REuse and Migration of legacy applications to Interoperable Cloud Services
REMICS
Small or Medium-scale Focused Research Project (STREP)
Project No. 257793

Deliverable D6.1

Verification, Testing and Models@Runtime Toolkit, Preliminary Release

Work Package 6

Leading partner: Fraunhofer

Author(s): Christian Hein, Franck Barbier, Tom Ritter

Dissemination level: Public

Delivery Date: 22nd September 2011

Final Version: 1.0

Versioning and contribution history

Version	Description	Contributors
0.1	Initial document	Christian Hein
0.2	Included section on Models@runtime	Franck Barbier
0.3	Put description text to all sections	Christian Hein
0.4	Internal review	Arnstein By, Antonin Abhervé
0.5	integration internal reviewer comments	Christian Hein
1.0	Final version	

Executive Summary

This deliverable represents a preliminary release of the toolkit demonstrating feasibility of the approaches of verification, testing and Models@Runtime. The scope of verification was limited to the aspect of static analysis of models based on model metrics. This toolkit preliminary release consists of the three parts including the following features:

- Metrino – Is a static analysis workbench for MOF-based models.
  - Implementation of Software Metric Metamodel (SMM) available in the toolkit with extensions
  - Initial set of metrics for the recovered and migrated models
  - First report generation regarding metrics
  - User Interface based on the Eclipse-Framework
- FOKUS!MBT – is a model-based testing workbench
  - Initial definition of test objectives for both cases
  - Initial generation of test cases in UTP from extracted UML system models for both cases (report automation degree)
  - Initial generation of test data from these models.
  - Some proof of concept for test execution.
- M@RT – Models@Runtime demonstration
  - Proof of Concept – demonstration

Each part is described in the following individual sections of this document.

Table of contents

EXECUTIVE SUMMARY **3**

TABLE OF CONTENTS **4**

1 STATIC ANALYSIS, TESTING AND MODELS@RUNTIME TOOLKIT **5**

1.1 VERIFICATION AND VALIDATION NEEDS IN REMICS 5
1.2 REQUIREMENTS 6
1.3 TOOLKIT ARCHITECTURE 6

2 STATIC ANALYSIS WORKBENCH - METRINO **7**

2.1 TOOL, DOCUMENTATION AND LICENSE 8
2.2 GETTING STARTED 8

3 MODEL-BASED TESTING WORKBENCH – FOKUS!MBT **9**

3.1 TOOL AND DOCUMENTATION 10
3.2 GETTING STARTED 10

4 MODELS@RUNTIME **12**

4.1 MODELS@RUNTIME 12
4.1.1 Terminology and Abbreviations 12
4.1.2 M@RT motivation and brief characterization 12
4.2 CASE STUDY 14
4.2.1 Underlying technologies 14
4.2.2 Case study's implementation in Java 15
4.2.3 Demonstration 16
4.3 REFERENCES 16

5 FUTURE WORK AND PLANS **16**

1 Static Analysis, Testing and Models@Runtime Toolkit

This document describes the WP6 toolkit on static analysis, testing and Models@Runtime. This document is part of the deliverable D6.1, which is the preliminary release of the toolkit. After first state-of-the-art analysis we limited the scope of the verification part to the static analysis aspects of the REMICS models.

1.1 Verification and Validation Needs in REMICS

The quality of a software system is influenced by a lot of factors. It is undisputed that in model-driven engineering the quality of models is crucial for the quality of the resulting software. Starting from design models, techniques like model transformation and code generation are used to produce automatically or semi-automatically software systems. Design models gain great importance for the software development process, therefore the developer, who models or design a system, has a difficult task and takes high responsibility.

A software system has to work in an adequate way after deployment phase. The developer needs evidence that the system is working correctly according to user requirements. Beside the typical requirement of having few bugs in the developed system, other stakeholders could be also interested in quality aspects of the system. In particular certification authorities have specific conditions which should be addressed within the development process as well.

Nevertheless, the discussion about the quality of software is not a new. A lot of verification and validation methods and techniques are already well-established which are focused on the quality aspects of software. For instance, in traditional development processes program code like C or C++ has to follow specific coding conventions to improve software quality and maintainability.

The WP6 toolkit is focussing on specific parts of these V&V techniques, in fact static analysis of models, model-based testing as well as Models@Runtime aspects. The general approach of integration these parts into REMICS is depicted by figure 1.

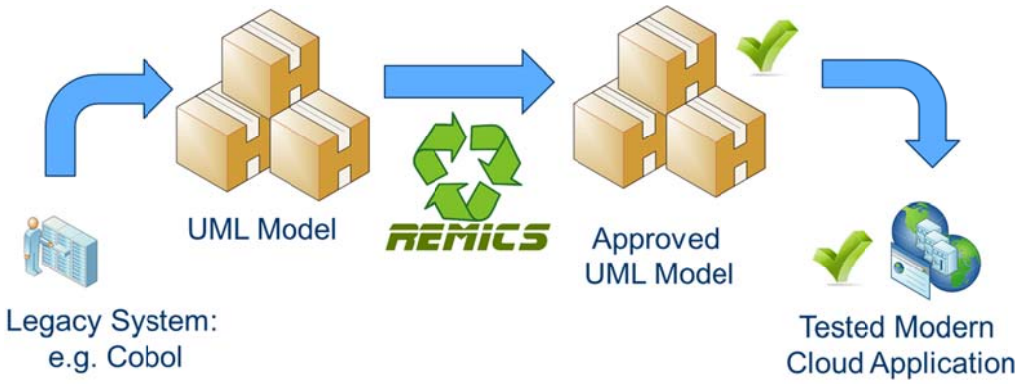

Figure 1 – General Idea of Verification and Validation in REMICS

The REMICS approach can be separated into three main phases: recovery, migration and deployment. Due to the fact that REMICS is following a model-driven approach, a lot of models will be created in each phase. Therefore, the WP6 toolkit can contribute in each phase with V&V techniques based on models in order to gain high-quality systems. Figure 1 shows the general idea on how this can be realized. The recovered information as well as migrated models can be analysed according to specific rules, conventions or guidelines. Furthermore, these models are also used to define test cases and test data in order to test the new modernized application based on the service-cloud paradigm.

Independent from the general need of high-quality systems, the REMICS case studies have specified needs regarding V&V. For instance, both case studies want to have fewer bugs in their systems after migration. First of all, this implies a measurement of the number of bugs before and after migration. And secondly, in particular, a mechanism to detect bugs at all. This indicates the application of model-based testing in both cases and the use of corresponding tools. Therefore, a list of relevant requirements for WP6 is given in the next section.

1.2 Requirements

In general, the toolkit addresses common software quality engineering aspects and goals, such as validation and verification of functionality, usability or maintainability. The adherence of these quality attributes contributes to high quality systems. Besides these common goals, the REMICS project have specified requirements for the REMICS tools, methods and case studies (see also D1.2). The following table shows a list of requirements from D1.2 which are related to the WP6 toolkit.

Table 1 – Requirements related to WP6 toolkit

Req.ID	Req. Title and description	(Sub)Goals	Priority	Milestone	Toolkit workbench relation
21	Preserve report data integrity after migration	Modernize legacy	High	M18	Specific test case
51	Reduce the number of bugs detected after installation	Fewer bugs	Major	M24	general goal for quality aspects
56	Model-based testing	Fewer bugs	High	M36	Need of Model-Based Testing approach
57	Increase knowledge of business logic across models	Extract business models	High	M36	can be addressed in static analysis workbench
58	Use model as documentation generation	Generate documents from models	Medium	M36	Results of toolkit as models -> can be reported
59	Document automation degree whenever possible	Well-integrated development environment	High	M18 and M36	General MDE goal. Provide toolkit functions as services
60	Integration of REMICS tools with the development environments of industrial cases	Well-integrated development environment	High	M36	Use of open standards and protocols

1.3 Toolkit Architecture

Verification and validation methods are diverse and can be applied in various areas in software engineering. Based on the given requirements (section 1.2), state-of-the-art analysis (REMICS deliverable D2.1) and case study analysis we limited the scope of the toolkit. Therefore, the REMICS

WP6 toolkit focuses on the following specific aspects on a verification and validation approach within a model-driven software development process.

- Static Analysis of models – is focusing on non-dynamic aspects of models and systems. Of interest are quantitative features of models, such as size metrics.
- Model-based Testing – in addition to static analysis it includes also dynamic aspects of models, moreover the generation of test cases or test data out of test specification and other qualitative aspects can be considered.
- Models@Runtime – is focusing on verification techniques at runtime. It aims to solve/find problems which do not occur during design/development time.

Based on these areas, the WP6 toolkit preliminary release consists of the three parts including the following features:

- Metrino – is a static analysis workbench for MOF –(OMG's Meta-Object Facility)[1] based models:
  - Implementation of Software Metric Metamodel (SMM) available in the toolkit with extensions
  - Initial set of metrics for the recovered and migrated models
  - First report generation regarding metrics
  - User Interface based on the Eclipse-Framework
- FOKUS!MBT – is a model-based testing workbench:
  - Initial definition of test objectives for both cases
  - Initial generation of test cases in UTP (UML Testing Profile) from extracted UML system models for both cases (report automation degree)
  - Initial generation of test data from these models.
  - Some proof of concept for test execution.
- M@RT – Models@Runtime demonstration
  - Proof of Concept – demonstration

Each part is described in the following individual sections of this document. Metrino and FOKUS!MBT are well integrated into the Eclipse-Framework. Both workbenches are using EMF (Eclipse Modeling Framework) models as input and output artefacts. In addition to that, several services are provided by both tools which can be integrated into a service-oriented architecture. Thus, the integration into existing development environments is not a complex task, due to the fact that open standards and protocols are used.

2 Static Analysis Workbench - Metrino

A static analysis workbench, named Metrino[2], is one part of the WP6 toolkit. Metrino is an integrated set of tools to support the validation and quality assurance of models based on OCL (Object Constraint Language) and SMM (Structured Metrics Metamodel). It can be used as an integrated tool through its Graphical User Interface based (GUI) on the Eclipse platform. It can be used for UML models as any instance of a Domain-Specific Modeling Language (DSL) based on MOF . The following figure depicts the general architecture of Metrino.

[1] http://www.omg.org/mof/

[2] http://www.modelbus.org/modelbus/index.php/metrino

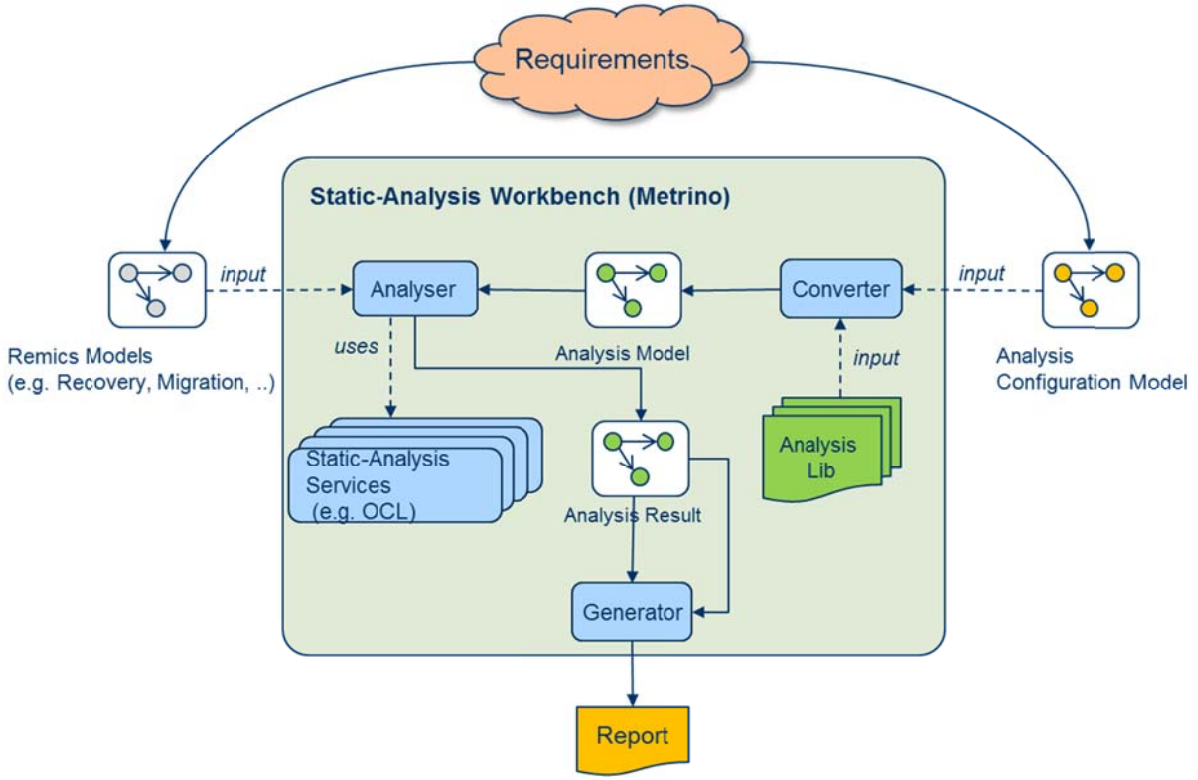

Figure 2 – Static Analysis Workbench Architecture

Based on requirements two types of models are the input for the static analysis workbench. On one hand the created and designed REMICS models (for instance recovery or migrated models) and on the other hand a specified analysis model based on SMM in order to analyse certain properties on the REMICS models. In addition to that, the workbench provides predefined standard metric sets for UML models, due to the fact that UML as well as UML profiles are the basis for the REMICS approach. Technically, the static analysis is realized by using an OCL engine and executing OCL queries on the used Models. The Result of the analysis can be exported and can be used for further evaluation within the testing process.

2.1 Tool, Documentation and License

The tool can be found on the REMICS platform at:

https://project.sintef.no/eRoomReq/Files/informatics/ReMiCS/0_8d018/de.fraunhofer.fokus.metrino_2.0.updatesite-TagBuild-20110826-1316.zip

A detailed user guide can also be found on the REMICS platform at:

https://project.sintef.no/eRoomReq/Files/informatics/ReMiCS/0_8d150/Metrino_InstallationGuide_v2.0.pdf

It is developed under a non-commercial evaluation license. The license agreement can be found at:

http://www.modelbus.org/modelbus/images/stories/fokus_evaluation-license-metrino.pdf

2.2 Getting Started

Metrino is based on the Eclipse Platform and therefore is distributed as a set Eclipse bundles. The software can be installed in any Eclipse based tool but it is recommended to use the “Modelling Tools edition” since it includes a lot of tools needed for Metrino. You may use the following link for getting an Eclipse Modeling distribution (Indigo release):

http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigor

In order to install Eclipse please unpack the contents of the archive to any location of your choice.

Metrino requires some additional software to be installed beforehand. After having installed and started Eclipse, you have to install the Model-To-Text engine Acceleo (see http://eclipse.org/acceleo/). As of the Helios Release of Eclipse, the IDE provides a comfortable mechanism to install the Modeling related software. Therefore, select the option “Install Modeling Components…” from the Eclipse Help menu. A detailed installation guide is provided with the tool.

To verify that the installation was successful do the following steps

- Open Eclipse Workbench
- Create new project
- Create a new Metrino Measure Model by opening the Metrino Wizard “File -> Other -> Metrino -> Measure Model” as shown in figure ….

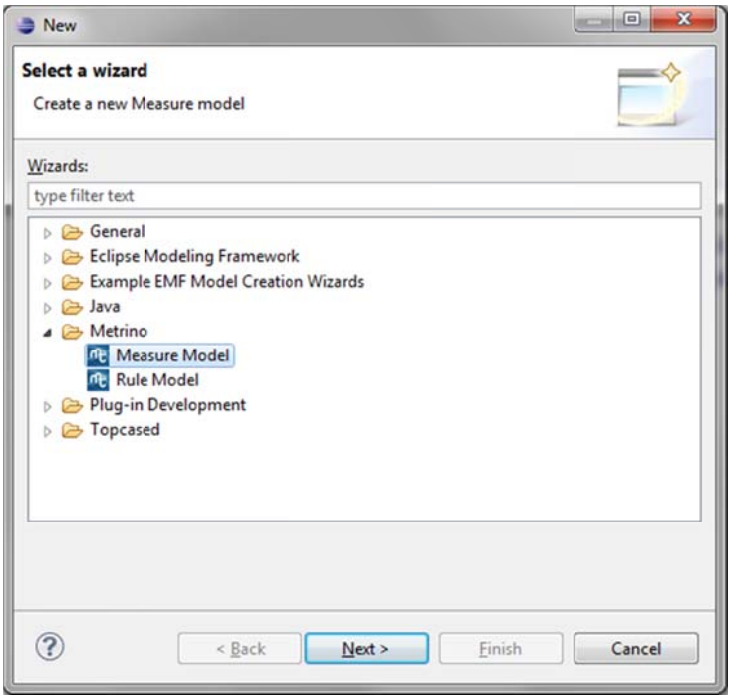

Figure 3 – Model-Based-Testing Workbench Architecture

In case of any problems you can contact Christian Hein or Tom Ritter from Fraunhofer FOKUS.

3 Model-Based Testing Workbench – FOKUS!MBT

Fokus!MBT is a set of compound tools which support the model-based paradigm for testing purposes. It establishes a tooling landscape for the specification, development and documentation of tailored model-based testing scenarios. Current Fokus!MBT methodologies are based on, but not limited to well-known and established standards like UML, SysML and the UML Testing Profile. That allows Fokus!MBT to be applied to a wide range of heterogeneous system development processes. Overall time to market is reduced by significantly decreasing the number of error-prone and resource-consuming manual tasks.

Fokus!MBT supports the testing process in a coherent way to capture the system’s requirements from the very beginning, using iterations of refinements, down to the test code and test execution. It is able to import different requirement representations, in particular the Requirements Interchange Format (ReqIF) and SysML. In combination with the UTP standard, Fokus!MBT allows for the creation of expressive test models. The explicit support of requirement driven testing scenarios and coverage analysis facilitates the assessment of system quality. This represents one of the most important

outcomes of a Fokus!MBT-driven test process. The following figures show the Fokus!MBT architecture.

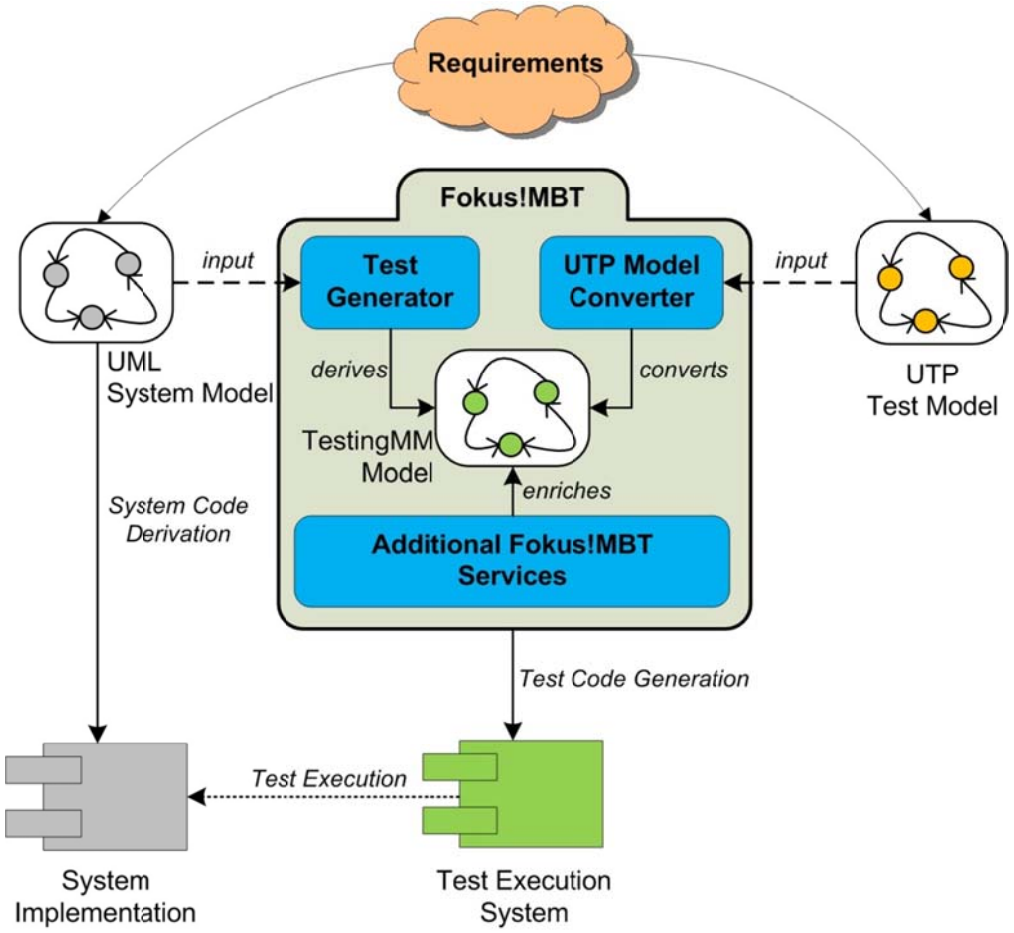

Figure 4 – Model-Based-Testing Workbench Architecture

3.1 Tool and Documentation

The tool can be found on the REMICS platform at:

https://project.sintef.no/eRoomReq/Files/informatics/ReMiCS/0_8d053/de.fraunhofer.fokus.testing.fokusmbt.updatesite_20010829_1505.zip

A detailed user guide for FOKUS!MBT can be found at:

https://project.sintef.no/eRoomReq/Files/informatics/ReMiCS/0_8d0aa/Fokus!MBT_UserGuide_010.pdf

It is developed under a non-commercial evaluation license.

3.2 Getting Started

The Fokus!MBT kernel simply provides the core features required for both implementing new (testing) services and creating user-defined, purposefully tailored tool environments. It is based on Eclipse Helios, but works with Indigo as well. Fokus!MBT is tested with the 32-bit version of Helios and Indigo on Windows XP and Windows 7.

As a recommendation, please use the Eclipse Modeling distribution as your underlying platform, since it contains a lot of stuff which may be needed for additional tool support. For a good foundation, be sure the following Eclipse modeling components are installed previously:

- Eclipse Modeling Framework (EMF) – the very foundational meta-modeling service framework including the meta-meta model Ecore (as an almost implementation of the OMG EMOF standard)
- Eclipse MDT UML2 (UML2) – an implementation of the UML2 v2.2 specification based on Ecore
- Eclipse MDT Object Constraint Language (OCL) – an implementation of the OMG OCL 2.2 specification based on Ecore
- Eclipse M2T Acceleo (Acceleo) – An implementation of the OMG MOF Model-to-Text-Transformation Language based on Ecore
- Eclipse M2M Query/View/Transformations operational (QVTo) – an implementation of the OMG QVTo specification based on Ecore

Additionally to these fundamental components, the following components have proved helpful from time to time – depending on what you want to achieve and what services are to be integrated into your Fokus!MBT tool chain:

- Eclipse MDT Papyrus – a graphical modeling environment for UML2 (and domain-specific) models
- Eclipse TMF XText – a generative framework for the creation of textual dsl's based on an input grammar in EBNF-like notation
- Eclipse GMF Graphical Modeling Framework (GMF) – comprises GMF Tooling, GMF Notation and GMF Runtime
- Eclipse EMFT Ecore Tools – a set of tools for the creation of Ecore metamodels
- Eclipse EMFT Compare – A framework for comparison of Ecore based models
- Eclipse Requirements Modeling Framework (RMF) – an implementation of the standards RIF1.1, RIF1.2 and the OMG adopted successor ReqIF 1.0.1 as well an optional GUI (ProR) to create ReqIF files

In order to verify that the installation was successful do the following steps:

- Open Eclipse Workbench
- create a new TestingMM project; opening the New Project wizard by clicking on File -> New Project (Ctrl + N). In the New Project wizard, expand the "Fokus!MBT" category and select "Fokus!MBT Editor Testing Model" as shown in figure

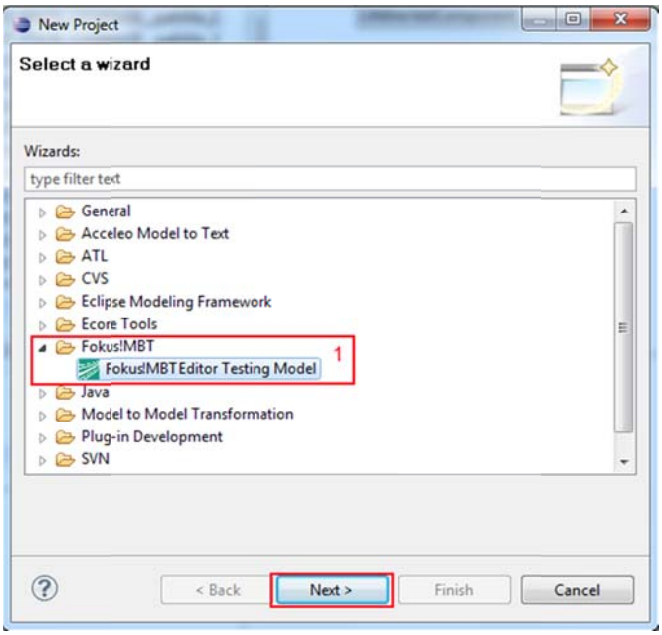

Figure 5 – FOKUS!MBT new project

In case of any problems you can contact Christian Hein or Tom Ritter from Fraunhofer FOKUS.

4 Models@Runtime

This document section intends to illustrate the idea of runtime verification in REMICS based on a novel research area within MDD named models at runtime (Models@runtime).

4.1 Models@Runtime

In the "Planned Contents of REMICS Deliverables – M12" document provided at M6, it is written: "Due to the very recent emergence (and thus weak maturity) of the notion of Models@runtime, REMICS aims to explore this notion in order to provide, at M12, a proof of concept. This proof is a simple application (deliverable) that has embedded (executable) models. In the logic of WP6, this proof of concept intends to assess the added values of runtime verification compared to MBT or static model analysis that in essence occurs at design time." With respect to this text extract, this section describes a Models@runtime realistic scenario along with its implementation in Java. For opportunistic reasons, it has been decided to reuse the case study presented in WP5 that illustrates the idea of service interoperability/mediation for REMICS. This creates a convergence with the different concepts/techniques/tools of WP5 and WP6/Models@runtime.

4.1.1 Terminology and Abbreviations

M@RT	Models at runtime
IasS	Infrastructure–as–a–service
PasS	Platform–as–a–service
PoC	Proof of concept
SOA	Service–Oriented Architecture

4.1.2 M@RT motivation and brief characterization

Verifying software systems at design/development time using model checking and, more traditionally, testing technologies, stumbles over – at execution time when software is in users' hands – unknown, unpredictable environment conditions and factors. Maintenance is easy to perform when software is

stopped but it is trickier when software is running, leading to the idea of (dynamical) software adaptation. However, disturbing conditions and factors have to be captured to be better understood and even controlled through correction actions without stopping execution.

The idea of software management (a.k.a. administration) encompasses software observation and control. When software systems are expressed in the form of models (Platform–Specific Models especially), there is great value in making these models persistent at runtime to have a tangible and intelligible comprehension support. For that, models must be executable (operational semantics) and embeddable through appropriate libraries and APIs.

The increasing demand on runtime verification comes from several reasons including:

- Mobility: cleaning up software from strong adherence to running platform features is often a dream. Ideally, software has to be designed free from technology facets, to be deployed and used in a very flexible way including mobility, i.e., the ability to move from one technological platform to another without costly adaptation actions at runtime. In the mobile phone for example, the Java slogan ("build once, run everywhere") does not apply even if mobile phones support similar Java Mobile Edition (Java ME) versions;
- SOA: in a world of services, software composition cannot be fixed once and for all at design/development time. Unanticipated, late composition of business applications and components with third–party components is a higher and higher concern in modern computing;
- Deployment: the notion of deployment amounts to transforming configuration parameter values of applications and components into runtime indicators[3], so that deployment platforms run applications and components in optimal conditions including resource needs and usages, performance (through load balancing especially), fault recovery based on transactions and security requirements fulfillment.

All of these changes show that traditional verification and testing at design/development time only solve a limited set of problems. Runtime verification (en.wikipedia.org/wiki/Runtime_verification) aims at complementing both of these two key concerns of software engineering.

Beyond, cloud computing in essence provides computing environments in which "old" configuration and preparation tasks relating to software execution environment management will be left, in the simplest cases, to cloud providers. In the more complicated cases, behind the ideas of PaaS and IaaS, one supposes that software administrators of the future will perform such tasks more easily and probably, remotely and programmatically. A great challenge relating to the verification and testing of applications in the cloud, will be the isolation of faults and the identification of their source between, in particular, the code and the effective setup and/or appropriate usage of the cloud computing environment provided. We mean, in distributed computing in general, many faults come from the inadequacy of applications' deployment parameter values with regard to the way platforms are installed/configured to host these applications. To have good matching rules and results, software administrators and software designers collaborate. In the cloud computing spirit, software designers will face up new difficulties to tune their applications' deployment parameter values because local knowledge about the way platforms are (can be) installed/configured will disappear. Having tools like Models@runtime may greatly help the diagnosis of problems and, possibly, the achievement non–disruptive maintenance. Models@runtime behaviors may be used as tracers for instance.

The vision of REMICS is to embed executable models in services to observe executions even control these executions through online software modifications (adaptation). Metamodels serve as metadata in the same ways programming languages, middleware… offer reflection support and APIs (introspection and intercession).

[3] This also covers on-the-fly code generation to map business code to its running environment, including the transformation of applications/components' deployment parameter values into technical code.

4.2 Case study

The case study (see also D5.1) is a *login protocol* concerning a client in relation to a predetermined service named “Service 1”. *Service 1* is able to allow/disallow login in one step (login and password validation). Another service named “Service 2” is able to do the same through two steps (login validation and password validation later).

Owing to *Client* cannot directly interact with *Service 2*, there is a fourth actor named *Mediator* that can act as a proxy between the two. For simplification, all connections are one–to–one connections but there is no special difficulty to extend this case study to have one–to–many and/or many–to–many connections.

There is no dedicated adaptation scenario in this case study or any interoperability/mediation illustration. Instead, this PoC only shows how models can be used to straightforwardly design each actor, how models are tangible and manipulable at runtime.

Figure 6 is an overview of the case study’s architecture. All state machines are depicted in D5.1 deliverable with key states and key requests and possible replies.

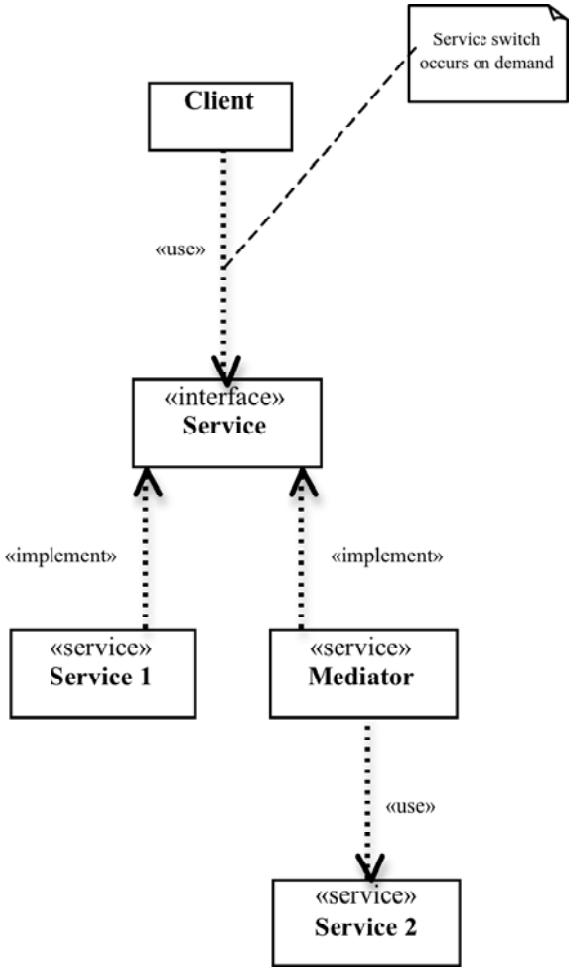

Figure 6 – PoC architecture

4.2.1 Underlying technologies

This PoC is based on the PauWare Java library (web.univ–pau.fr/~barbier/PauWare/PauWare_software), a set of Java classes and interfaces making up an execution engine for UML 2 State Machine Diagrams and UML 2 Sequence Diagrams. UML 2 State

Machine Diagrams are a variant of Harel's Statecharts that are used to model and to implement reactive software–intensive systems (see Section 4.2.2 about using PauWare).

The second technology used in this PoC is *Java Management eXtensions* (a.k.a. JMX). Instead of building from scratch a software management framework, it is definitely beneficial to use the JMX standard. Indeed, many facilities and surrounding tools are available thanks to JMX (here, a Web browser facility allowing the supervision and control of services and their embedded models through a Web console). Beyond, PauWare has a natural gateway to this technology.

4.2.2 Case study's implementation in Java

The code below shows how to embed a state machine into a service (as it happens, *Service 1*) by means of PauWare. The canonical *init_behavior* method includes all state initializations and setups like forcing a state to be an input state, attaching "entry", "exit" or "do" actions to states, etc. The *start* method starts up the state machine in expressing the allowed state machine's transitions especially. This method is called from an outside context at an appropriate time depending upon a given state machines' collaboration strategy.

Reading the code below must done in conjunction with reading the state machines in D5.1.

```
/** UML  state machine */
   protected AbstractStatechart _Wait_for_credentials;
   protected AbstractStatechart _Validate_credentials;
   protected AbstractStatechart_monitor _Service_1;
…
   protected void init_behavior() throws Statechart_exception {
      _Wait_for_credentials = new Statechart("Wait for credentials");
      _Wait_for_credentials.inputState();
      _Validate_credentials = new Statechart("Validate credentials").entryAction(this,
"validate_credentials");
   }

   protected void start() throws Statechart_exception {
      _Service_1 = new Statechart_monitor(_Wait_for_credentials.xor(_Validate_credentials),
"Service 1", true);
      _Service_1.fires("login", _Wait_for_credentials, _Validate_credentials, true, this, "valid",
null, AbstractStatechart.Broadcast);
      _Service_1.fires("login", _Wait_for_credentials, _Validate_credentials, true, this, "invalid",
null, AbstractStatechart.Broadcast);
      _Service_1.fires("valid", _Validate_credentials, _Wait_for_credentials, this, "if_valid", null,
this, "send_acknowledgment");
      _Service_1.fires("invalid", _Validate_credentials, _Wait_for_credentials, this,
"if_not_valid");
…
To close about implementation, one may look at the way events are processed:
/** UML events */
   public void login(Client client, String login, String password) throws Statechart_exception {
      _client = client;
      _client_login = login;
      _client_password = password;
      _Service_1.run_to_completion("login");
   }
```

The *run_to_completion* method of PauWare is a key element in the sense that it moves the state machine from one stable consistent state (most of the time, a set of states when states are parallel) to another.

4.2.3 Demonstration

This PoC is intended to be demonstrated on demand during the REMICS's first–year review. The approximate length of the demonstration is less than 15 min.

4.3 References

REMICS M@RT PoC can be downloaded from: www.FranckBarbier.com/REMICS/WP6 as a NetBeans project. Using this software requires any version of the NetBeans IDE (www.netbeans.org).

No additional download of PauWare and JMX is required.

5 Future Work and Plans

The second version of the WP6 toolkit is planned for M24 and will consist of the extended packages of existing parts. Furthermore, the results of the new tasks after enlargement of the REMICS project will also be integrated into the WP6 toolkit. In order to support the case study providers WP6 is going to collect information of the current test procedures by analysis and interviews as well as identifying the baseline data for verification and validation purposes. It is also planned to integrate the toolkit into the REMICS methodology provided by WP2 in order to integrate the toolkit into the REMICS tool chain. Therefore, the following features of the toolkit are planned for M24:

Integration of the new viewpoints from new tasks

- Test generation for high level acceptance
- Performance testing of cloud
- Agile testing methods

Metrino - Static analysis of models:

- Perform analysis of SoaML and CloudML models (according to metrics identified or updated from M12)

FOKUS!MBT – model-based testing

- Test execution engine
- Full test case generation for the migrated model

Models@Runtime:

- REMICS software components for supporting Models@Runtime as a library and an API (deliverable) to make some types of SoaML models executable and persistent at runtime. Depending on advances in the definition of a CloudML language, one may consider this applicability of this library/API for state-full cloud services implementation. Added values at this stage of Models@Runtime will be measured in relation with WP5 on interoperability. One may for instance consider mediation components as state-full services.

.