provide a unified façade interface to the applications built on top via the OperatorFacade component.
- Integration (composite): this is an SCA composite that is responsible for *orchestrating* the functionality provided by the underlying service platforms through the corresponding gateway, adapter, and proxy components and

providing the *façade interface* to the application built on top. This composite is running inside an Apache Tuscany Runtime.

- ERP (composite): this is an SCA composite that provides the adapter between the VSP and the ERP service platform. This composite is running inside an Apache Tuscany Runtime.
- WMS (composite): this is another SCA composite that provides the adapters for interacting between the VSP and the WMS service platform. This composite is running inside an Apache Tuscany Runtime.
- YMS (composite): this is an SCA composite that provides the adapters for interacting between the VSP and the YMS service platform. This composite is running inside an Apache Tuscany Runtime.
- RMS (composite): this is an SCA composite that provides the proxies for interacting between the VSP and the RMS service platform. This composite is also running inside an Apache Tuscany Runtime.
- WMS(Platform and UI): The WMS platform and UI are developed and running under Microsoft .NET framework and AppFabric in Windows and provides services via the Windows Communication Framework (WCF). The service invocations between VSP and WMS through the WMSProxy and WMSNotificationGateway are wrapped in JMS message exchanges.
- YMS (Platform and UI): The YMS platform is extracted from SAP real systems and running inside an SAP NetWeaver Cloud Platform. The communications between the VSP and YMS are performed via the YMSProxy by using RESTful service invocations and via YMSNotificationGateway by using JMS message exchange wrapping. The YMS UI is a Web-based application that can be accessible from anywhere via the Web browser. We illustrated in the review meeting that the truck drivers could easily access services provided by the YMS via the YMS UI using smartphones or tablets in order to make appointments for loading or unloading and to monitor the schedule and progress.

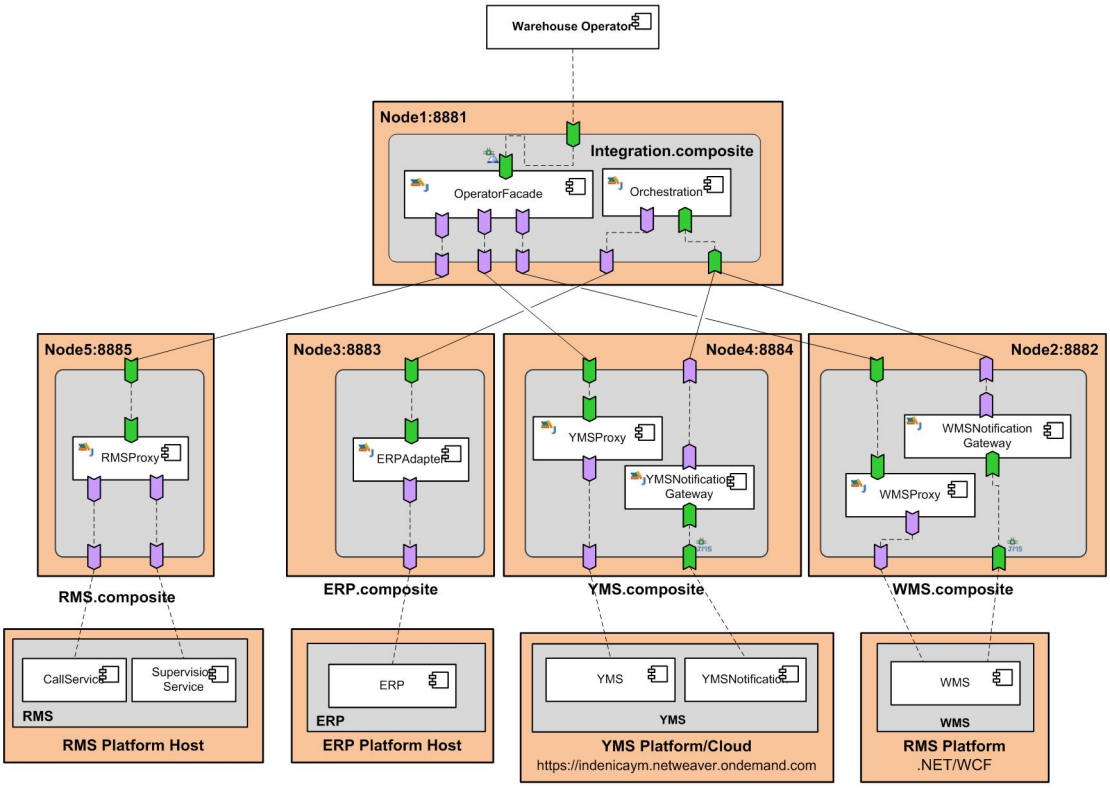

Figure 23 Schematic view of the deployment strategy of the VSP at runtime

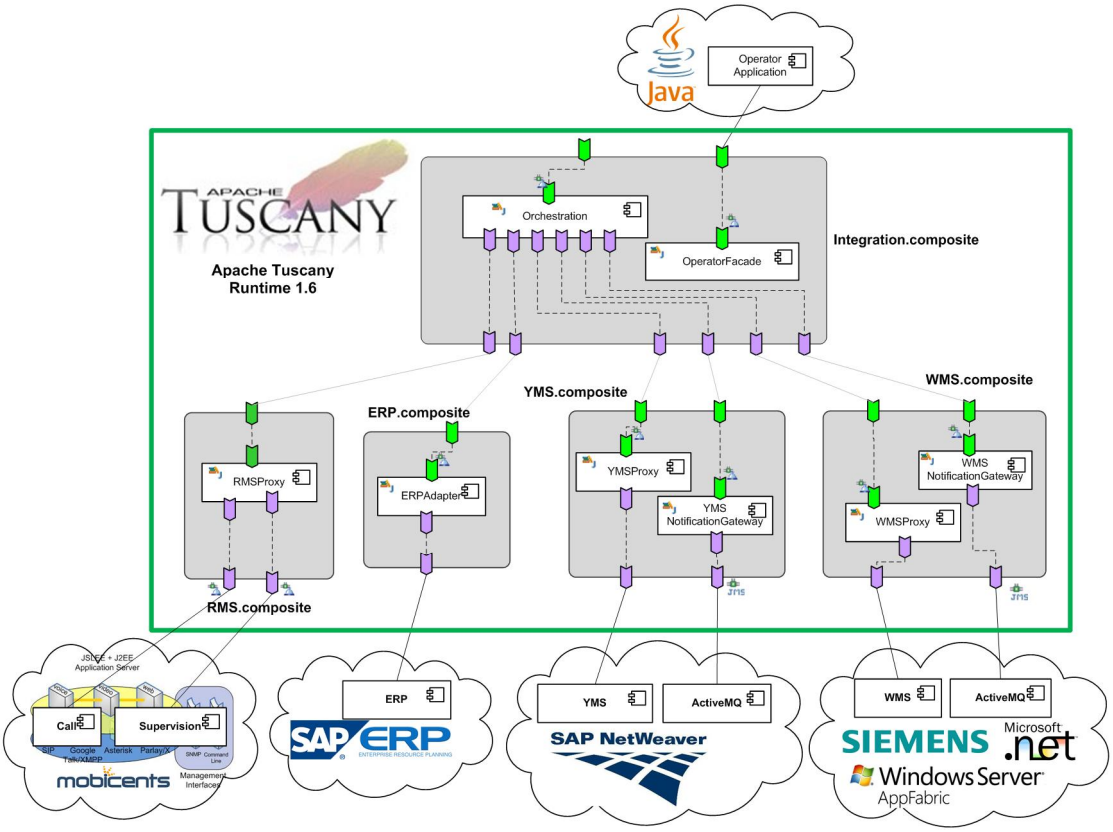

Figure 24 The actual deployment of the INDENICA integration scenario

7 Summary

In the third Year of INDENICA the third platform (RMS) has been integrated and evaluated in the overall scenario. Significant amount of integration work was enhanced by automatically generated code based on modelling in INDENICIA Tool Suite. Additional monitoring and adaptation capabilities have been implemented and used to govern the VSP at runtime.

Implemented service platforms and their capabilities have been presented including the sequence diagrams presenting the actual flow of information between the subsystems during scenario execution.

Table of Figures