



Engineering Virtual Domain-Specific Service Platforms

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

Tool Suite for Virtual Service Platform Engineering (Final)

Abstract

In INDENICA, the notion of Virtual Service Platforms (VSP) shall be leveraged to protect the investments into service-based applications against potential external negative influences and threats such as the heterogeneity of the involving service platforms, service discontinuation, service evolutions, etc. In D3.1, we already presented the view-based design time and runtime architecture for developing and tailoring VSPs and an interim version of the Tool Suite for Virtual Service Platform Engineering in D3.3.1. In this report, we introduce the final version of the tool suite with additional advanced features and tool integrations.

Document ID:	INDENICA – D3.3.2
Deliverable Number:	D3.3.2
Work Package:	3
Type:	Prototype
Dissemination Level:	PU
Status:	Final
Version:	2.0
Date:	2013-09-30
Contributing partners	SUH, UNIVIE, PDM, SAP, SIE, TUV, NDL

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

Version History

1.1	20. Apr 2013	Initial final version based on the interim version D3.3.1
1.2	06. May 2013	Updated Section 1, 2 & 3
1.3	22. May 2013	Added Section 2.2
1.4	07. Jun 2013	Revised the second version
1.5	20. Jun 2013	Revised Section 3
1.6	01. Aug 2013	Added more subsections for Section 3
1.7	13. Aug 2013	Revised Section 2 and 4
1.8	23. Aug 2013	Revised and added subsections to Section 3
1.9	10. Sep 2013	Added and revised sections 5 and 6
2.0	30. Sep 2013	Revised and finalized the deliverable

Document Properties

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

Table of Contents

Table of Contents	3
1 Overview of the Tool Suite for Virtual Service Platform Engineering	4
2 Tool Suite Implementation (Final)	4
2.1 Introduction.....	4
2.2 Summary of new features in the final release	5
3 User's Guide	6
3.1 Installing the Tool Suite.....	6
3.2 Start with a simple project	10
3.3 Creating a high-level Service Component View	12
3.4 Creating a low-level Service Component View (aka SCA View)	12
3.5 Mapping the high-level views to the low-level counterparts	13
3.6 Defining a Deployment Model	15
3.7 View model validation/constraint checking.....	16
3.8 Generate SCA Configuration	20
3.9 Generating code for proxies and adapters	21
3.10 Running SCA Configurations and Code with Apache Tuscany	25
4 Developer's Guide.....	25
5 Application of the Tool Suite in the INDENICA use case.....	26
5.1 High-Level Service Component View	26
5.2 Mapping DSL.....	26
5.3 SCA View (aka Low-level Service Component View for SCA).....	28
5.4 Deployment View	28
5.5 Generator DSL for SCA	28
5.6 Generator DSL for proxy and adapter generation	31
6 Conclusions	32
Table of Figures.....	33
References	34

1 Overview of the Tool Suite for Virtual Service Platform Engineering

The Tool Suite for Virtual Service Platform Engineering is based on the view-based design time and run-time architecture described in D3.1. The fundamental feature of the view-based architecture is to leverage the notation of architectural views to capture different aspects of a complex system and leverage the model-driven development paradigm to link those architectural views and generate code, configurations, runtime directives, and so on.

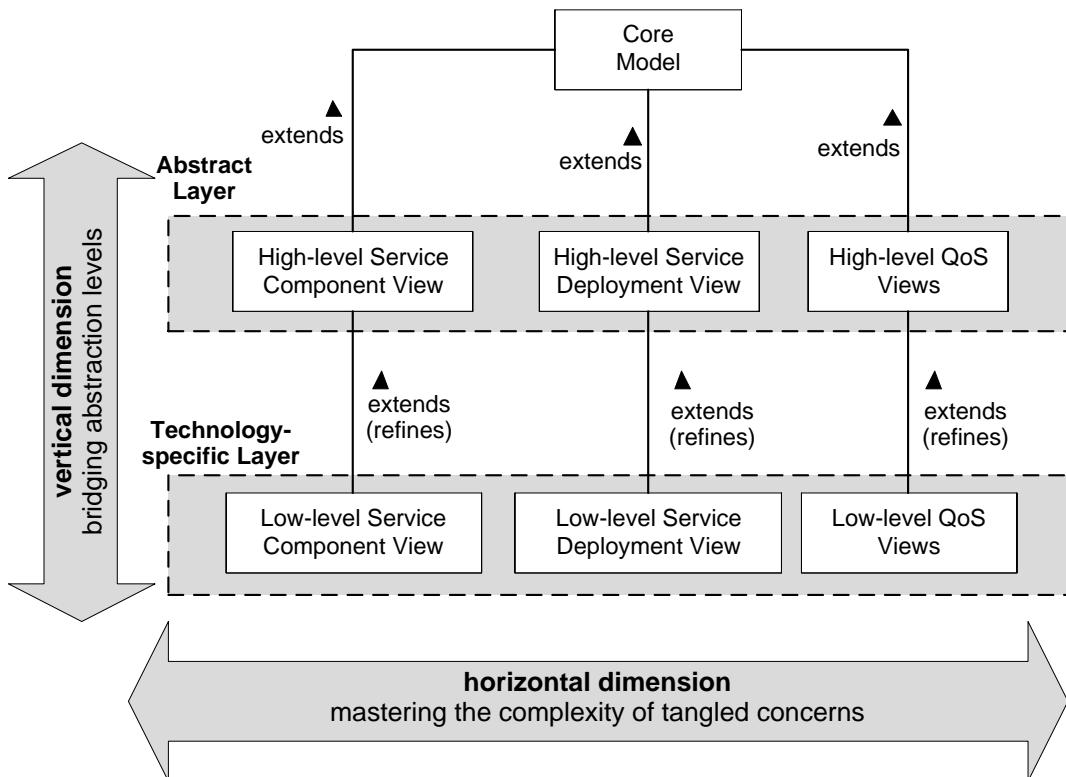


Figure 1 Overview of the view-based design time and runtime architecture

2 Tool Suite Implementation (Final)

In this document, we report the final release of the INDENICA Tool Suite for Virtual Service Platform Engineering (or Tool Suite for short, from now on). This version has been significantly enhanced based on the feedbacks and experiments we achieved from our partners from previous release as well as our on-going scientific results.

2.1 Introduction

As mentioned in the previous version, the Tool Suite implementation is based on Eclipse Modelling Framework (EMF), which is a popular framework for modelling and development of applications and systems.

The most advantage of using Eclipse Modelling Framework is that we gain better integration and interoperability with existing development tools developed based on

EMF Ecore, a MOF¹-compliant meta-model, and XMI² (XML Metadata Interchange), a standard for serializing models.

The Tool Suite provides different graphical and textual editors for creating and manipulating view models such as Service Component view, Deployment view, and so on. The template-based code generation rules are developed using the Xtend language provided by Eclipse M2T project. Using these rules, we can automatically generate virtual service platform configurations that can be deployed and executed in Apache Tuscany 1.6 for SCA.

The Tool Suite depends on following technologies and platforms:

Technology/Platform	Version	Website
Eclipse IDE	3.7+	http://eclipse.org
Eclipse Modelling Framework (EMF)	2.5+	http://eclipse.org/modeling/emf
Graphical Modeling Framework (GMF)	1.5+	http://www.eclipse.org/modeling/gmp
Xtend	2.3+	http://www.eclipse.org/xtend
Xtext	2.3+	http://www.eclipse.org/Xtext
Epsilon Validation Language	1.1+	http://www.eclipse.org/epsilon/doc/evl

For the convenience of the end-user, the Tool Suite has been developed and bundled in terms of Eclipse plug-ins. After installing the aforementioned required plug-ins, you can download the plug-ins bundle³ and decompress it into the folder "dropins" of the Eclipse installation.

Instead of searching and installing all aforementioned required plug-ins, which is tedious and error-prone, one can use the update site to install all necessary components and start using the Tool Suite right away (see 2.2.6.)

2.2 ***Summary of new features in the final release***

- Live model validation/constraint checking (part of [4])
- Mapping language and execution engine (c.f. [1])
- Integration with ADVISE of WP1 (c.f. D1.3.2) based on the conference paper [1])
- Integration with the runtime infrastructure WP4 (reported in the conference papers [2,3,4])
- More expressive, powerful code generation based on Xtend
- Code generation/view mapping launchers
- Online Tool Suite update site

¹ <http://www.omg.org/mof>

² <http://www.omg.org/spec/XMI>

³ <https://swa.univie.ac.at/~huytran/vbmf/vbmf-plugins-only.zip>

3 User's Guide

The features provided to the end-users by the INDENICA Tool Suite are including:

- Creating/manipulating a high-level Service Component View (software architects and/or domain experts)
- Defining/manipulating a mapping of the high-level Service Component View to a low-level Service Component View for SCA technology (software architects and developers)
- Creating/manipulating the low-level Service Component View (developers and/or IT administrators—regarding runtime middleware)
- Enriching/annotating the low-level Service Component View using the Generator DSL (software architects (very seldom), developers and IT administrators)
- Defining deployment strategy (IT administrators and/or developers)
- Defining integration interfaces (e.g., proxies, adapters, gateways, etc.) using the integration part of the Generator DSL (developers)
- Constraint checking on all models and revised the models according to reported errors (domain experts, software architects, developers)
- Using launchers to generate SCA configurations and Java code (developers)
- Augmented the generated code with hand-written code, e.g., concrete orchestration or business logic (developers)

3.1 Installing the Tool Suite

For the convenience of end-users, we create and maintain an online update site for the Tool Suite at the following address:

<http://indenica.swa.univie.ac.at/public/vb/update>

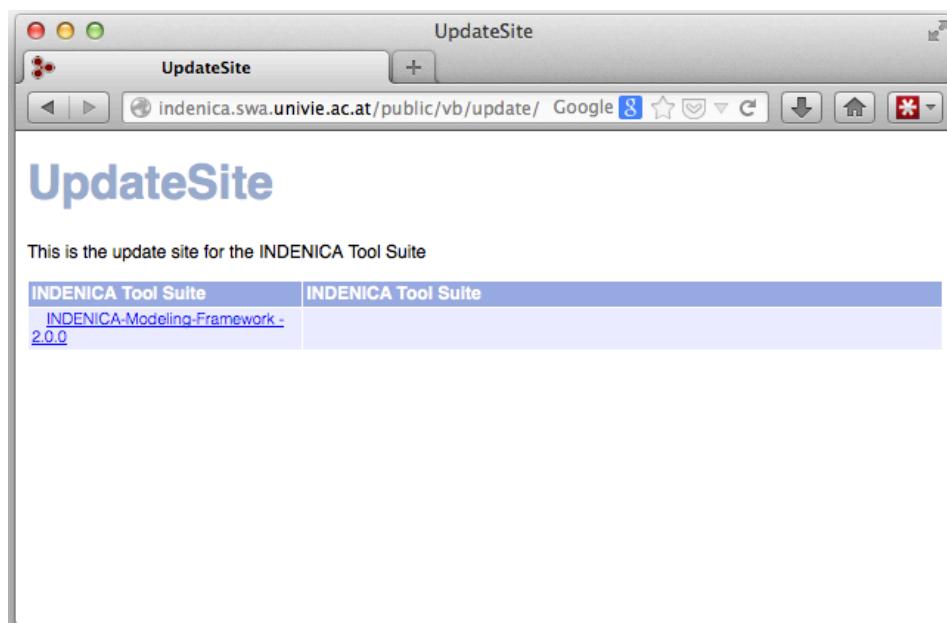


Figure 2 Update Site for INDENICA Modelling Tool Suite

Every release of the Tool Suite will be deployed to the update site and the end-users can directly install or update.

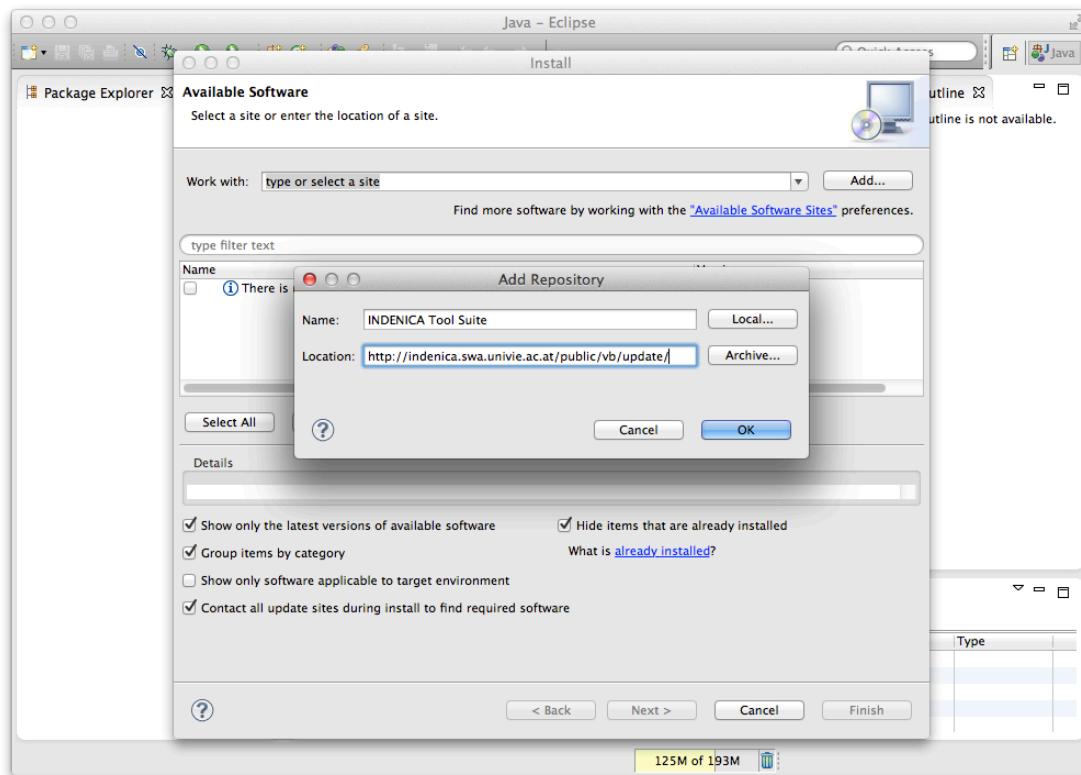
Assuming that all required frameworks and libraries of the Tool Suite have been installed, we can use the update site to directly install or upgrade the Tool Suite.

The required frameworks/libraries are including:

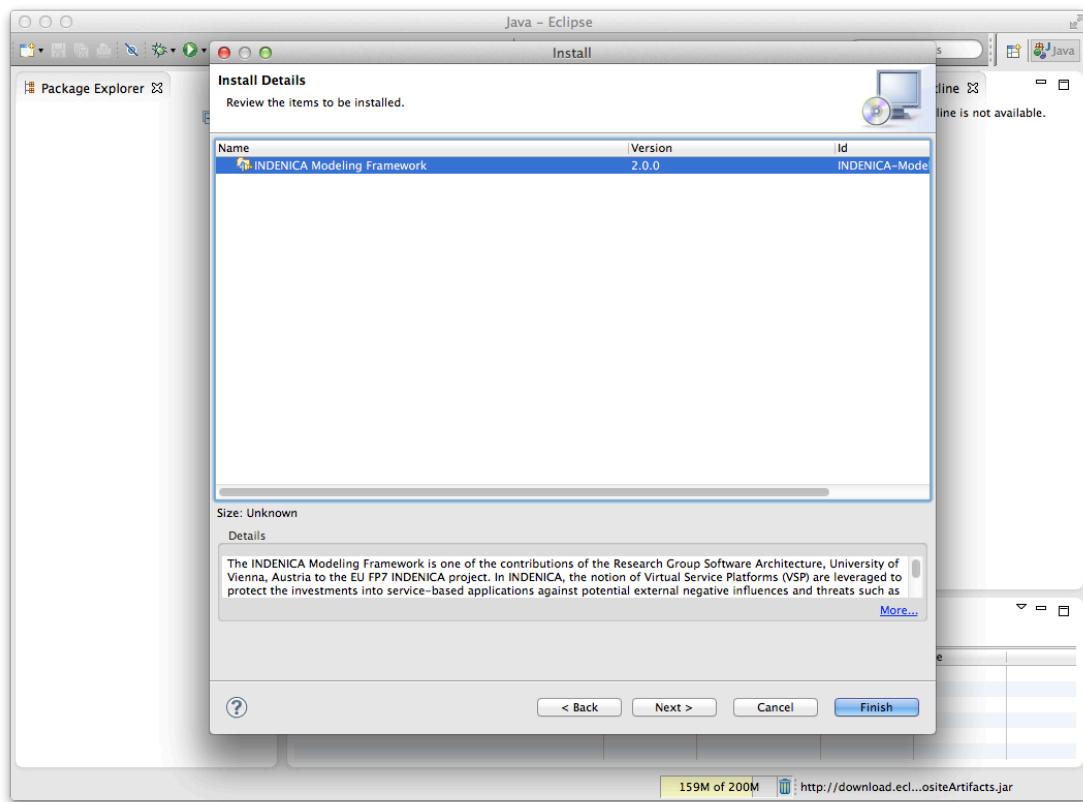
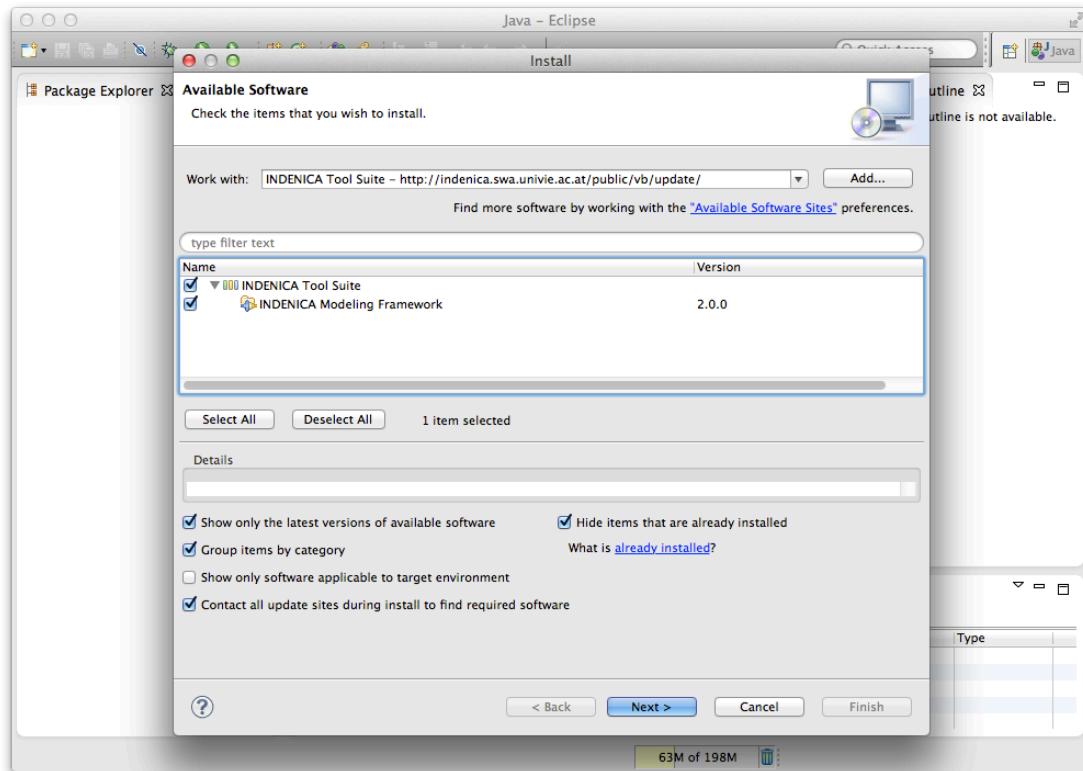
- Eclipse Modeling Framework (EMF)
- Epsilon Validation Language (EVL)
- Eclipse Xtext/Xtend 2.4
- Eclipse Graphical Modeling Framework (GMF)

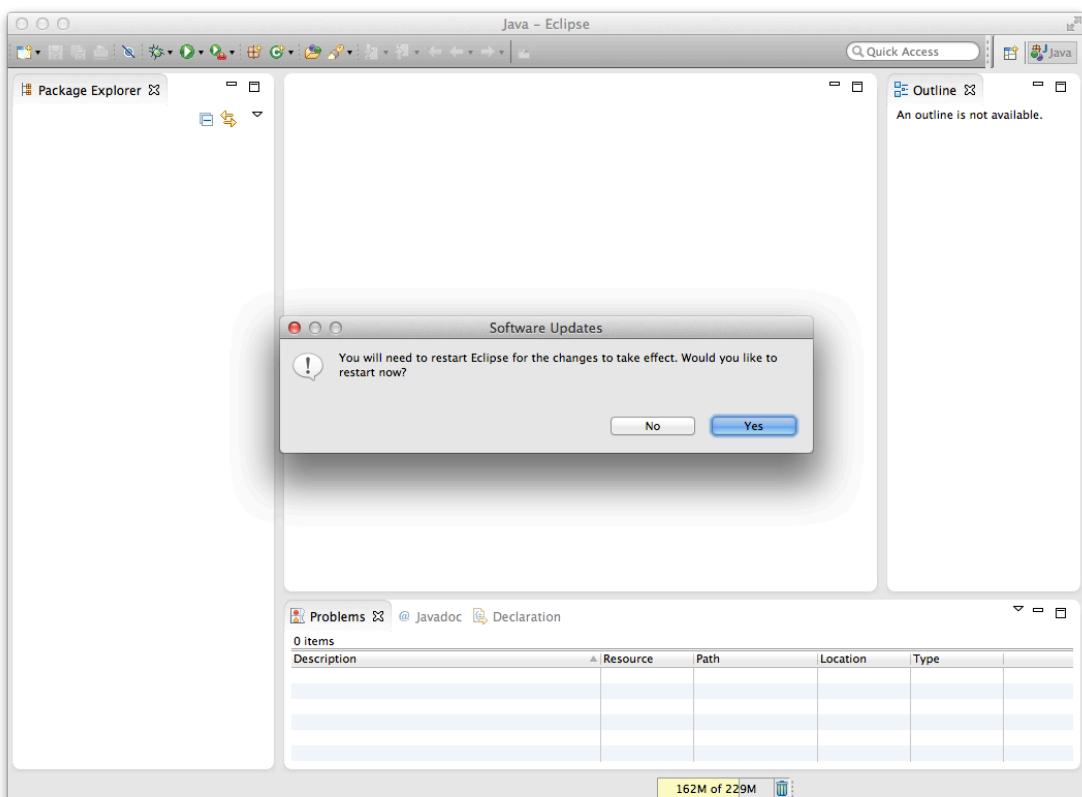
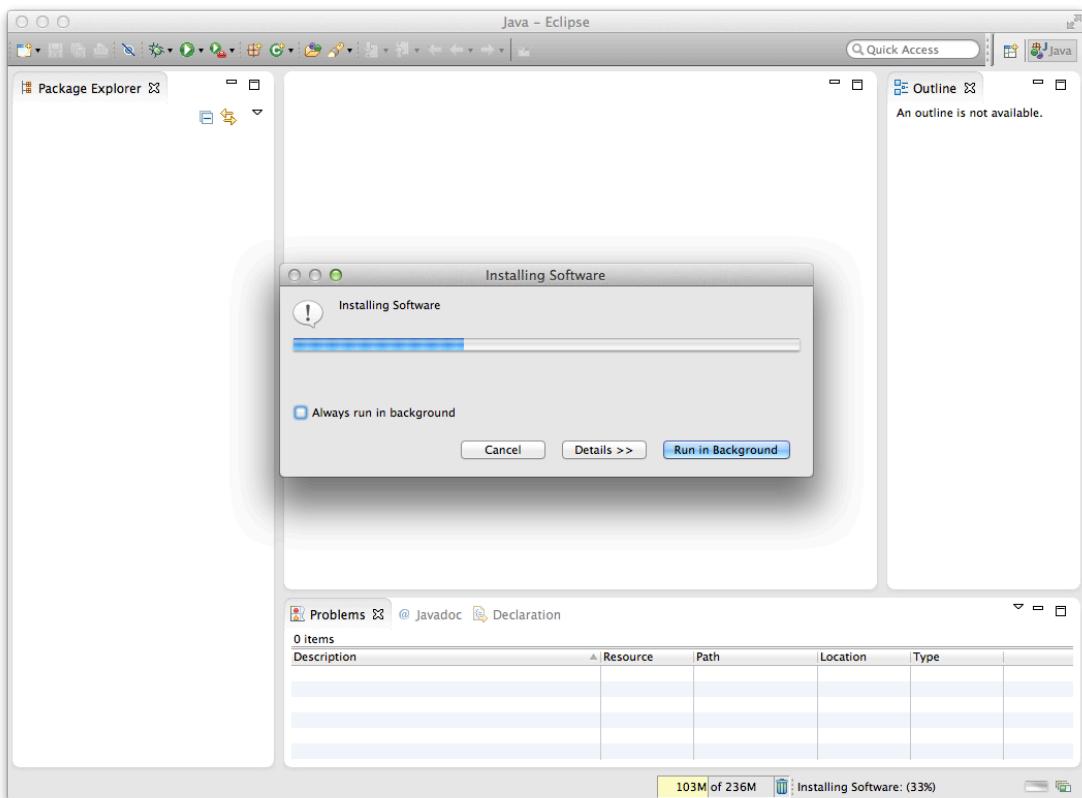
Open Eclipse, go to „Help“, „Install New Software...“, then choose „Add“, in order to add a new software update site. In the first text box, type „INDENICA Tool Suite“, in the second text box, type the update site URL and press „OK“:

<http://indenica.swa.univie.ac.at/public/vb/update>



In the dialog, choose „INDENICA Tool Suite“, and click „Install“

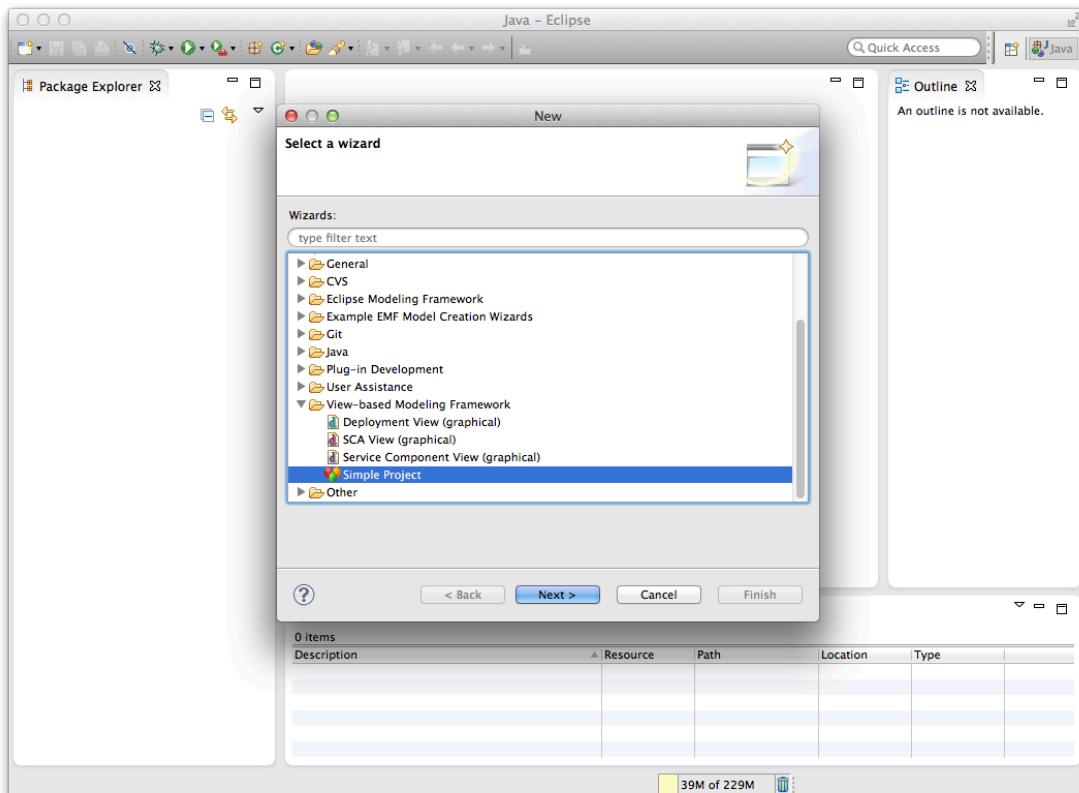




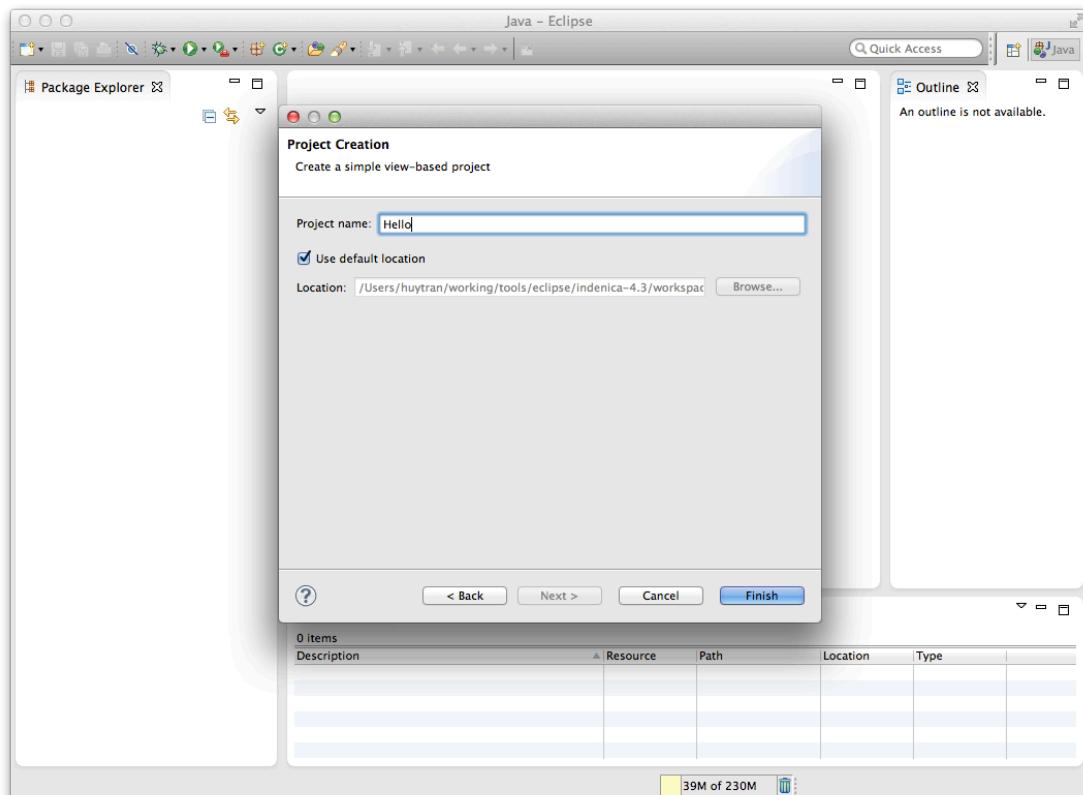
3.2 Start with a simple project

After installing the INDENICA Tool Suite, one can start modelling and developing virtual service platforms from scratch. Nevertheless, we provide a wizard that can help users to gain quicker start with an initial project setting with sample models.

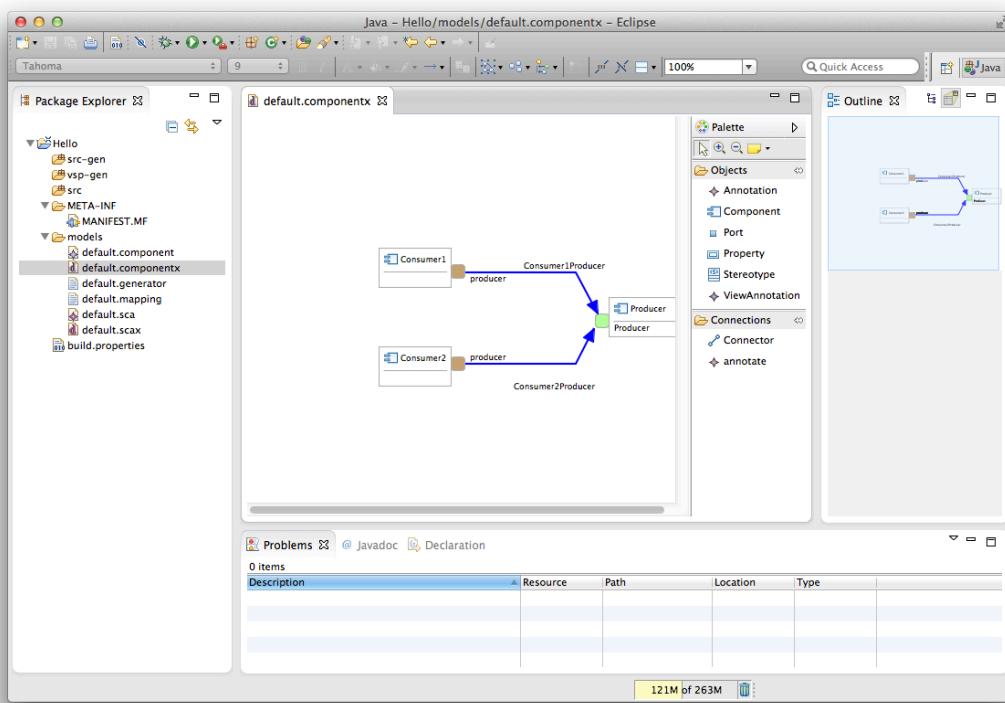
Let's create a new project by pressing „**Ctrl + N**“ (Windows and Linux) or „**cmd + N**“ (Mac OS X), then go to the category „**View-based Modeling Framework**“, choose „**Simple Project**“



then give the new project a name, for instance, „**Hello**“ and press „**Finish**“.

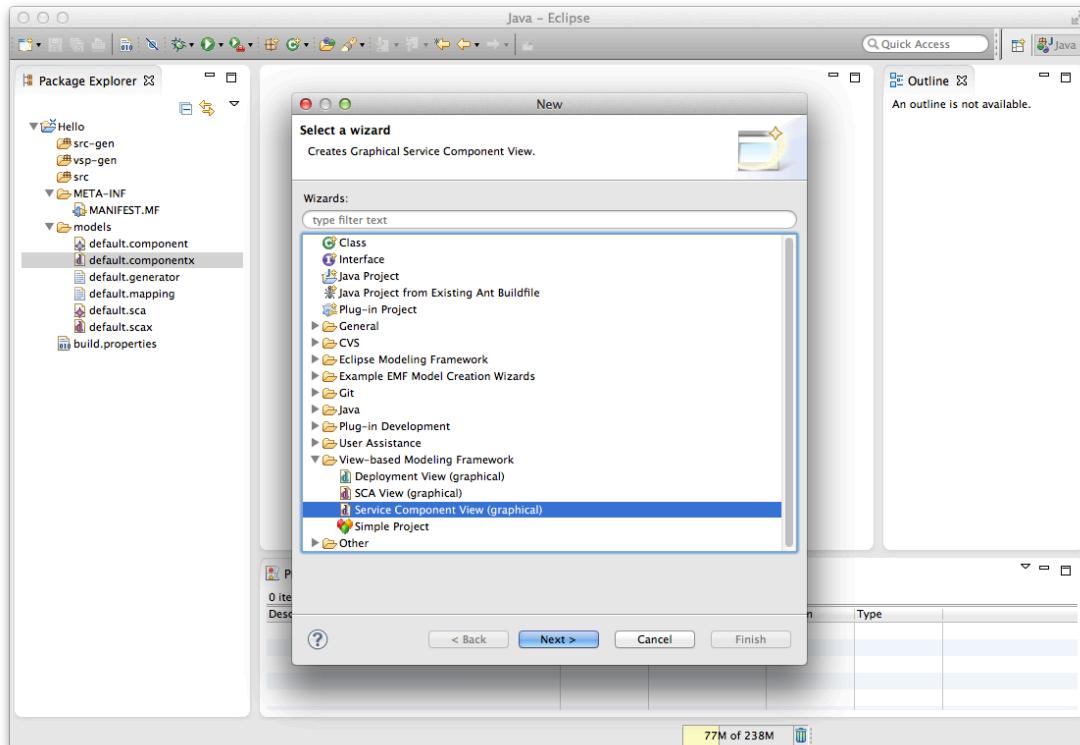


A new modelling project named „Hello“ will be created with some sample models and DSLs as a basic skeleton of a typical INDENICA modelling and development project.



3.3 Creating a high-level Service Component View

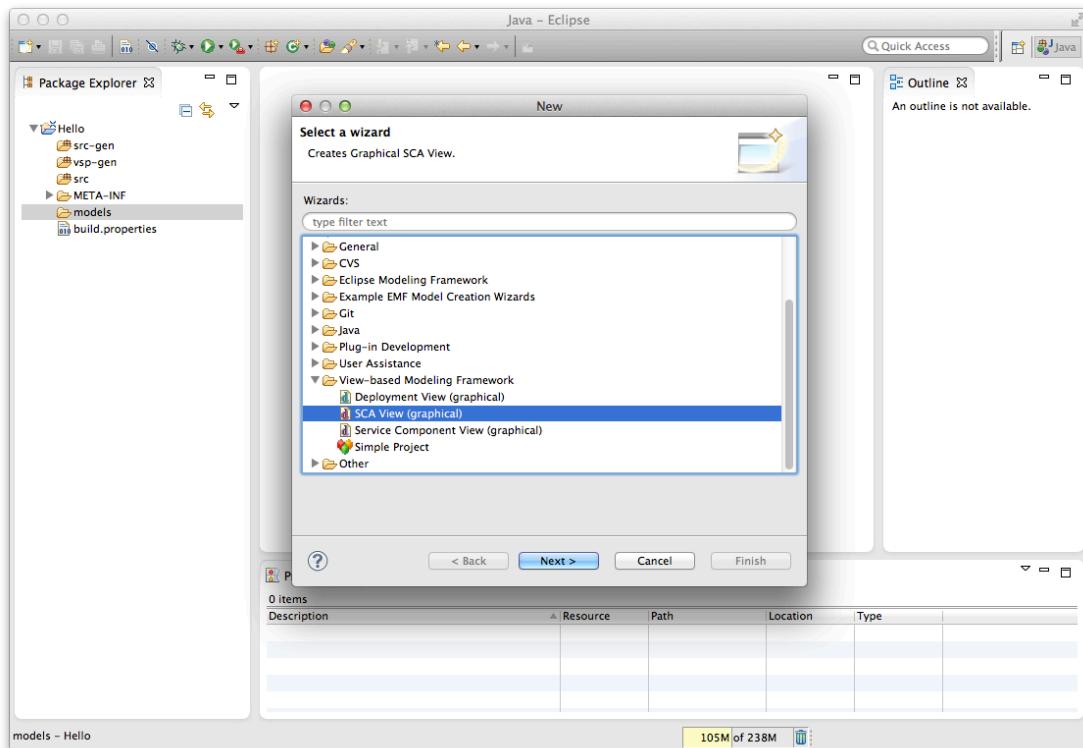
In case the user wants to create his own models, he can start with the wizard, choose a corresponding view model, such as Service Component View, SCA View, Deployment View, etc. He can start with a blank high-level Service Component View.



Then, adds the components, ports, and connectors to describe the architecture of the Virtual Service Platform.

3.4 Creating a low-level Service Component View (aka SCA View)

Similarly, the low-level Service Component View (SCA View) can also be created in the same manner.



3.5 Mapping the high-level views to the low-level counterparts

To save the effort of manually creating the SCA view, given the existing high-level counterpart created in the previous step, the Tool Suite supports view mapping.

Firstly, we need to define a mapping DSL that operationally specify how the high-level Service Component View will be refined to an SCA View. An excerpt of the mapping DSL for the case study [D5.1] is provided below.

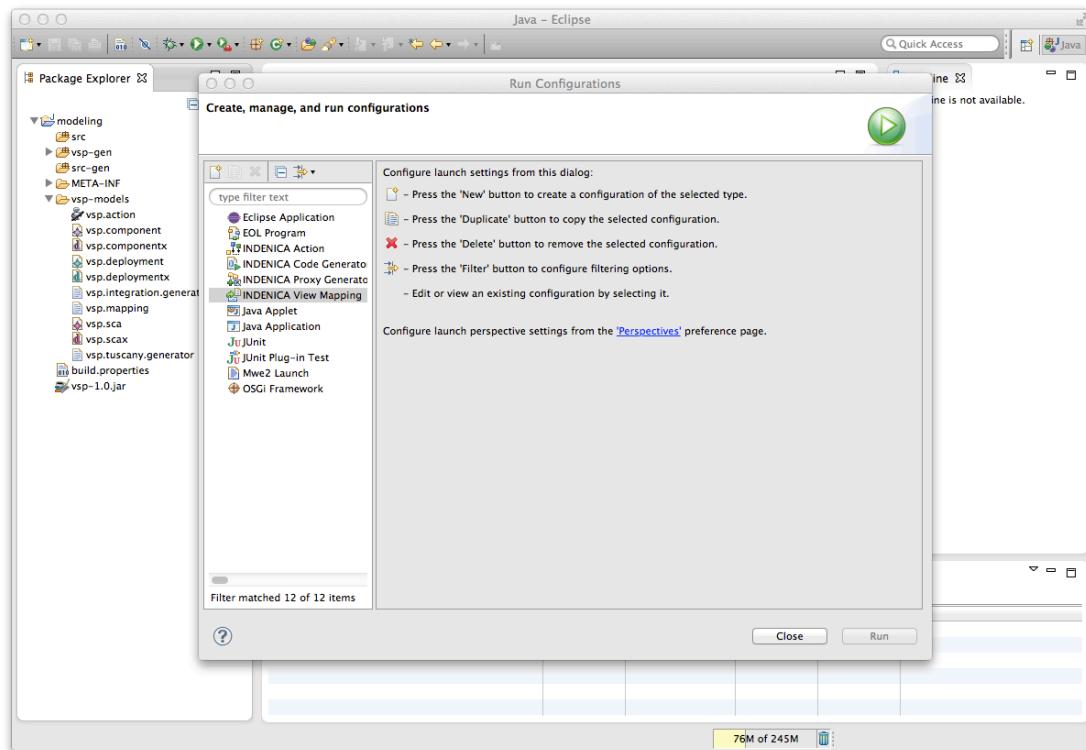
```
module vsp.mapping

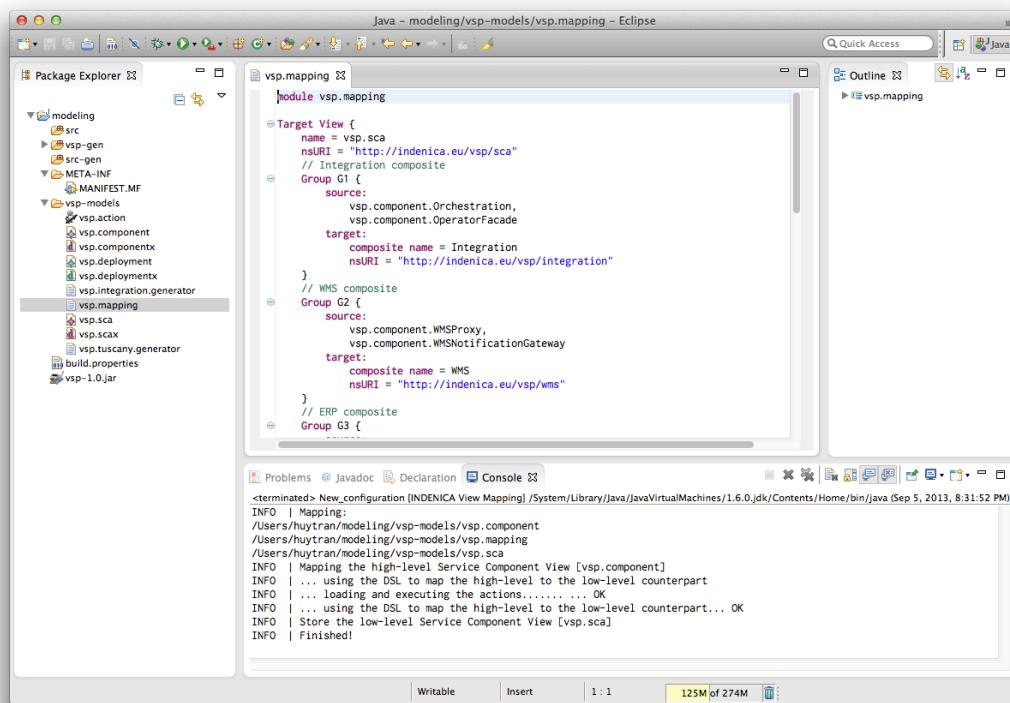
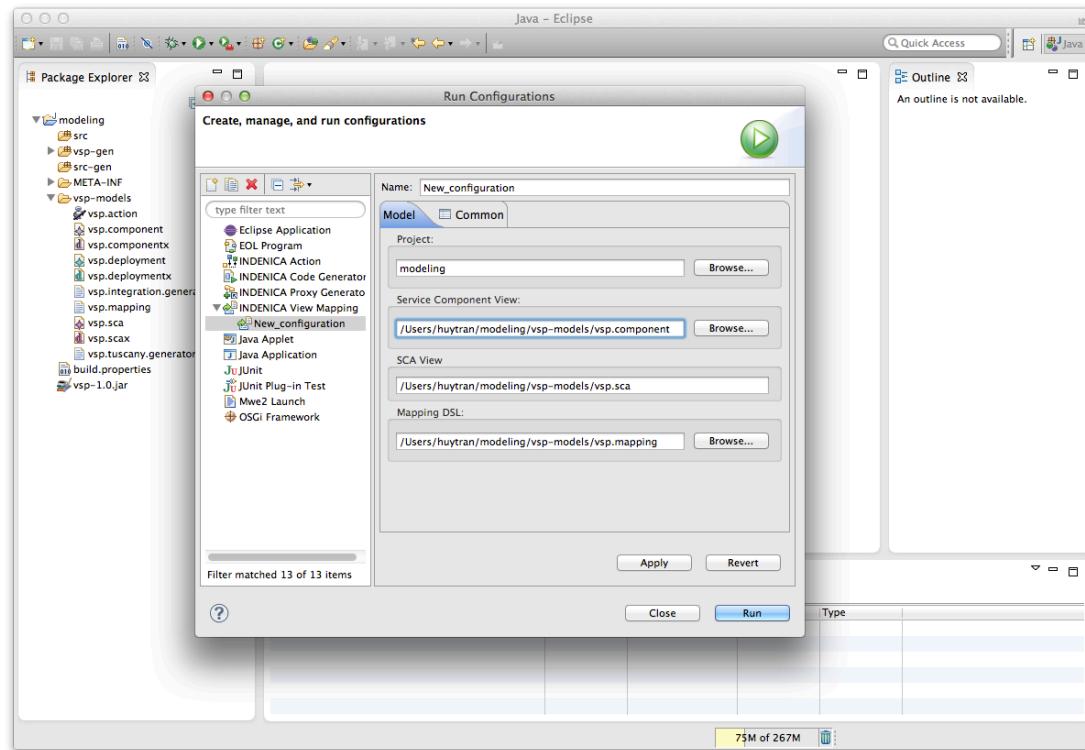
Target View {
    name = vsp.sca
    nsURI = "http://indenica.eu/vsp/sca"
    // Integration composite
    Group G1 {
        source:
            vsp.component.Orchestration,
            vsp.component.OperatorFacade
        target:
            composite name = Integration
            nsURI = "http://indenica.eu/vsp/integration"
    }
    // WMS composite
    Group G2 {
        source:
            vsp.component.WMSProxy,
            vsp.component.WMSNotificationGateway
        target:
            composite name = WMS
            nsURI = "http://indenica.eu/vsp/wms"
    }
    ...
}
```

We provide suitable Java APIs for producing an SCA view given an input high-level Service Component view and a mapping DSL. An example of using the API for mapping a high-level view according to a mapping DSL specification and storing the resulting low-level view in the specified path.

```
Mapping.map(High-level-Component_View, Mapping_DSL, Low-level-View-Path);
```

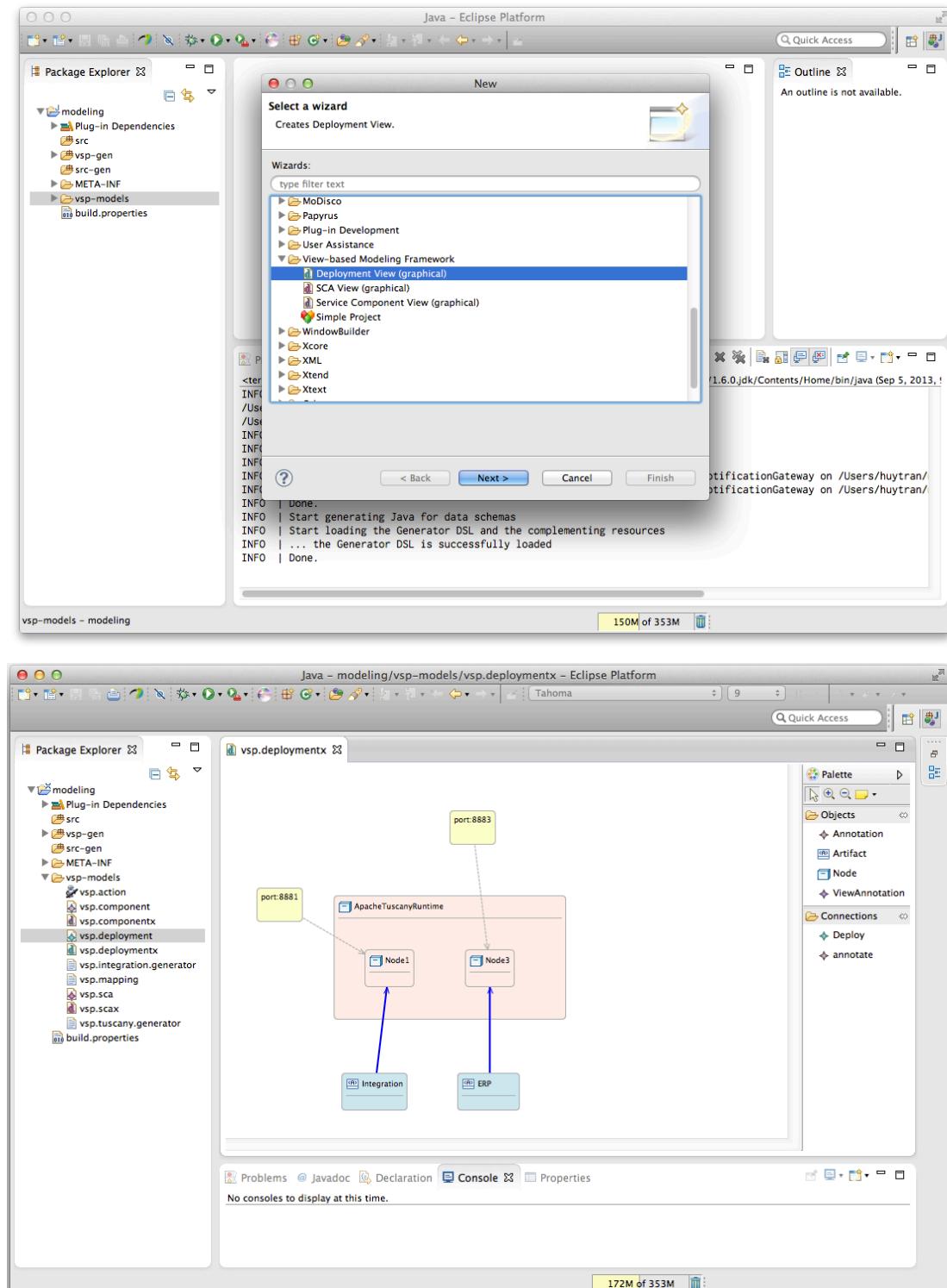
In the final version of the Tool Suite, we provide more easy to use launchers in Eclipse that the end-users can simple pick the view models and mapping DSLs and press a button to accomplish the mapping task.





3.6 Defining a Deployment Model

A Deployment view model is used by the administrators (perhaps together with the software architects) to denote where the software artefacts are hosted and even more specific, how to deploy particular artefacts.



3.7 View model validation/constraint checking

In the interim version, in order to ensure the integrity and consistency of view models, the Tool Suite enables the verification of view models using an OCL-based constraint checking mechanism. This is implemented by using the **Check** language.

For instance, we can write some simple OCL constraints for the Service Component view:

```

import core;
import component;
import sca;

context core::NamedElement ERROR
    "Name of " + this + " must not be empty" :
    name != null || name.length > 0;

context component::ComponentView ERROR
    "Port " + this + " is not a valid element of the Component View":
    element.typeSelect(component::Port).isEmpty
;

context component::Connector ERROR
    "Connector " + this + " must link a required port to a provided":
    source.kind == component::PortKind::REQUIRED
    && target.kind == component::PortKind::PROVIDED
;

context component::Property ERROR
    "Property " + this + " must have both name and value" :
    name != null && name.length > 0
    && value != null && value.length > 0;

context component::ComponentView ERROR
    "Ports are not valid elements of the Component View " + this :
    element.typeSelect(component::Port).isEmpty
;
```

Similarly, the Tool Suite provides relevant Java APIs for verification purposes. The following Java class illustrates the usage of the aforementioned APIs.

```

Issues issues = ComponentValidator.validateComponentView(CHECK_FILE,
COMPONENT_FILE);
if (issues != null && issues.hasErrors()){
    // do something with the issues
    // e.g., informing the developers
}
issues = ComponentValidator.validateScaView(CHECK_FILE, SCA_FILE);
if (issues != null && issues.hasErrors())
    // do something with the issues
}
```

In this release of the Tool Suite, the aforementioned model checking method is still available. Nevertheless, it is not quite visible to the end-users, and therefore, becomes less helpful when the end-users are working on the models.

Therefore, we develop a new feature for “live and dynamic verification and validation” for the Tool Suite. The verification and validation of view models are accomplished by using Eclipse Epsilon Validation Language (EVL), which is a powerful and expressive dynamic OCL-like language and can be integrated with any EMF-based editors or GMF-based editors to provide live validations and powerful quick fixes.

For instance, we can define similar OCL-based rules as before in EVL:

```

context NamedElement {
    constraint HasName {
        check : self.name.isDefined()
        message : 'Unnamed ' + self.eClass().name + ' not allowed'
    }
}

context Connector {
    critique SelfLoop {
        guard : self.satisfies('HasName')
        check : self.source.eContainer().name <> self.target.eContainer().name
        message : self.name + ' creates a self-referenced loop'
    }

    constraint RightDirection {
        guard : self.satisfies('HasName')
        check : self.source.kind = PortKind#REQUIRED and self.target.kind = PortKind#PROVIDED
        message : self.name + ' is invalid as a connector must go from a required to a provided port'
    }

    critique Duplication {
        check : not Connector.allInstances.exists(c | c <> self and c.source = self.source and c.target = self.target)
        message : self.eClass().name + '\' ' + self.name + '\' might be duplicated'
    }
}

context Port {
    critique LowerCaseRequiredPortName {
        guard : self.satisfies('HasName')
        check : self.kind = PortKind#REQUIRED implies self.name.firstToLowerCase() = self.name
        message : 'Required port ' + self.name + ' should start with an lower-case letter'
        fix {
            title : 'Rename to ' + self.name.firstToLowerCase()
            do {
                self.name := self.name.firstToLowerCase();
            }
        }
    }

    critique UpperCaseProvidedPortName {
        guard : self.satisfies('HasName')
        check : self.kind = PortKind#PROVIDED implies self.name.firstToUpperCase() = self.name
        message : 'Provided port ' + self.name + ' should start with an upper-case letter'
        fix {
            title : 'Rename to ' + self.name.firstToUpperCase()
            do {
                self.name := self.name.firstToUpperCase();
            }
        }
    }

    constraint NotBelongingToView {
        guard : self.satisfies('HasName')
        check :
ComponentView.allInstances.select(v|v.element.includes(self)).isEmpty()
        message : 'Port ' + self.name + ' cannot belong to the view '
    }
}

context Property {

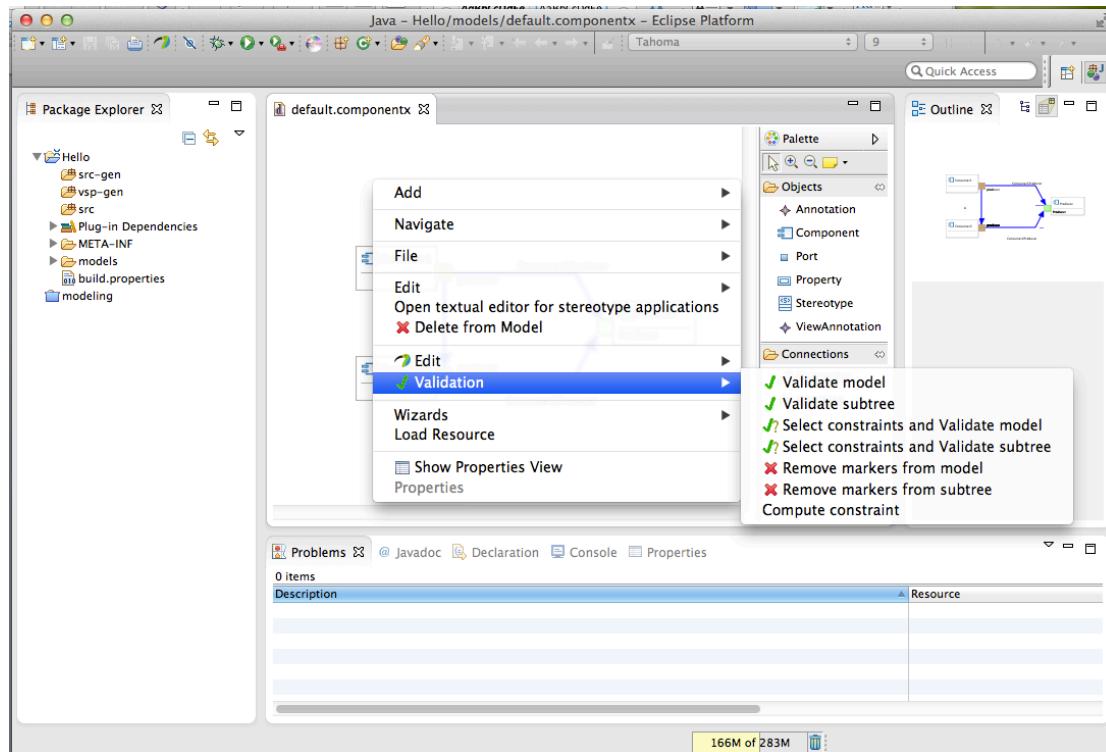
```

```

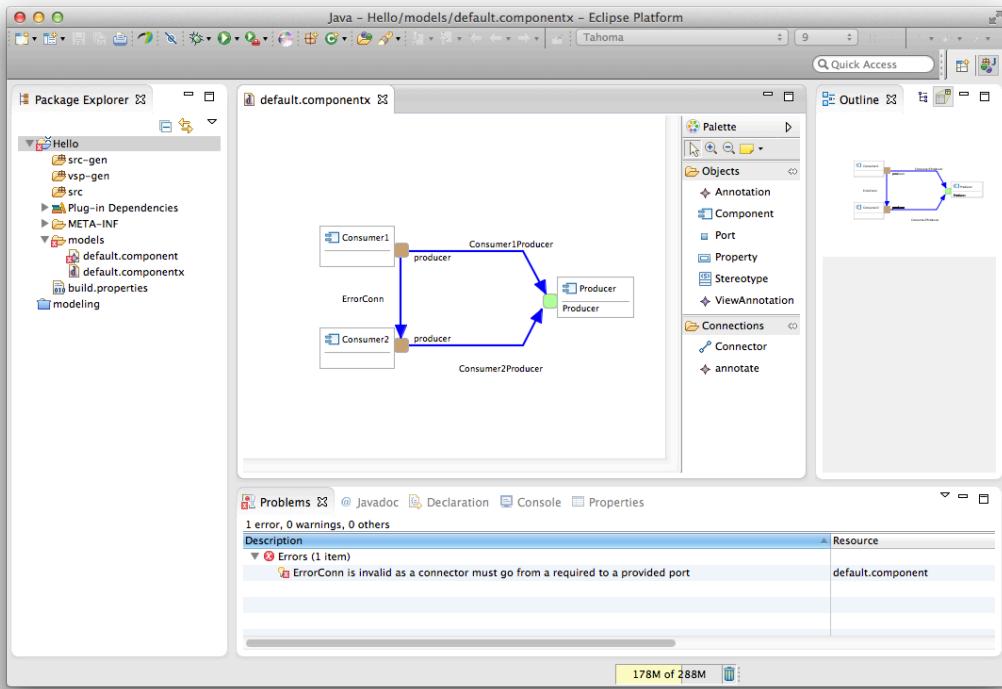
constraint HasNameAndValue {
    check : self.name.isDefined() and (self.value.isDefined() or
self.defaultValue.isDefined())
    message : "Property " + self.name + " must have a valid (default)
value"
}

```

In the graphical view model editors, for instance, the Service Component View, one can Right-Click and select “**Validation**”, and choose an appropriate option to start validating the model.



Otherwise, whenever the models are saved, the validators are also triggered to check and inform the errors in the “**Problem View**”.



For instance, we just add a connector named “**ErrorConn**” from a required port to another required, which is against the rule “**RightDirection**”. A corresponding error is shown in the “Problem View”. Click on these errors can bring up the models and the elements to be active for fixing the problems.

In the final release, we define model consistency rules for the provided views such as Service Component View, SCA View, Deployment View, etc. Nevertheless, defining rules for other views can be done in the same manner using EVL.

3.8 Generate SCA Configuration

After achieving the SCA view (either as a refinement of the high-level Service Component View or a fresh one built from scratch), we need to define necessary SCA-specific annotations using the SCA DSL.

To do so, we just create a new file with the extension „.generator“. The Generator DSL editor will be triggered and will support us to formulate the DSL. An example of the Generator DSL is shown in the following listing.

```
module vsp.generator
{
    bindings {
        Web Service { // binding of the Orchestration
            uri : "http://127.0.0.1:8881/Orchestration"
        }
        binds {
            vsp.sca.Integration.Orchestration.Orchestration,
            vsp.sca.YMS.YMSNotificationGateway.orchestration,
            vsp.sca.WMS.WMSNotificationGateway.orchestration
        }
        Web Service { // binding of the OperatorFacade
            uri : "http://127.0.0.1:8881/OperatorFacade"
        }
}
```

```

    binds {
        vsp.sca.Integration.OperatorFacade.OperatorFacade
    }
    ...
}

interfaces {
    interface eu.indenica.vsp.orchestration.ws.Orchestration describes {
        vsp.sca.Integration.Orchestration.Orchestration,
        vsp.sca.WMS.WMSNotificationGateway.orchestration,
        vsp.sca.YMS.YMSNotificationGateway.orchestration
    }
    interface eu.indenica.vsp.operatorfacade.ws.OperatorFacade describes {
        vsp.sca.Integration.OperatorFacade.OperatorFacade
    }
    interface eu.indenica.vsp.wmsproxy.WMSProxy describes {
        vsp.sca.WMS.WMSProxy.WMSProxy,
        vsp.sca.Integration.OperatorFacade.wmsProxy,
        vsp.sca.Integration.Orchestration.wmsProxy
    }
    ...
}

implementations {
    class eu.indenica.vsp.impl.OrchestrationImpl implements
        vsp.sca.Integration.Orchestration
    class eu.indenica.vsp.impl.OperatorFacadeImpl implements
        vsp.sca.Integration.OperatorFacade
    class eu.indenica.vsp.impl.ERPAdapterImpl implements
        vsp.sca.ERP.ERPAdapter
    class eu.indenica.vsp.impl.WMSProxyImpl implements vsp.sca.WMS.WMSProxy
    class eu.indenica.vsp.impl.YMSProxyImpl implements vsp.sca.YMS.YMSProxy
    ...
}
}

```

The main responsibility of the Generator DSL is to specify the implementations of SCA components, the bindings of every SCA ports, and the representative interfaces of the bindings.

We note that the Generator DSL, which is developed based on the Xtext framework, provides very powerful editing assist, for instance, code completion, live error checking, cross-referencing, etc.

3.9 Generating code for proxies and adapters

In order to integrate various underlying service-based platforms, proxies and adapters are often required for different tasks such as performing delegating invocations, translating input and output data, abstracting the underlying platforms and acting on behalf of the users, etc.

The Tool Suite provides means for developers to define and generate proxies and adapters using a part of the Generator DSL. For instance, an example of defining the Generator DSL for the INDENICA case study, in which we need adapters and proxies for interacting with remote service-based platforms such as the Yard Management System and Warehouse Management System, is shown as following.

```

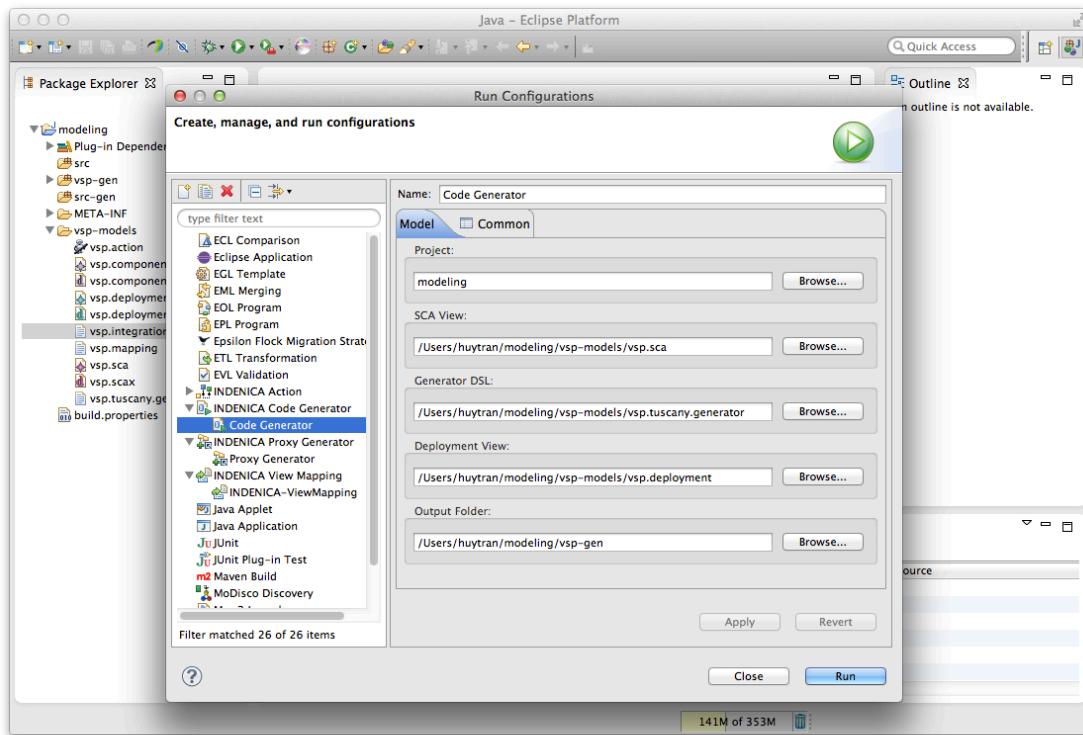
module vsp.^integration
{
    integration {
        JMS adapter
    }
    eu.indenica.vsp.wmsnotificationgateway.WMSNotificationGateway {
        broker-url : "tcp://127.0.0.1:61616"
        queue : "vspcalls"
    }
}

```

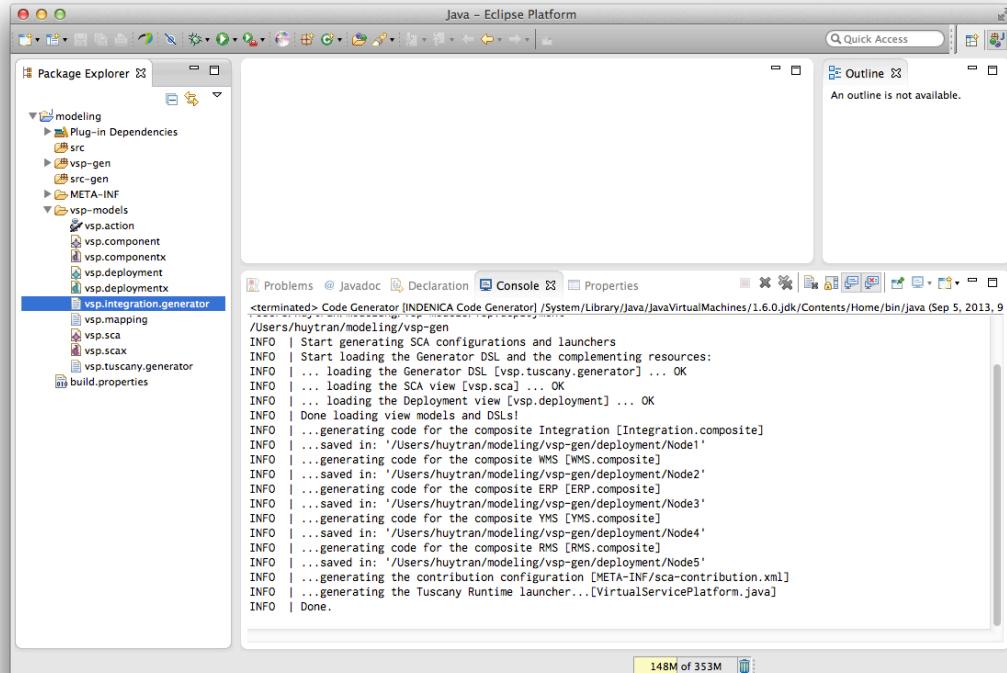
```
operations {
    checkReady (data : readyEvent)
    notifyFinished (data : finishedEvent)
}
kind : TextMessage
}
JMS adapter
eu.indenica.vsp.ymsnotificationgateway.YMSNotificationGateway {
    broker-url : "https://indenicaym.netweaver.ondemand.com/yms/jms"
    queue : "eu.indenica.vsp.NotificationGateway"
    operations {
        notifiedTruckArrived (data : TruckCheckIn)
    }
    kind : TextMessage
    username : "guest"
    password : "guest"
}
}
data {
    namespace "http://indenica.eu/yms" =>
eu.indenica.vsp.ymsnotificationgateway {
    complex TruckCheckIn {
        bookingId : string
        driverId : string
    }
}
namespace "http://indenica.eu/wms" =>
eu.indenica.vsp.wmsnotificationgateway {
    complex readyEvent {
        eventType : string /* ready */
        numberOfBoxes : integer
        toBeCleaned : boolean
        ready : boolean
    }
    complex finishedEvent {
        eventType : string /* finished */
        toBeCleaned : boolean
    }
}
}
```

In order to generate code for the virtual service platforms as well as the accompanying proxies and adapters, the end-users can use the launchers provided by the Tool Suite.

Creating a code generator launcher is similar to that of the view mapping.

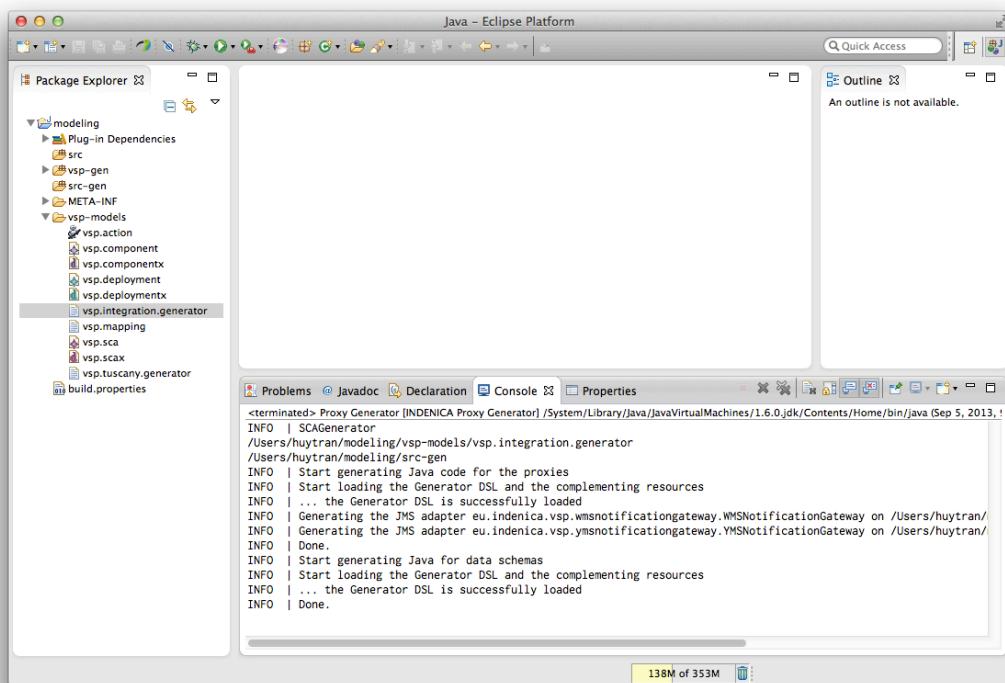
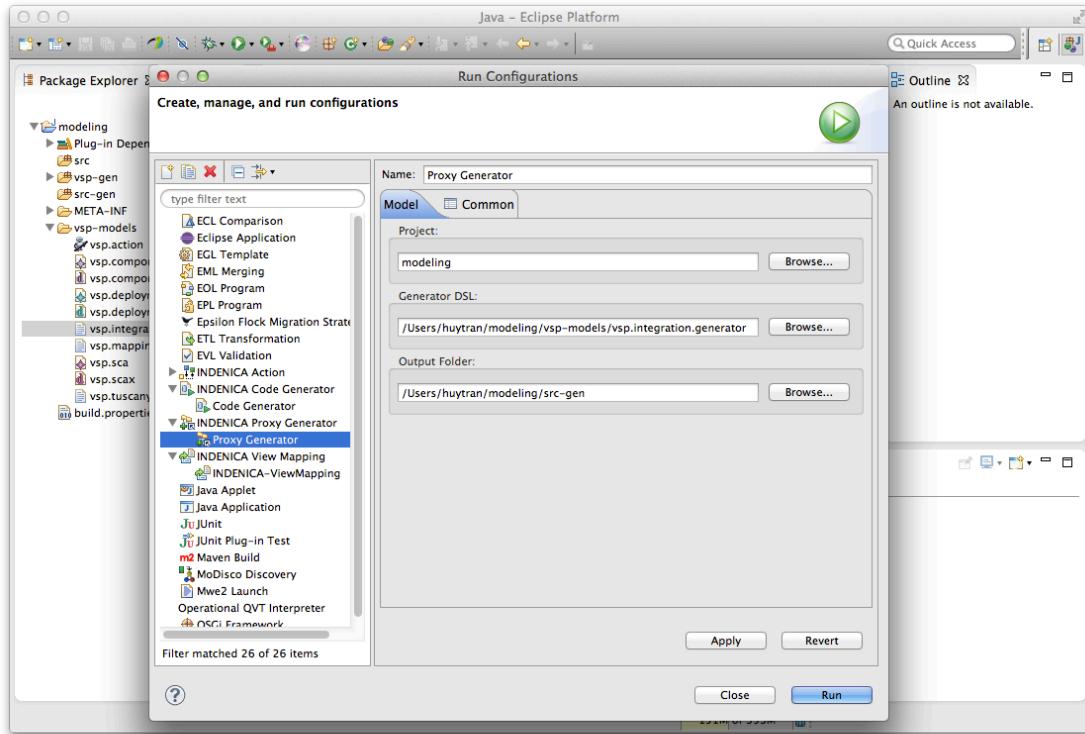


Here we choose the input SCA view (created in Section 3.4), a Deployment View (created in Section 3.6), a Generator DSL as created in Section 3.8.



The code generator will execute and produce Apache Tuscany SCA 1.0 configuration and code.

In the same manner, we create a launcher for proxy generator and use it to generate Java code for the proxies and adapters.



Nevertheless, the Code Generator APIs partially developed in the interim version and completed in this version can also be used programmatically to produce the same results.

```
ScaGenerator.generate(SCA_VIEW, GEN_DSL, DEPL_VIEW, TARGET_FOLDER);
```

3.10 Running SCA Configurations and Code with Apache Tuscany

For testing and development purpose, we can temporarily run the Virtual Service Platform by the generated code.

```
package launcher;

import org.apache.tuscany.sca.node.SCANode;
import org.apache.tuscany.sca.node.SCANodeFactory;

public class VirtualServicePlatform {
    public static void main(String[] args) throws Exception {
        final SCANode node =
SCANodeFactory.newInstance().createSCANodeFromClassLoader(null,
Thread.currentThread().getContextClassLoader());
        node.start();
        Thread.currentThread().join();
    }
}
```

Nevertheless, the generated configurations and code, including proxies and adapters, will form the skeleton for a virtual service platform, i.e., the interfaces and the interacting among architectural components. The actual implementation of the constituting components, which must be described manually by the developers according to particular requirements of the applications built on top, and can be incorporated with the generated parts in order to form the whole functioning virtual service platforms.

4 Developer's Guide

The whole source code of the Tool Suite for Virtual Service Platform Engineering is provided in the following INDENICA subversion repository (authentication needed):

<https://repository.sse.uni-hildesheim.de/svn/Indenica/workpackages/wp3/implementation/viewbased>

Further details and guidance for the Tool Suite for Virtual Service Platform Engineering can be found (along with a demonstration movie) at:

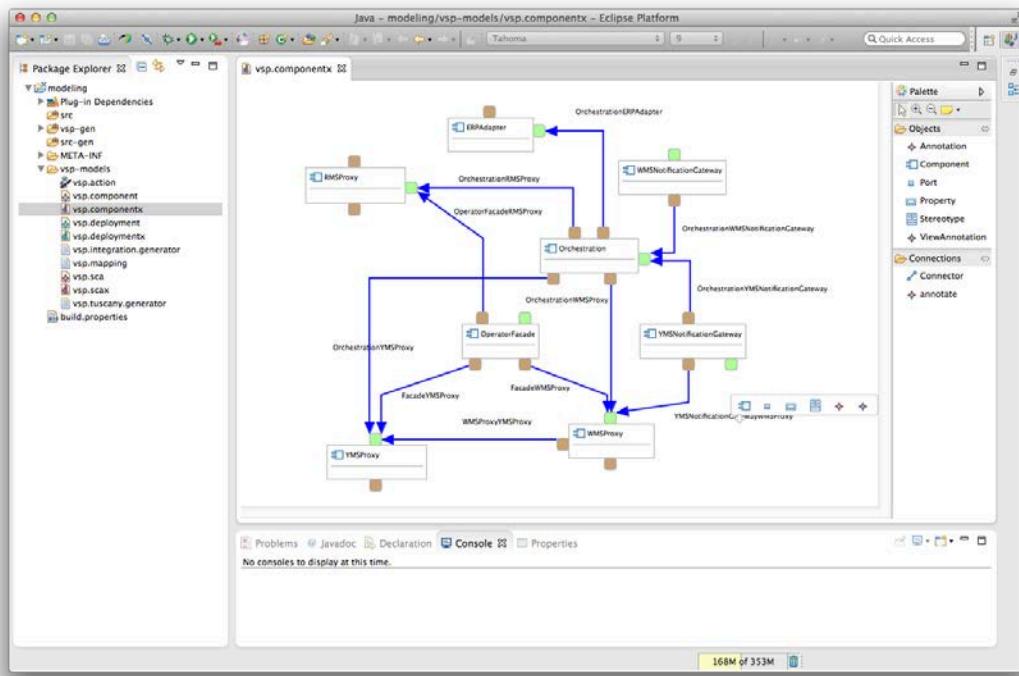
<https://swa.univie.ac.at/View-based Modeling Framework>

We note the significant changes since the interim version is that the code generation rules have been totally migrated from the legacy Xpand language to Xtend language (which is more powerful, extensible, substantially faster, and integrated well with Java, EMF, and Xtext).

5 Application of the Tool Suite in the INDENICA use case

The Tool Suite is fully functional and has been used to develop the virtual service platform for integrating a Yard Management System, a Warehouse Management System, and a Remote Management System together and providing façade interfaces for a Warehouse Operator Application. More details of the use case and the implementation can be found in the deliverables [D5.3.1] and [D5.3.2]. In this section, we show the corresponding view models and DSLs that are developed in the context of the use case.

5.1 High-Level Service Component View



5.2 Mapping DSL

```
module vsp.mapping

Target View {
    name = vsp.sca
    nsURI = "http://indenica.eu/vsp/sca"
    // Integration composite
    Group G1 {
        source:
            vsp.component.Orchestration,
            vsp.component.OperatorFacade
        target:
            composite name = Integration
            nsURI = "http://indenica.eu/vsp/integration"
    }
    // WMS composite
    Group G2 {
        source:
            vsp.component.WMSProxy,
            vsp.component.WMSNotificationGateway
        target:
    }
}
```

```

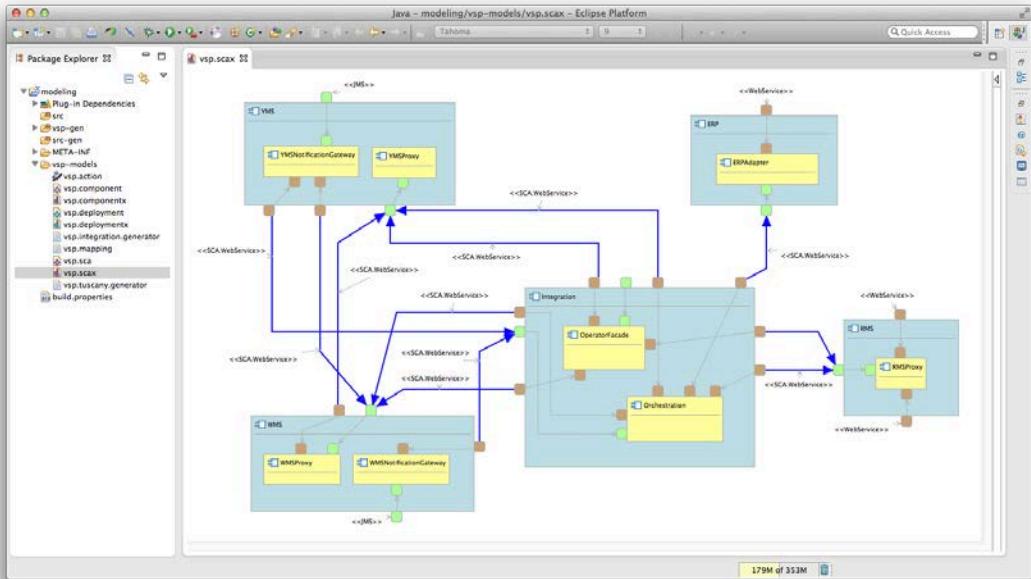
        composite name = WMS
        nsURI = "http://indenica.eu/vsp/wms"
    }
// ERP composite
Group G3 {
    source:
        vsp.component.ERPAdapter
    target:
        composite name = ERP
        nsURI = "http://indenica.eu/vsp/erp"
}
// YMS composite
Group G4 {
    source:
        vsp.component.YMSProxy,
        vsp.component.YMSNotificationGateway
    target:
        composite name = YMS
        nsURI = "http://indenica.eu/vsp/yms"
}

// RMS composite
Group G5 {
    source:
        vsp.component.RMSProxy
    target:
        composite name = RMS
        nsURI = "http://indenica.eu/vsp/rms"
}

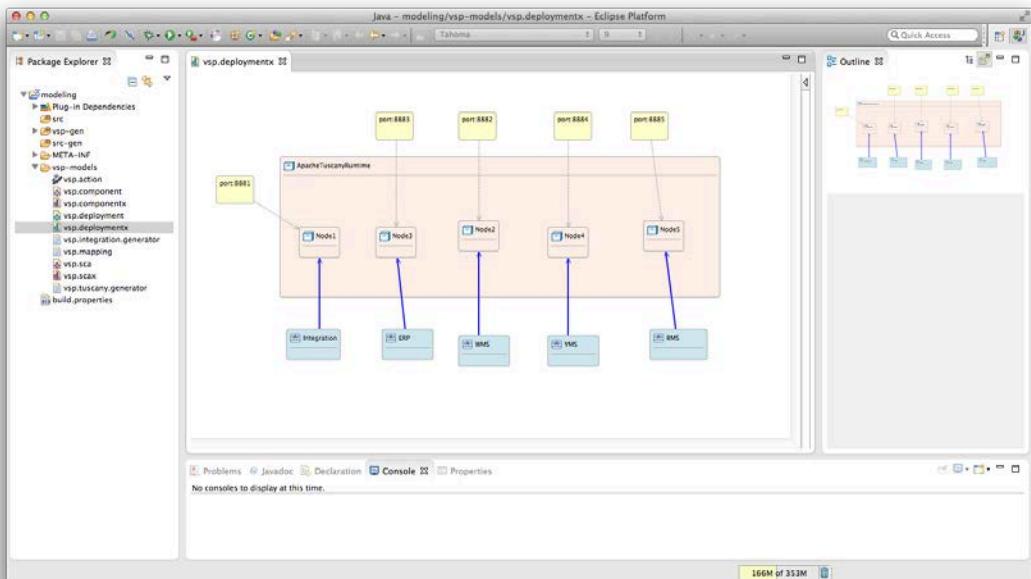
// add some stereotypes
add stereotype <<"WebService">> to vsp.component.RMSProxy.callService
add stereotype <<"WebService">> to
vsp.component.RMSProxy.supervisionService
add stereotype <<"WebService">> to vsp.component.ERPAdapter.erp
add stereotype <<"JMS">> to
vsp.component.WMSNotificationGateway.WMSNotificationGateway
add stereotype <<"JMS">> to
vsp.component.YMSNotificationGateway.YMSNotificationGateway
add stereotype <<"SCA.WebService">> to
vsp.component.OrchestrationERPAdapter
add stereotype <<"SCA.WebService">> to vsp.component.OrchestrationYMSProxy
add stereotype <<"SCA.WebService">> to vsp.component.OrchestrationYMSProxy
add stereotype <<"SCA.WebService">> to vsp.component.FacadeYMSProxy
add stereotype <<"SCA.WebService">> to vsp.component.FacadeWMSProxy
add stereotype <<"SCA.WebService">> to vsp.component.WMSProxyYMSProxy
add stereotype <<"SCA.WebService">> to
vsp.component.OrchestrationYMSNotificationGateway
add stereotype <<"SCA.WebService">> to
vsp.component.YMSNotificationGatewayWMSProxy
add stereotype <<"SCA.WebService">> to vsp.component.OrchestrationRMSProxy
add stereotype <<"SCA.WebService">> to
vsp.component.OrchestrationWMSNotificationGateway
}

```

5.3 SCA View (aka Low-level Service Component View for SCA)



5.4 Deployment View



5.5 Generator DSL for SCA

```
module vsp.generator
{
    bindings {
        Web Service { // binding of the Orchestration
            uri : "http://127.0.0.1:8881/Orchestration"
        }
        binds {
            vsp.sca.Integration.Orchestration.Orchestration,
            vsp.sca.YMS.YMSNotificationGateway.orchestration,
            vsp.sca.WMS.WMSNotificationGateway.orchestration
        }
    }
}
```

```

Web Service { // binding of the OperatorFacade
    uri : "http://127.0.0.1:8881/OperatorFacade"
}
binds {
    vsp.sca.Integration.OperatorFacade.OperatorFacade
}

/* Internal WS binding of the WMSProxy */
Web Service {
    uri : "http://127.0.0.1:8882/WMSProxy"
}
binds {
    vsp.sca.Integration.OperatorFacade.wmsProxy,
    vsp.sca.Integration.Orchestration.wmsProxy,
    vsp.sca.WMS.WMSProxy.WMSProxy,
    vsp.sca.YMS.YMSNotificationGateway.wmsProxy
}
/* Internal WS binding of the ERPAdapter */
Web Service {
    uri : "http://127.0.0.1:8883/ERPAdapter"
}
binds {
    vsp.sca.Integration.Orchestration.erpAdapter,
    vsp.sca.ERP.ERPAdapter.ERPAdapter
}
/* Internal WS binding of the YMSProxy */
Web Service {
    uri : "http://127.0.0.1:8884/YMSProxy"
}
binds {
    vsp.sca.Integration.OperatorFacade.ymsProxy,
    vsp.sca.Integration.Orchestration.ymsProxy,
    vsp.sca.WMS.WMSProxy.ymsProxy,
    vsp.sca.YMS.YMSProxy.YMSProxy
}

/* Internal WS binding of the RMSProxy */
Web Service {
    uri : "http://127.0.0.1:8885/RMSProxy"
}
binds {
    vsp.sca.RMS.RMSProxy.RMSProxy,
    vsp.sca.Integration.Orchestration.rmsProxy,
    vsp.sca.Integration.OperatorFacade.rmsProxy
}

/* Remotely binding to the ERP service */
Web Service {
    uri : "http://127.0.0.1:9999/ERP"
}
binds {
    vsp.sca.ERP.ERPAdapter.erp
}

/* Remotely binding to the SupervisionServiceAPI*/
Web Service {
    uri : "http://127.0.0.1:9999/SupervisionServiceAPI"
}
binds {
    vsp.sca.RMS.RMSProxy.supervisionService
}
Web Service {
    uri : "http://127.0.0.1:9999/CallService_API"
}
binds {
    vsp.sca.RMS.RMSProxy.callService
}

```

```

    /* Remotely JMS binding to the YMSNotification: for receiving JMS
messages */
JMS
{
    uri : "jms:eu.indenica.vsp.NotificationGateway"
    initialContextFactory :
"org.apache.activemq.jndi.ActiveMQInitialContextFactory"
    jndiURL : "https://indenicaym.netweaver.ondemand.com/yms/jms"
} binds {
    vsp.sca.YMS.YMSNotificationGateway.YMSNotificationGateway
}

    /* Remotely JMS binding to the WMSNotification: for receiving JMS
messages */
JMS
{
    uri : "jms:vspcalls"
    initialContextFactory :
"org.apache.activemq.jndi.ActiveMQInitialContextFactory"
    jndiURL : "tcp://127.0.0.1:61616"
} binds {
    vsp.sca.WMS.WMSNotificationGateway.WMSNotificationGateway
}
}

interfaces {
    interface eu.indenica.vsp.orchestration.ws.Orchestration describes {
        vsp.sca.Integration.Orchestration.Orchestration,
        vsp.sca.WMS.WMSNotificationGateway.orchestration,
        vsp.sca.YMS.YMSNotificationGateway.orchestration
    }
    interface eu.indenica.vsp.operatorfacade.ws.OperatorFacade describes {
        vsp.sca.Integration.OperatorFacade.OperatorFacade
    }
    interface eu.indenica.vsp.wmsproxy.WMSProxy describes {
        vsp.sca.WMS.WMSProxy,
        vsp.sca.Integration.OperatorFacade.wmsProxy,
        vsp.sca.Integration.Orchestration.wmsProxy
    }
    interface eu.indenica.vsp.ymsproxy.YMSProxy describes {
        vsp.sca.YMS.YMSProxy,
        vsp.sca.Integration.OperatorFacade.ymsProxy,
        vsp.sca.Integration.Orchestration.ymsProxy,
        vsp.sca.WMS.WMSProxy.ymsProxy
    }

    interface eu.indenica.vsp.erpapter.ERPAdapter describes {
        vsp.sca.ERP.ERPAdapter.ERPAdapter,
        vsp.sca.Integration.Orchestration.erpAdapter
    }

    interface eu.indenica.platform.erp.ws.ERP describes {
        vsp.sca.ERP.ERPAdapter.erp
    }

    interface eu.indenica.vsp.rmsproxy.RMSProxy describes {
        vsp.sca.RMS.RMSProxy,
        vsp.sca.Integration.OperatorFacade.rmsProxy,
        vsp.sca.Integration.Orchestration.rmsProxy
    }

    interface com.nextdaylab.indenica.rms.service.call.CallServiceAPI
describes {
        vsp.sca.RMS.RMSProxy.callService
    }
}

```

```

interface
com.nextdaylab.indenica.rms.service.supervision.SupervisionServiceAPI
describes {
    vsp.sca.RMS.RMSProxy.supervisionService
}

interface eu.indenica.vsp.ymsnotificationgateway.YMSNotificationGateway
describes {
    vsp.sca.YMS.YMSNotificationGateway.YMSNotificationGateway
}
interface eu.indenica.vsp.wmsnotificationgateway.WMSNotificationGateway
describes {
    vsp.sca.WMS.WMSNotificationGateway.WMSNotificationGateway
}
implementations {
    class eu.indenica.vsp.impl.OrchestrationImpl implements
vsp.sca.Integration.Orchestration
    class eu.indenica.vsp.impl.OperatorFacadeImpl implements
vsp.sca.Integration.OperatorFacade
    class eu.indenica.vsp.impl.ERPAdapterImpl implements
vsp.sca.ERP.ERPAdapter
    class eu.indenica.vsp.impl.WMSProxyImpl implements vsp.sca.WMS.WMSProxy
    class eu.indenica.vsp.impl.YMSProxyImpl implements vsp.sca.YMS.YMSProxy
    class eu.indenica.vsp.impl.YMSNotificationGatewayImpl implements
vsp.sca.YMS.YMSNotificationGateway
    class eu.indenica.vsp.impl.WMSNotificationGatewayImpl implements
vsp.sca.WMS.WMSNotificationGateway
    class eu.indenica.vsp.impl.RMSProxyImpl implements vsp.sca.RMS.RMSProxy
}
}

```

5.6 Generator DSL for proxy and adapter generation

```

module vsp.^integration
{
    integration {
        JMS adapter
eu.indenica.vsp.wmsnotificationgateway.WMSNotificationGateway {
        broker-url : "tcp://127.0.0.1:61616"
        queue : "vspcalls"
        operations {
            checkReady (data : readyEvent)
            notifyFinished (data : finishedEvent)
        }
        kind : TextMessage
    }
    JMS adapter
eu.indenica.vsp.ymsnotificationgateway.YMSNotificationGateway {
        broker-url : "https://indenicaym.netweaver.ondemand.com/yms/jms"
        queue : "eu.indenica.vsp.NotificationGateway"
        operations {
            notifiedTruckArrived (data : TruckCheckIn)
        }
        kind : TextMessage
        username : "guest"
        password : "guest"
    }
}

data {
    namespace "http://indenica.eu/yms" =>
eu.indenica.vsp.ymsnotificationgateway {
    complex TruckCheckIn {
        bookingId : string
        driverId : string
    }
}

```

```
        }
    namespace "http://indenica.eu/wms" =>
eu.indenica.vsp.wmsnotificationgateway {
    complex readyEvent {
        eventType : string /* ready */
        numberofBoxes : integer
        toBeCleaned : boolean
        ready : boolean
    }
    complex finishedEvent {
        eventType : string /* finished */
        toBeCleaned : boolean
    }
}
namespace "http://indenica.eu/rms" => eu.indenica.vsp.rms {
    complex VoiceAndVideoCallInitiationRequest {
        adminId : string
        userSipId : string
        receiverSipIds : string *
        callParameters : string
    }
    complex SendTextMessageToUserRequest {
        adminId : string
        textMessage : string
        userSipId : string
    }
    complex ConnectToCameraWithVideoOnlyRequest {
        adminId : string
        userSipId : string
        cameraSipId : string
        callParameters : string
    }
}
}
```

6 Conclusions

So far we presented the final version of the INDENICA Tool Suite for Virtual Service Platform Engineering, which is released at M36. During this time period, we significantly added enhanced features for the Tool Suite such as the dynamic and live model verification and validation, faster code generation, better UI support via launchers and stable development APIs. We added the mapping language and engine for assisting the developers in translating high-level to low-level view models. We also enhanced the integration with the architectural design decisions support developed in the scope of WP1 (via the Architectural Knowledge Transformation Language reported in D1.3.2) [1] as well as better support for deployment and runtime infrastructure developed in WP4 (see the Deployment Manager and Monitoring reported in D4.2.1 and D4.2.2) [2,3,4]. The Tool Suite has been used to fully model and generate code and configurations for the virtual service platform that can integrate three service-based platforms provided by our industrial partners.

Table of Figures

Figure 1 Overview of the view-based design time and runtime architecture.....	4
Figure 2 Update Site for INDENICA Modelling Tool Suite.....	6

References

- [1] Lytra, H. Tran, and U. Zdun, "Supporting Consistency between Architectural Design Decisions and Component Models through Reusable Architectural Knowledge Transformations," in *7th European Conference on Software Architecture (ECSA)*, pp. 224-239, 2013.
- [2] C. Inzinger, W. Hummer, I. Lytra, P. Leitner, H. Tran, U. Zdun, and S. Dustdar, "Decisions, Models , and Monitoring – A Lifecycle Model for the Evolution of Service-Based Systems," in *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, PP. 185-194, 2013.
- [3] H. Tran and U. Zdun, "Event Actors Based Approach for Supporting Analysis and Verification of Event-Driven Architectures," in *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, Vancouver, Canada, pp. 217-226, 2013.
- [4] S. Tragatschnig, H. Tran, and U. Zdun, "Change Patterns for Supporting the Evolution of Event-based Systems," in *21st International Conference on Cooperative Information Systems (CoopIS)*, pp. 283-290, Graz, Austria, 2013.