Agenda
Structure of the talk

1. Who we are?
2. What’s the problem: build toolchains for 50 product lines
3. What’s the solution:
   1. 1IDE
   2. Software Product Line
   3. Eclipse - Wazaabi
4. Feedback from the field
Who am I?
Short introduction

Head of the EU R&D Architecture
@Huawei European Research Center
Driving projects for the transition from Telco to Service provider with new technologies

Research director
@ EURA NOVA
Make the link between Research & Customer challenges

Committer on open source Projects
Distributed MoM, Distributed Data Processing, Model-Based UI
RoQ-Messaging, AROM
I come From Belgium
Very Well known for …

Manenken Pis
Ok that’s simple but we get clothes for him!

Top 1 of the 11 Europe’s most bizarre building
http://www.cnn.com/2013/01/18/travel/europe-bizarre-buildings

French Belgian Fries
I come From Belgium

ERC – IDE & Soft Engineering Team

Carrier-Software Division
EU R&D Architecture (Platform)

What is that?
What kind of Software do you do?
Carrier Software & Core Network at Huawei

Telecom Operator Solutions

Enterprise Solutions

Devices Hardware & software

Platform R&D

What kind of Software do we develop?

Beyond Your Expectations
Software domains for a Telecom operator
The three main areas

INTERNET
World

NETWORK
World

IT
World

CRM
SCM
BSS
User Profile
Software domains for a Telecom operator
The three main area

- INTERNET World
- VAS
  - Over the top Competitors
- NETWORK World
- VAS

Enterprise Competitors

IT World

CRM
SCM
BSS
User Profile

and Your Expectations
Collaboration with Telco for better global operations
360° coverage enabling global Intelligent business operations

As a result the entire product lines cover a wide area of products
IDE problem Statement

Modeling & IDEs challenges to support more than 50 Product lines
Cover these 3 domains has a price
A wide range of product lines based on platforms

Each of them needs a Toolchain
Configuration, Edition, Dev, Deployment, etc.

Direct Consequence of the Telco Software evolution
Consequences
High costs of replicated features

Which costs?
- The cost of each Toolchain
- The cost of maintenance & evolution
- The cost of feature replication
- The cost of IDE teams – learning curve, size of the teams

What can we do?
What are the common features shared among Toolchains?
Is there any way to share a common framework?
How to define the different Toolchain particularities?
How to make it smart to avoid too much code maintenance?
Basic idea
Building an IDE component Framework
Step 1: identifying feature patterns
What is shared between the toolchains?
Step 1: identifying feature patterns
Summarized in DSL patterns and features

<table>
<thead>
<tr>
<th>Graphical layout</th>
<th>Data pattern</th>
<th>Data type</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organized text</td>
<td>Textual DSL</td>
<td>String</td>
<td>/</td>
</tr>
<tr>
<td>(language editor)</td>
<td></td>
<td>+ DSL Meta-model</td>
<td>/</td>
</tr>
<tr>
<td>UML class diagram</td>
<td>Classes (generic)</td>
<td>Class [*]</td>
<td>/</td>
</tr>
<tr>
<td>UML state diagram</td>
<td>State machine</td>
<td>Node [*]</td>
<td>Type + label</td>
</tr>
<tr>
<td>Tree browser</td>
<td>Tree</td>
<td>Node [*]</td>
<td>/</td>
</tr>
<tr>
<td>Decision tree</td>
<td>Tree</td>
<td>Node [*]</td>
<td>Rules</td>
</tr>
<tr>
<td>Table</td>
<td>Array</td>
<td>Class</td>
<td>/</td>
</tr>
<tr>
<td>Merging table</td>
<td>Tree</td>
<td>Node [*]</td>
<td>/</td>
</tr>
<tr>
<td>Tree-table</td>
<td>Tree</td>
<td>Node [*]</td>
<td>/</td>
</tr>
<tr>
<td>Form / properties</td>
<td>Single node</td>
<td>Class</td>
<td>/</td>
</tr>
<tr>
<td>Wizard</td>
<td>Graph</td>
<td>Node [*]</td>
<td>/</td>
</tr>
<tr>
<td>Web site</td>
<td>Graph</td>
<td>Node [*] + UI</td>
<td>Label (buttons)</td>
</tr>
<tr>
<td>BPEL (Agent modeling)</td>
<td>Complex</td>
<td>ABM</td>
<td>/</td>
</tr>
</tbody>
</table>

Possible graphical layout

- Text editor, Language editor
- Form
- Table (list of Objects), List (list of primitives)
- Radio buttons, Combo box
- Wizard (succeeding forms)
- Tree, table, tree-table
- Table, tree-table
- Form + dialogs, tree, tree table
- Table + dialogs, tree, tree table
- Tree, decision tree
- Graph, wizard, state diagram
- Class diagram
- State diagram
Step 2: Software engineering techniques
Not re-inventing the wheel while adopting flexible and dedicated approach

Software product Line
Adapted from factory and assembly product line to Software
Building highly customized products from a base platform
Adding a component layer on top of Eclipse

Applying the Software Product Line concepts
Variability Management
Individual systems are considered as variations of a common theme. This variability is made explicit and must be systematically managed. Designing variation within the model.

Architecture-centric
The technical side of the software must be developed in a way that allows taking advantage of similarities among the individual systems. Try to come back on the extensions of the reference architecture.

Two-life-cycle approach
The individual systems are developed based on a software platform. These products—as well the platform—must be engineered within their own life cycles.

Basic Example: The project explorer

Defining the variability

**What should be configured?**

Defining the structure for a project type including:
- Virtual nodes,
- Sorters, filters,
- Double click handlers,
- Many other things ....

![Project Explorer Diagram](image-url)
Our Approach of the SPL
Constraints & objectives

**No CODE GENERATION !!**
R&D management constraint. We can use models in domain platform but NO code generation for application IDE

Component could be aggregated
We should be able to build high level component from other components

Must propose at least an XML-like configuration
Components should offer DSL for configuration and later editors
Our Approach of the SPL
Consequences & design challenges

No CODE GENERATION !!

Component could be aggregated

Must propose at least an XML-like configuration or DSL

No plug-in generation
No UI generation
No extension point generation
Reference between components
Dependency injection for component (HL)
Manipulating lighter XMI configuration

Challenges
How to create property sheets?
How to create model-based wizards and dialogs?
Just imagine to configure a project explorer and a simple BPMN editor with a property sheet for each activity without code generation & extension points!

How to inject a model-based UI in a dialog?
How to manage their dependency?
How can we manage the instance pool for injection?
How to inject the same validation rules in the BPMN editor and Form Editor?
Our Approach of the SPL
Consequences & design challenges

No CODE GENERATION!!

Component could be aggregated

Must propose at least an XML-like configuration

Model-based Component

Dependency Injection Fwk

Configuration DSL

Dynamic Extension point Fwk

Mustaard & Wazaabi

M2M With Wazaabi

I will focus the rest of the presentation of those components
Dependency injection framework
Basic concepts - Injecting high level components

```java
public class MyProperty extends PropertySheet implements Component<PropertySheet> {

    //Injected form for this prop sheet
    @CreateComponent("location:/myformId")
    private Form form;

    public MyProperty () {
        //Do my prop sheet init stuff
    }
}
```

```xml
<?xml version="1.0" encoding="ASCII"?>
<Dialog xmlns:dialog.config=http://dialog/config.ecore id="dialogtest.test1">
    <form id="dialogtest.formTest1"></form>
</Dialog>
```
Examples of Configurable components

Configurable components
- Project Explorer
- Forms
- Content Assistant
- Data Modeler
- Diagram designer
- Development Process
- Preference

Illustrations

Domain platform
- Model Management
- Dependency injection Framework
- Tooling for configuration
Examples of IDE Created
Illustrations

SOA IDE BME
Examples of IDE Created

Illustrations

SOA eBUS Workflow
Examples of IDE Created

Illustrations

SOA BPMN Editor
Using Wazaabi model-based UI as fundamental component of the SPL approach

Focus on Wazaabi-based components
What is Wazaabi?
Where does it come from …?

Olivier Moises, the Wazaabi’s father @Shenzhen office (China)

Wazaabi is Born …
What is Wazaabi?
Model-based UI – An Eclipse Project

Is it declarative or model-based?
Actually it is both of them! The key idea is to use the declarative support for building the model. As a result you can define your UI by declaration and you will get a model of your UI.

Hum, so it's not live, it's MDA?
No, it’s live. That's a major difference comparing to the traditional MDA approach which consists in generating code, in this case UI code, from a Model. In wazaabi, the UI is a live model, the same model that you have defined at the design phase. An engine, is then responsible for rendering your model as the final UI.

You can bind UI & business model!

```java
//create a pushButton
PushButton pushButton = WidgetsFactory.eINSTANCE.createPushButton();
pushButton.setText("Hello World");
// append the button to composite's children
list.composite.getChildren().add(pushButton);
```
Binding 2 Models in Wazaabi
The Wazaabi burger Factory example

Step 1: Create the model
Define the business elements

http://blog.euranova.eu/?p=317
Step 2: Binding the order model to the bill model
As soon as an order change, we compute the bill. It is a way to synch up 2 models based on events.

On the “Change event” of the drinkEntries we bind an event handler that will sync the drink price on the bill.
Step 2: Binding the order model to the bill model

Binding between different models.

On the “Create event” of the TimeStamp attribute we bind an event handler that will sync the Kitchen planning.
Let’s see a bit of code for the binding!

```java
//Create source and target of the binding
ListParameter source = Factory.createSource();
IntegerParameter target = Factory.createTarget();

//Init source target values
source.setValue("@elist");
target.setValue("bill[1]/burgerTotalPrice/@integer");

//Configure the binding
binding.getParameters().add(source);
binding.getParameters().add(target);

//Instantiate the handlers
EDPHandler action = Factory.createEDPHandler();
action.setUri("urn:java:org.wazaabi.Burger2Bill.java");
binding.getExecutables().add(action);
order.getHandlers().add(binding);

//Define the event on which we attach the handler
Event event = Factory.createEvent();
event.setId("core:model:change");
binding.getEvents().add(event);
```

We can chain the handlers by adding more of them.
Binding UI and Models in Wazaabi
Basic mechanics

Binding UI is the same!
As Wazaabi provides engine for Model UI, binding a business model and a UI is reduced to bind 2 (or more) models in the same Way!

```java
//Create source and target of the binding
StringParameter source = Factory.createSource();
StringParameter target = Factory.createTarget();
//Init source target values
source.setValue("@text");
target.setValue("../TextComponent[1]/@text");
//Configure the binding
binding.getParameters().add(source);
binding.getParameters().add(target);
//Instantiate the Converters
Converter converter = Factory.createConverter();
action.setUri("urn:java:platform.wazaabi.Conveter");
```
Binding UI and Models in Wazaabi

Basic mechanics

Out of the box handler flavors
Wazaabi provides out of the box various Converters & validators
Why is it useful for our SPL vision?
Key factors for choosing Wazaabi

1. Declarative configuration fit (M2M)
As our components use models, that’s much more easier to use M2M for transforming component model in UI.

2. No code generation
The model interpretation enables us to never generate code instead we manipulate object models, binding models and UI models.

3. No prior knowledge of SWT or JFace
We manipulate Wazaabi concepts without coding Uis.

Usages:
1. Dialogs & wizards from application models
2. Property sheet from applications models for flow chart designer elements
3. Property sheets for project explorers
4. Preference page from preference configuration
The Tools we create on Top of Wazaabi
1. Mustaard – Textual DSL for Wazaabi

Wazaabi Xtext DSL – Not a 1-1 mapping
1. Json-like syntax
2. We implemented an intermediate layer for guessing what are the default configuration.

```plaintext
ProgressBar {
  value 11
  IntRule { propertyName "maximum" value 100}
  IntRule { propertyName "minimum" value 100}
  orientation {value HORIZONTAL}
}
```

We support 99% of SWT & JFace
The Tools we create on Top of Wazaabi

1. Mustaard – Language shortcuts

```
importTheme "<uri to the Theme file>.th"
```
The Tools we create on Top of Wazaabi

1. Mustaard – Language shortcuts – Extending Xtext

By reflection the P. dispatcher (Mystique) calls the corresponding Wazaabi constructor.

If a method has been defined for overriding the default behavior, the PD will call the interpret method.
The Tools we create on Top of Wazaabi

1. Mustaard – Textual DSL for Wazaabi

Concept of Themes

- All Containers will look like this
- All TextComponents with the class TextComponentClass 1 will look like this
- All other TextComponents will look like this

Features

- Declare Styles and Binding only once
- Create variable to parameterize the theme
- Create classes to apply style and binding only on selected objects
- Syntax Jason-Like
- Quick overview of the rendering
The Tools we create on Top of Wazaabi

2. Model 2 Wazaabi Cartridge

Cartridge oriented
From an Ecore we can generate a Wazaabi UI model. We can choose different transformation strategies based on the cartridge we select. The cartridges are extensible.
The Tools we create on Top of Wazaabi

2. Model 2 Wazaabi Cartridge
Feedback from the field
Lessons learned
Lesson 1: Tooling

Tooling is mandatory

You can have the greatest declarative, modular, architecture

This is useless without an easy-to-use graphical tooling

Vs
Lesson 2: User feedback
Getting ASAP feedback from DSL users

User feedback is very important when building a DSL

Especially when having 2 levels of indirections
Needs are different
Skills & background are different
Lesson 3: Software does not stop to version 1.0
Be prepared to changes and to propagate them

Changes must be *correctly* propagated
2-3 breaks between versions can kill the trust with customers

**Propagated a change?**
Propagation through the different levels on top of a component

Design a bullet-proof Life cycle management for components
Storage, update, tracing modifications, quality process, migration management, etc.
Lesson 4: Keep it simple is complex
Easy to use, simple & maintainable are very hard in real world!

Configuration language
What kind of DSL?
What configuration? What configuration processes?
Thinking about developer viewpoint and habits

Eclipse veterans do NOT work like newbie
Different needs of tooling, different habits, different expectations, etc.

Permanent war for simplification
Number of words to type in DSL, number of config files, number of step for building features, etc.
1IDE Phase 1 started in Q1 2011 & finished in Q1 2012
Team of 8 Architects in Belgium & 10 developers in China
Cost to build an IDE decreased of 80% (for the IDE we targeted in phase 1)

Try to avoid re-inventing the wheel
Phased Approach – step by step
Looking existing solution in Eclipse Ecosystem & Software engineering
Try to keep it simple as much as possible
Thank you

www.huawei.com