Engineering Virtual Domain-Specific Service Platforms

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

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

Abstract

This document describes the implemention 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 – D5.3.1
Deliverable Number:	D5.3.1
Work Package:	5
Type:	Deliverable
Dissemination Level:	PU
Status:	final
Version:	1.0
Date:	2012-09-30
Author(s):	SIE, SAP, NDL

Project Start Date: October 1st 2010, Duration: 36 months

Version History

1.0	30. Sep 2012	Final Version

Document Properties

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

Table of Contents

Table of Contents 3
1 Introduction 4
2 Integration Scenarios 5
2.1 Check In 5
2.2 Error Handling 6
2.3 Unloading 7
2.4 Check Out 8
3 Warehouse Management System 9
3.1 Overall Architecture 9
3.2 Used Technologies 11
3.3 Available Services 11
4 Yard Management System 13
4.1 Overall Architecture 13
4.2 Used Technologies 15
4.3 Available Services 16
5 Technical Integration 18
5.1 Overview 18
5.2 Architectural Design and Implementation 19
5.3 Deployment and Enactment 21
6 Outlook 24
Table of Figures 25

1 Introduction

This document describes the current state of the implementation of the service platform from the partners SAP and Siemens. As outlined in previous documents, SAP is responsible for a Yard Management Platform while Siemens provides the counterpart, which is a Warehouse Management 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 and WMS).

- The integration is shortly outlined in the following section.

2 Integration Scenarios

The integration of the Yard Management System (YMS) and the Warehouse Management System (WMS) 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, which handle processes in the yard itself, need to be managed. 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 warehouse is busy, 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 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.

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.

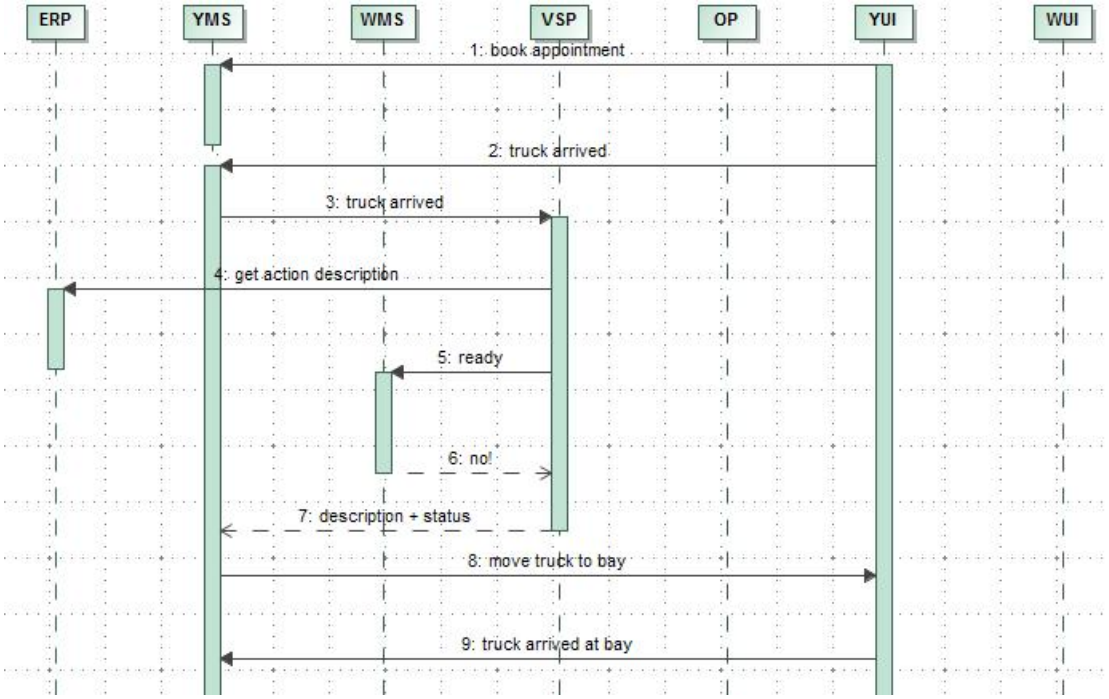

Figure 1 Check In Process

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 checks repeatedly for the WMS to be ready and when it finally is, he manually notifies the VSP, that the trailer can now be sent to the dock.

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.

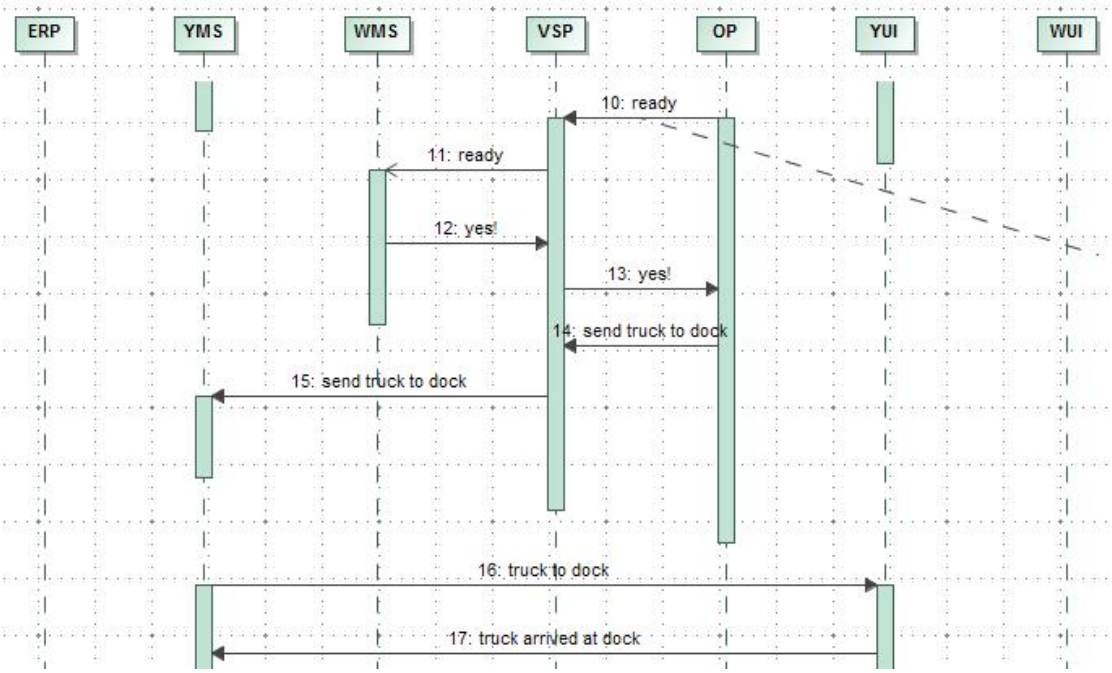

Figure 2 Error Handling Process

2.3 Unloading

When being notified that unloading of the track can be started, the VSP creates a new unloading order (see Figure 3). 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.

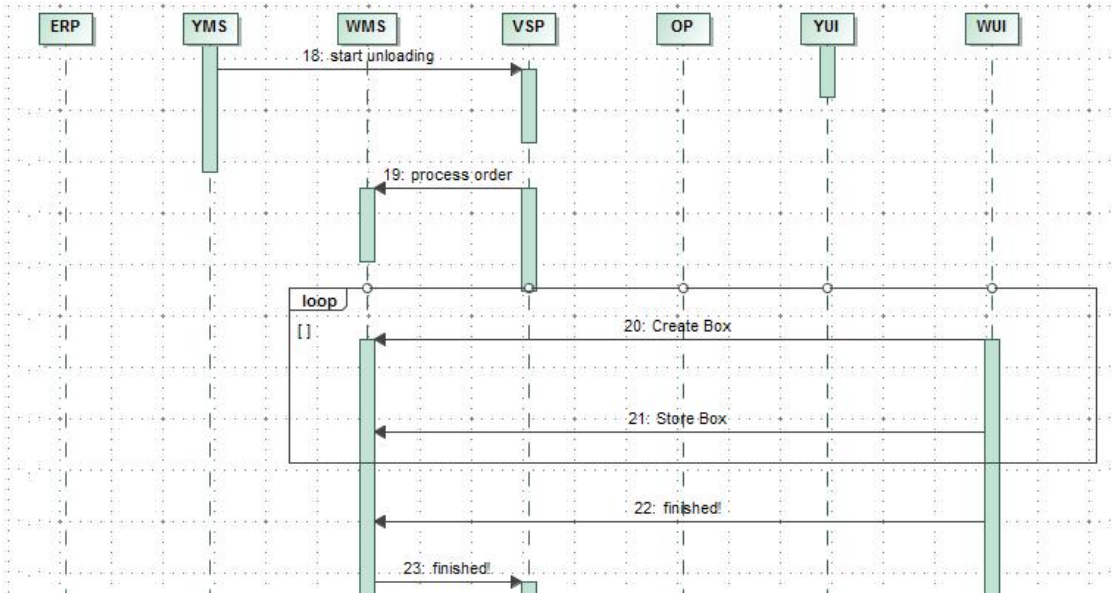

Figure 3 Unloading Process

2.4 Check Out

After unloading is finished (see Figure 4) the VSP 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.

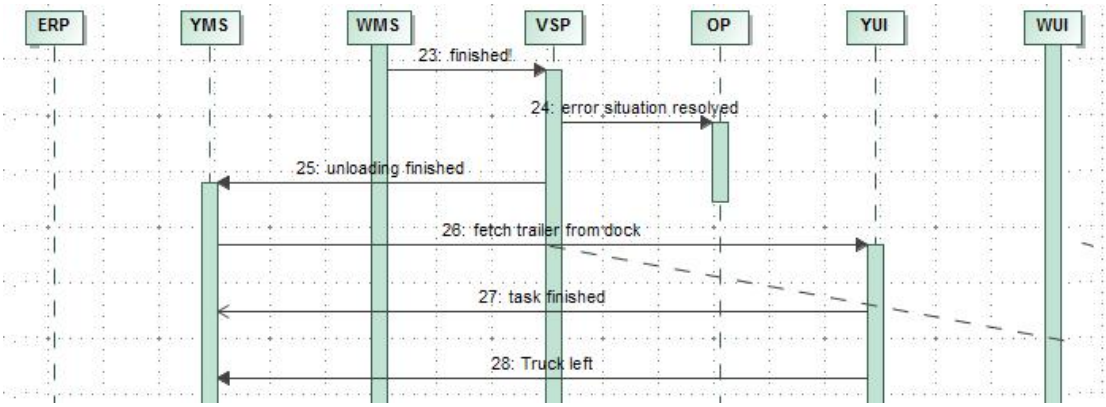

Figure 4 Check Out Process

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)

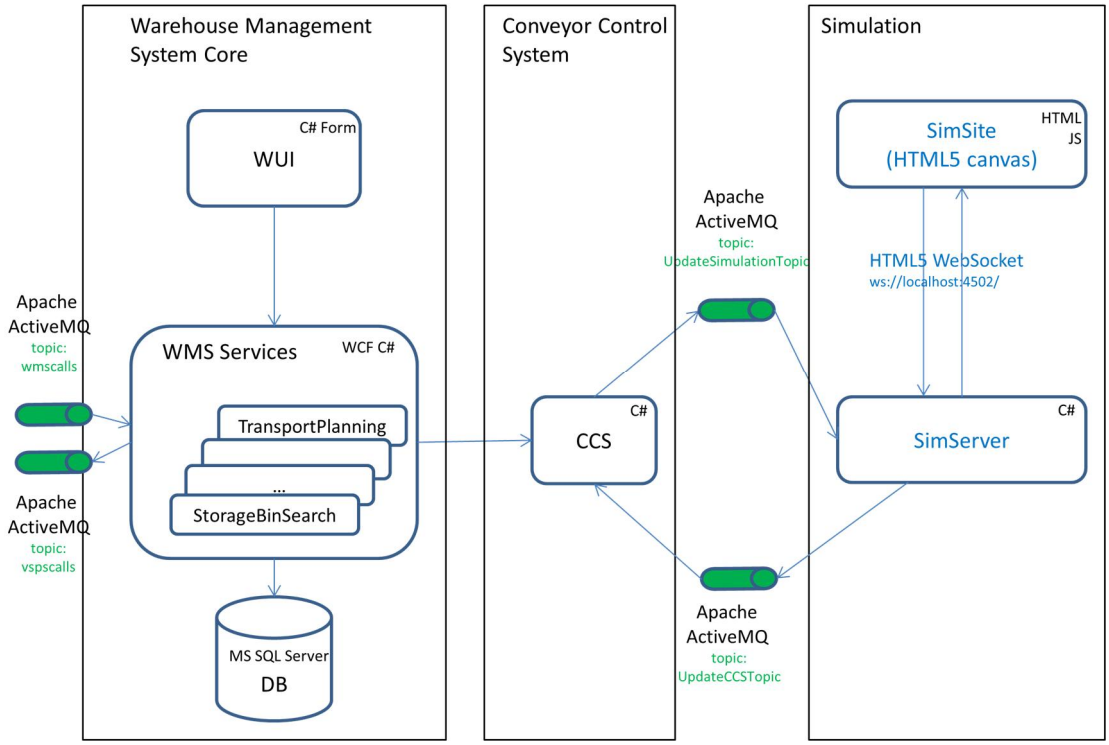

Figure 5 Warehouse Management System (WMS) Architecture

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.

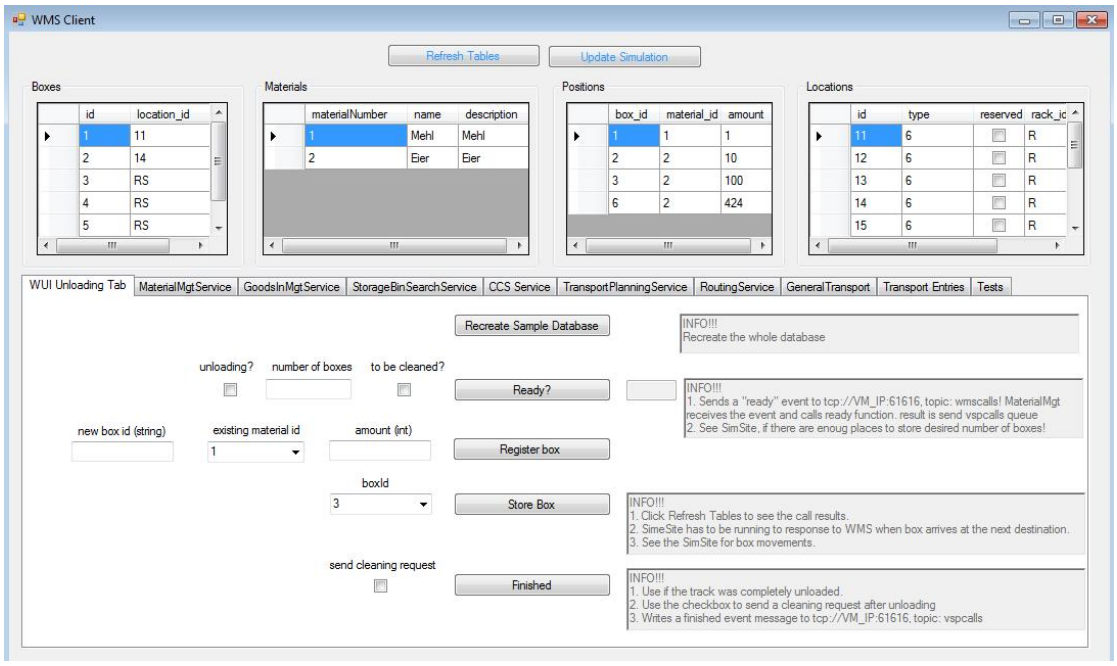

Figure 6 Warehouse User Interface (WUI)

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.

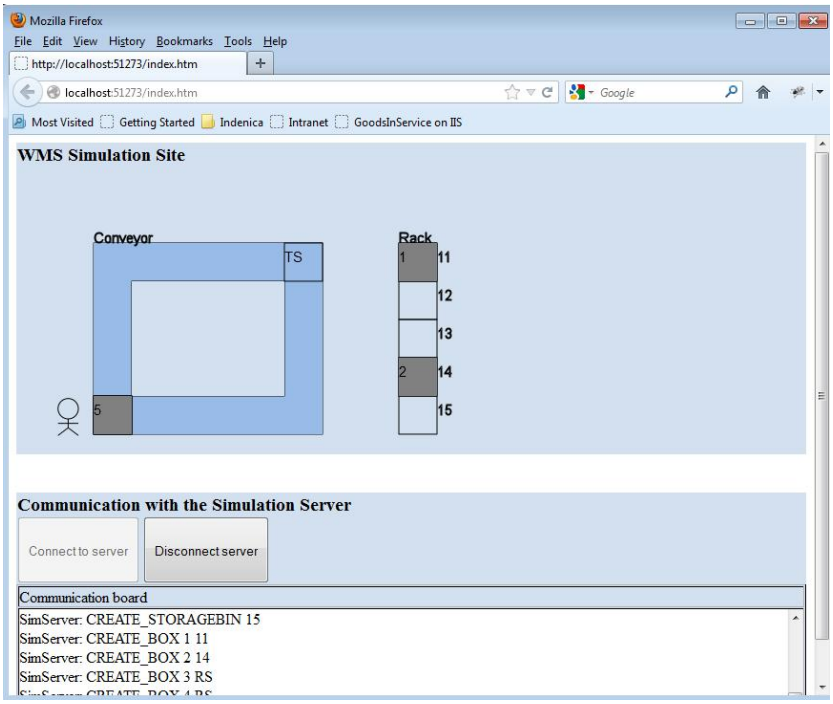

Figure 7 Simulation Site (SimSite)

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 (EF)

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 Table 1 gives 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);	Creates a new 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);	Creates a new reserved or unreserved location
Material Mgt Service bool CreateTransportentry(string boxId, string binId);	Creates a new transport entry
Material Mgt Service bool DeleteTransportentry(string boxId);	Deletes the transportentry

Material Mgt Service bool ReserveBin(string binId);	Reserves the bin
Material Mgt Service bool UnReserveBin(string binId);	Unreserves the bin
Material Mgt Service bool SetLocation(string boxId, string binId);	Sets location of the box to the binId
Material Mgt Service bool Ready(bool unloading, int numberOfBoxes, bool toBeCleaned);	Checks if WMS can store the given number of boxes
Material Mgt Service void Finished(bool toBeCleaned);	Informs the VSP that unloading was finished and if the track has to be cleaned or not
GoodsIn Service bool RegisterBox(string boxId, string materialNumber, int amount);	Creates a new box and insertes 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 in
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 endlocation

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 NetWeaver Cloud. 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.

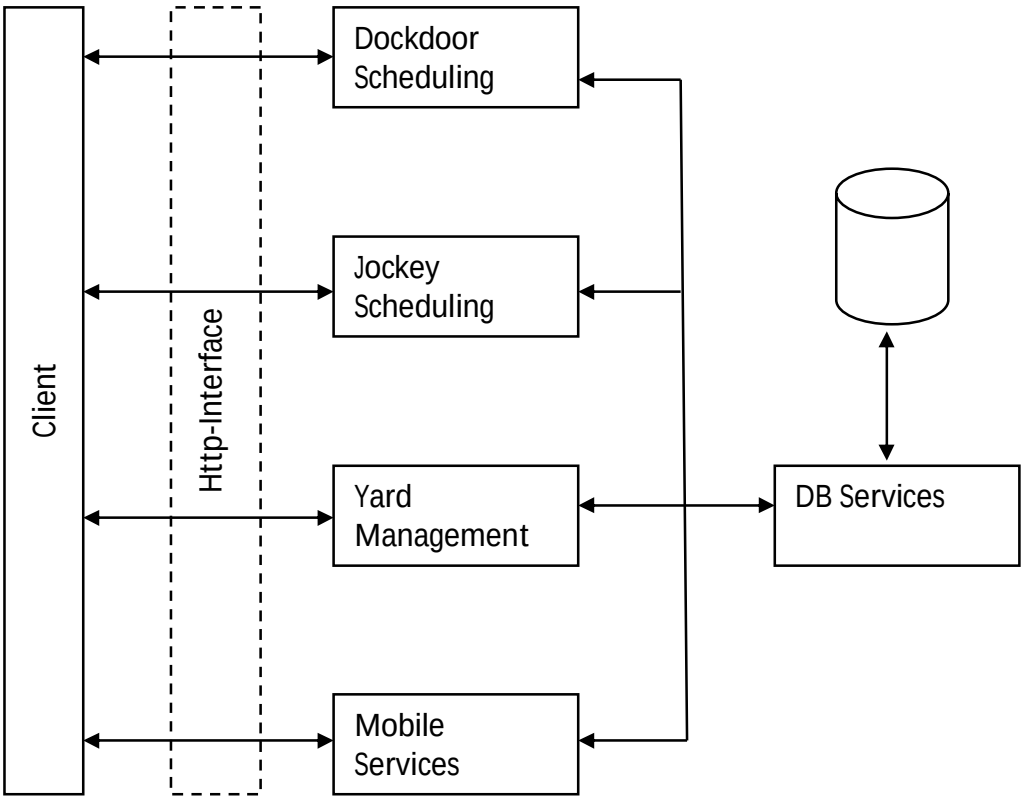

Figure 8 Architecture of Yard Management Server

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.

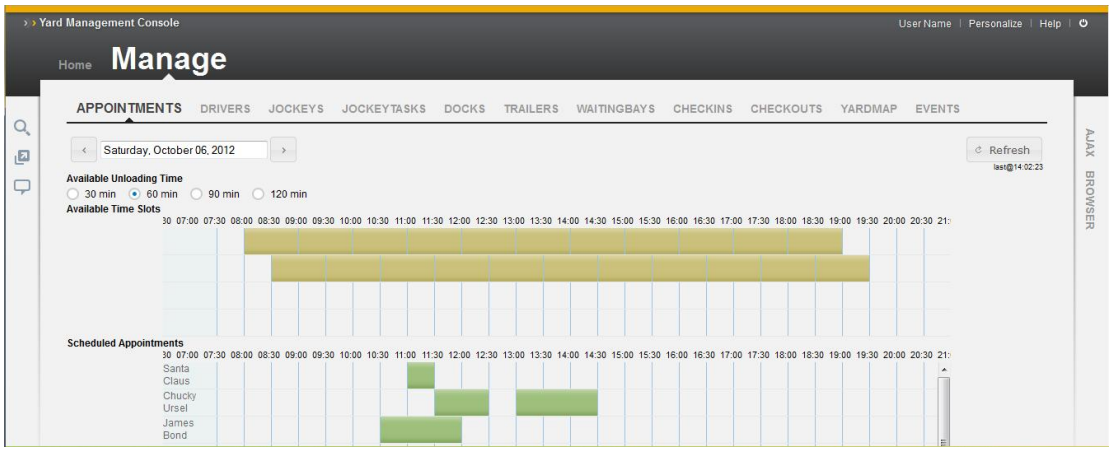

Figure 9 Yard Management Interface

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

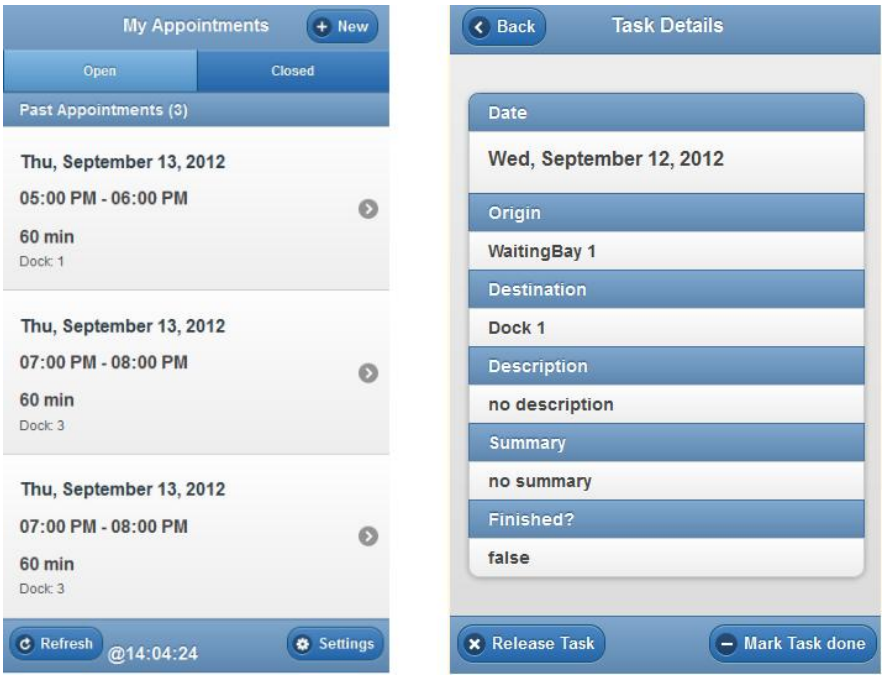

Figure 10 Driver Interface (l), Jockey Interface (r)

4.2 Used Technologies

The current use case application runs on SAP NetWeaver Cloud 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/from dock 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

5 Technical Integration

As discussed in Section 2 the scenario for integration was taken from the real world and the process has already been explained. The technical integration includes several different communication mechanisms and techniques.

5.1 Overview

In Figure 11, we present an overview of the aforementioned 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.

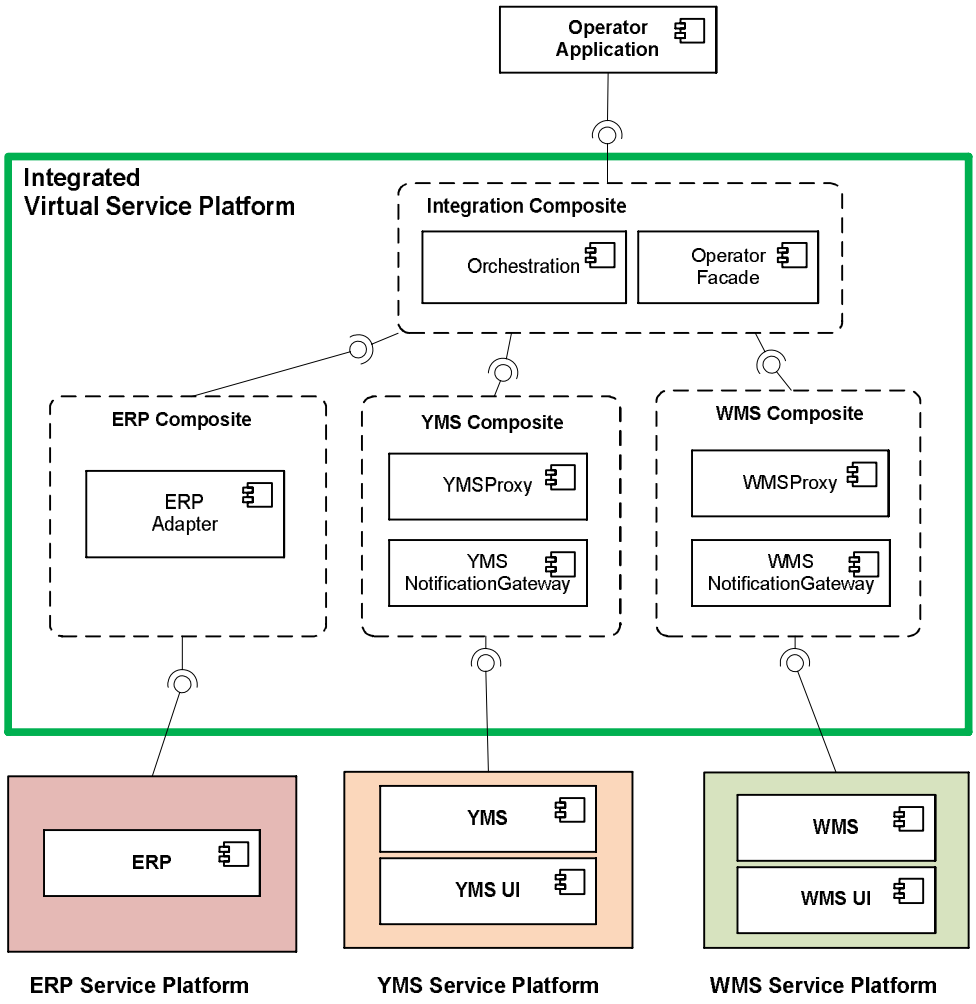

Figure 11 Overview of the INDENICA integration scenario

5.2 Architectural Design and Implementation

Based on the requirements described using the IRET tool (developed by PDM, see D1.2.2), the platform variability captured using the EASy-Producer (developed by SUH, see D2.2.1 and D2.4.1), and the architectural design decisions captured by the ADvISE tool (developed by UNIVIE, will be reported in D1.3.2), we use the service component view model provided by the VbMF tool suite (reported in D3.1 and D3.3.1) to model the high-level architecture of the integrated VSP (i.e., the green box in Figure 11). The high-level architecture of the VSP described using VbMF—as shown in Figure 12—is 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, and WMSNotificationGateway. Indeed, during the course of the 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 only need to annotate the previous Web service-based "proxy" components as "JMS" proxy components in the high-level view model and leverage the templates described in VbMF 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.

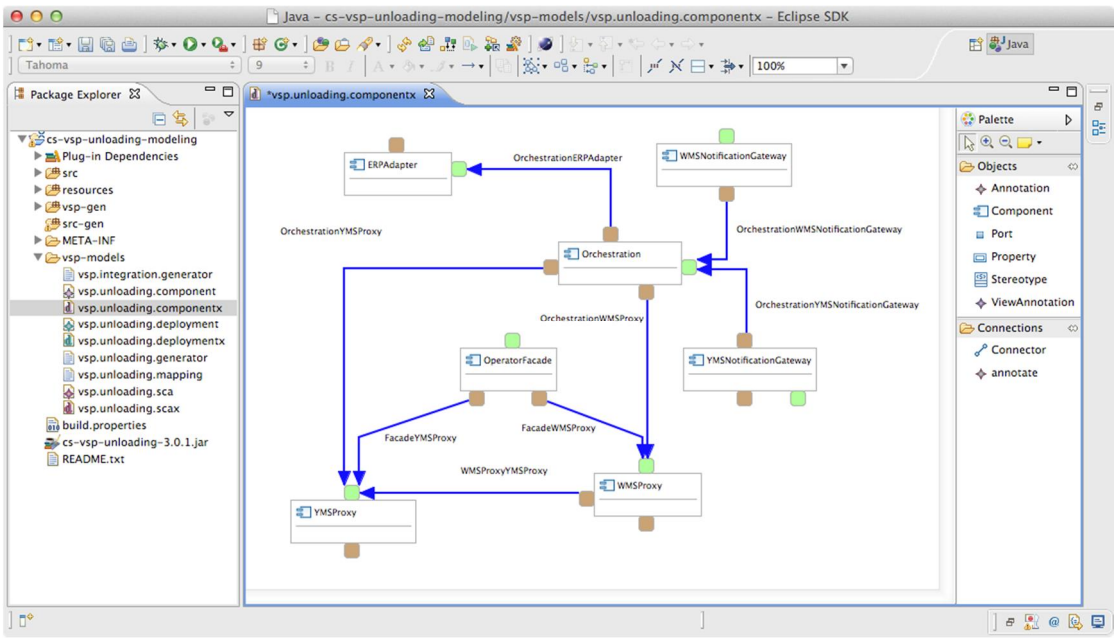

Figure 12 High-level modeling of the integrated VSP using VbMF Tool Suite

Once the high-level service component view is modeled, we can carry out formal validations using the OCL-like languages supported by VbMF (see §3.3 in D3.3.1). Then, we use VbMF's Mapping DSL (see §3.2 in D3.3.1) to refine the high-level view model to the low-level counterpart, which is specific for the Apache Tuscany SCA technology that we exemplify to deploy and run the VSP. The low-level view model is shown in Figure 13.

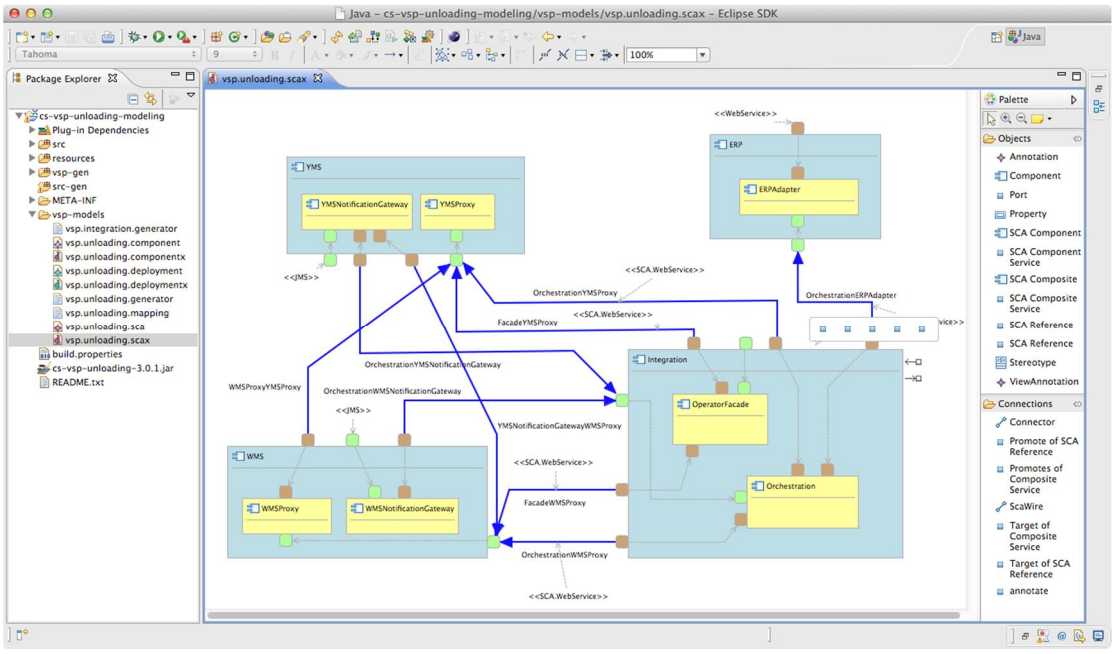

Figure 13 Low-level modeling of the integrated VSP for SCA 1.x using VbMF Tool Suite

We note that the components are refined with more specific details and grouped into "composites" that are the notion of component container in SCA. In addition, we use the SCA DSL (or so-called 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 §3.4 in D3.3.1). Based on the specification in the low-level view model (plus the Deployment view model presented in the subsequent section), the implementation and configuration of the VSP will be generated automatically (see §3.4 in D3.3.1).

In Figure 14, we show the generated Java code of the VSP components described above. Besides, SCA composite configurations and bindings are also generated. VbMF 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 14 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.

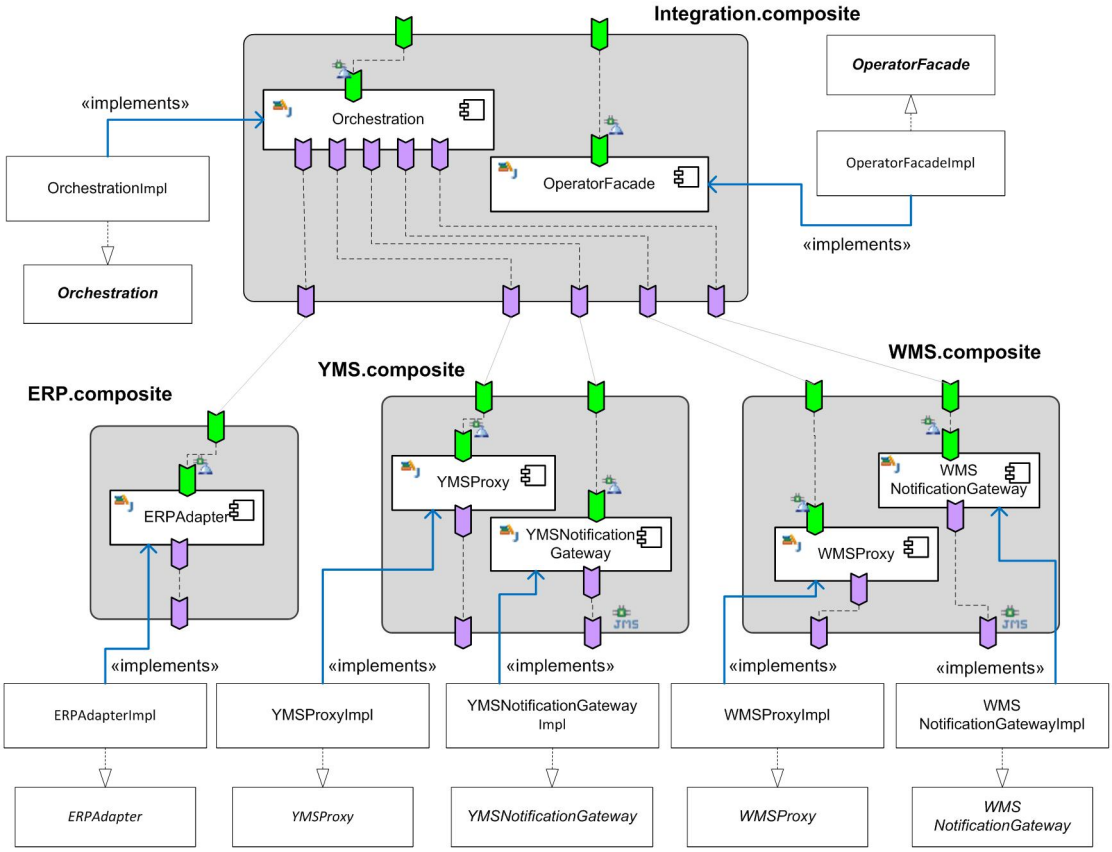

Figure 14 Schematic view of the generated VSP implementation

5.3 Deployment and Enactment

After describing and enriching the low-level view model of the VSP, we must define the deployment of the VSP into the Apache Tuscany Runtime. This can be done using the Deployment view model of VbMF Tool suite as shown in Figure 15. Essentially, for better load balancing and distribution, we decide to deploy each SCA composite described in the low-level view model in Figure 13 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.

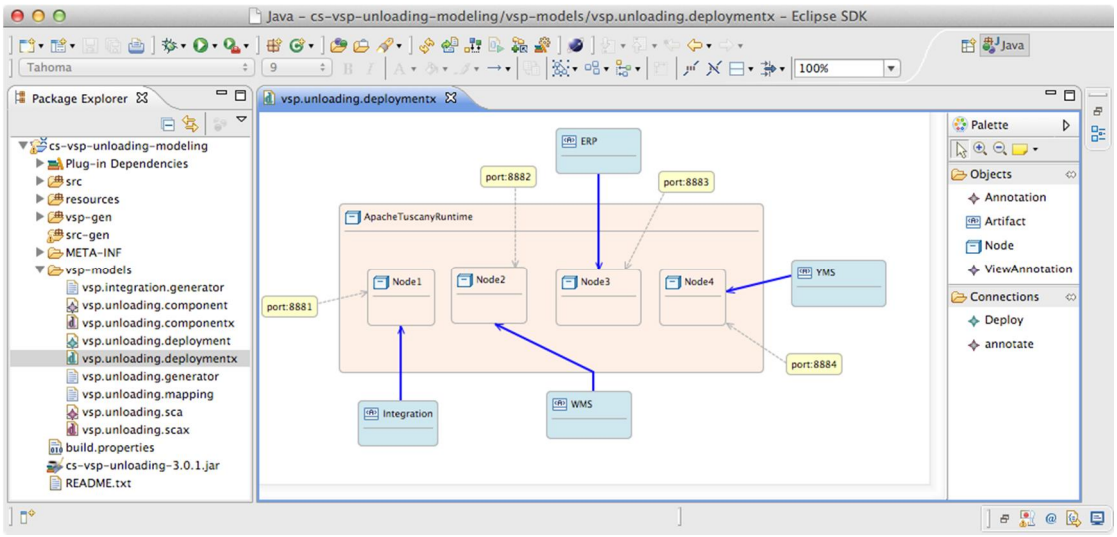

Figure 15 Modeling the deployment of the integrated VSP into Apache Tuscany Runtime

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 16 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 adapter 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 adapter between the VSP and the YMS service platform. This composite is 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.

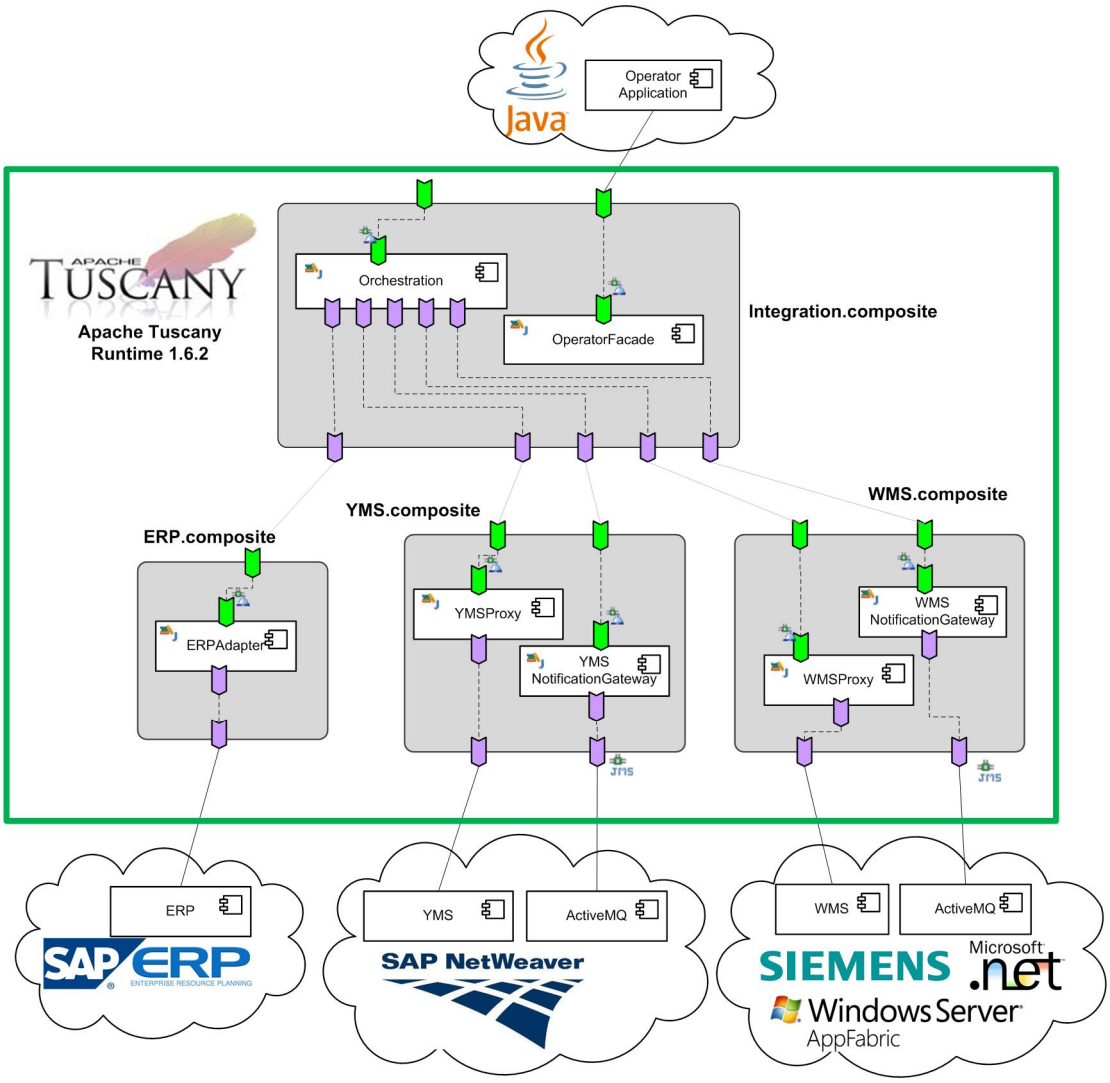

Figure 16 The actual deployment of the INDENICA integration scenario

6 Outlook

In Year 3 of the INDENICA project a third technical platform will be integrated into the overall scenario. In addition to that integrated variability models will be created. Also the support for monitoring will be improved and runtime adaption will be implemented, respectively.

Table of Figures

Figure 1 Check In Process 6
Figure 2 Error Handling Process 7
Figure 3 Unloading Process 7
Figure 4 Check Out Process 8
Figure 5 Warehouse Management System (WMS) Architecture 9
Figure 6 Warehouse User Interface (WUI) 10
Figure 7 Simulation Site (SimSite) 10
Figure 8 Architecture of Yard Management Server 13
Figure 9 Yard Management Interface 14
Figure 10 Driver Interface (l), Jockey Interface (r) 15
Figure 11 Overview of the INDENICA integration scenario 18
Figure 12 High-level modeling of the integrated VSP using VbMF Tool Suite 20
Figure 13 Low-level modeling of the integrated VSP for SCA 1.x using VbMF Tool Suite 20
Figure 14 Schematic view of the generated VSP implementation 21
Figure 15 Modeling the deployment of the integrated VSP into Apache Tuscany Runtime 22
Figure 16 The actual deployment of the INDENICA integration scenario 23