Conception and Development Tools for SCOrWare - Version 1.0

Stéphane Drapeau, Gaël Blondelle, Damien Fournier, Philippe Merle, Marc Pantel, Djamel Belaïd, Samir Tata, Etienne Juliot, Marc Dutoo

SCOrWare Project

Conception and Development Tools for SCOrWare - *Version 1.0*

Funded by

AGENCE NATIONALE DE LA RECHERCHE
ANR

http://www.agence-nationale-recherche.fr

Colophon

Date	September 28 2007
Deliverable type	Specification
Version	1.0
Status	Revision
Work Package	2
Access Permissions	Consortium Confidential

Editor

Obeo	Stéphane Drapeau

Authors

EBM WebSourcing	Gaël Blondelle
INRIA	Damien Fournier
	Philippe Merle
IRIT	Marc Pantel
INT	Djamel Belaid
	Samir Tata
Obeo	Stéphane Drapeau
	Etienne Juliot
Open Wide	Marc Dutoo

Table of Contents

List of Figures

1 Introduction

The SCOrWare project objectives are to develop:

- A runtime platform for deploying, executing, and managing SCA based applications.
- A set of development tools for modeling, designing, and implementing SCA based applications.
- A set of demonstrators.

This document specifies the set of tools used to design, develop, test, and deploy elements to build a distributed architecture compound of components, services, and business services. These tools are based on standards like MDA (Model Driven Architecture) [1], DSL (Domain Specific Language) [2], SCA (Service Component Architecture), and are build on top of Eclipse. Some of these tools complete currently available tools proposed in the SOA Tools Platform (STP) project [3], and others are specific to the SCOrWare project. Section 1.1 lists existing STP tools, and Section 1.2 gives an overview of the contributions of the developed SCOrWare tools, and a summary of next chapters.

1.1 Eclipse SOA Tools Platform Project

STP sub projects

The goal of STP is to provide an integrated developer tooling platform for infrastructures based on Service Oriented Architecture (SOA). Until now, STP uses as model the SCA specification. STP is composed of 5 sub projects:

1. **Core Frameworks** (CF) define EMF models that conform to the SCA specification for service assembly as well as Java language components for SCA syntax support.
2. **SOA System** (SOAS) provides tools and frameworks for assembling, building, packaging, and deploying services to runtime containers. In addition support is provided for the definition and association of policy to services prior to deployment.
3. **Service Creation** (SC) handles the management of the relationship between the SOA model tooling provided by STP and the actual implementation tooling(s). The ultimate goal is to support the development of SOA network via tools fully integrated for top-down and bottom-up approaches as well as a mix of both in an agile way.
4. **BPEL 2 Java** (B2J) provides tools to translate BPEL into executable Java classes. It also defines a standard framework to which these executable Java classes can be deployed.
5. **BPMN** (BPMN) provides an editor and a set of tools to model business processes diagrams using the BPMN notation.

STP Incubator

To complete these 5 sub-projects, an **incubator** is going defined. Technologically related contributions from multiple organizations that do not have any connection to existing STP sub-projects are suitable candidates to be incorporated into this incubator. Currently, 4 new components are in the pipeline of the STP incubator:

- SCA Visual Composite Editor: it constructs top-down and bottom-up SCA composites. This editor is described in Section 4.1.
- Cimero 2 Enterprise Integration Patterns Editor: this is an editor for Enterprise integration patterns that has attached a generation framework that can produce standard JBI 1.0 components for deployment.
- SOA Policy Editor: this tool allows editing WS-Policy standard documents and assertions. A first contribution is a form-based editor that integrates representations of policy instances with documentation. A second contribution is an extension of the WTP WSDL editor and usuals a similar visual metaphor for

interactions.

- SOA meta model: This meta model will replace the SCA meta model used in the Core Framework sub-project. This contribution is described in the next paragraph about reconsideration of the meta model to use.

At the moment, the source code of these tools is passing the IP process.

Reconsideration of the meta model to use

At the core of STP is the SCA meta model (the core framework sub-project) which is the basis for all tools. However, the SCA meta model code has not received much attention since the initial contribution, it is out of date and does not appear to be an appropriate match to the sub-project requirements for modelling. Then, it is considered to replace it by an **hybrid model**.

This hybrid model will allow meta-data to be exchanged between various editors within the STP project. It is designed to be flexible and extensible. The hybrid model approach would displace the SCA as being the core, although each project could optionally use it if they wished to exchange data. The hybrid model is to serve as unifying baseline for SOA projects, as it attempts to model not just services, components and bindings, but also other concepts that are part of common SOA modelling and infrastructure technologies. In this respect it is better than the SCA meta model, which is mostly about component construction, composition and policy.

Few tools for SCA

The most part of the tools developed in STP focuses on web services thought the use of Apache CXF [4]. CXF is an open source services framework that helps to build and develop services using front-end programming API. CXF supports a variety of web service standards including SOAP, the Basic Profile, WSDL, WS-Addressing, WS-Policy, WS-Reliable Messaging, and WS-Security.

Furthermore, some of the tools are not complete. For example, initially, it was proposed that the BPMN sub project will provide a generator from BPMN to BPEL. Currently, this feature is not implemented.

1.2 Contributions

Because of the lack of existing SCA tools of STP, this work package proposes some new tools needed for modeling, designing, and implementing SCA based applications. Some of these tools are contributed as new tools to the STP project, others add new features to existing STP tools, and others are particular to the SCOrWare project.

Proposed tools are classified according to 6 categories:

- **Tools for the common meta model between platform and tooling**. The common meta model between platform and tooling, described in [5] (Chapter 3), needs tools to be represented in memory, to be read, to be written, to be validated, etc. This meta model is the starting point for all our developed SCA tools. Chapter 2 contains specification of these tools. These specifications are related to task 2.1.
- **Development tools**. Tools that simplify the SCA developer tasks are important in order for this technology will be adopted by the largest community. While graphical designer and other tools mainly target end-user and components assembler, it is clear that developers also need tools to be more efficient. These tools that are related to task 2.2 are described in Chapter 3. Four tools are proposed: (1) a wizard that simplifies the development of new SCA components, (2) an editor that helps the creation of SCA assembly files, (3) a tool for Java developers to add SCA annotations in Java code, and (4) a BPMN to BPEL generator.
- **Graphical designers**. These tools target end-users and component assembler. These tools that are related to task 2.3 are described in Chapter 4. The proposed graphical designer provides both bottom-up and top-down methods for constructing graphically SCA composites. Tools related to Business Process are also specified.
- **Test and deployment tools**. These tools concern test and deployment of SCA based applications. First, it is proposed to integrate the SCOrWare runtime in Eclipse to test SCA based applications. Second, a graphical designer is specified. It provides a quick and easy way to construct graphically the deployment

plan of SCA applications. This graphical designer can generate deployment configuration files that can be used by the tool proposed in [5] (Chapter 8 Section 2 - SCA System Deployment with FDF). These tools, related to task 2.4, are described in Chapter 5.

- **Tools for searching and semantic composition of services** (Chapter 6). These tools help, developers and integrators that build on the SCOrWare platform, finding and better reusing SCA services and components. This chapter contains specifications related to task 2.5.
- **Tools for choreography of process that use generic services** (Chapter 7). The objective of these tools is to obtain a workflow and orchestration solution that is able to work with generic services. These solutions must be enough flexible to respect development time and runtime requirements. This chapter contains specifications related to task 2.6.

Figure 1 gives an overview of the tools specified in this document and their links with Eclipse.

Chapter 8 is about the integration of proposed tools. Chapter 9 concludes this document.

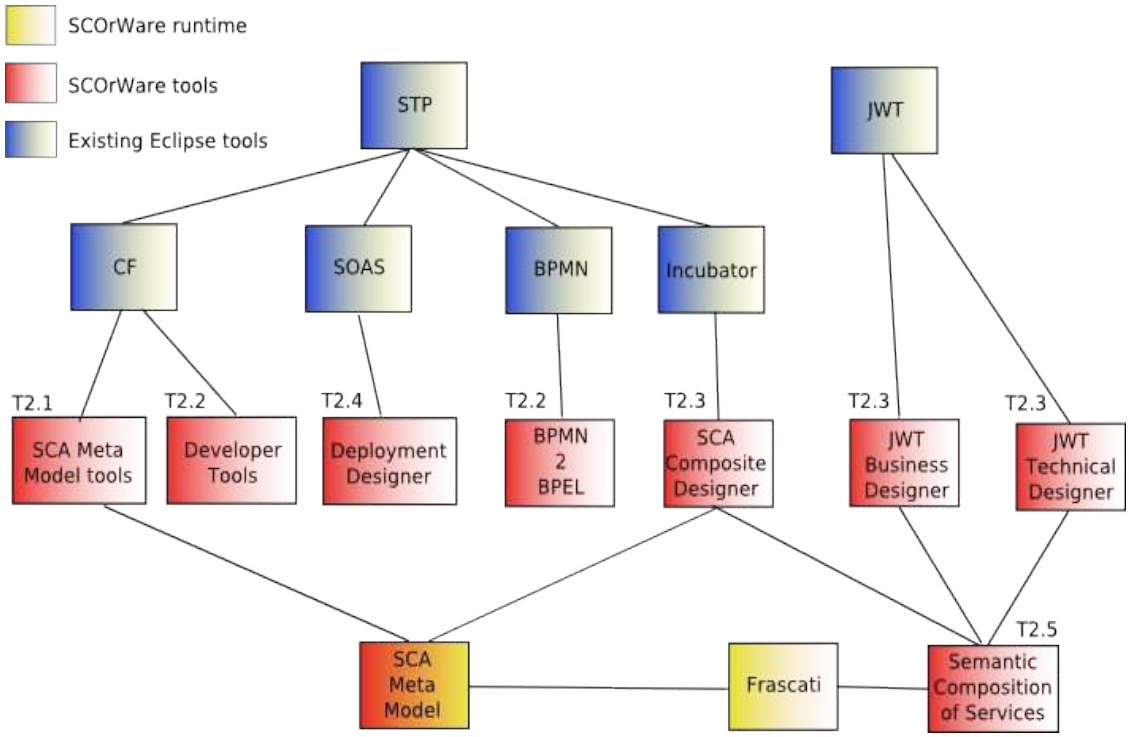


Figure 1: Overview of the SCOrWare Tools

2 Common meta model between platform and tooling

This Chapter concerns task 2.1. The purpose of this task is to develop tools to manipulate the common meta model for the SCOrWare runtime and the SCA tools. Section 2.1 presents the objectives, Section 2.2 the specifications and Section 2.3 the implementation.

2.1 Objectives

To develop tools to manipulate the common meta model we take advantage of the Eclipse Modeling Framework (EMF) [6]. EMF is a modeling framework and code generation facility to build tools and other applications based on a structured data model. From a model specification, described in XMI, EMF provides tools and runtime support to produce:

- The model as a set of Java classes and
- A set of adapter classes that enable viewing and command-based editing of the model.

2.2 Specification

EMF needs only the SCA meta model defined in [5] (Chapter 3) to build automatically desired tools. In Figure 3 is shown the diagram representing the SCA meta model.

Meta model additions

Some existing links between elements in the XSDs are not represented by references. For example, the SCA meta model element *Service* has a field *promote* that links it with a *ComponentService.* Tools need to know explicitly that a reference exists between these two elements.

Then, for each field representing a link between SCA elements, we add a reference as depicted in Figure 2. These new references are not saved but calculated from the field representing the link. These added references appear in Figure 3.

SCA Element	Field representing a link	Reference added
Service	The promote field identifies the promoted S*ervice*	promote2 that references the promoted *ComponentService*
Reference	The promote field identifies the promoted *Reference*	promote2 that references the promoted *ComponentReference*
Wire	The source field names the source component reference	source2 that references the promoted *ComponentReference*
Wire	The target field names the target component service	target2 that references the promoted *ComponentService*

Figure 2: Meta Model Additions

Additional validation rules

EMF allows to add additional validation rules that are not managed by the meta model. In fact, we need to add additional validation rules that appear in the SCA specification documents but that are not considered by the XSD scheme shown in [5] (Chapter 3). We distinguish 2 types of rules:

- The rules which apply on the SCA assembly description files and
- The rules which verify coherence between the SCA assembly description files and the implementation.

In the following are presented rules found in the SCA specification document [7] (the number between parenthesis is the line number of the rule in the specification document)

Rules which apply on the SCA assembly description files

- Uniqueness of names:
  - The component name must be unique across all the components in the composite (L142).
  - The name of a composite reference must be unique across all the composite references in the composite (L1325).
  - The name of a composite service must be unique across all the composite services in the composite (L1498).
  - The name attribute allows distinction between multiple binding elements on a single service or reference. The default value of the name attribute is the service or reference name. When a service or reference has multiple bindings, only one can have the default value; all others must have a value specified that is unique within the service or reference (L2319) .
- Multiplicity:
  - Multiple target services (target is an attribute of a service) are valid when the reference has a multiplicity of 0..n or 1..n. (L1581) (L203) (L211).
  - The value specified for the multiplicity attribute of a composite reference has to be compatible with the multiplicity specified on the component reference, i.e. it has to be equal or further restrict (L1364).
  - The target attribute of a composite reference is a list of one or more target services, depending on multiplicity setting (L1342).
  - The same component reference may be promoted more than once, using different composite references, but only if the multiplicity defined on the component reference is 0..n or 1..n. The multiplicity on the composite reference can restrict accordingly (L1392).
- Visibility:
  - For a composite used as a component implementation, wires can only link sources and targets that are contained in the same composite (irrespective of which file or files are used to describe the composite). Wiring to entities outside the composite is done through services and references of the composite with wiring defined by the next higher composite (L1630).
  - Components within the composite cannot be referenced directly by the using component. The using component can only connect wires to the services and references of the used composite and set values for any properties of the composite. The internal construction of the composite is invisible to the using component (L1852).
  - Composite services involve the promotion of one service of one of the components within the composite (L1059).
  - Composite references involve the promotion of one or more references of one or more components (within the composite) (L1061).
  - When a component is constrained by a constrainingType (via the "constrainingType" attribute), the entire componentType associated with the component and its implementation is not visible to the

containing composite. The containing composite can only see a projection of the componentType associated with the component and implementation as scoped by the constrainingType of the component. For example, an additional service provided by the implementation which is not in the constrainingType associated with the component cannot be promoted by the containing composite (L2188).

- Valid URI:
  - Valid URI schemes are <component-name>/<reference-name> for the attribute source of a wire. The specification of the reference name is optional if the source component only has one reference (L1621).
  - Valid URI schemes are <component-name>/<service-name> for the attribute target of a wire. The specification of the service name is optional if the target component only has one service with a compatible interface (L1625) .
  - The value of the promote attribute of a composite reference is a list of values of the form <component-name>/<reference-name> separated by spaces. The specification of the reference name is optional if the component has only one reference (L1328).
  - The promote attribute value of a composite service is of the form <component-name>/<service-name>. The service name is optional if the target component only has one service (L1501).
- Validity:
  - A composite used as a component implementation must honor a completeness contract. The concept of completeness of the composite implies that (L1857):
    - the composite must have at least one service or at least one reference.
    - each service offered by the composite must be wired to a service of a component or to a composite reference
  - Two or more component references may be promoted by one composite reference, but only when (L1395):
    - the interfaces of the component references are the same, or if the composite reference itself declares an interface then all the component references must have interfaces which are compatible with the composite reference interface
    - the multiplicities of the component references are compatible, i.e one can be the restricted form of the another, which also means that the composite reference carries the restricted form either implicitly or explicitly
    - the intents declared on the component references must be compatible – the intents which apply to the composite reference in this case are the union of the required intents specified for each of the promoted component references. If any intents contradict (e.g. mutually incompatible qualifiers for a particular intent) then there is an error.
  - If the attribute “WiredByImpl” of a reference is set to true, then the reference should not be wired statically within a composite, but left unwired (L220).
  - If the attribute “WiredByImpl” of a composite reference is set to true, then the reference should not be wired statically within a using composite,but left unwired (L1351).
  - uri attribute of a binding is optional for references defined in composites used as component implementations, but required for references defined in composites contributed to SCA domains (L2307).

Rules which verify coherence between the SCA assembly description files and the implementation

- Name and type matching with implementation:
  - The name of a service has to match a name of a service defined by the implementation (L164).

- The name of a reference has to match a name of a reference defined by the implementation (L192)
- The name of the property has to match a name of a property defined by the implementation (L279). The property type specified must be compatible with the type of the property declared by the implementation (L274).

- Compatible interface:
  - If an interface is specified for a service, it must provide a compatible subset of the interface provided by the implementation (L177).
  - If an interface is specified for a reference, it must provide a compatible subset of the interface provided by the implementation (L226).
  - If an interface is specified for a service composite it must be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service (L1512).
  - If an interface for a composite reference is specified it must provide an interface which is the same or which is a compatible superset of the interface declared by the promoted component reference (L1358).
  - A wire may only connect a source to a target if the target implements an interface that is compatible (see document) with the interface required by the source (L1634).
  - A wire can connect between different interface languages (e.g. Java interfaces and WSDL portTypes) in either direction, as long as the operations defined by the two interface types are equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and faults/exceptions map to each other (L1648).
- Validity:
  - When an implementation is constrained by a constrainingType it must define all the services, references and properties specified in the corresponding constrainingType. The constraining type's references and services will have interfaces specified and may have intents specified. An implementation may contain additional services, additional optional references and additional optional properties, but cannot contain additional non-optional references or additional non optional properties (a non-optional property is one with no default value applied) (L2182).
  - The constrainingType can include required intents on any element. Those intents are applied to any component that uses that constrainingType. In other words, if requires="reliability" exists on a constrainingType, or its child service or reference elements, then a constrained component or its implementation must include requires="reliability" on the component or implementation or on its corresponding service or reference. Note that the component or implementation may use a qualified form of an intent specified in unqualified form in the constrainingType, but if the constrainingType uses the qualified form, then the component or implementation must also use the qualified form, otherwise there is an error (L2196).
  - The rule which forbids mixing of wires specified with the target attribute with the specification of endpoints in binding sub elements of the reference also applies to wires specified via separate wire elements (L1592).

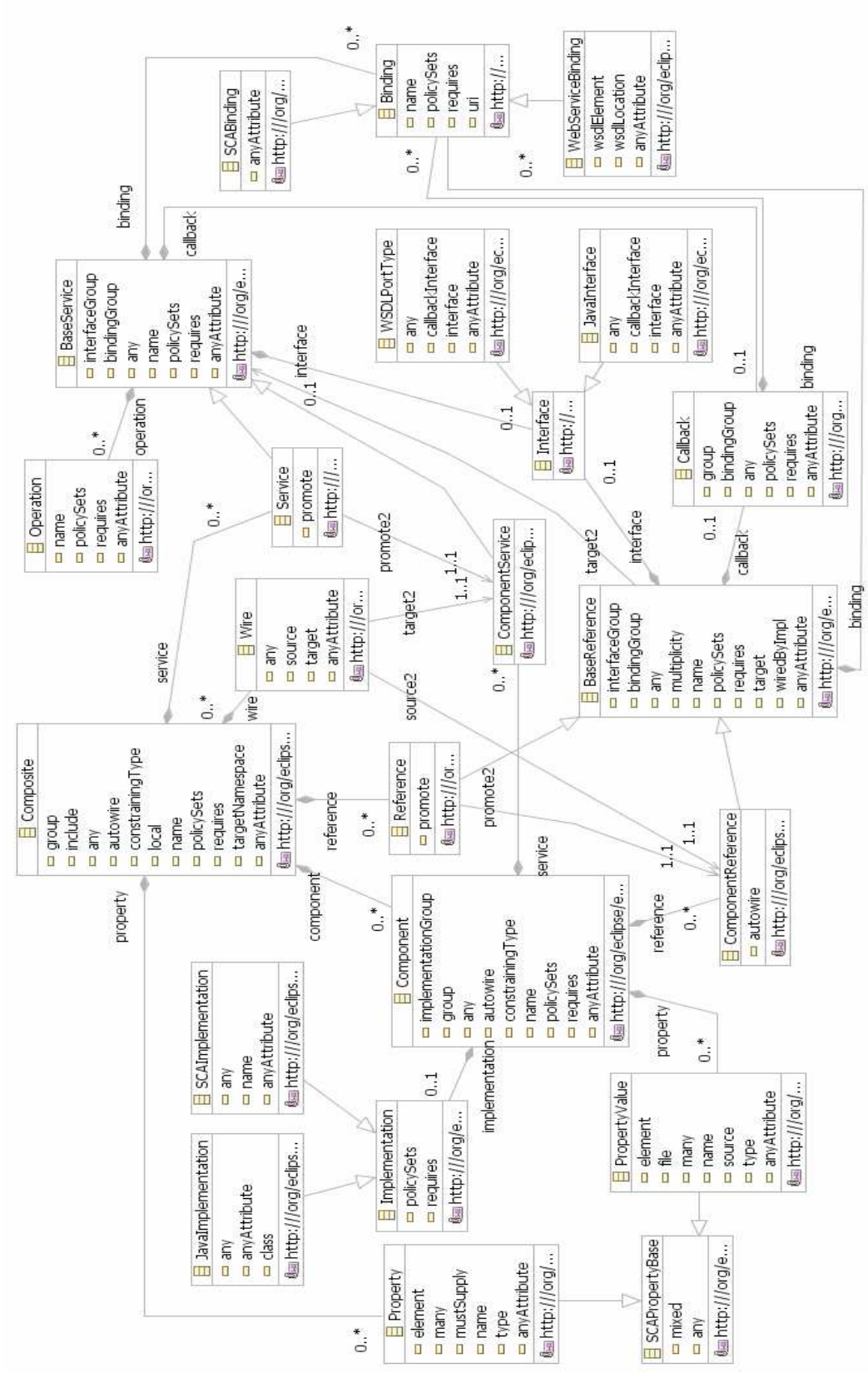


Figure 3: SCA meta model

2.3 Implementation

To build the meta model code and related tools (parser, editor, ...), we use the EMF framework.

Generate ecore meta model from XSD

First, to define the meta model we use the EMF framework to generate it from the XSD files. In the properties view, we can adjust the properties of the meta model elements.

Code generation

Second, to do useful work with the generated meta model, we have to generate a number of artifacts. First of all, we need to generate the ***genmodel*** (Figure 4 (a)). This is basically a decorator model around our meta model that "decorates" a number of extra properties on top of it. Once we have generated the *genmodel*, we select the root element, and execute “Generate All” from the context menu. This will generate (Figure 4 (b)):

- The **implementation classes for the meta model**. Each meta model element has a corresponding interface and implementation. Each generated interface contains getter and setter methods for each attribute and reference of the corresponding model class. Each generated implementation class includes implementations of the getters and setters defined in the corresponding interface, plus some other methods required by the EMF framework. The generated get method simply returns an instance variable representing the attribute. The set method, although a little more complicated, needs to send change notification to any observers that may be listening to the object. The generated accessors for references, especially two-way ones, are a little more complicated and start to show the real value of the EMF generator. See [8] for more explanations about the generated code.
- The **.edit project** that contains a number of generic reusable classes for building editors. It provides:
  - Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
  - A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.
  - A code generator capable of generating everything needed to build a complete editor plug-in. We can then customize the code however we like, without losing our connection to the model.

  See [9] for more explanations about the generated code.
- The **.editor project** that contains a tree-based editor, specific to that meta model. After running this plug-in, we can define instances of our meta model with a tree view.
- Finally, the **.test project** that contains a number of tests for our meta model.

Additional validation rules

Third, we add additional validation rules with the **framework EMF validation** [10] to check these additional constrains. The EMF validation framework provides a mean to evaluate and ensure the well-formedness of EMF models.

Validation comes in two forms: batch and live. While batch validation allows a client to explicitly evaluate a group of *EObjects* and their contents, live validation can be used by clients to listen on change notifications to *EObjects* to immediately verify that the change does not violate the well-formedness of the model. Client contexts can be specified to ensure that certain constraints do not get executed outside of a certain context. A context is defined by the client to set up their own boundaries from other clients.

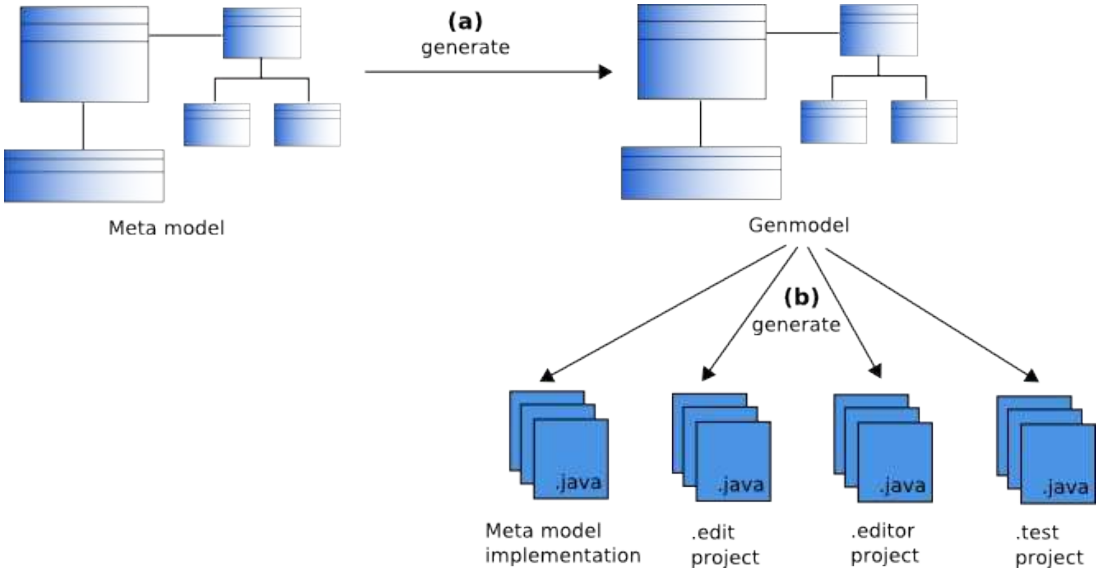


Figure 4: Use of EMF Framework

3 Development tools

This section concerns task 2.2. Section 3.1 presents tools for service and component creation for Java developers. Section 3.2 presents a code generator to generate BPEL from BPMN.

3.1 Tooling for service and component creation targeting Java developers

3.1.1 Objectives

Tools (for development, test, deployment and administration) that ease the use of a middleware like the SCOrWare runtime are as important as the quality of the middleware itself in order that the middleware will be adopted by the largest community.

We target unified and homogeneous development tools that support both the JBI approach and the creation of SCA components to be deployed in the PEtALS based runtime of SCOrWare.

While modelers and other tools mainly target end-user and components assembler, it is clear that developers also need tools for the SCOrWare platform so that they can be more efficient in the creation of the SCA components.

3.1.2 SCA Component Creation

Typically, a developer wants to control all the code he writes, but like productivity tools like wizard, generators and “development kits” that let him the full control of what he generates and just write the skeletons of code to write.

That's exactly this type of tools we provide for the creation of Java SCA Components.

In order to build this tools, we extend and eventually contribute to the tools provided by the STP project from Eclipse.

Hereafter are some screenshots (Figure 5 and Figure 6) of the current versions of the Java SCA Service Creation tool in STP.

This version is somewhat simplistic :

- It only supports two types of bindings, with Web Service or RMI, and it create empty Java class skeletons in an adequate directory structure.
- It supports Tuscany, but without the ability to give access to the deployment of SCA components directly to Tuscany

We will extend this tool in three directions :

- Support new types of Bindings, especially a binding to JBI, which will be the more natural one when using the SCOrWare Runtime
- Unification of JBI and SCA tools
- Generation of more elaborated code by providing “typical” components templates.

The generator for the SCA Java service code will be developed using Acceleo. Thus, the wizard fills a simple meta model which is passed to the Acceleo generator to create most of the “syntactic sugar” for the SCA Java service.

With the experience of the work done in PEtALS during the JOnES project, we know that the creation of SCA components, like the generation of JBI components can be much more productive if we “categorize” the components types, and provide a higher layer abstraction for each types. Categories will group together typical components according to their main characteristics like : “conversationnality”, “parametrizability”, ... Then, we can gather the more information from the developer and generate a lot of code that would be boring and error prone.

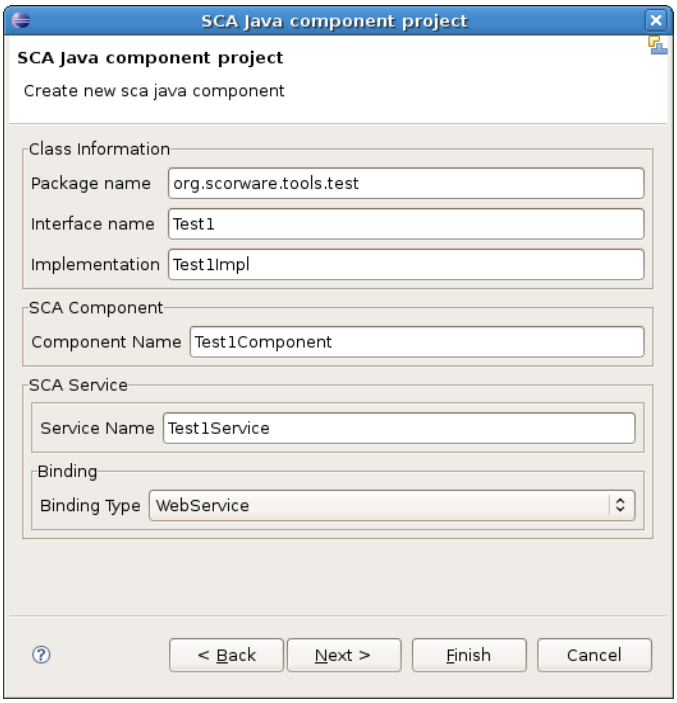


Figure 5: STP - Wizard to create a new SCA Java component

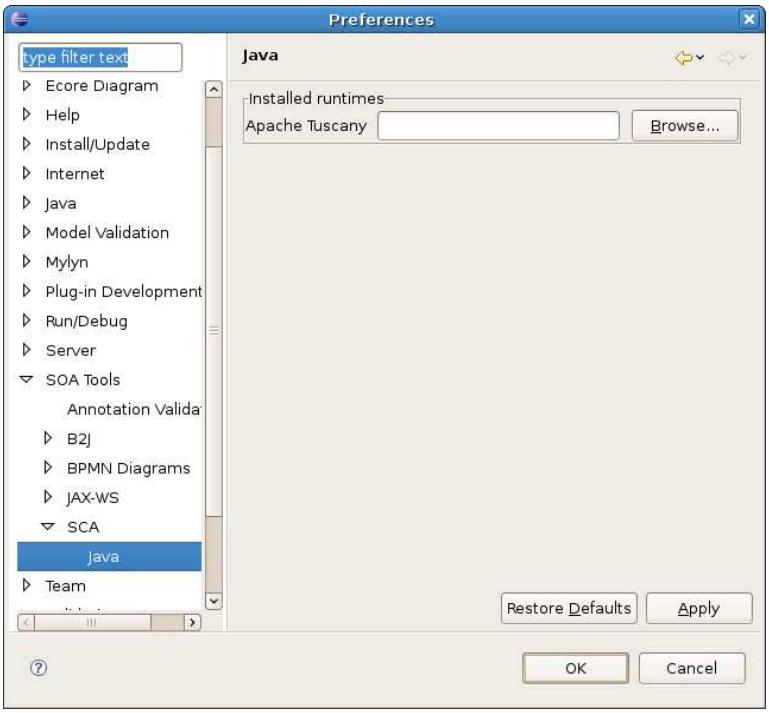


Figure 6: STP - SCA preferences page

Of course, we will add the support for Tinfi/PEtALS in the STP project so that components will be testable and deployable in Tinfi/PEtALS, just the way they are supposed to do in the short future with Tuscany.

The following pictures (Figure 7) gives an extract of a wizard designed for JBI component creation. It allows selection of component type, and is extensible via the extension mechanism to enable the creation of more and more specific components.

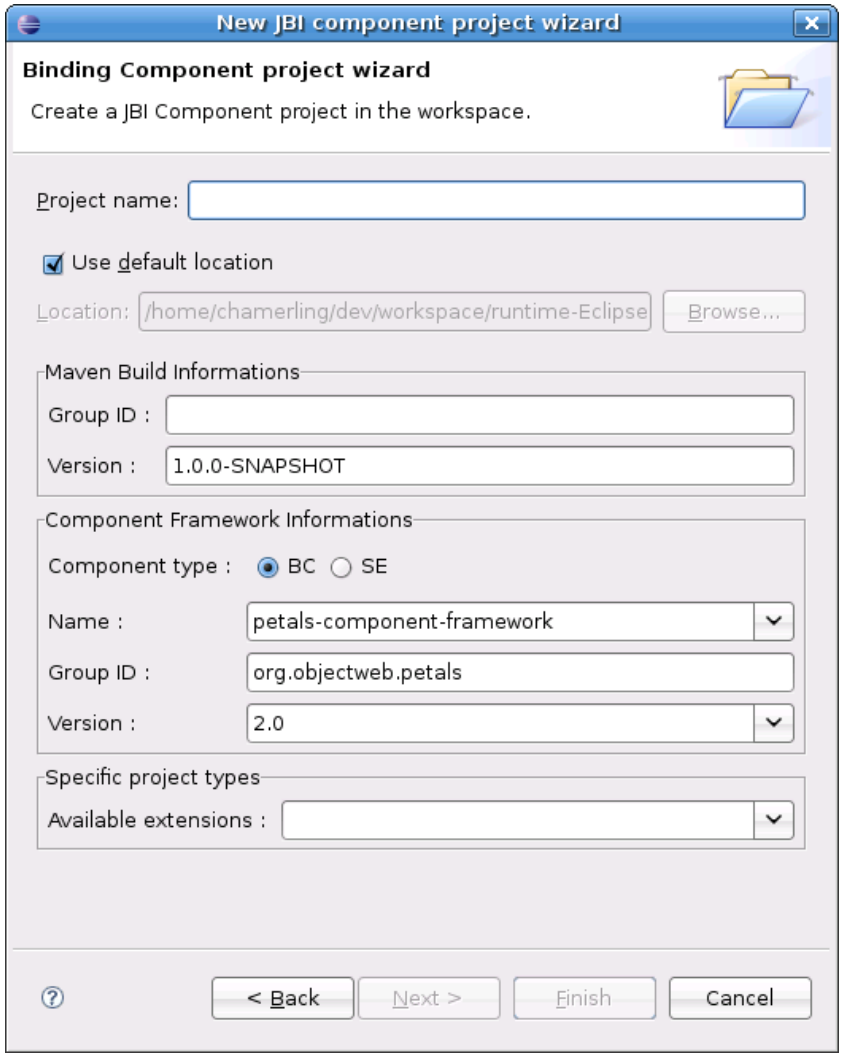


Figure 7: Wizard to create new JBI component

3.1.3 SCA Assembly Editor

We think that developers will need a more direct editor for the SCA assembly than the graphical editor.

The stake here is not only to support the SCA assembly schema, but also to help developers create their assembly by providing a tool that provides auto-completion for the XML attributes and values according to the “known” SCA components and their characteristics.

For example, in the Figure 8, the auto-completion must be context sensitive and propose the acceptable values to fill fields like “properties” and “references”.

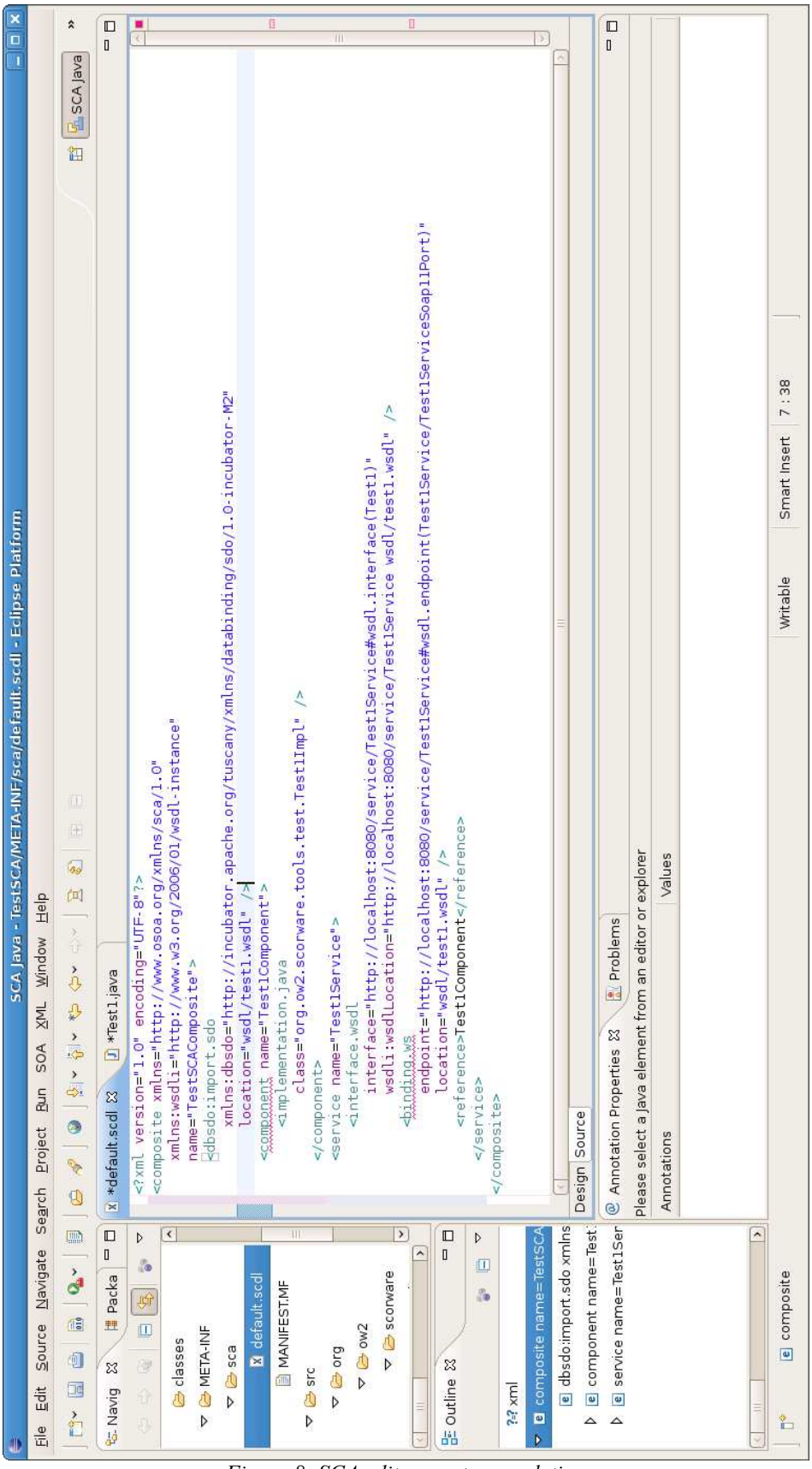


Figure 8: SCA editor - auto completion

A strong constraint, is that this SCA assembly editor must be able to be synchronized with the graphical SCA composite modeler provided by task 2.3.

The next picture (Figure 9) shows the wizard designed for JBI SU and SA development. We expect to extend it so that it is possible to create the same type of tool to ease the creation of SCA assemblies. In this type of wizard, each tab deals with a part of the deployment descriptor.

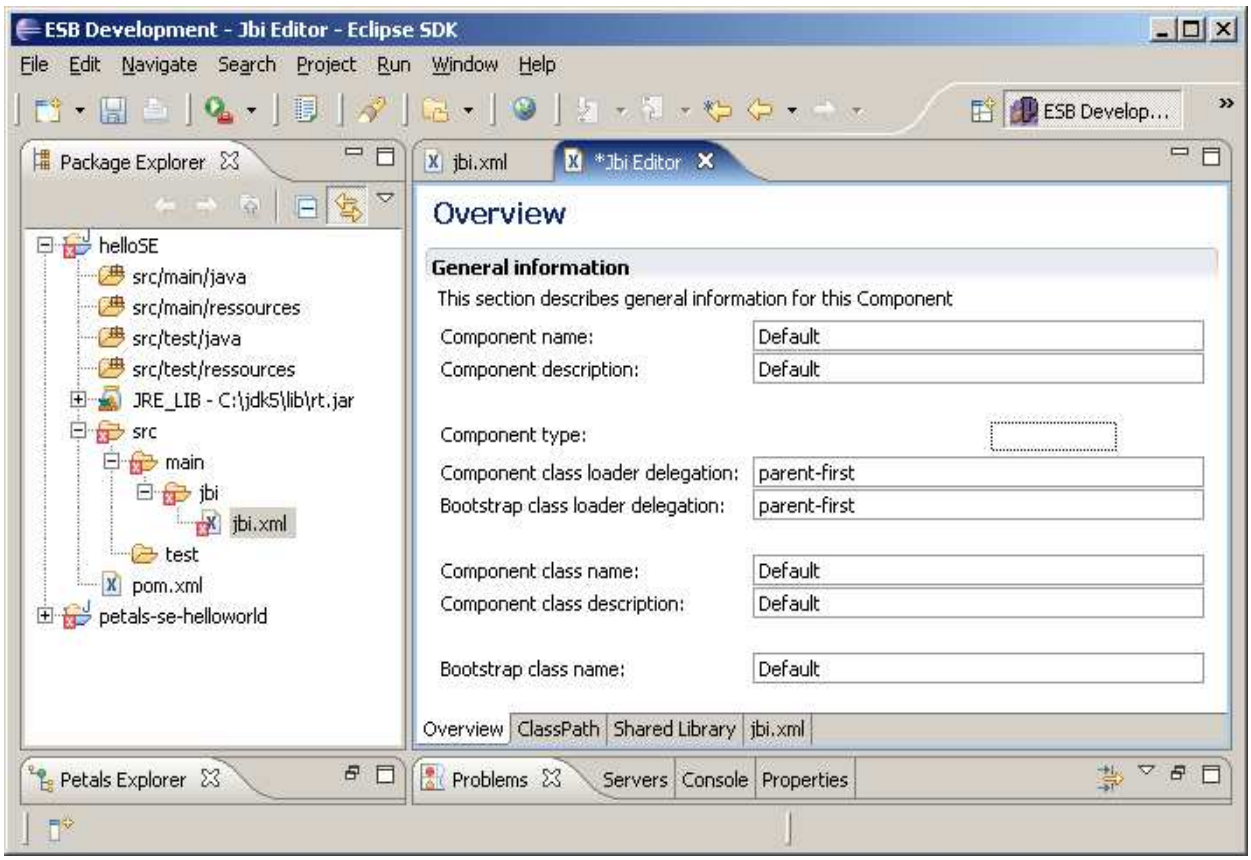


Figure 9: JBI Editor

3.1.4 SCA annotations support in Eclipse Java editor

This tool will provide support for SCA annotations for Java services, and for the additional annotations introduced in the SCOrWare platform if necessary.

Supporting such annotations means that the tool provides annotation completion for people that write code directly or want to annotate existing classes to make them SCA components.

3.2 BPMN to BPEL generator

3.2.1 Objectives

Another starting point for SOA is to develop a set of detailed business requirements or Business Process Models (BPM). To analyze enterprise systems the best way is to make a representation of work practices in the form of business process models. The Business Process Modeling Notation (BPMN) [11] is the classic way for representing such models.

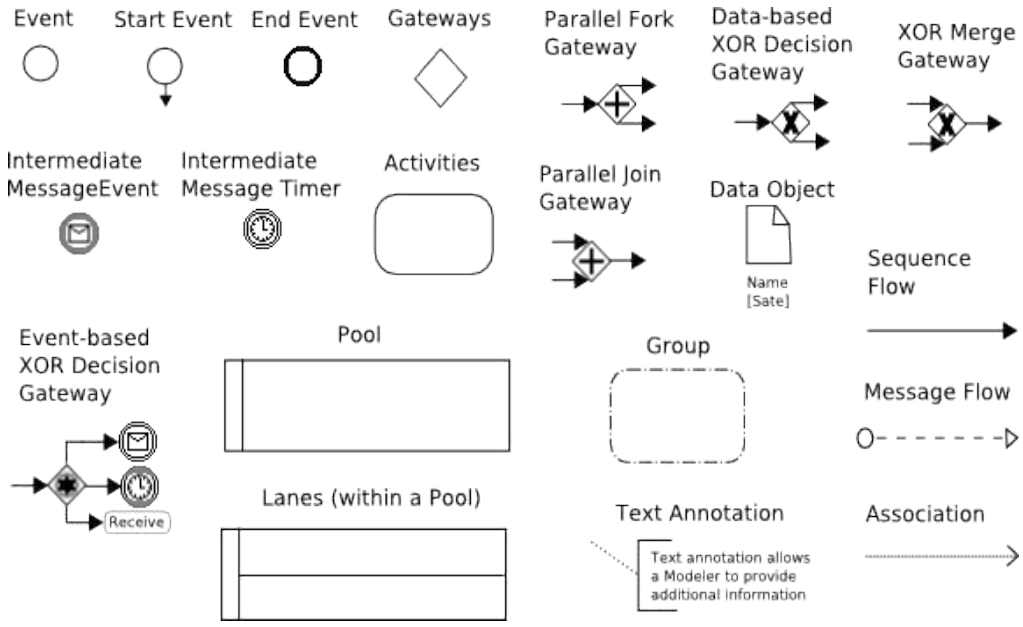


Figure 10: Sub set of BPMN elements

BPMN is a standardized graphical notation for drawing business processes in a workflow. Figure 10 depicts a sub set of BPMN elements. BPMN models are mainly intended for communication and decision-making between domain analysts, but often they are also used as input to software development projects.

The development methods for process-oriented systems rely upon detailed process definitions that are executed by process engines. These process definitions refine BPMN models by adding data manipulation, application binding and other implementation details. A major standard for process implementation is the Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short) [12][13]. Accordingly, a natural method for end-to-end development of process-oriented systems is to translate BPMN models to BPEL definitions for subsequent refinement. BPEL is a vendor-neutral specification being developed by OASIS to specify business processes as a set of interactions between web services.

In the SCOrWare project, we propose a code generator to generate BPEL from BPMN. This generator will be developed using Acceleo [14]. From a BPMN model and the template that maps BPMN to BPEL, Acceleo will generate the BPEL code.

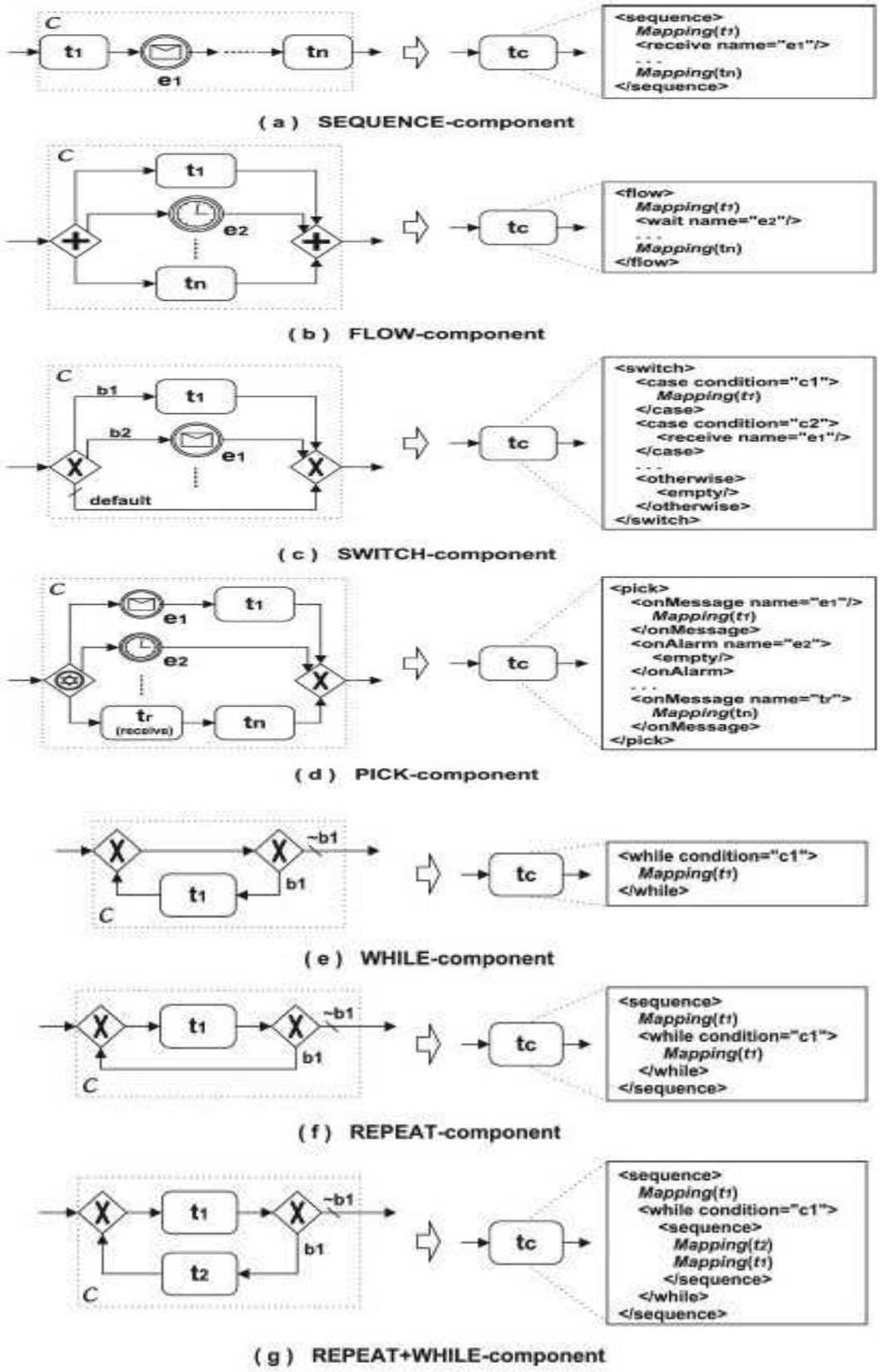


Figure 11: BPMN/BPEL

3.2.2 Specification

To generate BPEL code from BPMN, Acceleo needs as parameters the BPMN meta model and a template that maps BPMN to BPEL.

BPMN meta model

Since there is no official meta model of BPMN available, we propose to use as BPMN meta model the one proposed in Eclipse STP [15]. Figure 12 presents the diagram of the BPMN meta model. Next, is presented a short description of the BPMN meta model:

- **Activity**. An activity is work that is performed within a business process. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Process Model are: Task, and Sub-Process.
  - **Task**. A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail. There are specialized types of Tasks sending, receiving, user-based,... Markers or icons can be added to Tasks to identify the type of Task.
  - **Sub-Process**. Sub-Processes enable hierarchical Process development. A Sub-Process is a compound activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail (a Process) through a set of sub-activities. For a collapsed version of a Sub-Process, the details of the Sub-Process are not visible in the Diagram. A "plus" sign in the lower-center of the shape indicates that the activity is a Sub-Process and has a lower-level of detail. For an expanded version of a Sub-Process, details (a Process) are visible within its boundary. There are two types of Sub-Processes: Embedded and Independent (Re-usable).
- **Event**. An Event is something that "happens" during the course of a business process. These Events affect the flow of the Process and usually have a trigger or a result. They can start, interrupt, or end the flow:
  - **Start Events** indicate where a Process will begin. There are different "Triggers" that indicate the specific circumstances that start the Process. None Start Events are used to mark the start of Sub-Processes or when the start is undefined. Any one of the Triggers included in a Multiple Start Event will start the Process.
  - **Intermediate Events** occur after a process has been started and before a process is ended. There are different "Triggers" that indicate the specific circumstances of the Event. They can be placed in the normal flow of the Process or attached to the boundary of an activity.
  - **End Events** indicates where a process will end. There are different "Results" that indicate the specific circumstances that end the Process. None Start Events are used to mark the start of Sub-Processes or when the start is undefined.
- **Gateway**. Gateways are modeling elements that are used to control how Sequence Flows interact as they converge and diverge within a Process.
  - **Exclusive Gateways** (Decisions) are locations within a business process where the Sequence Flow can take two or more alternative paths. This is the fork for a process. Only one of the possible outgoing paths can be taken when the Process is performed. There are two types decision mechanism: Data (e.g., condition expressions) Events (e.g., the receipt of alternative messages). They are also used to merge Sequence Flow.
  - **Exclusive Gateways based on data**. These are the most commonly used type of Gateways. They can be shown with or without an internal "X" marker. Without is the most common use. The Gateway (Decision) creates alternative paths based on defined conditions.
  - **Exclusive Gateways based on events**. This type of Decision represents a branching point in the process where the alternatives are based on events that occurs at that point in the Process, rather than conditions. The Multiple Intermediate Event is used to identify this Gateway. The Event that follow the Gateway Diamond determine the chosen path. The first Event triggered wins.

- **Connection**.
  - A **Sequence Flow** is used to show the order in which the activities will be performed in a Process.
  - A **Message Flow** is used to show the flow of messages between two entities.
  - An **Association** is used to associate data, information and artifacts with flow objects.
- **Artifact**. Artifacts provide the capability to show information beyond the basic flow-chart structure of the Process. There are currently three standard Artifacts in BPMN: **Data Objects**, **Groups**, and **Annotations**. A modeler or tool can extend BPMN by defining new Artifacts.
- **Swimlane**. Swimlane helps partition and organize activities. There are two main types of swimlanes: Pool and Lane:
  - **Pools** represent Participants in an interactive (B2B) Business Process Diagram.
  - **Lanes** represent sub-partitions for the objects within a Pool.

BPEL description

BPEL is essentially an extension of imperative programming languages with constructs specific to web service implementations. BPEL processes are exposed as WSDL services and message exchanges map to WSDL operations. WSDL can be derived from partner definitions and the role played by the process in interactions with partners. Partner links are instance of typed connector. Partner link type specifies required and/or provided portTypes and channel along which a peer-to-peer conversation with a partner takes place.

A BPEL process definition relates a number of activities. An activity is basic or structured.

Basic activities correspond to atomic actions :

- **receive**. Do a blocking wait for a matching message to arrive
- **reply**. Send a message in reply to a formerly received message
- **invoke**. Invoke a one-way or request-response operation
- **assign**. Update the values of variables or partner links with new data
- **validate**. Validate XML data stored in variables
- **empty**. No-op instruction for a business process
- **throw**. Generate a fault from inside the business process
- **rethrow**. Forward a fault from inside a fault handler
- **exit**. Immediately terminate execution of a business process instance
- **wait**. Wait for a given time period or until a certain time has passed
- **compensate**. Invoke compensation on an inner scope that has already completed
- **extensionActivity**. Wrapper for language extensions

Structured activities enable the presentation of complex structures:

- **flow**. Contained activities are executed in parallel, partially ordered through control links
- **sequence**. Contained activities are performed sequentially in lexical order
- **while**. Contained activity is repeated while a predicate holds
- **repeatUntil**. Contained activity is repeated until a predicate holds
- **pick**. Block and wait for a suitable message to arrive (or time out)
- **forEach**. Contained activity is performed sequentially or in parallel, controlled by a specified counter variable

- **if then else**. Select exactly one branch of activity from a set of choices
- **scope**. Associate contained activity with its own local variables, fault handlers, compensation handler, and event handlers

From BPMN to BPEL:

[16] presents a mapping to BPEL that is derived by analyzing the BPMN objects and the relationships between these objects. It is a good starting point to generate BPEL code from a BPMN model. However, mapping BPMN diagrams to BPEL processes is not an easy task because of the structural disparity between BPMN and BPEL. BPEL is a block structured language. In contrast, BPMN is a constrained, but relative free form graph. There are no fundamental difficulties in mapping a BPEL process to an BPMN diagram. But it is not always possible to map a BPMN diagram directly to a BPEL process. Without analyzing and redrawing such diagram flow structures, it is practically impossible to map all processes correctly. To map more BPMN diagrams onto BPEL we will use the solution proposed in [17].

[17] proposes to decompose the business process diagram into components. A component is a subset of the business process diagram that has one entry point and one exit point. Then they map components onto suitable BPEL blocks. For example, a component holding a purely sequential structure should be mapped onto a BPEL sequence construct while a component holding a parallel structure should be mapped onto a flow construct.

The goal of [17] is also to generate readable BPEL code. For this purpose, BPEL structured activities (sequence, flow, switch, pick, and while) have the first priority if the corresponding structures appear in the business process diagram. Components that can be suitably mapped onto these five structured constructs are considered well structured.

[17] depicts some examples (see Figure 11) of mapping well structured components onto the corresponding BPEL structured activities. A component C is replaced by a single task tc attached to the corresponding BPEL translation of C. In Figure 11(a) to 11(e), mappings of the five components (sequence, flow, switch, pick, and while) are straightforward. In a pick component (Figure 11(d)), an event-based XOR decision gateway must be followed by receive tasks or intermediate message or timer events. For this reason, a sequence-component (Figure 11(a)) cannot be preceded by an event-based XOR decision gateway. In Figure 11(f) and 11(g), the two components, repeat and repeat+while, represent repeat loops. A while loop (see Figure 11(e)) evaluates the loop condition before the body of the loop is executed, so that the loop is not executed if the condition is initially false. In a repeat loop, the condition is checked after the body of the loop is executed, so that the loop is always executed at least once. In Figure 11(f), a repeat loop of task t1 is equivalent to a single execution of t1 followed by a while loop of t1. In Figure 11(g), a repeat loop of task t1 is combined with a while loop of task t2, and both loops share one loop condition. Task t1 is always executed once before the initial evaluation of the condition, which is followed by a while loop of sequential execution of t2 and t1.

To map not well structured components onto BPEL, [17] proposes to translate such components into a scope activity by exploiting the event handler construct in BPEL. An event handler is an event-action rule associated with a scope, and thus the corresponding translation approach is based on event-action rules. As the first step in the event-action rule-based translation, they generate a set of preconditions for each object within a component. The term precondition is used to capture one possible way that leads to the execution of an object, and thus a set of preconditions associated with the object encodes all possible ways for reaching that object. They propose an algorithm for generating all precondition sets for a component. In the second step, they transform the precondition sets with their associated objects into a set of event-action rules. As the last step, [17] translates event-action rules to BPEL.

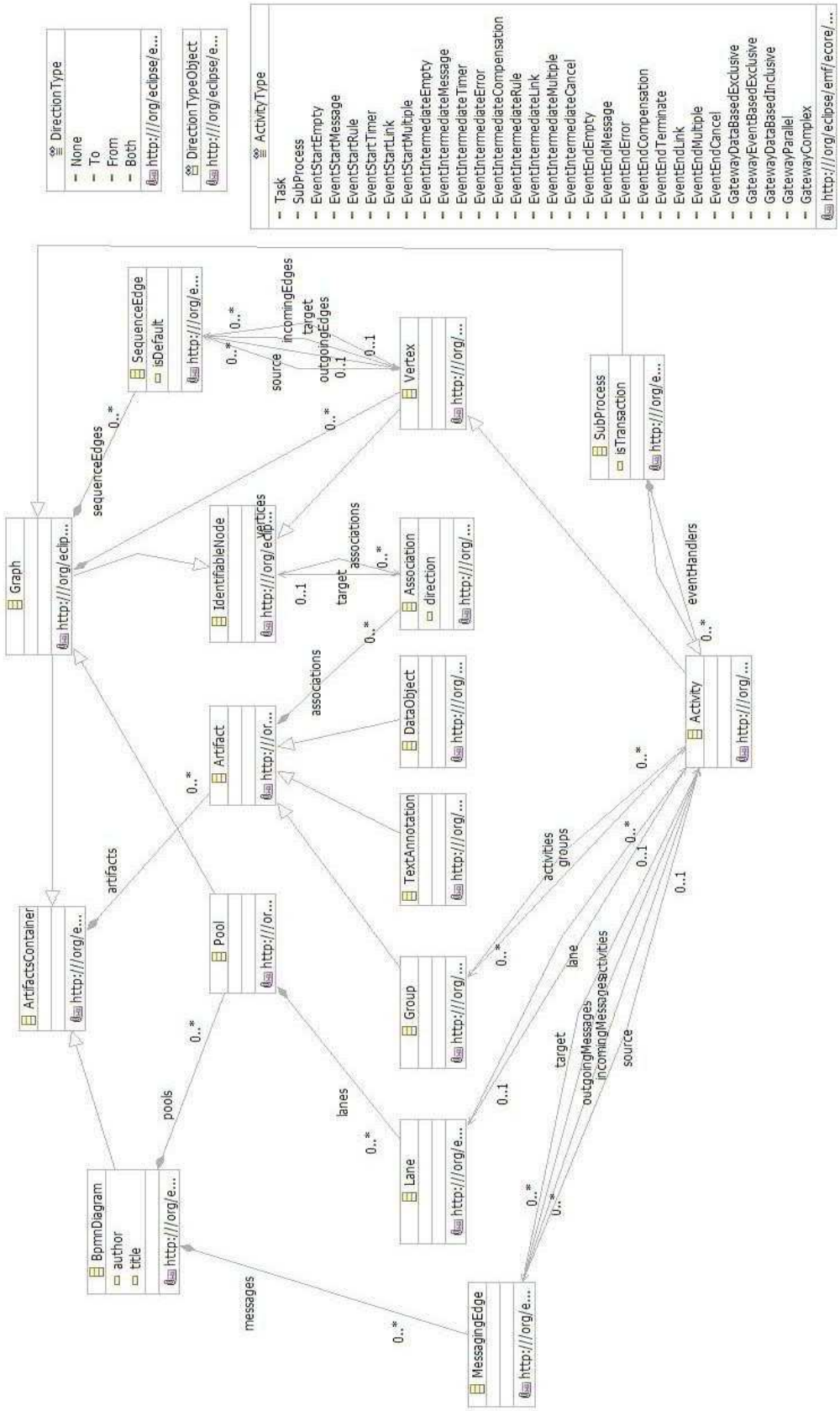


Figure 12: BPMN meta model

3.2.3 Implementation

Generating source code can be powerful, but the program that writes the code can quickly become very complex and hard to understand. One way to reduce complexity and increase readability is to use templates. The Eclipse Modeling Framework (EMF) project contains a very powerful tool for generating source code: JET (Java Emitter Templates) [18]. JET uses a JSP-like syntax (actually a subset of the JSP syntax) that makes it easy to write templates that express the code to generate. JET is a generic template engine that can be used to generate SQL, XML, Java source code and other output from templates. JET accepts objects passed in arguments as well as Java code to create variables, to produce loops, etc.

However to implement our BPMN to BPEL generator we will use Acceleo [14]. Like JET, Acceleo is a code generator integrated in Eclipse that transforms models into code using templates. Acceleo allows to browse models whatever the meta model on which they are based (UML, ecore, GenModel,…). But, contrary to JET, the simplified syntax of Acceleo, reduced to the minimum number of key words, makes it possible to concentrate on the target code and the parsing of the model. In addition, Acceleo has already tools for the edition of templates, the traceability of the code, the synchronization between code and model, etc.

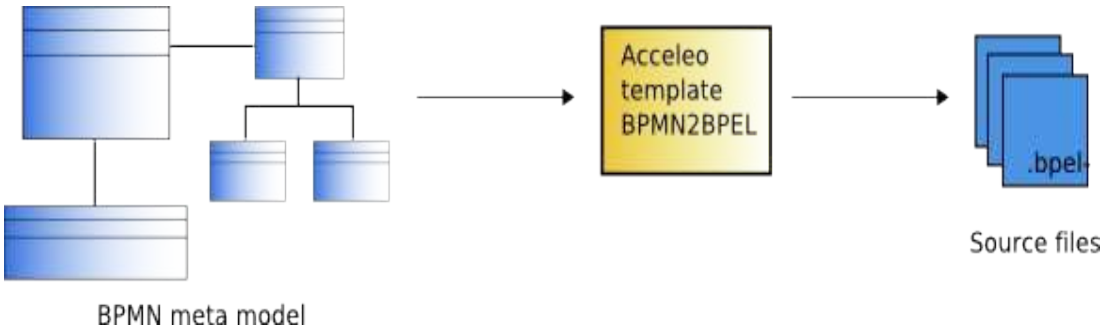


Figure 13: BPEL generation from BPMN model with Acceleo

Acceleo uses as parameters a meta model and a generation template (Figure 13). In our case, we use the BPMN meta model presented above. A generation template is a pattern that should show clear and proper static text from variable elements. The generation template must follow the rules defined in the algorithm presented above. The template is divided in different parts:

- The heading section defines the meta-model on which the template will apply. Here the URI corresponds to the one of our BPMN meta-model. The heading section also includes Imports that allow to include other templates or Java Services. Java services will be used to implement the algorithm to transform BPMN to BPEL.
- The template is divided into a certain number of scripts. A Script is the elementary unit of a template. It is applied on a model element to produce some text. Every script will be evaluated on a given element type. A script is composed of static areas that will be generated as is in the generated file and areas surrounded by <% and > signs that are dynamic areas. These dynamic areas correspond to the expression evaluation on the current object of the meta model.

4 Graphical designers

This section presents graphical tools for Eclipse that simplify the SCA developer task. Section 4.1 specifies the SCA composite designer. This graphical editor helps the development and the configuration of SCA assemblies. Section 4.2 introduces Business Process (BP) tooling and integration in SCOrWare. Section 4.3 presents the JWT business designer, and Section 4.4 the JWT technical designer.

4.1 SCA Composite designer

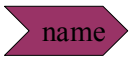
4.1.1 Objectives

The SCA Composite editor is a graphical tool to construct SCA composite files. The tool will provide both bottom-up and top-down methods for constructing standard SCA 1.0 composites. This graphical editor will respect the graphical representation proposed in the SCA specifications. This Eclipse tool will be developed with the GMF framework and from the SCA meta model.

Graphical representation of a component

For some SCA artifacts (service, reference, property, component, composite, wire, promote) the specification document [7] proposes a graphical representation. For the SCA artifacts not represented in the specification (implementation, interface and binding) we propose our own graphical representation. The SCA document specification depicts:

- A service by the following green figure: 

. The service name is on the top of the figure. The same figure is used for a component service and for a composite service. The only difference is in the properties where a composite service has a property “promote”.
- A reference by the following purple figure: 

. The reference name must be on the figure. The same figure is used for a component reference and for a composite reference. The only difference is in the properties where a composite reference has a property “promote”.
- A property by a yellow rectangle: 

. The property name must be on the figure. The same figure is used for a component property and for a composite property.
- A component by a blue rounded rectangle (Figure 14). The component name must be at the center of the component. Component services straddle the left side of the component, component references straddle the right side of the component and properties straddle the up of the component.

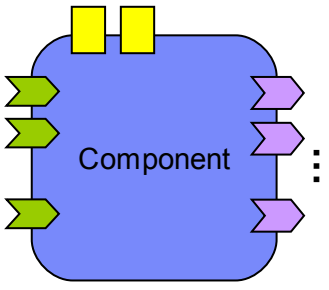


Figure 14: Graphical representation of an SCA component

- A composite by a light blue rounded rectangle (Figure 15). The composite name must be on the left corner of the composite graphical representation. Components are represented inside the composite. Composite services straddle the left side of the composite, composite references straddle the right side of the composite and properties straddle the up of the composite.

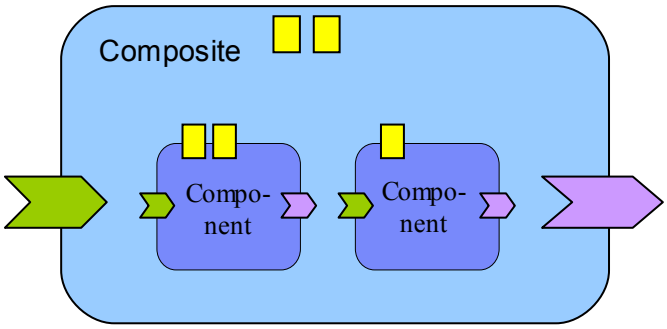


Figure 15: Graphical representation of an SCA composite

- A wire by a black line. Two types of wire are possible: the first one is a wire object, and the second one uses the target property of a reference to set the wire. Wires can only connect a component reference to a component service.
- A promote link by a black dash line. A promote link can be added between a composite service and a component service or between a composite reference and a component reference.

The specification document does not propose a graphical representation for:

- A binding. We propose to represent this SCA element by a red circle on a component/composite service/reference. Bindings can be only added to a service or a reference (of a composite or a component). Several bindings can be added to the same service/reference.
- An implementation. We propose to represent this SCA element by a yellow circle on a component. An implementation can be only associated with a component and only one implementation can be associated to a specific component.
- An interface. We propose to represent this SCA element by an orange circle on a component/composite service/reference. Interfaces can be only added to a service or a reference (of a composite or a component). Only one interface can be added to the same service/reference.

4.1.2 Specification

The SCA composite designer must allow to construct SCA composite configuration files in a top-down manner and in a bottom-up manner.

Top-Down method

This tool works in a *top-down* manner, allowing the creation of composites first, and then the generation of model code.

The process to construct a composite configuration is the following:

- First, a wizard is used to create a new composite configuration. The user picks-up a name for the composite file.

- The SCA composite editor is open with the graphical representation of the new created composite.
- The user can add SCA elements (component, service, reference, wire, binding, implementation, interface, ...). For this, creation tools are available at the right side of the editor in the creation tools palette.
- Interface, implementation and binding can be added also by drag and drop of existing files in the workspace.
- A right-click on an implementation, binding or interface representation proposes a menu item that allows to open the element with the corresponding editor. For example right-click on a BPEL implementation open the file with the BPEL editor.
- A right-click on a component service (or a component reference) proposes a menu item that allows to promote it automatically. For this, the corresponding composite service (or composite reference), its name (the same as the promoted element) and the promote link are created.

Bottom-up method

This tool works also in a *bottom-up* fashion, discovering components that have been developed in code and producing a graphical representation. The process to construct a composite configuration by introspection of the workspace is the following:

- First, like in the top-down approach the wizard allows to pick-up a name for the composite file.
- The graphical editor is open with the newly created composite.
- Then, a right click on the composite representation proposes a menu item to import components from the workspace.
- A new wizard is open. This wizard (like in Figure 16) proposes to select components to be created out of existing implementations. Then, the graphical representation of selected components are created inside the composite. Each component is created with Service, Reference, Implementation and Interface according to information found by introspection of selected components.
- After that, properties on the components, wire and promote links can be added.
- Like in the top-down method:
  - A right-click on an implementation, binding or interface representation proposes a menu item that allows to open the element with the corresponding editor and
  - A right-click on a component service (or a component reference) proposes a menu item that allows to promote it automatically.

Use semantic broking service

This tools will be integrated with the tools developed in Chapter 6 to pick-up services corresponding to a certain set of semantic criteria.

Meta-model

The meta model used to construct this tool is a subset of the SCA meta model. This subset contains all elements relative to the *Composite* element.

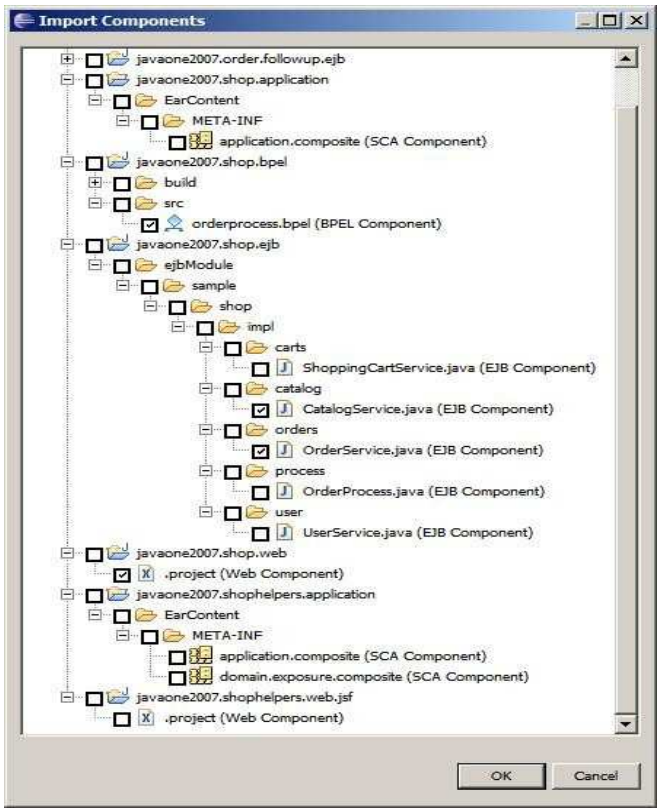


Figure 16: Wizard to select existing components to add to a new SCA composite

4.1.3 Implementation

To build this editor we will use the GMF framework [19]. We start from our meta model (the *.ecore* file) and the *.genmodel* defined above in Section 2.3 (Figure 17 (a)). A number of additional models have to be defined:

- A **model defining the graphical notation** (Figure 17 (c)) including shapes, decorations and graphical nodes and connections. This is called the *.gmfgraph* model.
- A **model for the editor's palette** (Figure 17 (b)) and other tooling, called the *.gmftool* model.
- A **mapping model** (Figure 17 (d)) that binds these two models to the domain meta model. The two models defined above are technically independent of your domain meta model.

From all of these additional models, GMF creates the ***.gmfgen* model** (Figure 17 (e)) – a "low level" model that the code generator uses as an input, finally creating the .diagram project which contains our desired editor.

The Tool Model

The *.gmftool* model defines only a set of palette entries. The palette is the set of buttons that allows to add model elements to our model. So we need a creation tool for each of the meta model elements that we want to be able to place onto the editor.

The Graph Model

The graph model defines several things:

- One is the set of figures defined in *FigureGalleries*. Colors, line, and static decorations are also defined here.
- We also define graph nodes and connections.
- We also define compartments. Compartments are sections in nodes that can be collapsed and themselves contain graphs (or lists of elements).
- Finally, we define diagram labels used to show text associated with graphical elements.

The Mapping Model

This is the most complex model. Here we map the tool definition and the graph definition to the domain meta model. To be able to map the different elements, we have to add these other resources to the editor:

- For each meta model element, that we want to map directly onto the diagram surface, we have to define first a Top Node Reference.
- Attached to a Top Node Reference, we add a normal Node Mapping. It contains information about the model element to map and the property, in which the set of these elements is stored in the container.
- Attached to a Node Mapping there's a Label Mapping. This one associates the label defined in the graph with the respective model element properties.
- We can also have a Child Reference that identifies the set of children that should be shown.
- The Child Reference can be associated with the Compartment, to ensure that the child collection is actually shown in the respective compartment.

All of this have to be mapped with a number of properties. The editors for doing that are just the usual tree editors which make all of that stuff a bit cumbersome. There are additional, more specialized editors in the GMF pipeline that should make this process simpler.

Generating the Editor

Now we can create the *.gmfgen* model from the set of models we just create. Finally, from the *.gmfgen* model, we can generate the diagram code – this will be contained in the *.diagram* project.

Specific code must be developed for the service and reference figures. These figures must represent the shapes defined by the SCA specification document. GMF does not take into account natively these types of shape. Moreover, specific code must be developed to represent component and composite according to the specification document. This code must allow to represent services and references on the component and composite border.

Figure 18 shows a prototype of the SCA Composite designer contributed to the STP project by Obeo.

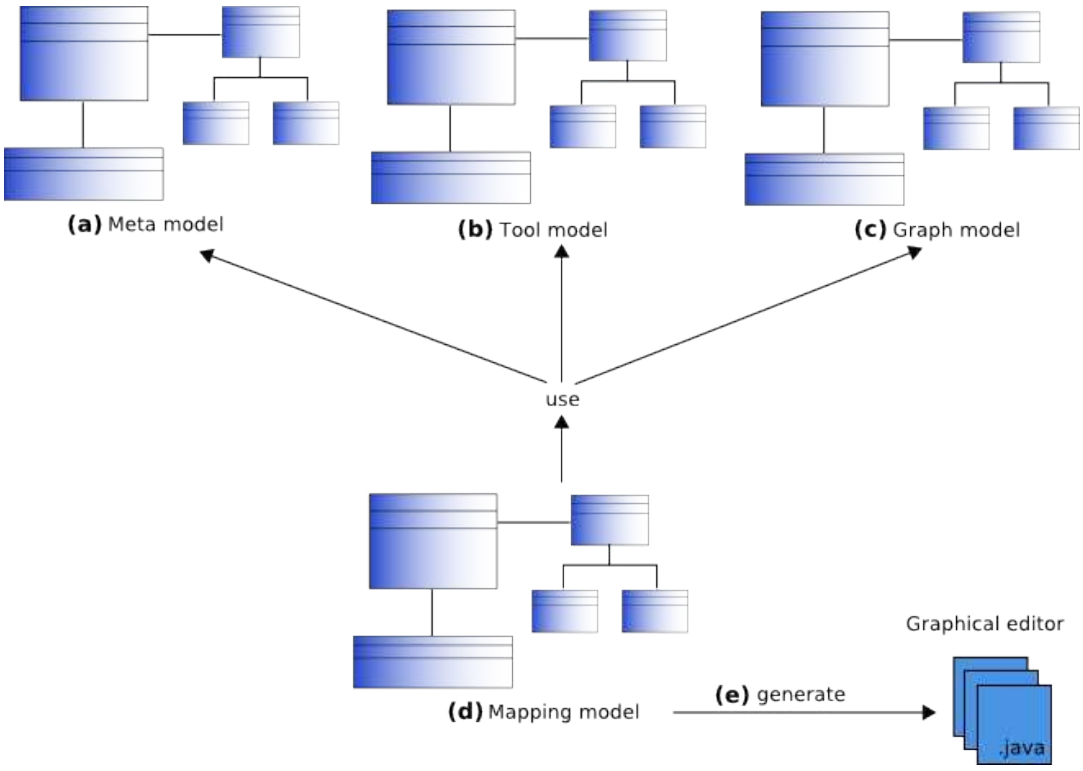


Figure 17: Use of GMF to generate graphical editor

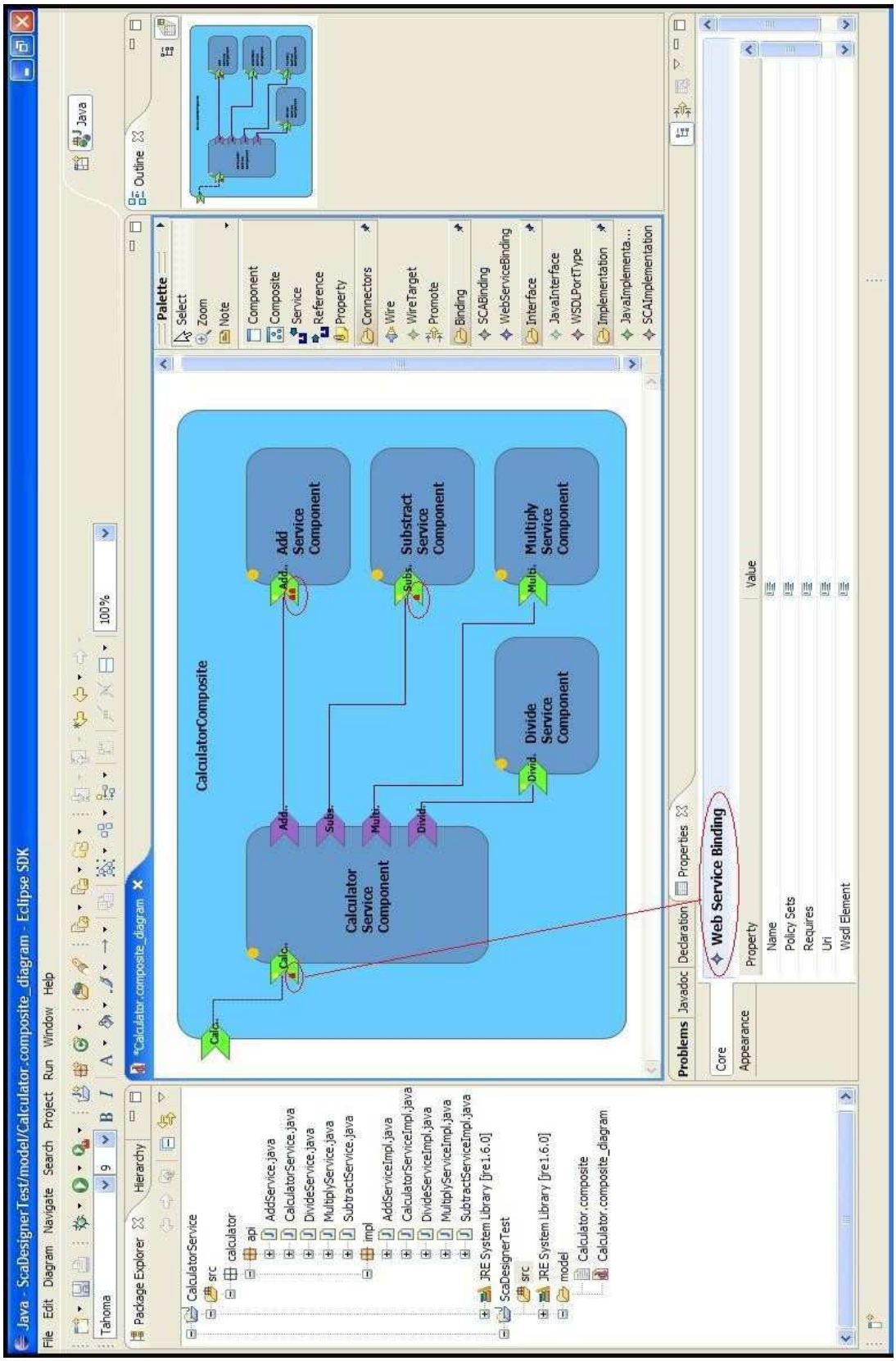


Figure 18: SCA Composite designer prototype

4.2 Preliminary note about BP tooling and integration in SCOrWare

JWT for SCOrWare tooling suite

JWT's goal in the context of the SCOrWare platform is to provide a top-down business-oriented approach to modeling business processes and implementing business tasks using the SCOrWare framework. In a more concrete point of view, this should include providing a high-level process view, providing tooling for building processes using SCOrWare services, providing graphical tooling for the semantic service and process engine.

JWT's larger scope objectives – that is, a unified approach to BPM development tooling and especially design, implementation using a BP language and testing – are in line with the SCOrWare specific objectives, since SOA as a methodology and SCA as a technology are in their own way addressing the same kind of problematics.

What the BP tooling suite comprise

The workflow and orchestration tool suite is provided by JWT, its designer and the hereby specified additional developments.

Nowadays JWT is available as a complete workflow solution, featuring a BP designer, using its own BP format and coming along with a standalone simulator application allowing BP execution.

The top-down approach that BP brings to SOA features two different BP design use cases:

- The BP analyst's (where the prominent standard is BPMN)
- And the BP developer's (with formats as BPEL, XPDL etc.).

In order to support both use cases, as well as increase its standard compliance, it will feature a BPMN compliant business designer on one side, and support a standard BP format on the other side. This will be achieved with the help of JWT's format interoperability and meta model genericity.

Other parts of the SCOrWare BP solution

This part is about BP designers.

See further for :

- JWT integration with SCOrWare services
- JWT designer integration
- JWT process execution integration
- Service registry (implemented on top of the SCA service / component semantic provider)
- Process registry
- Process execution monitoring

About standard BP Engine choice and integration

Obviously if we want business processes to run within the SCOrWare platform, a business process engine will be required that is able to orchestrate SCA / SCOrWare services (service call, data binding, even having a BP engine API made available to the SCOrWare platform).

As for now, JWT provides a simulator that supports its own BP format. This simulator is based on the jBoss jBPM technology.

A study will be done by Open Wide to choose the BP engine of choice for JWT / SCOrWare. This engine will have to be able to support the SCOrWare use cases and the SCOrWare SCA and BP feature set.

Note : Open Wide has experience with many BP engines, including Bonita, which is a fellow ObjectWeb project

and has been integrated with the Petals ESB by Open Wide in the context of the JoNES project. Moreover the next generation of Bonita will be built using the Process Virtual Machine core that promises to unify BP development and deployment - and actually will be the basis of the next generation of jBoss jBPM as well.

Following this study, said BP Engine will be integrated in the second year of the project with JWT and the set of companion tools developed for SCOrWare by Open Wide.

Tying everything back together : the big picture

Here is a diagram (Figure 19) showing how the different BP models (including BP representation models, graphical editors meta models, BP language format models) actually interact within the JWT for SCOrWare BP solution, thanks to JWT's key notion of *pivotal meta model* :

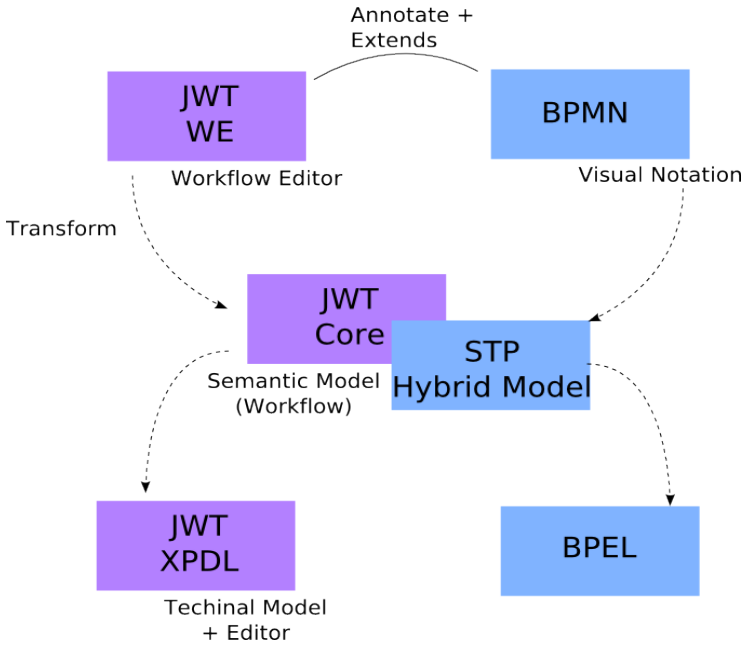


Figure 19: JWT / BPMN / BPEL / STP Hybrid Model

The JWT core meta model acts as a pivotal meta model in that every other meta model for which there is a bijective transformation with it has access to the full JWT features, just like the core meta model has – and therefore both may be transparently swapped. This is showed in the middle line of the diagram.

The upper line of the diagram shows business process representation editors. Those are targeted at business analyst rather than developers. Those editors allows to edit information that may be used, thanks to a transformation (usually one way), as a starting point for more developer oriented work done in the JWT technical editor or on a pivotal meta model.

On the lower line of the diagram stand the technical tools, allowing to produce Bps in their own language format, possibly even targeted at a specific platform, like SCOrWare. Those are produced through another transformation from a pivotal meta model (usually one way), and they may also have their own editor.

4.3 JWT business designer

4.3.1 Objectives

The objective is to develop the tooling required by the BP analyst use case of the top-down approach that BP brings

to SOA.

This will be achieved with the help of JWT's format interoperability and meta model genericity.

It will concretely be a BPMN compliant design tool.

In the first year, it will simply consist in integrating the BPMN Editor provided by the Eclipse STP BPMN project.

4.3.2 Specifications

About BPMN for SCOrWare

BPMN is a graphical notation for modeling business processes. It is meant to have all the expressive power needed for the business analyst to design its business processes. As such a SCOrWare integrated BPMN designer is the business oriented tool of choice in the first step of a top-down approach of BP design in the SCOrWare platform.

However it is important to note that it is not a BP language in itself : for example, it has no text format representation and it can't be executed. Concretely, it talks about shapes and edges with BP related meaning like task, transition and so on ; but there is no implementation information.

The point of a BPMN diagram is what it means, therefore we will have

1. to specify this meaning in the context of SCOrWare (see BPMN subset afterwards)
2. to define a textual (XML) format for ex. exchange with other tools (see EMF meta model)

BPMN subset and meaning

A subset of BPMN that will be supported first will be defined. This subset has to be enough for the needs of the SCOrWare partners and the common uses of SCOrWare. The aim is to already have a fully functional one in year one. This study will be done by Open Wide with the help of Obeo, IRIT and the JWT community.

This subset will be able to represent

- activity (task),
- action and implementation specific extensions (service call etc.),
- guarded transitions (decisions),
- loops,
- business role,
- data.

BPMN graphical modeling tool

The aim is to provide a BPMN modeling solution that is consistent and integrated with the SCOrWare platform.

We choose the EMF / GMF technology as the technology of choice of this BPMN editor.

In year one, we will prototype said solution by

- building on the Eclipse STP BPMN editor using its flexible annotation framework
- and by developing tools allowing its edited diagrams to be transformed in SCOrWare formats (see other parts of this document)

Note : in year two, thanks to the information provided by the use and analysis of this prototype, a choice will be made between extending the prototype to entirely support BPMN and with more features or building a dedicated BPMN designer will be built in EMF / GMF and use its own EMF meta model based on the BPMN subset validated in the prototype.

BPMN integration with technical processes

A study comparing JWT and this BPMN's meta models and explaining how to match one with the other will be done jointly by Open Wide, IRIT, Obeo and the JWT community.

Since BPMN share with JWT the transition-based process paradigm, such a transformation is quite straightforward once the meaning of BPMN shapes in JWT has been decided and the information required by JWT added as annotations.

A transformation from this BPMN meta model to the JWT native meta model will be provided by Obeo on the basis on the previously mentioned study and integrated as a graphical tool.

See the “model transformation” part of the present document.

Note : the JWT meta model that will be the transformation target will be as following (Figure 20) :

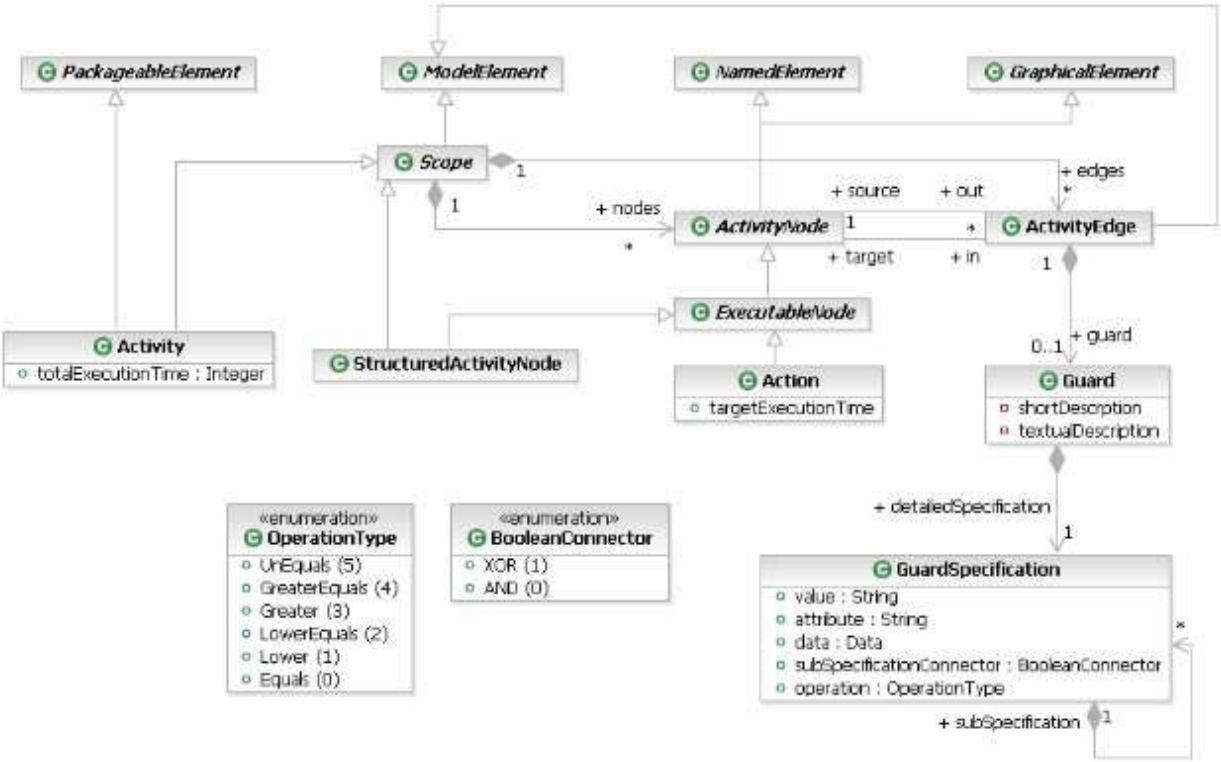


Figure 20: JWT Meta Model

A study comparing the various solutions for Business Process modeling and its relation to the SPEM modeling language from OMG (used for Development Process modeling) will be conducted by IRIT. The main purpose is to define how BP could be used to implement distributed process driven CASE tools. The study will explore the syntactic and semantic differences between the various languages, define model transformations and weaving allowing to relate them, and apply them on the case of some real development process defined in the TOPCASED project relying on the Eclipse Process Framework.

4.3.3 Implementation

Study for establishing a BPMN subset consistent with the requirement of SCOrWare

This study will be done by specifying the SCOrWare requirements in terms of a business oriented representation of business processes. It will be based on the existing studies comparing BPMN to other BP representations and to BP

languages (see part about BPMN to JWT transformation and Annex).

This includes validating that this subset, along with annotations, will be enough to support all features of the SCOrWare BP solution.

The output is a document detailing the validation, along with a commented set of three to five BPMN diagrams representing the different main use cases (ex. BP using "generic" service through specific protocol binding, BPMN using loops) in SCOrWare.

Extending Eclipse STP BPMN to support this subset

The Eclipse STP BPMN editor provides a simple, fully flexible annotation framework. This framework allows to enrich the BPMN diagram (and therefore its XML / EMF serialization) with additional information beyond what BPMN specifies, in the manner of collections of properties and their values. This additional information may typically be "implementation hints" allowing to map the BPMN diagram to a concrete implementation, namely to a "concrete", technical BP language like XPDL or BPEL, and to a concrete execution platform, i.e. BP engine or more generally in our context the SCOrWare platform.

We will extend the Eclipse STP BPMN editor this way to support said subset and every information required for transforming it to a format specific to the SCOrWare platform.

Those annotations will be for example :

on Task Activity,

- actionImplementation
- actionParameters
- actionValues

BPMN to JWT Transformation

See the other parts of this document.

Note : this transformation will be developed using Acceleo.

Since BPMN share with JWT the transition-based process paradigm, such a transformation is quite straightforward once the meaning of BPMN shapes in JWT has been decided and the information required by JWT added as annotations.

This transformation will be included as an Eclipse plug-in in order to be available to the user along the BPMN editor. We will develop this integration.

4.4 JWT technical designer

4.4.1 Objectives

JWT's goal in the context of the SCOrWare platform is to provide a top-down, business-oriented approach to modeling business processes and implementing business tasks using the SCOrWare framework. In a more concrete point of view, this should include providing a high-level process view, providing tooling for building processes using SCOrWare services, providing graphical tooling for the semantic service and process engine...

JWT's larger scope objectives – that is, a unified approach to BPM development tooling and especially design, implementation using a BP language and testing – are in line with the SCOrWare specific objectives, since SOA as a methodology and SCA as a technology are in their own way addressing the same kind of problematics.

4.4.2 Specification

XPDL Capable designer

The aim of the technical designer is to provide a transition-oriented (“workflow”) BP graphical designer compatible with the JWT tool suite.

The target BP format will be XPDL. Compared to the native JWT BP format, it has the advantage of being standardized and as such is an open door to let SCOrWare processes be executed on more “serious” BP engines.

This will be done by Open Wide, with the help of the JWT partners and community.

In year one, we will simply prototype XPDL support within JWT by developing a transformation from the JWT meta model to the XPDL format.

Note : in year two, the specification of the SCOrWare XPDL solution will be updated thanks to the inputs provided by the prototype, either by completing the integrated SCOrWare platform feature set and the integrated engine feature set, or by developing a new editor using the EMF / GMF technology and built on a meta model mutualized with the “business” designer's.

BPEL : Tooling for BPEL developer and users

A BPEL designer is a big endeavor on itself. Moreover, it implies a range of issues that have not been definitely addressed for now, like choosing a preferred BPEL execution platform for SCOrWare, defining the level of SCA / BPEL specification supported by it etc.

Therefore the issue of BPEL tooling will be handled in the second stage (year) of the SCOrWare project, by providing a JWT to BPEL transformation.

4.4.3 Implementation

JWT to XPDL transformation

See the other parts of this document.

In year one, we will develop this transformation using Acceleo.

This transformation will be included as an Eclipse plug-in in order to be available to the user along with the JWT editor. We will develop this integration.

A study comparing the most adequate BP description languages (BPEL, XPDL, JWT, ...) and execution engines for the project will be conducted by IRIT in relation with Open Wide. This study will be supported by prototype transformations between the core concepts of the languages. The key point of the study will concern the implementation in the other languages of the concepts that are available in some languages and not in the others.

5 Test and deployment tools

This Chapter concerns task 2.4. Section 5.1 specifies tools for deployment, Section 5.2 tools for test, and Section 5.3 presents the graphical deployment designer allowing generation of deployment files in FDF or Tunes format.

5.1 Tools for deployment

The tooling for deployment covers several aspects :

- Deployment on a development and integration platform.
- Deployment on a pre-production platform, and deployment on a production platform.

The first type of deployment implies a full integration of the SCOrWare runtime, namely Tinfi embedded in PEtALS, to make it easy to deploy a component on it in order to test it.

This deployment tools needs a very strong integration, both with the PEtALS runtime but also with the JBI Service Engine embedding Tinfi in PEtALS.

It includes the hot deployment of SCA composites on Tinfi/PEtALS (see Figure 21).

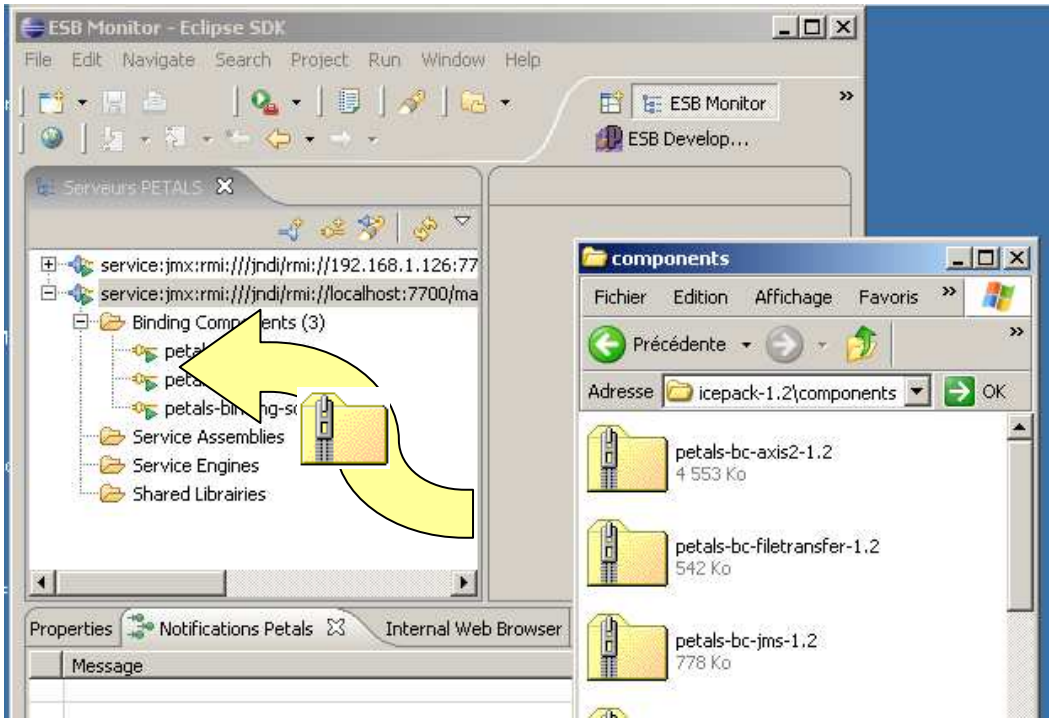


Figure 21: ESB Monitor

Deployment on pre-production platforms will typically be done with tools like FDF.

5.2 Tools for test

Testing components is also very important for developers in Eclipse. Beyond potential adaptations or extensions to usual testing frameworks like JUnit, we want to propose visual tools for testing which introspect the components,

ask the user for values for properties and parameters, and call the component, eventually recording the return values for further comparison.

Such tools exists in WTP in order to test Web Services as it is shown in the picture below (Figure 22). We will develop an identical tool to test SCA components, and extend it functionally to be able to test SCA composites. This last step implies some work to drill-down the composite when a test is recorded so that it is possible to introspect the in and out values of a given component during the test.

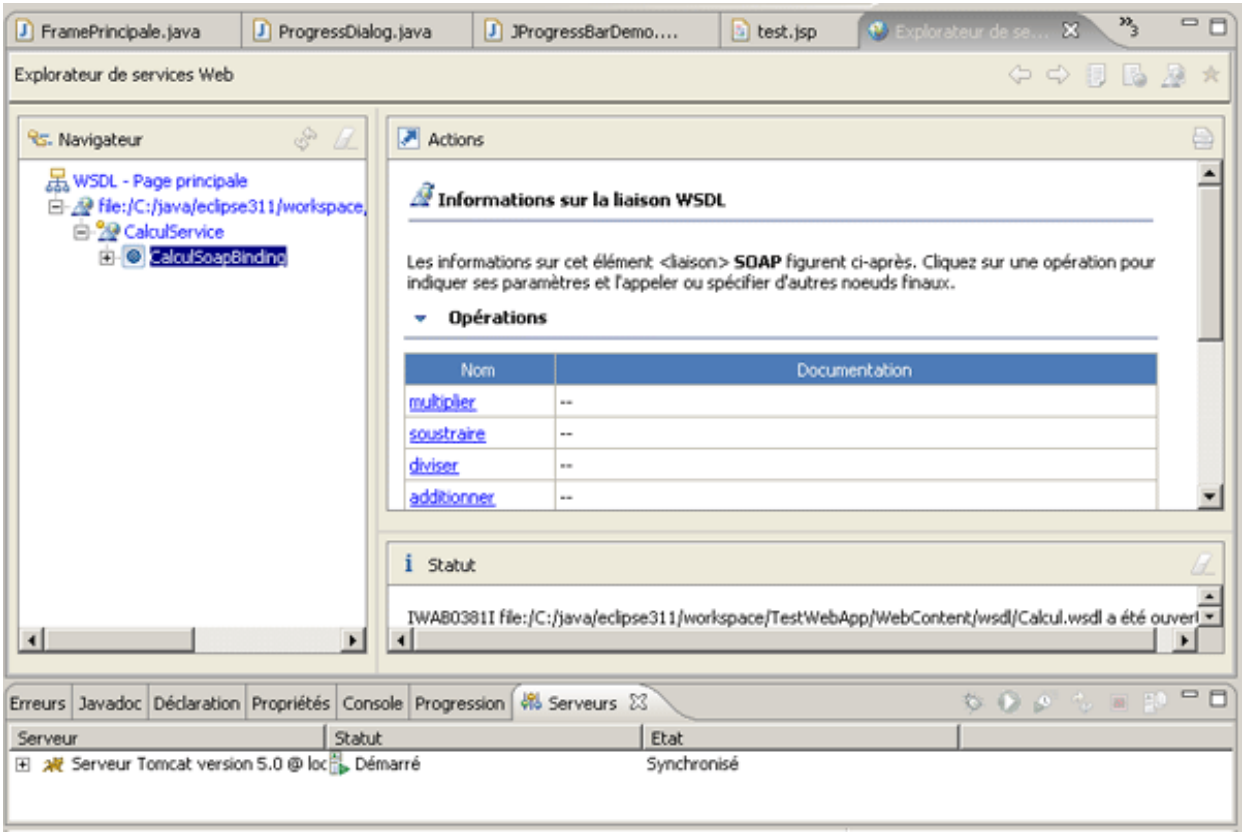


Figure 22: Web Services test with WTP tool

5.3 Deployment designer

5.3.1 Objectives

This tool provides a quick and easy way to construct graphically a “generic” deployment plan of SCA applications. The deployment plans are generic in the meaning that they are independent of any deployment tool. From these plans, this tool generates deployment plans specific to some deployment tools like FDF or Tune (IRIT new version of Jade).

This tool allows :

- To create new generic deployment plans.
- To modify existing generic deployment plans.
- To generate deployment plans for FDF and Tune.

- To import FDF or Tune deployment plans.

5.3.2 Specification

First, we define a meta model for the deployment of software archives, SCA runtime archives and SCA applications on a set of machines. This is the meta model for the generic deployment plans. Second, we propose a graphical representation for the elements of this meta model and functional specifications for the graphical deployment designer.

Deployment meta model

Deployment plans must respect the following meta model (Figure 24 represents graphically this meta model):

- **DocumentRoot**: the root of a deployment plan is composed of
  - One reference to the local repository (LocalRepository) that contains softwares (e.g. JDK) and SCA runtime archives that can be deployed.
  - 1..n references to the machines where SCA applications must be installed (SCAHost).
  - 1..n SCA archives containing one or more SCA composite description and their implementation (SCAPackage).
- **LocalRepository**: the local repository contains
  - 0..n software archives (SoftwareArchive).
  - 1..n SCA runtime archives (RuntimeArchive).
- **SoftwareArchive** (a software archive) and **RuntimeArchive (**a SCA runtime archive) are described by the path and the name of the archive.
- **SCAHost**: describes a machine with information allowing to access it and upload softwares, SCA runtime, and SCA applications. It is composed of
  - The name of the host, the type of transfer, protocol and shell to use.
  - One user (**User**). A user is described by its user name, its password and/or its key.
  - 1..n reference to the SCA runtimes (SCARuntime) that are installed on the host.
  - 1..n reference to the software archives that are installed on the host. For each one the directory where to install it (Software).
- **SCARuntime** (**Software**) represents a particular SCA runtime archive (software archive) installed on a particular host.
- **SCAPackage**: represents an archive containing one or more SCA composite description and their implementation. It is composed of
  - The path and the name of the archive.
  - A reference to the SCA runtime.
  - 1..n references to the Composite that form part of this archive.
- **Composite**: this element extends Composite defined in the SCA specification.
  - composite: the name of the composite file.
  - 0..n references to other Composite. These references represent business dependencies between SCA composites.

Graphical deployment designer

This tool allows to describe the deployment of software archives, SCA runtime archives and SCA applications on a set of machines. Elements of the deployment meta model defined above are represented on follows:

- A local repository is represented by a rectangle. This figure is divided in compartments where each compartment contains a JDK archive or a SCA runtime archive.
- A machine is represented by the figure defined in Figure 23. The name of the machine is on the top of the figure.

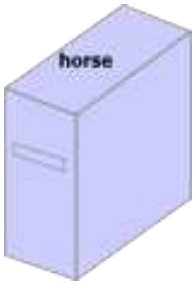


Figure 23: Graphical representation of a machine

- User of a machine is represented by the following figure:  . This figure is on the bottom left of a machine.
- Deployment of SCA runtime are represented by an orange rectangle on the machine representation.
- The Composite is represented in the same way as a Composite in the Composite Graphical Designer (Section 4.1). This designer allows to redefine Service, Reference, and Property of a Composite. A Wire between SCA applications is represented by a black line between service and reference.
- Business deployment dependencies between SCA applications are represented by a red line and a label to show the dependency type.

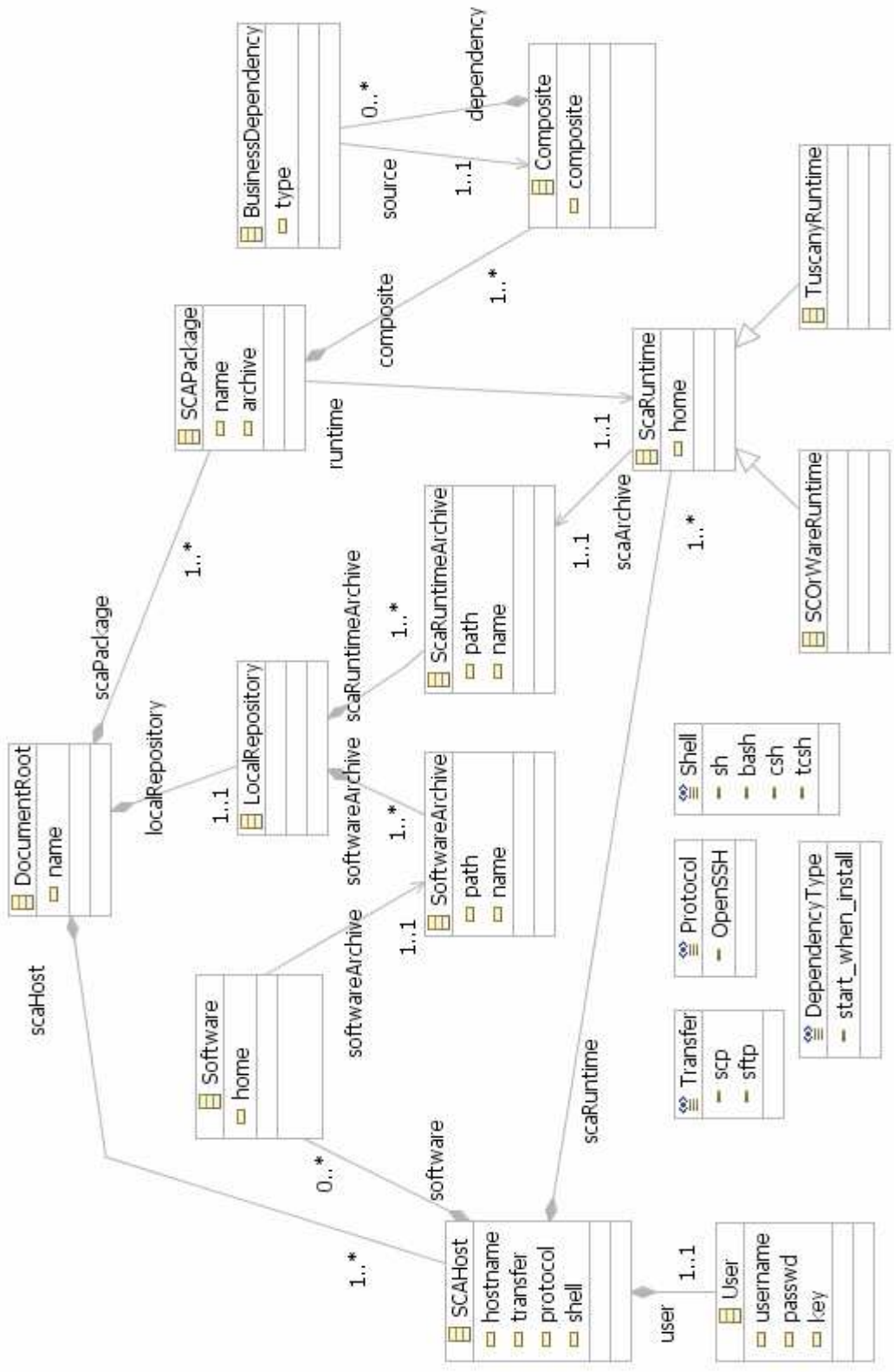


Figure 24: Deployment meta model

5.3.3 Implementation

The deployment graphical designer will be implemented with the GMF framework as shown in Section 4.1.3. The generation of FDF and Tune configuration files will be implemented with Acceleo as shown in Section 3.2.3.

FDF Mapping

The first part of an FDF file defines the system architecture with information allowing to access the machines and upload software. For each host, FDF needs the host name, the user, the transfer method, the protocol, the shell and the list of softwares to install. These informations are in the meta model elements: SCAHost, User, and SoftwareArchive.

The second part of an FDF file contains a description of the SCA runtime. For each host, one or more SCA runtime can be described. For each SCA runtime description, FDF needs the SCA runtime archive path and name, and the home directory where the SCA archive will be installed. These information are in SCAHost, SACRuntimeArchive and SCARuntime meta model elements.

The third part contains the description of the SCA applications.

The figure 25 shows an example of an Acceleo template that generates the description of the hosts in the FDF format.

```
1  # Description of hosts
2  Hosts = INTERNET.NETWORK {
3   <%for (scaSystem){%>
4    <%hostname%> = INTERNET.HOST {
5      hostname = INTERNET.HOSTNAME(<%hostname%>);
6      user     = INTERNET.USER(<%user.username%>,<%user.passwd%>,<%user.key%>);
7      transfer = TRANSFER.<%transfer%>;
8      protocol = PROTOCOL.<%protocol%>;
9      shell    = SHELL.<%shell%>
10     software {
11         java    = JAVA.JRE {
12         archive = JAVA.ARCHIVE(<%java.javaArchive.path%>
                                   <%java.javaArchive.name%>);
13         home    = JAVA.HOME(<%java.installDir%>);
14       }
15     }
16   }
17  <%}%>
18 }
```

Figure 25: Example of Acceleo template for FDF

Tune Mapping

One of the purpose of the SCOrWare project is to establish bridges between the various technologies involved in the project. Tune will therefore rely on some services provided by FDF in order to implement autonomic administration. Currently, Tune relies on UML models in order to express the system architecture and to configure the autonomic components. SCOrWare will allow to improve this solution by relying on dedicated tools both for expressing and modeling the system architecture and deployment.

The DSL and graphical designer will be implemented by Obeo with the help of INRIA (FDF tools) and IRIT (Tune tools).

Figure 26 shows a prototype of the deployment designer developed by Obeo.

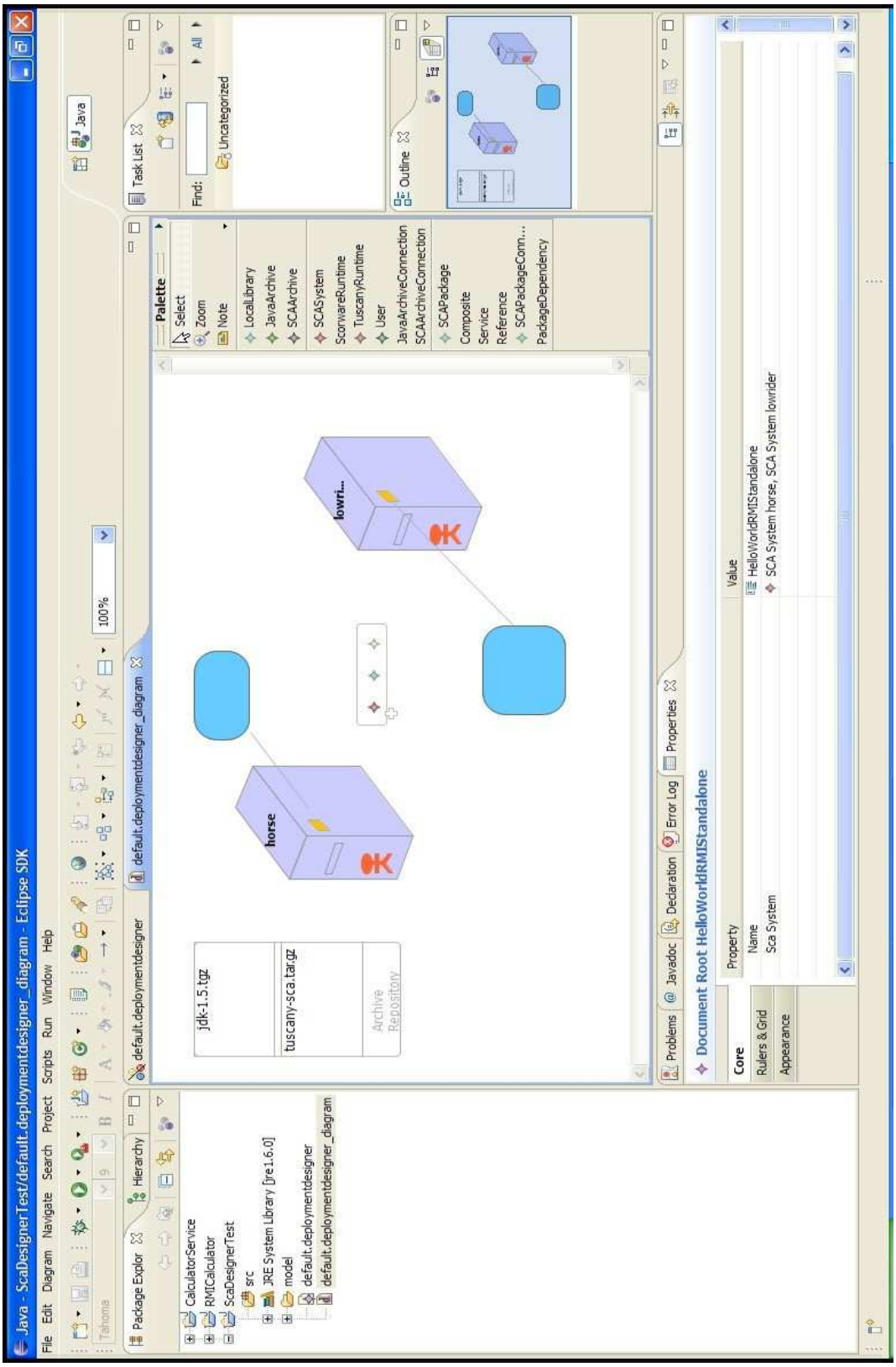


Figure 26: Deployment designer prototype

6 Tools for searching and semantic composition of services

The objective of the task 2.5 is to help the developer and integrators building on the SCOrWare platform to find and better reuse SCA services / components in a consistent manner.

Consistency is maintained thanks to a SCA / SCOrWare service / component registry.

The feature is provided at the same time in the SCA Composite Designer to the SCA integrator and in the JWT Designer to the business process developer.

Note that the workflow and orchestration tool suite is provided by JWT, its designer and the hereby specified additional developments.

6.1 Semantic composition of services using the Trading Service

6.1.1 Objectives

In the near future, services will be present everywhere and can be used by anyone. The diversity of services will allow a user to select the service -among many of the available services- which matches best his requirements and which is adapted most to the resources available to him. Service providers will provide services for a variety of devices having different capabilities. For an end-user, the choice of the best available service will not be so obvious. It is possible that a certain desired service is available to the user but he is not able to use it because it depends on some other services which are not available at the moment. The missing services can be replaced by similar available services, provided by a competitor; however a novice user does not know how to do that. In fact, the application has already been defined in terms of static services, so even if a replacement service is found, it cannot be used. On the other hand, if the user application is capable of replacing the missing service by another one dynamically, user need not to even aware of presence or absence of any services.

Such an application needs to be described in terms of abstract services which will be resolved into concrete services at the time of execution of the application, i.e. the service descriptions should be independent of their implementations.

Our Approach

The motivation behind our work is to propose a model for creating such applications whose composition, in terms of services, is static only at an abstract level and the exact concrete composition is determined at runtime. In this regard, we propose an infrastructure for dynamic composition of applications based on abstract service descriptions.

We would also like to mention that our contribution is not about proposing dynamic deployment of services nor do we plan to provide anything related to dynamic composition or orchestration of services. We assume that the application has already been defined in terms of the SCA composites, i.e. a composition of abstract services; however, we need to ensure the composition of application in terms of such concrete (SCA) services that best match the semantics of the abstract services. We also assume that the concrete services are already available during service selection process.

6.1.2 Specification

Application Description

- **Abstract Composite Description**

In SCA, composites encapsulate services. A service offered by a composite is denoted by the service element in the composite. A required service is denoted by the reference element. To be able to reason about the services, we have to annotate services with reference to a semantic model, such as OWL or UML. This choice is motivated by the fact

that applications developers can use any ontology language to annotate services (such as UML or OWL) unlike in OWL-S or WSMO. Using SASCDL (see semantic trading service description, task 1.5), we suggest how to add semantic annotations to various parts of a SCDL document like services and their properties, components and their properties, and references. SASCDL defines a namespace called *sawsdl* and an extension attribute called *modelReference* so that relationships between SCA components and concepts in another semantic model are handled.

Figure 27 below shows the description of an abstract SCA composite. It only contains the service and component description as well as their properties. It also shows the semantic annotations added to each element.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite
   xmlns="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
   xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
   name="bigbank.account">

   <service name="AccountService"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Account">
      <reference>AccountServiceComponent</reference>
   </service>

   <property name="currency" type="xsd:string"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Currency">
   USD
   </property>

   <component name="AccountServiceComponent"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Account”>

      <property name="currency" source="$currency"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Currency"/>

      <reference name="accountDataService"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountData">
      AccountLoggerDataServiceComponent
      </reference>
      <reference name="stockQuoteService"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#StockQuote">
      StockQuoteServiceComponent
      </reference>
   </component>

   <component name="AccountLoggerDataServiceComponent"
      sawsdl:modelReference="http://www.int-
edu.eu/sawsdl/ontology/bigbank#AccountLoggerData">

      <reference name="accountDataService"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountData">
      AccountDataServiceComponent
      </reference>
      <reference name="accountLoggerService"
         sawsdl:modelReference="http://www.int-
edu.eu/sawsdl/ontology/bigbank#AccountLogger">
      AccountLoggerServiceComponent
      </reference>
   </component>

   <component name="AccountDataServiceComponent"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountData"/>
   <component name="AccountLoggerServiceComponent"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountLogger"/>
   <component name="StockQuoteServiceComponent"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#StockQuote"/>
</composite>
```

Figure 27: Abstract composite description

- **Concrete Composite Description**

The abstract composite description only specifies the services provided or required by it and does not show any detail on how the service is implemented, how to invoke it and which protocol should be used for this purpose. This essential information is provided in the concrete composite descriptions. A concrete composite description has the same set of services as defined in the abstract composite description and, in addition, also specifies the implementation details such as interfaces, classes, bindings and endpoints. Figure 2 shows the concrete composite description for the composite in figure 28.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite
   xmlns="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
   xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
   name="bigbank.account">
   <service name="AccountService"
      sawsdl:modelReference="http://www.intedu.eu/sawsdl/ontology/bigbank#Account">
      <interface.wsdl
         interface="http://www.bigbank.com/account#wsdl.interface(AccountService)"
         wsdli:wsdlLocation="http://www.bigbank.com/account wsdl/AccountService.wsdl" />
      <binding.ws
    endpoint="http://www.bigbank.com/account#wsdl.endpoint(AccountService/AccountServiceSOAP)"
         conformanceURIs="http://ws-i.org/profiles/basic/1.1"
         location="wsdl/AccountService.wsdl"/>
      <reference>AccountServiceComponent
       </reference>
   </service>
   <property name="currency" type="xsd:string"
       sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Currency">USD
    </property>
   <component name="AccountServiceComponent"
      sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Account”>
      <implementation.java
         class="bigbank.account.services.account.AccountServiceImpl" />
      <property name="currency" source="$currency"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#Currency"/>
      <reference name="accountDataService"
         sawsdl:modelReference="http://www.intedu.eu/sawsdl/ontology/bigbank#AccountData">
      AccountLoggerDataServiceComponent
      </reference>
      <reference name="stockQuoteService"
         sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#StockQuote">
       StockQuoteServiceComponent
       </reference>
    </component>
    <component name="AccountLoggerDataServiceComponent"
       sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountLoggerData">
       <implementation.java
          class="bigbank.account.services.accountlogger.AccountLoggerDataServiceImpl"/>
       <reference name="accountDataService"
          sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountData">
       AccountDataServiceComponent
       </reference>
       <reference name="accountLoggerService"
          sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountLogger">
       AccountLoggerServiceComponent
       </reference>
    </component>
    <component name="AccountDataServiceComponent"
       sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountData">
       <implementation.java
          class="bigbank.account.services.accountdata.AccountDataServiceDASImpl"/>
    </component>
    <component name="AccountLoggerServiceComponent"
       sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#AccountLogger">
       <implementation.java
          class="bigbank.account.services.accountlogger.AccountLoggerServiceImpl"/>
    </component>
    <component name="StockQuoteServiceComponent"
       sawsdl:modelReference="http://www.int-edu.eu/sawsdl/ontology/bigbank#StockQuote">
       <implementation.java
          class="bigbank.account.services.stockquote.StockQuoteServiceImpl"/>
    </component>
</composite>
```

Figure 28: Concrete composite description

6.1.3 Architecture

The basic architecture of our framework is shown in Figure 29. The Application Builder's task is to take the abstract application description and generate the corresponding concrete application. It needs assistance of the Trading Service for selection of the component implementation to be included in the output application. The implementations (concrete services) are present in service repository.

Service Selection: The Application Builder launches the service selection process by acquiring all the components specified in the composite description and requests the Trading Service to find a corresponding implementation for each one of them. This has been shown in Algorithm 1. The Trading Service first iterates over all the component descriptions and uses each component's *key* to find the corresponding components with implementations. Since more than one component implementation may exist, at the moment we only choose the first one among the list of returned components. The implementation of this component is an SCDL fragment which is then added to the abstract component description. Once implementations are found for all the components, the Trading Service returns the concrete composite to the Application Builder.

Algorithm 1

```
ResolveComposite( AbstCompositeDescription )
Begin
   //initialize concrete components same as abstract components
   //then find their implementations
   ConcCompositeDescription = AbstCompositionDescription;
   ComponentList = GetAllComponents(ConcCompositeDescription);

   For each Component_i in ComponentList
      Key = key (component_i);
      Components = findComponents(key); //Find matching components
      Component = Components(1); //Select the first one
      Implementation = GetComponentDetails(Component);
      //add implementation to make it concrete component
      AddImplementation(Component_i, Implementation)
   End for

   // return the concrete component list
   Return ConcCompositeDescription;
End
```

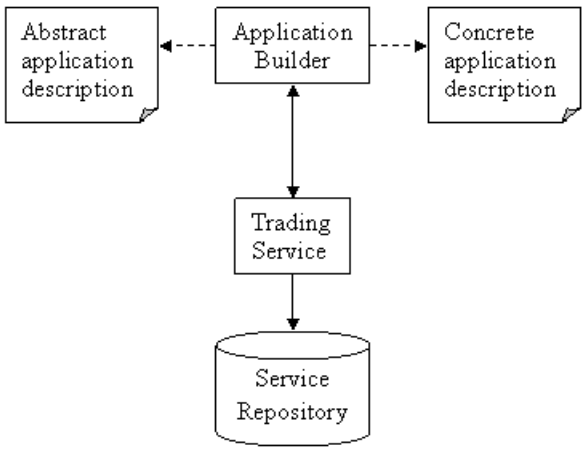


Figure 29: Architecture

6.2 Integration as a usable, consistent feature set within the SCOrWare development tools

6.2.1 Objectives

The aim is to achieve :

- Integration within the SCA Composite Designer.
- Integration within the JWT Designer.

It will be developed as an Eclipse plug-in to the JWT designer and will use the SCA / SCOrWare service registry and service semantic provider.

6.2.2 Specification

Integration within the SCA Composite Designer.
This will be done by Obeo with the help of the INT, IRIT and Open Wide. The aim is to help develop SCA composites using SCA / SCOrWare services, by

1. Allowing to graphically search SCA / SCOrWare services corresponding to a certain set of semantic criteria and
2. Allowing SCA composites to graphically integrate and configure the semantic service courtage service, so as to semantically define a composition of services to call.

This will be done on the basis of the common, mutualized work described below.

Integration within the JWT Designer.
This will be done by Open Wide with the help of the INT and IRIT. The aim is to help develop SCA composites using SCA / SCOrWare services, by

1. Allowing to graphically search SCA / SCOrWare services corresponding to a certain set of semantic criteria and
2. Allowing SCA composites to graphically integrate and configure the semantic service courtage service, so as to semantically define a composition of services to call.

SCA / SCOrWare service semantic provider
This is the service developed by the INT in lot 1.

SCA / SCOrWare service registry implementation

The SCA / SCOrWare service semantic provider features semantic search algorithm but requires to be implemented on top of a service registry implementation. The aim of such a service registry implementation is to store and provide SCOrWare services and their semantic annotations, i.e. Concretely SASCDL files (SCDL files along with the WSDL and SAWSDL files it needs).

This registry is useful

1. at development time to maintain consistency across the set of service used in an SCA architecture, by letting the development tools (like JWT or SCA Designer using the SCA component semantic search interface) peek in it
2. at runtime but only for the features that are using it, like the service semantic provider.

It will be developed by Open Wide, INT, and IRIT using Open Source components allowing to manage component and their services' definitions and standards like UDDI, EbXML, Web Services.

One implementation alternative studied by the INT is a UDDI compliant registry, based on the Open Source UDDI implementation Juddi, using a database storing tables whose design allows to store SCA components, and SOAP web services as a remote interface.

Another implementation alternative studied by Open Wide would be an EbXML compliant registry, based on Sun's reference implementation omar / ebxmlrr, using the standard EbXML database but an SCA-specific ebXML ontology to describe the document types and relations required to store and ably serve SCDL, WSDL and / or SAWSDL files. This alternative would be more powerful, notably in terms of organizing and classifying service and components, but still needs to be reviewed in order to answer questions like : would this solution perform well on semantic service queries, for example that feature criteria on SCA components as well as on their services, even though they are defined in different files ?

A command-line based basic administration tool is required, in order to be able to fill or remove components from the registry. It will be developed in Java and ant.

Note that at first a trivial implementation (memory-stored registry) would be enough to be able to integrate the rest of the solution. This is what will be done in first year.

SCA / SCOrWare service semantic provider / registry administration interface

This interface will be developed as an Eclipse plug-in.

It will feature a view integrating the semantic browsing and search interface described below.

It has been developed in lot 1 by INT.

Note : this plug-in will only be developed in second year. In first year, the batch loading (command line based) mechanism will be used instead and will be enough.

SCA / SCOrWare service semantic provider integration as SCA component

This allows to easily use an SCA component wrapping the SCA / SCOrWare service service semantic provider and allowing to use it at runtime to call a development time semantically described, runtime chosen service. It has been developed by the INT in lot 1.

SCA / SCOrWare service semantic provider SCA component interface

The SCA Designer allows to use and configure said SCA component and the service semantic provider service it wraps.

This interface will be developed jointly by Obeo, INT, IRIT, and Open Wide and used as well in the SCA Editor as in the JWT Editor.

SCA / SCOrWare service semantic search interface

This will be developed as an Eclipse EMF/GEF/GMF plug-in.

It will feature graphical panels and fields allowing to change the semantic search engine configuration.

This feature will be provided as a dialog including both

- an Eclipse view allowing to “browse” the registry (list all SCA services and / or components), by using the semantic service provider in a basic manner to provide an all-around useful feature
- and an Eclipse view providing full featured SCA / SCOrWare service semantic search, i.e. Providing fields and combo boxes for all search criteria the search algorithm allows.

It may be used in two ways :

- first provide or select the service definition the target service should comply with, as well as data mapping information (SCDL – WSDL - SAWSDL) ; then add implementation information about which

service will actually be targeted i.e. which SCA component from the defined architecture provides it. In this case there is no runtime decision made about which service implementation (component) to use since the decision has already been made at development time.

- or provide or select the service definition the target service should comply with, as well as data mapping information, and all manner of semantic annotations describing what kind of implementation should be find and used at runtime (SCDL – WSDL – SAWSDL). This alternative actually consists in configuring the runtime SCA component that wraps the semantic service provider with this information.

Integration of service semantic search interface in the SCA Designer

The SCA Designer tool set will allow to include an SCA component whose definition is taken from the registry, ex. Insert > SCA Component, in a feature similar to File > Open.

This feature will be provided as a dialog including both an Eclipse view allowing to browse the registry (list all SCA services and / or components) and the Eclipse view defined above (SCA / SCOrWare service semantic search interface) which allows to do a semantically configured search.

The administration interface mentioned above will be integrated as well.

See more in the SCA Designer part.

Integration of service semantic search interface in the JWT Designer

The JWT Process Designer will allow to choose an SCA service provided by an SCA component to be the target of a JWT Action. The definition of this SCA Service and its call may be provided explicitly or taken from the SCA / SCOrWare service registry, so as to belong to a consistent SCA architecture.

Both browsing and semantic search views will be integrated as an Eclipse plug-in so as to be available in the JWT Designer, and from there specify SCA service / components as an action's implementation thanks to SCA / JWT integration (see next chapter).

SASCDL Editor

This editor would complete the registry feature set.

Having in mind a full featured solution, a diagram interface could be developed using EMF GMF technologies.

However it is not a priority and is not funded, so for now it will be provided as a simple typed XML editor.

Depending on the other implementation choices, a less costly alternative will be to allow to edit semantic information as annotations on the BPMN or SCA designer.

6.2.3 Implementation

Two kind of solution can be used in the semantic trading of services for deciding if a given service fits the requirements of the user: general solutions relying on existing WEB SEMANTIQUE technologies which can be used in any context but are limited in their expressiveness; and domain dedicated solutions which can use the specific semantic properties of the application domain in order to have sophisticated description of the services and comparison procedures.

IRIT will adapt in SCOrWare the scientific computing dedicated semantic service trader that have been developed in the TLSE project. IRIT will rely on SCOrWare semantic description formalism in order to define the properties required for its trading technology and will adapt its trading algorithm to use this new description and provide the results which will be used in the graphical wizards.

7 Choreography of process that use generic services

The objective is a workflow and orchestration solution (tooling and runtime) that is able to work with generic services and flexible enough to meet development time and runtime provided requirements (see also task 3.4).

Orchestration is to be provided through a workflow / business process engine. Service genericity is to be achieved through supporting interaction with SCA services / components and bindings. Flexibility is helped at development time by semantic process search and allowed at runtime by the semantic process matching service.

The orchestration tool suite is provided by JWT, its designer and the hereby specified additional developments.

7.1 JWT / SCOrWare process semantic search feature

7.1.1 Objectives

The aim will be to achieve integration as a usable, consistent feature set within the SCOrWare development tools. In order to do so, we will integrate semantic process choreography within the JWT Designer.

7.1.2 Specification

Introduction
This will be done by Open Wide with the help of the INT. The aim is to help develop SCA composites using SCA / SCOrWare services, by

1. Allowing to graphically search SCA / SCOrWare services corresponding to a certain set of semantic criteria and
2. Allowing SCA composites to graphically integrate and configure the semantic service courtage service, so as to semantically define a composition of services to call.

It will be developed as an Eclipse plug-in to the JWT designer and will use the JWT / SCOrWare process registry and process semantic provider (matcher ?).

JWT / SCOrWare process semantic registry
Its aim is to store and provide JWT / SCOrWare processes and their semantic annotations.

This registry is useful at development time to maintain consistency across the set of processes used in an JWT / SCA architecture, by letting the development tools (typically the JWT Designer) peek in it.

It will be developed by Open Wide and the INT using Open Source components allowing to manage process definitions and standards like UDDI, EbXML, Web Services, extending the similar work done for SCA component / service registry.

Note that the choice of EbXML omar / ebxmlrr technologies for both registries would allow to build more easily on the work done for the semantic service registry, and from there extend it at will.

A command-line based basic administration tool is required, in order to be able to fill or remove components from the registry. It will be developed in Java and ant.

Note that at first a trivial implementation (memory-stored registry) would be enough to be able to integrate the rest of the solution. This is what will be done in first year.

JWT / SCOrWare process semantic provider
This service is provided by the process semantic registry.

SCA / SCOrWare process semantic search interface
This will be developed as an Eclipse EMF GEF GMF plug-in.

This interface must be developed so as to display all fields and actions required by the process semantic search engine.

This feature will be provided as a dialog including both

- An Eclipse view allowing to "browse" the process registry (list all JWT processes), by using the semantic service provider in a basic manner to provide an all-around useful feature
- And an Eclipse view providing full featured SCA / SCOrWare process semantic search, i.e. Providing fields and combo boxes for all search criteria the search algorithm allows. This Eclipse view allows to choose a process among those returned by the search algorithm (they are displayed by a view similar to the browse feature's), and this sets the process as a subprocess within the selected JWT Activity.

Integration of process semantic search interface in the JWT Designer
This interface must be integrated as a wizard action in the JWT "technical" designer.

The JWT Process Designer will allow to choose an existing JWT process to become a subprocess within a given JWT Activity . The definition of this JWT process may be provided explicitly or taken from the SCA / SCOrWare service registry, so as to belong to a consistent SCA architecture.

Both process browsing and semantic search views will be integrated as an Eclipse plug-in so as to be available in the JWT Designer, and from there specify a JWT process as an action's implementation thanks to SCA / JWT integration.

7.1.3 Implementation

Interface definition and simple implementation of JWT / SCOrWare process semantic registry
INT and Open Wide will define the Java interfaces and develop a simple implementation of the process semantic registry. In year one, there will be a memory and file based trivial implementation in order to validate the architecture.
In year two, based on the results of the year one prototyping and the choice made for the service registry implementation, we may use the omar /ebxmlrr off-the-shelf EbXML as an implementation foundation.

SCA / SCOrWare process semantic search interface
Open Wide with the help of the INT will develop a "semantic browsing" Eclipse view allowing to navigate in the registry, its components and their services, and a "semantic search" Eclipse view allowing to enter semantic search parameters, execute a semantic search in the registry using the previously defined interfaces and displaying its results in a table.

Both will allow to select one component and / or service.

This will be developed as an Eclipse EMF GEF GMF plug-in.

Integration of process semantic search interface in the JWT Designer
A wizard interface will be developed by Open Wide around the process semantic search interface. This wizard will be made available in the JWT Editor in order to specify a process to be used as subprocess in a JWT Activity.

7.2 "Generic" SCA / SCOrWare service integration in BP solution

7.2.1 Objectives

The aim is to provide tooling for "generic" services, that is services beyond "mere" web services, thanks to the cross platform and cross language capacities of SCA and especially SCA protocol bindings.

This "Generic" service tooling is achieved through the SCA component and binding integration within JWT (design and execution, as seen above) as well as the standard BP engine.

This "Generic" service tooling will thus provide an extensible mechanism for supporting JWT Actions implementations beyond web service calls, e.g. generic connectors to XML over several layer (JMS, HTTP, ...), EDI, ...

7.2.2 Specifications

SCA / SCOrWare service integration in JWT process definition and modeling
This will be done by Open Wide in the context of the SCOrWare project, with the help of the JWT community.

It will consist in :

- An SCA service / component service provider and its registry implementation. This is actually the semantic service provider, enriched by a set of utilities enabling getting the components in their EMF model from the SCDL description said service returns, and introspecting them to get the service that was initially looked for. It will integrate in JWT in a generic manner, as one kind among others of Action implementation providers.
- A generic extension of the JWT model so as to be able to manage SCA / SCOrWare service call as a JWT Action. This is done by 1. defining a generic way to extend the "standard" JWT Action and provide the definition of a specific Action implementation ; 2. develop a specific JWT Action model extension supporting SCA / SCOrWare service calls, using the SCA / SCOrWare service / component EMF model to specify them.
- An administration interface allowing to define, display and save target SCA services (provide WSDL) and their optional semantic annotations (provide SAWSDL), as well as specify the data binding. This is actually the interface that has been defined in the previous chapter. Note : it has to bee seen whether this interface and its underlaying service may be mutualized at some level with the work done on the BindingFactory in lot 1, especially by IRIT.
- An interface allowing to browse and semantically search SCA services / components in the registry. This is actually the interface that has been defined in the previous chapter.

NB. It will be implemented, if possible, as a two step functionality : first, provide or select the service definition the target service should comply with, as well as data mapping information ; then add implementation information about which service will actually be targeted i.e. which SCA component from the defined architecture provide it.

SCA / SCOrWare service integration in JWT process execution engine
This will be done by Open Wide in the context of the SCOrWare project, with the help of the JWT community.

JWT processes will be able to call SCA services.

Conversely SCA services will be able to call JWT processes, and react on JWT events.

In year one, we will simply develop support for calling SCA services within JWT process execution.

Tooling for SCA protocol bindings
Specific tooling will be developed by Amadeus with the help of Open Wide that will allow the developer to graphically choose, set and configure a chosen set of specific protocol binding implementations.

About Data binding

SCA drives a line between "protocols" and "data binding". We note that data binding is first and foremost and integration issue, and that the choices of SCOrWare and also more globally of the markets of SCA, ESB, JBI and service-oriented integration are targeted toward XML-compatible data binding.

Such data binding that would be of interest in SCOrWare and for generic services encompass bare XML as well as

DOM, SDO, JAXB (meaning it allows generic as well as domain specific models), thanks to various XML transformation and marshaling technologies. The Apache Tuscany project provides a collection of such data bindings, including SDO, JAXB and XMLBeans.

Therefore we will restrain to the support of one XML-compatible data binding for each protocol, since others may be made available through the use of available Tuscany SCA Databindings or “classic” XML manipulation.

Selecting target protocols

The aim of this selection is offering the greatest coverage in terms of protocol features, so there is enough supported protocols to support any set of requirements. For example, some use cases may require great transport performances, whereas others may prefer reliability or openness.

Protocols preferred by SCOrWare will here be a primary choice. These are web services and JBI, as well as EJB compatibility.

Web services provide an unrivaled compatibility potential.

JBI is especially powerful, since it may itself bridge toward an unlimited set of supported protocols. Therefore it may itself support and kind of requirement that the chosen SCA / JBI integration at least supports. This last point is illustrated by the fact that a high performance protocol made available by a JBI bridge may be interesting if we're using a “native” (local memory) SCA-JBI integration, but that it would be less interesting if we'd be using a lower performance Web service-based SCA-JBI integration.

JMS provides versatility since it covers a large number of use cases, and at the same time reliability.

A peculiar category of requirements that would be of great interest to support is business data format and protocols, like EDI. Supporting those is as much a protocol issue as a data format issue. This could be done using “classic” XML – EDI integration and web service protocol, or using a JBI – EDI integration. Since the EDI format is business- and use case- dependent, we will only provide here generic tooling for those. This will be done in year two by Amadeus with the help of Open Wide.

About protocol binding implementations

SCOrWare specific protocol binding implementations will be chosen in priority.

The Apache Tuscany project provides a set of additional, interesting protocol binding implementations. These includes Web Service, EJB and JMS.

JBI support (interesting, see above) could be provided by the Petals ESB that the SCOrWare partner EBM WebSourcing develops.

The selected protocol bindings and implementations

We select Web Service / SCOrWare and JMS / Tuscany because the complement well each other, as well as JBI as a second choice because of the ways it opens. We also select EDI / XML as business-oriented format and protocol of choice.

For these, we will develop graphical tooling (see below) that will be integrated into the workflow tool suite as an Eclipse plug-in.

Specification of a protocol binding implementation configuration's integration in the JWT meta model

A protocol binding implementation's configuration is modeled in the JWT meta model as a specific extension of the JWT Action concept, as described above. Therefore it reuses their SCA / SCOrWare EMF meta model.

Specification of a protocol binding implementation's graphical tooling

A protocol binding implementation's graphical tooling consists in :

- A graphical interface allowing to display and edit the above mentioned protocol binding implementation configuration's meta model.
- An integration of this one in the JWT tooling suite and / or the SCA editor.

7.2.3 Implementation

Tools for the semantic service provider (registry)

In year one, we will develop tools allowing to load and facilitating introspection of the component and service model from the registry. A simple JWT Action implementation provider wrapper will also be developed for the registry.

Action extensibility meta model

In year one, we will define a simple way to extend the JWT meta model to specify specific Action implementations and their required implementation information.

In order to achieve this goal, we will build on the features of the EMF meta model framework.

SCA / SCOrWare - JWT Runtime integration

This will be done by Open Wide in the context of the SCOrWare project, with the help of the JWT community.

JWT processes will be able to call SCA services.

Conversely SCA services will be able to call JWT processes, and react on JWT events.

In order to have maximum efficiency, we will use the RMI protocol to let the JWT engine and the SCOrWare platform communicate.

In year one, we will simply develop support for calling SCA services within JWT process execution.

Web service protocol binding implementation configuration tooling

We will develop the JWT meta model extension that specifies the Web Service binding configuration.

We will develop its companion Eclipse UI and integrate it in JWT.

JMS protocol binding implementation configuration tooling

This will be done in year two.

EDI configuration generic utilities

This will be done in year two.

Other protocol binding implementation configuration tooling

This may be done in year two.

Other protocols may be supported, or the existing ones better integrated.

7.3 Model transformations

7.3.1 Objectives

This tool generates BPMN code from JWT code, and JWT code from BPMN code.

7.3.2 Specification

The JWT meta model is presented Figure 20 and the BPMN meta model in Section 3.2.2.

The following table depicts the mapping that we propose between the main concepts of BPMN and JWT. To define this mapping, we adopt the most relevant aspects for a comparison of Business Process Meta models proposed in [20]. This article presents a comparison between a simple Business Process Meta model, BPBM (Business Process Definition Meta model), EPC (Event-driven Process Chains), List/Korherr, UML2 Activity Diagram and JWT.

Concept	**BPMN**	**JWT**
General concepts		
Process	Business Process Diagram	Activity
Process behavior	?	No distinction
Link to another process	Collapsed Sub Process Expended Sub Process	Activity Link Node
Included process	Collapsed Sub Process Expended Sub Process	Structured Activity Node
Group	?	Group
Activity	Task Activity	Action
Transition	Flow ?	Activity Edge
Guard on transition	Event ?	Guard/Guard Specification
Loops	Activity Looping Sequence Flow Looping Multiple Instances	-
Control nodes		
Process start	Start Event Start Message Event Start Timer Event Start Rule Event Start Link Event Start Multiple Event	Initial Node
Process finish	End Event End Message Event End Error Event End Compensation Event End Link Event End Terminate Event End Multiple Event	Final Node
Process flow abort	Intermediate Cancel Event End Cancel Event	No distinction
XOR split	Data Based Gateway	Decision Node

XOR join	Data Based Gateway	Merge Node
AND split	Parallel Gateway	Fork Node
AND join	Parallel Gateway	Join Node
OR split	Inclusive Gateway	-
OR join	Inclusive Gateway	-
IOPE		
Input data	Event Message Flow	Data
Output data	Event Message Flow	Data
Events		
Event	Event	Event
Message event	Start Message Event Intermediate Message Event End Message Event	Message Event
Timer event	Start Timer Event Intermediate Timer Event	Timer Event
Rule event	Start Rule Event Intermediate Rule Event	-
Link event	Start Link Event Intermediate Link Event End Link Event	-
Multiple event	Start Multiple Event Intermediate Multiple Event End Multiple Event	-
Compensate event	Compensation Association Association Intermediate Compensation Event End Compensation Event	-
Error event	Intermediate Error Event End Error Event	-
Business specific		
References	-	Reference
Business function	-	Function
Role	Pool Lane	Role
Organization	-	Organization Unit
Application	-	Application
Parameter	-	Parameter
Interactions		
Interaction	?	-
Message channel	Event	-
Interaction role	Pool (when the pool is used as a	-

	black box)	
Flow binding	Input Maps attribute of Sub Process Output Maps attribute of Sub Process	-

Figure 30: BPMN/JWT Mapping

7.3.3 Implementation

Like in Section 3.2.3, Acceleo is used to generate BPMN from JWT and JWT from BPMN. The mapping defined in the section above will be implemented as an Acceleo template.

8 Integration

This chapter is related to the Task 2.7, and is dedicated to the integration work.

This chapter will be completed in the next version of this document when prototypes will be available and integrated together.

9 Conclusion

This document specifies tools developed by the different SCOrWare partners. Some of these tools do not aim a particular SCA runtime but complete existing tools of the STP project:

- SCA Component Creation Tool (Section 3.1.2),
- SCA Assembly Editor (Section 3.1.3),
- SCA Annotation Editor (Section 3.1.4),
- BPMN2BPEL (Section 3.2),
- SCA Composite Designer (Section 4.1),
- Tools for deployment (Section 5.1),
- Tools for test (Section 5.2).

The other tools are specific to SCOrWare and allow to better cover SOA needs:

- JWT Business Designer (Section 4.3),
- JWT Technical Designer (Section 4.4),
- Tools for Searching and Semantic Composition of Services (Section 6),
- Transformation between JWT and BPMN (Section 7.3).

All these tools simplify the work of architects, analysts, and developers by automating tedious tasks at different steps of the SCA applications construction. Figure 31 shows an overview of different roles and SCOrWare tools involved in the development of SCA applications.

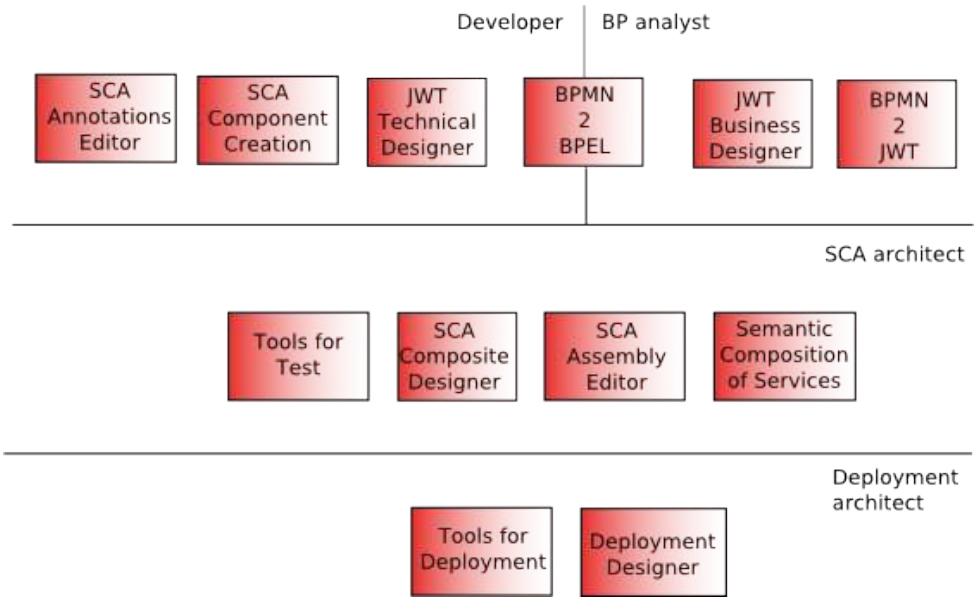


Figure 31: Roles and tools involved in SCA applications development

10 References

1: OMG, OMG Model Driven Architecture, http://www.omg.org/mda/

2: Wikipedia, Domain-specific programming language, http://en.wikipedia.org/wiki/Domain-specific_programming_language

3: The Eclipse Foundation - STP Project, SOA Tools Platform Project (STP), http://www.eclipse.org/stp/

4: , Apache CXF: An Open Source Service Framework, http://incubator.apache.org/cxf/

5: Philippe Merle, SCOrWare Project - SCA Platform Specifications - Version 1.0, 2007

6: The Eclipse Foundation - EMF Project, Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf/

7: Open Service Oriented Architecture collaboration, SCA Assembly Model Specification V1.00, 2007

8: The Eclipse Foundation - EMF Project, The Eclipse Modeling Framework (EMF) Overview, http://dev.eclipse.org/viewcvs/indextools.cgi/org.eclipse.emf/doc/org.eclipse.emf.doc/references/overview/EMF.html

9: The Eclipse Foundation - EMF Project, The EMF.Edit Framework Overview, http://dev.eclipse.org/viewcvs/indextools.cgi/org.eclipse.emf/doc/org.eclipse.emf.doc/references/overview/EMF.Edit.html

10: The Eclipse Foundation - EMF Validation Sub Project, Eclipse Modeling Development Tools subcomponent Validation, http://www.eclipse.org/modeling/emf/?project=validation#validation

11: Object Management Group/Business Process Management Initiative, Business Process Modeling Notation (BPMN) Information, http://www.bpmn.org/

12: Business Process Execution Language for Web Services version 1.1, Business Process Execution Language for Web Services version 1.1, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

13: Jordan Diane and Evdemon John, Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/

14: Obeo, Acceleo, http://www.acceleo.org

15: The Eclipse Foundation - STP Project, STP BPMN Modeler: BPMN object model, http://www.eclipse.org/stp/bpmn/model/index.php

16: Object Management Group, Business Process Modeling Notation (BPMN) Specification, 2004

17: Ouyang, Chun and van der Aalst, Wil M.P. and Dumas, Marlon and ter Hofstede, Arthur H.M., From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way, 2006

18: The Eclipse Foundation - M2T Project, JET, http://www.eclipse.org/modeling/m2t/?project=jet

19: The Eclipse Foundation - GMF Project, Eclipse Graphical Modeling Framework (GMF) , http://www.eclipse.org/gmf/

20: Florian Lautenbacher, Comparison of Business Process Metamodels, 2007