The LearnLib in FMICS-jETI

Tiziana Margaria
Chair of Services and Software Engineering,
University of Potsdam,
margaria@cs.uni-potsdam.de

Harald Raffelt Bernhard Steffen
Chair of Programming Systems
University of Dortmund, Germany
harald.raffelt@cs.uni-dortmund.de
steffen@cs.uni-dortmund.de

Martin Leucker
Institut für Informatik
TU München
München, Germany
leucker@in.tum.de

Abstract

The FMICS-jETI platform is a collaborative, service-based demonstrator of tools and techniques for the analysis of industrial critical systems. It is the FMICS Working Group contribution to the Verified Software Initiative.

In this paper, we extend the scope of the FMICS-jETI platform to address the integration of heterogeneous and legacy tools and technologies. *We show how to integrate 1) CORBA, a language independent standard for the interoperability of heterogeneous functionalities distributed over a network, 2) active model learning technologies, via the LearnLib, as a model extrapolation technique that uses testing to explore a black box system and CORBA as a communication mechanism, and 3) third party applications built on top of the LearnLib, in this case Smyle, a tool that synthesizes design models by learning from examples, that uses the LearnLib as learner core.*

1 Introduction

One of the goals of FMICS, the ERCIM Working Group on Formal Methods for Industrial Critical Systems (FMICS) [45], is to transfer and promote the use formal methods technology in industry. The ongoing Verified Software Initiative Grand Challenge [16] offers a great opportunity to reach this goal, resulting in a more robust and solid software industry. The FMICS-jETI platform[1] concretizes the collective effort of the FMICS WG by offering a collaborative demonstrator of the FMICS techniques and tools based on the jETI technology[2]. FMICS-jETI provides as repository a collection of verification tools stemming from the activities of the FMICS working group and facilities to orchestrate them in a remote and simple way. At the same time FMICS-jETI itself is a contribution to the VSI repository and thus to the Grand Challenge.

We focus on correctness at the model level, rather than at the software (more generally at the coding) level. An adequate repository should therefore contain not only analyzed or proven correct software, but principally *tools* (themselves software artifacts) that help establishing the correctness of the software in question starting from the requirements, specifications, and models.

Based on jETI [40], the new generation of ETI [38, 23], the core FMICS partners have set up the FMICS-jETI platform as a collaborative, service-based demonstrator that

- illustrates the wide applicability of the jETI technology for lightweight remote integration of tools into the repository
- shows how to provide tools to the repository, by registration and remote provision,
- demonstrates how to experiment with local and remote tools to solve cooperative verification tasks
- shows how to orchestrate different tools (possibly a mix of local and remote ones) which were not originally designed to cooperate, to address more complex case studies. This may require the availability of mediators, to cover semantic gaps between the tools.

FMICS-jETI so far includes verification tools based on model checking techniques, like GEAR [3, 17], applications of model checking to dataflow analysis, as in [22, 12], and parallel model checking [7].

In this paper, we show how to extend the scope of the FMICS-jETI platform to address the integration of *heterogeneous and legacy tools and technologies*. We are in fact integrating

- CORBA, a language independent standard for the interoperability of heterogeneous functionalities distributed over a network

[1] `jeti.cs.uni-dortmund.de/fmics/index.php`
[2] `jabc.cs.uni-dortmund.de/plugins/jeti_en.html`

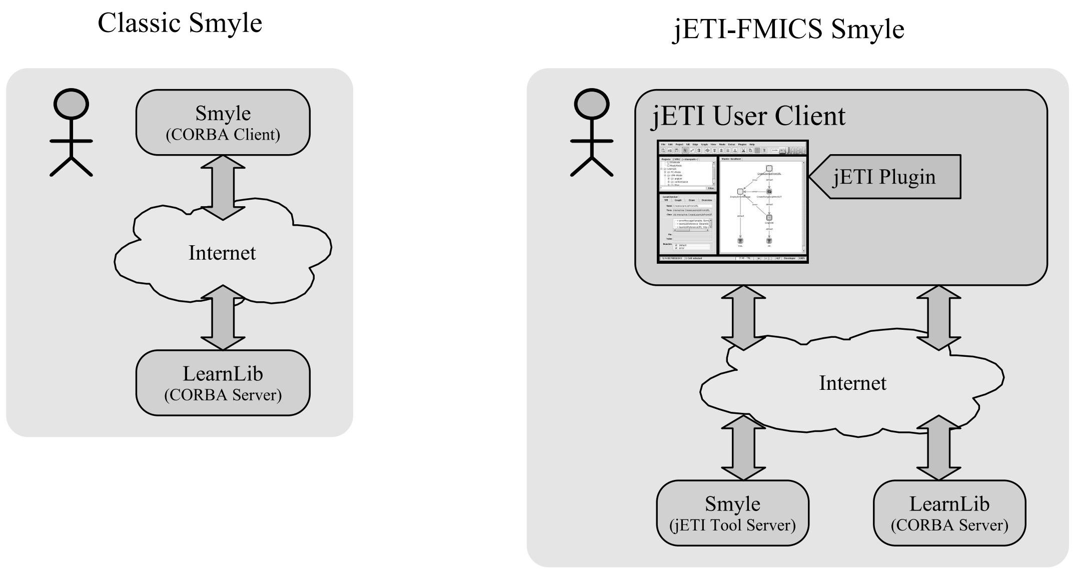


Figure 1. Smyle Classic and jETI-FMICS Version

- active model learning technologies, via the LearnLib [42, 36] as a model extrapolation technique that uses testing to explore a black box system and CORBA as a communication mechanism, and
- third party applications built on top of the LearnLib, in this case Smyle [4], a tool that synthesizes design models by learning from examples, that uses the LearnLib as learner core.

These tools and techniques have been successfully used before outside the jETI technology. We are currently using this case study as a blueprint for guidelines on how to bring CORBA-compliant tools and complex, communicating applications into (FMICS-)jETI.

In the following, we first present the current and the jETI-based architecture (in Sect. 2), then we recall the essential description of the jABC/jETI platform (in Sect. 3), before detailing on the ongoing integration of CORBA applications via its interface description language (in Sect. 4). We then briefly present the LearnLib (in Sect. 5 and 6) and Smyle (in Sect. 7) from the point of view of a typical FMICS-jETI user. We finally conclude in Sect. 8.

2 LearnLib/Smyle Architecture Overview

The communication architecture of Smyle and the LearnLib is depicted in Fig. 1.

Currently, a Smyle user directly interacts with Smyle, that in turn communicates with the LearnLib directly, via the CORBA interface. Thereby the LearnLib acts as CORBA server and Smyle acts as the client. Remote method invocations of the Learnlib are hard-wired in the sources of Smyle, so there is no flexibility in modifying, extending, or evolving Smyle without programming effort.

Within FMICS-jETI we envisage a looser, service based coupling of Smyle and the LearnLib. In this service-oriented setting, the LearnLib as well as the Smyle services are integrated into the jETI-FMICS platform as collections of (possibly remote) services that provide platform independent learning and model discovery functionalities. The Smyle developer interacts with the FMICS-jETI platform via the GUI of the jABC/jETI environment, and is responsible for designing the Smyle process by combining basic services offered by Smyle and the LearnLib into an orchestration or choreography that serves the desired purpose.

This approach offers a much higher versatility in customizing and extending the Smyle process, allows the easy integration of other tools in the global process, and reduces the programming effort in creating and then maintaining and evolving applications like Smyle.

The service core of the FMICS-jETI platform is the jABC/jETI framework, that we describe next.

3 jABC/jETI

The jABC framework [17, 46] is an environment for model-driven service orchestration based on lightweight process coordination. With its predecessors, it has been used over the past 12 years for business process and service logic modelling in several application domains, including telecommunications, bioinformatics, supply chain management, e-commerce, collaborative decision support systems, as well as for software and system development. In this paper, we restrict us to the jABC facilities relevant to (re-)designing the Smyle/LearnLib tool in terms of a complex service orchestration: a learning approach for synthesizing design models from example scenarios that are given as message sequence charts.

In jABC, orchestration and choreography of services happen on the basis of the processes they realize in the respective application domain. These processes embody the business logics, and are expressed themselves as (executable) process models. These jABC process models are called Service Logic Graphs (SLGs). Services are described by their (possibly hierarchical) SLG together with the atomic services they use. These basic service types are called SIBs (Service Independent building blocks).

Semantically, jABC models are control flow graphs, internally interpreted as Kripke Transition Systems [32], extended with fork/join parallelism. This provides a kernel for a sound semantical basis for formalisms like BPNM, BPEL, UML activity diagrams, and dataflow graphs, and constitutes a *lingua franca* adequate for the analysis and verification of properties, e.g. by model checking [32].

A SIB represents an atomic functionality provided as basic service. Within jABC, domain-specific SIB palettes are shareable among projects, and organized in a project-specific structure and with project-specific terminology. This is a simple way for adopting or adapting to different ontologies within the same application domain. Domain-specific SIB palettes are complemented by a library of SIBs that offer basic functionality (e.g. SIBs for I/O or memory handling), control structures (as used here) or handling of data structures like matrices (e.g. in our previous bioinformatics applications [26]).

A service orchestration is modelled in the jABC graphically, it is executable by interpretation via the Tracer plugin or it can be compiled to some target language via the GeneSys plugin. Once compiled, the orchestration lives independently of the jABC and can be imported in other frameworks.

SIBs can be local or remote. The jETI framework (Java Electronic Tool Integration) [20, 41, 2] enhances the jABC to support seamless integration of remote services such as SOAP based web services [21] and CORBA [34] applications. An essential capability of the jETI framework is its ability to generate basic service types, i.e. SIBs, from service interface descriptions provided in

- WSDL (Web Services Description Language), in case of SOAP based web services, and
- IDL (Interface Definition Language), in case of CORBA applications
- plain XML, in case of services exported on the basis of legacy systems.

The latter is the case for example for other FMICS-jETI services, like the jETI Version of our GEAR model checker and of jMosel [43], and of Alpha-spin [47]. So called REST services (for REpresentational State Transfer services) can be treated in this way too.

3.1 Choreography

jABC originated in the context of the verification of distributed systems [32], therefore SLGs are inherently adequate as choreography models. The SIBs can physically run in a distributed architecture. They communicate directly or with a shared space (called the context). The SLGs are fully hierarchical: SIBs can themselves be implemented via SLGs. The macro mechanism described in [39] allows defining what communication actions of an SLG are visible to the environment (for choreography). Orchestration is as far as the jABC is concerned just a degenerate, because localized, case of choreography.

Most applications of the jABC/jETI, like those concerning the LearnLib and those in bioinformatics, inherently require the coordination of several participating partners, often including humans, to collectively accomplish a goal. In this sense, SLGs like Smyle's define choreographies.

3.2 jETI Architecture Overview

jETI's tool integration philosophy is service-based: it replaces the usual requirement of ”physical” tool integration by very simple registration and publishing. This allows the provisioning of tool functionalities in a matter of minutes: fast enough to be fully demonstrated during our presentation. Moreover, whenever the portion of a tool's API which is relevant for a new version of a functionality remains unchanged, version updating is fully automatic.

This registration / publishing approach is implemented via the following three components.

1. a **Tool Configurator**, where tool providers register their tool functionalities just by filling out a simple template form, or by publishing the WSDL of a preexisting web service or the IDL of a CORBA application,

2. the **jETI Component Server**, which (a) automatically generates the SIBs, which are Java classes, from these specifications and (b) organizes all the registered tool functionalities, including the corresponding version control. In the future it will also include user authentication, authorisation, and accounting.

3. the **jETI client**, which loads the SIBs from the jETI Component Server and provides them as services, together with the jABC service development environment for orchestrating the tool functionalities.

4. a **Tool Executor**, which is able to steer the execution of the specified tools at the tool providers' site.

We explained previously how to integrate external tools by means of the XML Tool description [28], which was also applied to REST-style external services, like those offered by statistical analysis packages based on the R tool, as shown in [18].

We currently deal with Web services equipped with a WSDL description both in the scope of the Semantic Web Service Challenge [48], as shown in [30, 19], and for applications in the bioinformatics domain, as for example in [27].

In this paper we concentrate only on the new extension to CORBA applications.

4 jETI-enabling CORBA Applications

The integration of CORBA applications inside the jETI client requires only a valid IDL file. IDL files contain interface descriptions that specify the set of possible operations a client may request through that interface. Seen from the jETI point of view, the IDL file provides a syntactic description of how to access the service and of its set of operations. An example of an interface description is depicted in Fig. 2, which describes the LearnLib service that initializes Angluin's algorithm L^*. The init operation accepts the size of the input alphabet (*sigmaSize*) and initializes its internal data structures. After execution, it returns a set of membership queries via the output parameter *queries*. If the initialization fails due to lack of memory the exception *OutOfMemory* is raised.

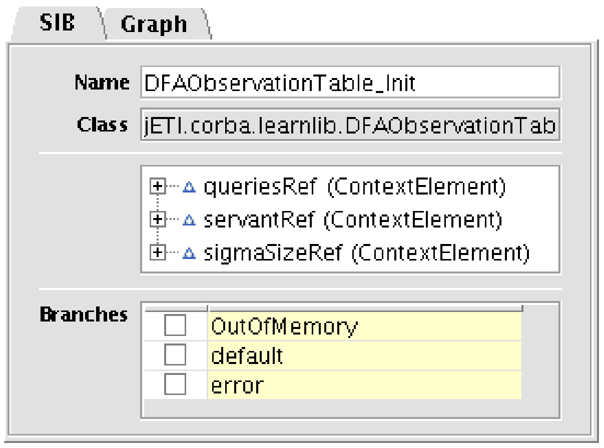
```
module learnlib {
  interface DFAObservationTable {
  void init(
    in unsigned long sigmaSize,
    out DFAQueryTree queries
  ) raises (OutOfMemory) ;

  /*  some more methods */
  };
};
```

Figure 2. A CORBA Interface Definition

CORBA operations are similar to methods in object oriented programming languages: they may have input and output parameters, a return value, and they can throw exceptions to signal error conditions. jETI's SIB generator extracts the information about the interfaces and operations and creates the SIBs accordingly. Since a SIB represents an atomic functionality, it generates one SIB for each operation.

Parameters and the return value are handled as hierarchical SIB parameters: they enable the user to freely define where to store input and output values for the CORBA service, using the preexisting graphical user interface of the jABC. The user view of the generated SIB is shown in Fig. 3, as provided by the jABC SIB inspector.

As for method calls, the invocation of CORBA operations requires an object that deals with the state of the service. Therefore the SIB has as additional parameter the reference to this object (the servant).



Name: DFAObservationTable_Init
Class: jETI.corba.learnlib.DFAObservationTab

queriesRef (ContextElement)
servantRef (ContextElement)
sigmaSizeRef (ContextElement)

Branches:
☐ OutOfMemory
☐ default
☐ error

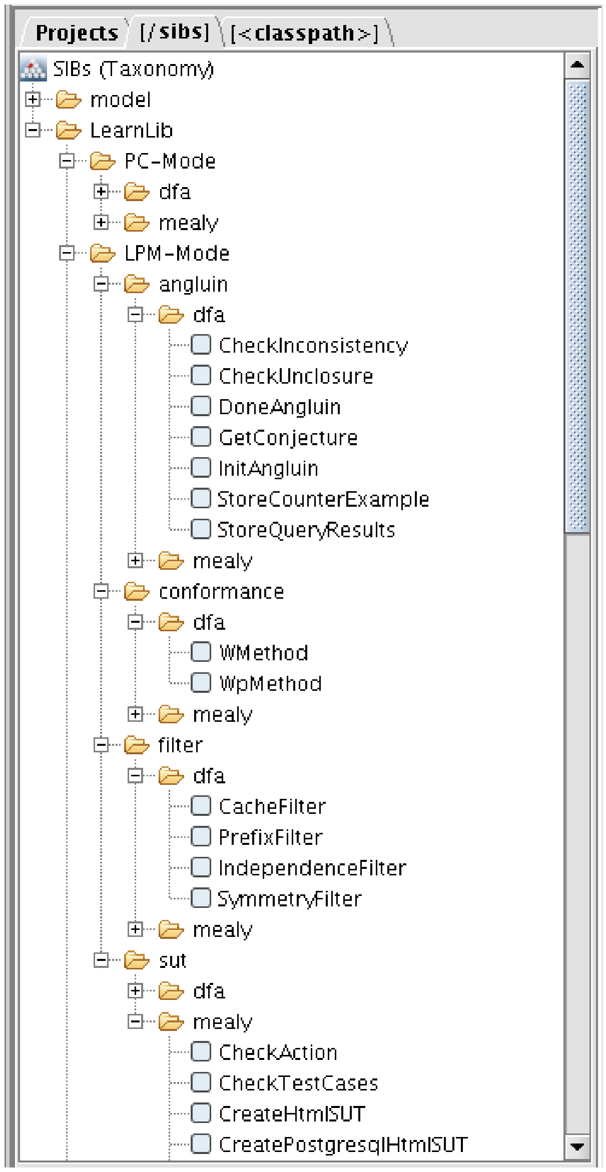
Figure 3. User view of the generated SIB: its jABC SIB inspector

CORBA exceptions are mapped to SIB branches, which steer the control flow in the jABC. In the example, the default branch indicates normal execution of the service, the OutOfMemory branch corresponds to the exception defined in the interface, and error is the generic branch used in the jABC as default to cover all other failures, such as network communication problems.

Our current implementation of the CORBA-to-jETI service importer uses the IDL-to-Java compiler *idlj* of the Java Software Development Kit to generate a Java client library, then analyzes that library and extracts the relevant information to generate the SIBs. This way the approach can easily be adapted to other remote invocation protocols, such as RMI and SOAP.

5 The LearnLib

LearnLib is a library of tools for automata learning. It is implemented in C++, maintained under Linux and Solaris, and it currently consists of 150 classes and almost 50.000 lines of code. Originally, LearnLib has been designed to systematically build finite state machine models of real world systems. In the meantime, it also became a platform for experimenting with different learning algorithms and to statistically analyze their characteristics in terms of learning effort, run time and memory consumption.



```
Projects  [/sibs]  [<classpath>]
SIBs (Taxonomy)
  model
  LearnLib
    PC-Mode
      dfa
      mealy
    LPM-Mode
      angluin
        dfa
          CheckInconsistency
          CheckUnclosure
          DoneAngluin
          GetConjecture
          InitAngluin
          StoreCounterExample
          StoreQueryResults
        mealy
      conformance
        dfa
          WMethod
          WpMethod
        mealy
      filter
        dfa
          CacheFilter
          PrefixFilter
          IndependenceFilter
          SymmetryFilter
        mealy
      sut
        dfa
        mealy
          CheckAction
          CheckTestCases
          CreateHtmlSUT
          CreatePostgresqlHtmlSUT
```

Figure 4. LearnLib basic services: jABC's overview in the SIB Taxonomy

Machine learning deals in general with the problem of how to automatically generate system descriptions. The field of *automata learning*, also called regular extrapolation [14] or regular inference [11], is of particular interest for soft- and hardware engineering [10], [31], [44], [35], [9]. Recent attention has been devoted to these techniques in the context of assume-guarantee [8] and of interface refinement for compositional verification [13].

We have used automata learning techniques in a number of contexts, e.g. to automatically construct models of Web applications as demonstrated in [36] and to enhance incomplete specifications of biological systems [24].

Automata learning tries to construct a deterministic finite automaton that matches the behavior of a given target automaton on the basis of observations of the target automaton and perhaps some further information on its internal structure [14, 37, 42]. Here we only say that our realization is based on Angluin's learning algorithm L^* from [1].

L^*, also referred to as an *active* learning algorithm, learns deterministic finite automata by *actively* posing *membership* queries and *equivalence* queries to the target automaton in order to extract behavioral information, and by refining successively an own hypothesis automaton based on the answers. A membership query *tests* whether a string (a potential run) is contained in the target automaton's language (its set of runs), and an equivalence query compares the hypothesis automaton with the target automaton for language equivalence, in order to determine whether the learning procedure was (already) successfully completed. In this case the experimentation can stop.

The LearnLib provides various means to carry out and optimize this learning.

6 The LearnLib in jABC/jETI

Fig. 4 shows the SIB palette of the LearnLib in the jABC: this is the catalogue of the services provided by the LearnLib to any jABC/jETI user as SIBs. They are taxonomically classified according to criteria that ease the intuitive retrieval to users not familiar with the LearnLib facilities.

The LearnLib Service Catalogue. The LearnLib has

- a *pre-configuration mode* (PC-Mode), which allows the user to pre-configure an optimized learning setting for both DFA and Mealy models, and

- a *learning process modelling mode* (LPM-Mode), which enables the user to control the entire learning process, the context-specific choice of optimizations, strategies of search, as well as the setting of interaction points for a truly interactive learning process.

Since these two modes are mutually exclusive, the top level classification distinguishes between the PC-Mode and the LPM-Mode services.

In the LPM mode, the LearnLib provides three sublibraries of services:

- The *automata learning* library (called Angluin) contains the basic learning algorithms for both DFA and Mealy models. Here we find for example the LearnLib SIBs initAngluin and GetConjecture that are used in the Smyle application.

- the *filter* library provides several strategies (cache, prefix, symmetry, independence) to reduce the number of queries [29], and

- the *approximative equivalence queries* library (called Conformance) is based on the generation of conformance test suites for the conjectures of the learning algorithms according to different methods.

Additionally, a service library SUT concerns SIBs providing the functionalities needed to access and manage a generic System Under Test.

The Smyle Application. As shown in Fig. 5, the jETI enabled Smyle application is built in the LPM-Mode, as a SLG that uses the LearnLib services, own services, and jABC/jETI basic services.

Here, SLGs are are used to model the entire learning process, which comprises the modelling of conditional or interactive behaviour. The nodes represent arbitrary statements, in particular including any atomic functionalities of the LearnLib, and the edges the control flow of the service logic, i.e. in which sequence and under which condition they are executed. Before we discuss this SLG in detail, we summarize the purpose of Smyle.

7 Smyle

Smyle (**S**ynthesizing **M**odels b**y** **L**earning from **E**xamples) [4], is a tool for synthesizing design models by learning from example scenarios that are given as message sequence charts. It addresses the requirement elicitation phase within the typical software engineering development cycle. Popular requirement engineering methods, such as the Inquiry Cycle and CREWS [33], exploit use cases and scenarios to specify the system's requirements. Sequence diagrams are also at the heart of the UML. Other formal methods-based approaches like R2D2C use traces (in this case CSP traces) to formulate scenarios [15]. A scenario is a partial fragment of the system's behavior, describing the system components, their message exchange and concurrency. Their intuitive yet formal nature has resulted in a broad acceptance. Scenarios can be either positive or negative, indicating a possible desired or unwanted system behavior, respectively. Different scenarios together form a more complete description of the system behavior.

The subsequent design phase in software engineering is a major challenge as it is concerned with a paradigm shift between the *requirement* specification—a partial, overlapping and possibly inconsistent description of the system's behavior—and a conforming *design model*, a complete behavioral description of the system (at a high level of abstraction). During the synthesis of such design models, usually automata-based models that focus on intra-agent communication, conflicting requirements will be detected and need to be resolved. Typical resulting changes to requirements specifications include adding or deleting scenarios, and fixing errors that are found by a thorough analysis (e.g., model checking) of the design model. Obtaining a complete and consistent set of requirements together with a related design model is thus a highly iterative process.

In previous work, we have shown how to enhance requirement elicitation and requirement completion in the R2D2C approach with the learning technology implemented in the LearnLib. [24, 25].

Here, we show how Smyle uses the LearnLib within the *Smyle modelling approach* (*SMA*, for short), to address the gap between scenario-based requirement specifications and design models by exploiting learning algorithms for the synthesis of design models from scenario-based specifications. Since message-passing automata (MPA, for short) [6] are a commonly used model to realize the behavior as described by scenarios, Smyle adopts MPA as design model.

The technical heart of *SMA* (see [4]) is a procedure that interactively infers an MPA from a given set of positive and negative scenarios of the system's behavior provided as message sequence charts (MSCs). This is achieved by generalizing Angluin's learning algorithm for deterministic finite-state automata (DFA) towards specific classes of bounded MPA, i.e., MPA that can be used to realize MSCs with channels of finite capacity.

SMA supports the *incremental generation* of design models. Learning of initial sets of scenarios is feasible. On addition or deletion of scenarios, MPAs are adapted accordingly in an automated manner. Thus, synthesis phases and analysis phases, supported by simulation or analysis tools such as *MSCan* [5], complement each other in a natural fashion. Furthermore, on establishing the inconsistency of a set of scenarios, *diagnostic feedback* in the form of a counterexample can guide the engineer to evolve his requirements.

7.1 Smyle and the LearnLib

Currently, *Smyle*, which can be freely downloaded at `http://smyle.in.tum.de`, is is written in Java and makes use of the `LearnLib` library via its CORBA interface. Thus, the LearnLib is basically used like a standard library in this case providing learning functionality. The main difference is that the learning functionality is not integrated into *Smyle* at compile time but at run time. While this requires to have an Internet connection to the LearnLib's location, this design choice has several advantages: *Smyle* immediately profits from ongoing improvements or bug fixes of the learning library – completely transparent to the user of *Smyle*. Furthermore, learning large systems typically asks for machines with sizable memory. A remotely

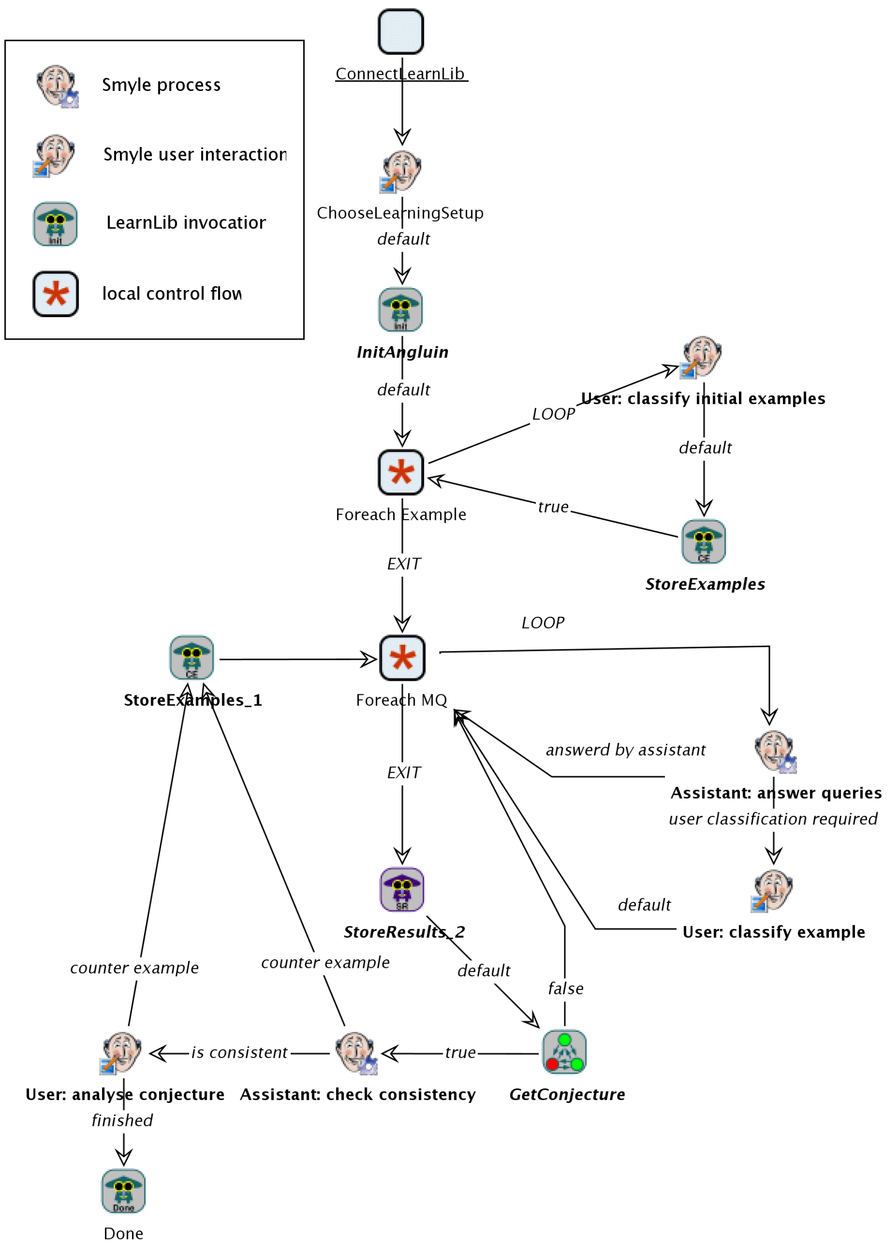


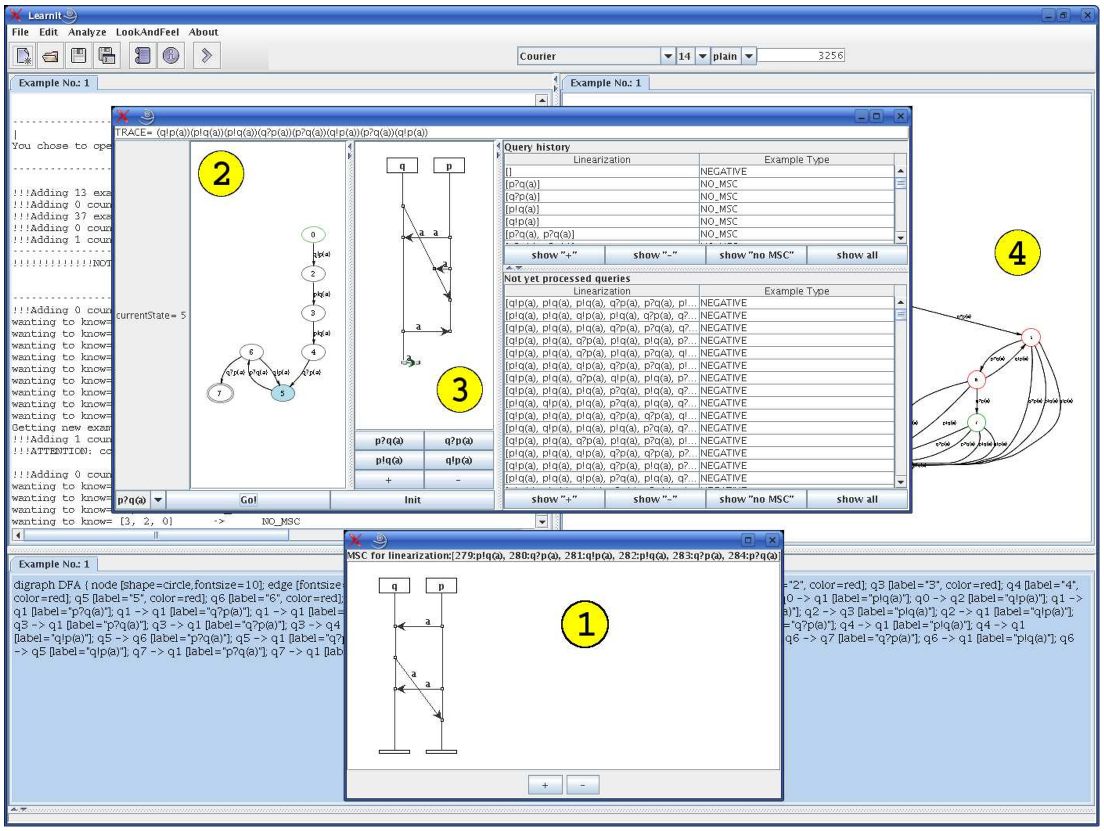
Figure 5. Learning Process Modeling Mode: Design of Smyle as jABC Choreography

running LearnLib shifts this issue to the location that usually supports it and can deal with this issue better than the typical user of *Smyle* on its local machine.

7.2 Smyle as a jABC Choreography

The SMA approach can be elegantly and intuitively explained along its LPMM model in the jABC. As shown in Fig. 5, Smyle's SLG coordinates (1) a user, who executes the Smyle user interaction SIBs, (2) Smyle processes, (3) LearnLib services, and (4) generic control activities provided by the jABC, like loop controls and store activities. In detail, once connected to the LearnLib (SIB ConnectLearnLib,

- the user is initially asked to specify the learning setup (ChooseLearningSetup) and the LearnLib is started (InitAngluin).
- After having selected a language type (existentially/universally) and a channel bound B, the user provides a set of MSCs. These MSC specifications must



```
--------------
|
You chose to ope
--------------
!!!Adding 13 exa
!!!Adding 0 coun
!!!Adding 37 exa
!!!Adding 0 coun
!!!Adding 1 coun
--------------
!!!!!!!!!!!!!NOT

--------------
!!!Adding 0 coun
wanting to know=
wanting to know=
wanting to know=
wanting to know=
wanting to know=
wanting to know=
wanting to know=
wanting to know=
wanting to know=
Getting new exam
!!!Adding 1 coun
!!!ATTENTION: cc

!!!Adding 0 coun
wanting to know=
wanting to know=
wanting to know=
wanting to know= [3, 2, 0]        ->        NO_MSC
```

TRACE= (q!p(a))(p!q(a))(p!q(a))(q?p(a))(p?q(a))(q!p(a))(p?q(a))(q!p(a))

Query history

Linearization	Example Type
[]	NEGATIVE
[p?q(a)]	NO_MSC
[q?p(a)]	NO_MSC
[p!q(a)]	NO_MSC
[q!p(a)]	NO_MSC
[p?q(a), p?q(a)]	NO_MSC

Not yet processed queries

Linearization	Example Type
[q!p(a), p!q(a), p!q(a), q?p(a), p?q(a), p!...	NEGATIVE
[p!q(a), p!q(a), q!p(a), p!q(a), q?p(a), q?...	NEGATIVE
[q!p(a), p!q(a), p!q(a), q?p(a), p?q(a), q?...	NEGATIVE
[q!p(a), p!q(a), q?p(a), p!q(a), p!q(a), p?...	NEGATIVE
[q!p(a), p!q(a), q?p(a), p!q(a), p?q(a), q!...	NEGATIVE
[p!q(a), p!q(a), q!p(a), q?p(a), p?q(a), p!...	NEGATIVE
[q!p(a), p!q(a), q?p(a), q!p(a), p!q(a), p?...	NEGATIVE
[q!p(a), p!q(a), p!q(a), p?q(a), p!q(a), q?...	NEGATIVE
[p!q(a), q!p(a), p!q(a), p?q(a), p!q(a), q?...	NEGATIVE
[q!p(a), p!q(a), p!q(a), q?p(a), q?p(a), q!...	NEGATIVE
[p!q(a), q!p(a), p!q(a), p!q(a), p?q(a), q?...	NEGATIVE
[q!p(a), p!q(a), q?p(a), p!q(a), p?q(a), p!...	NEGATIVE
[p!q(a), p!q(a), q!p(a), q?p(a), q?p(a), p?...	NEGATIVE
[q!p(a), p!q(a), p!q(a), q?p(a), p!q(a), p?...	NEGATIVE
[p!q(a), p!q(a), q!p(a), p?q(a), p!q(a), q?...	NEGATIVE

MSC for linearization:[279:p!q(a), 280:q?p(a), 281:q!p(a), 282:p!q(a), 283:q?p(a), 284:p?q(a)]

```
digraph DFA { node [shape=circle,fontsize=10]; edge [fontsize
color=red]; q5 [label="5", color=red]; q6 [label="6", color=red];
q1 [label="p?q(a)"]; q1 -> q1 [label="q?p(a)"]; q1 -> q1 [label=
q3 -> q1 [label="p?q(a)"]; q3 -> q1 [label="q?p(a)"]; q3 -> q4
[label="q!p(a)"]; q5 -> q6 [label="p?q(a)"]; q5 -> q1 [label="q?
-> q5 [label="q!p(a)"]; q7 -> q1 [label="p?q(a)"]; q7 -> q1 [lab
```

```
="2", color=red]; q3 [label="3", color=red]; q4 [label="4",
q0 -> q1 [label="p!q(a)"]; q0 -> q2 [label="q!p(a)"]; q1 ->
a)"]; q2 -> q3 [label="p!q(a)"]; q2 -> q1 [label="q!p(a)"];
="q?p(a)"]; q4 -> q1 [label="p!q(a)"]; q4 -> q1
q6 -> q7 [label="q?p(a)"]; q6 -> q1 [label="p!q(a)"]; q6
```

Figure 6. *Smyle* **screenshot**

then be classified as *positive* (i.e., MSCs contained in the language to learn) and *negative* (i.e., MSCs not contained in the language to learn). This happens via user interaction with Smyle (User: classify initial examples and StoreExamples).

- After submitting these examples, all linearizations (see Fig. 6 (1)) are checked for consistency with respect to the properties of the learning setup. Violating linearizations are stored as negative examples.

- Now the learning algorithm starts. The *Learner* continuously communicates with the *Assistant* in order to gain answers to membership queries. This halts as soon as a query cannot be answered by the *Assistant*.

- In this case, the *Assistant* forwards the inquiry to the user, displaying the MSC in question on the screen. The user must classify the message sequence chart as positive or negative (cf. Fig. 6 (1)).

- The *Assistant* checks the classification for validity wrt. the learning setup. Depending on the outcome of this check, the linearizations of the current MSC are assigned to the positive or negative set of future queries. Moreover, the user's answer is passed to the *Learner* which then continues his question-and-answer game with the *Assistant*.

- If the `LearnLib` proposes a possible automaton, the *Assistant* checks whether the learned model is consistent with all queries that have been categorized but not yet been asked. If it encounters a counter-example, it presents it to the learning algorithm which, in turn, continues the learning procedure until the next possible solution is found.

- In case there is no further evidence for contradicting samples, a new frame appears (cf. Fig. 6 (2,3)). Among others, it visualizes the currently learned automaton (2,4) as well as a panel for displaying MSCs (3) of runs of the system described by the automaton. The user is then asked if it agrees with the solution and may either stop or introduce a new counter-example proceeding with the learning procedure.

8 Conclusions

We have shown how to extend the FMICS-jETI platform for the integration of *heterogeneous and legacy tools and* *technologies*. With automatic integration of CORBA applications from their IDL interface descriptions, we capture a significant segment of legacy network-based applications. We have used this integration technology to provide the LearnLib, a model extrapolation technique that uses testing to explore black box systems, as a service catalogue to the FMICS-jETI users. We have shown how Smile, which builds on LearnLib functionality, could be redesigned as a jABC service choreography that coordinates own and LearnLib services, thus illustrating how preexisting third party applications built on top of the LearnLib can be refactored in a service-oriented fashion.

From the point of view of the Verified Software Initiative, the Corba integration allowed us to extend the coverage of the FMICS-jETI platform to capture not only model checking-based analyses, but also requirement elicitation, model synthesis, testing, and automata learning.

Acknowledgements: We thank Benedikt Bollig, Joost-Pieter Katoen, and Carten Kern for many useful comments on Smyle [4].

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 2(75):87–106, 1987.

[2] A. Arenas, J. Bicarregui, and T. Margaria. The FMICS view on the verified software repository. In *Proc. Integrated Design and Process Technology (IDPT)*, San Diego (USA), June 26-29 2006. Society for Design and Process Science.

[3] M. Bakera and C. Renner. jABC model checking plugin. `http://www.jabc.de/modelchecking/`, seen Apr. 2007.

[4] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. Replaying play in and play out: Synthesis of design models from scenarios by learning. In O. Grumberg and M. Huth, editors, *Proceedings of the* 13^{th} *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, Braga, Portugal, Mar. 2007. Springer.

[5] B. Bollig, C. Kern, M. Schlütter, and V. Stolz. MSCan: A tool for analyzing MSC specifications. In *TACAS 2006*, volume 3920 of *Lecture Notes in Computer Science*, pages 455–458. Springer, 2006.

[6] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. of the ACM*, 30(2):323–342, 1983.

[7] L. Brim and M. Leucker. Parallel model checking and the FMICS-jETI platform. In *Proc. of* 12^{th} *Int. Conf. on Engineering of Complex Computer Systems (ICECCS'07)*, Auckland, New Zealand, July 11 - 14 2007.

[8] S. Chaki and O. Strichman. Optimized l*-based assume-guarantee reasoning. In *(to appear) Proc. of the* 19^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 276–291, 2007.

[9] J. E. Cook, Z. Du, C. Liu, and A. L. Wolf. Discovering models of behavior for concurrent systems. Technical report, New Mexico State University, Deppartment of Computer Science, August 2002. NMSU-CS-2002-010.

[10] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *(TOSEM) ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[11] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, September 2005.

[12] M. del Mar Gallardo, C. Joubert, P. Merino, and D. Sanán. On the fly model checking for C programs with extended CADP in FMICS-jETI. In *Proc. of* 12^{th} *Int. Conf. on Engineering of Complex Computer Systems (ICECCS'07)*, 2007.

[13] M. Gheorghiu, D. Giannakopoulou, and C. Păsăreanu. Refining interface alphabets for compositional verification. In *Proc. of the* 19^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 276–291, 2007.

[14] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In H. W. R. Kutsche, editor, *Proc. of the* 5^{th} *Int. Conf. on Fundamental Approaches to Software Engineering (FASE'02)*, volume 2306 of *Lecture Notes in Computer Science*, pages 80–95, Heidelberg, Germany, April 2002. Springer-Verlag.

[15] M. G. Hinchey, J. L. Rash, and C. A. Rouff. A formal approach to requirements-based programming. In *Proc. of the* 12^{th} *IEEE Int. Confand Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, pages 339–345, Washington, DC, USA, 2005. IEEE Computer Society.

[16] C. Hoare and J. Misra. Vision of a grand challenge project. In *Verified Software: Theories, Tools, Experiments*, July 2005.

[17] S. Jörges, C. Kubczak, R. Nagel, T. Margaria, and B. Steffen. Model-driven development with the jABC. In *Proc. of Haifa Verification Conference 2006 (HVC'06)*, LNCS, Haifa, Israel, October 23-26 2006. IBM, Springer Verlag.

[18] C. Kubczak, T. Margaria, A. Fischer, and B. Steffen. Biological LC/MS preprocessing and analysis with jABC, jETi, and xcms. In *(to appear) Proc. of.* 2^{nd} *IEEE-EASST Int. symp. On Leveraging Applications of formal methods, verification, and validation, (ISoLA'06)*, pages 308–313, Paphos (CY), Sept. 15-19 2006. IEEE CS Press.

[19] C. Kubczak, R. Nagel, T. Margaria, and B. Steffen. The jABC approach to mediation and choreography. In *Semantic Web Services Challenge 2006, Phase I Workshops, DERI*. Stanford University, Mar. 2006.

[20] C. Kubczak, B. Steffen, and T. Margaria. The jABC approach to mediation and choreography. 2^{nd} *Semantic Web Service Challenge Workshop.*, June 2006.

[21] Y. Lafon. W3c web services activity. `http://www.w3.org/2002/ws/`, seen Apr. 2007.

[22] A.-L. Lamprecht, T. Margaria, and B. Steffen. Data-flow analysis as model checking within the jABC. In A. Mycroft and A. Zeller, editors, *Proc. of* 15^{th} *Int. Conf. on Compiler Construction, (CC'06)*, volume 3923 of *Lecture Notes in*

Computer Science, pages 101–104, Vienna, Austria, March 30-31 2006. Springer.

[23] Special section on the electronic tool integration platform. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 1, November 1997.

[24] T. Margaria, M. G. Hinchey, H. Raffelt, J. Rash, C. A. Rouff, and B. Steffen. Completing and adapting models of biological processes. In *Proc. of IFIP Conf. on Biologically Inspired Cooperative Computing (BiCC '06)*, Santiago (Chile), 2006.

[25] T. Margaria, M. G. Hinchey, J. L. Rash, C. A. Rouff, and B. Steffen. Enhanced requirements-based programming for complex model completion. In *Proc. of* 5^{th} *Int. Conf. on Applications of Physics in Financial Analysis (APFA5)*, Torino (I), Jun.-Jul. 2006.

[26] T. Margaria, C. Kubczak, M. Njoku, and B. Steffen. Model-based design of distributed collaborative bioinformatics processes in the jabc. In 11^{th} *IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS '06)*, pages 169–176, Stanford, CA, August 2006. IEEE CS Press. iccecs-bio06.

[27] T. Margaria, C. Kubczak, M. Njoku, and B. Steffen. Model-based design of distributed collaborative bioinformatics processes in the jABC. In *Proc. of* 11^{th} *IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06)*, pages 169–176. Stanford University, CA (USA), IEEE Computer Society Press, August 2006.

[28] T. Margaria, C. Kubczak, B. Steffen, and S. Naujokat. The FMICS-jETI platform: Status and perspectives. In *(to appear) Proc. of* 2^{nd} *IEEE-EASST Int. symp. On Leveraging Applications of formal methods, verification, and validation (ISoLA'06)*, pages 414–418, Paphos (CY), Sept. 15-19 2006. IEEE CS Press.

[29] T. Margaria, H. Raffelt, and B. Steffen. Knowledge-based relevance filtering for efficient system-level test-based model generation. *Innovations in Systems and Software Engineering*, 1(2):147–156, July 2005.

[30] T. Margaria, C. Winkler, C. Kubczak, B. Steffen, M. Brambilla, S. Ceri, D. Cerizza, E. D. Valle, F. Facca, and C. Tziviskou. The SWS mediator with WebML/Webratio and jABC/jETI: A comparison. In *Proc of* 9^{th} *Int. Conf. on Enterprise Information Systems (ICEIS'07)*, Funchal (Portugal), June 12–16 2007. (to appear).

[31] L. Mariani and M. Pezzè. A technique for verifying component-based software. In *Proc. of Int. Workshop on Test and Analysis of Component Based Systems (TACoS'04)*, pages 17–30, March 2004.

[32] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. In G. F. A. Cortesi, editor, *Proc. of Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 330–354, Venice, Italy, Sept. 1999. Springer Verlag.

[33] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *ICSE 2000*, pages 35–46. ACM, 2000.

[34] I. Object Management Group. Omg's corba website. `http://www.omg.org/corba/`, seen Apr. 2007.

[35] D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. In J. Wu, S. T. Chanson, and Q. Gao, editors, *Proc. of the Joint Int. Conference on Formal Description Techniques for Distributed System and Communication/Protocols and Protocol Specification, Testing and Verification FORTE/PSTV '99:*, pages 225–240. Kluwer Academic Publishers, 1999.

[36] H. Raffelt and B. Steffen. Learnlib: A library for automata learning and experimentation. In L. Baresi and R. Heckel, editors, *Proc of* 9^{th} *Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2006)*, volume 3922 of *Lecture Notes in Computer Science*, pages 377–380. Springer, 2006.

[37] B. Steffen and H. Hungar. Behavior-based model construction. In S. Mukhopadhyay and L. Zuck, editors, *Proc. of the* 4^{th} *Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *Lecture Notes in Computer Science*, pages 5–19. Springer-Verlag, 2003.

[38] B. Steffen, T. Margaria, and V. Braun. The electronic tool integration platform: concepts and design. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 1:9–30, November 1997. ETI1.

[39] B. Steffen, T. Margaria, V. Braun, and N. Kalt. Hierarchical service definition. In *Annual Review of Communication*, pages 847–856. Int. Engin. Consortium Chicago (USA), IEC, 1997.

[40] B. Steffen, T. Margaria, and R. Nagel. jETI: A tool for remote tool integration. In N. Halbwachs and L. D. Zuck, editors, *Proc. of* 11^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05):*, volume 3440 of *Lecture Notes in Computer Science*, Edinburgh, UK, April 2005. Springer Verlag.

[41] B. Steffen, T. Margaria, and R. Nagel. Remote Integration and Coordination of Verification Tools in jETI. In *Proc. of* 12^{th} *IEEE Int. Conf. on the Engineering of Computer Based Systems (ECBS 2005)*, pages 431–436, Greenbelt (USA), April 2005. IEEE Computer Soc. Press. ECBS-jeti.

[42] B. Steffen, T. Margaria, H. Raffelt, and O. Niese. Efficient test-based model generation of legacy systems. In *Proc. of the* 9^{th} *IEEE Int. Workshop on High Level Design Validation and Test (HLDVT'04)*, pages 95–100, Sonoma (CA), USA, November 2004. IEEE Computer Society Press.

[43] C. Topnik, E. Wilhelm, T. Margaria, and B. Steffen. jmosel: A stand-alone tool and jABC plugin for m2l(str). In *Proc. SPIN 2006, 13th International SPIN Workshop on Model Checking of Software*, volume 3925 of *LNCS*, pages 293 – 298, Vienna (A), April 2006. Springer Verlag.

[44] T. Xie and D. Notkin. Mutually enhancing test generation and specification inference. In A. Petrenk and A. Ulrich, editors, *Proc. of* 3^{rd} *Int. Workshop on Formal Approaches to Testing of Software (FATES'03)*, volume 2931 of *Lecture Notes in Computer Science*, pages 60–69. Springer Verag, 2004.

[45] Home page of the ERCIM working group on formal methods for industrial critical systems (FMICS). `http://www.inrialpes.fr/vasy/fmics/`.

[46] jABC website. `http://www.jabc.de`, seen Apr. 2007.

[47] Hompage of αspin. `http://www.lcc.uma.es/~gisum/fmse/tools/`.

[48] Semantic web services challenge. `http://sws-challenge.org`, seen Apr. 2007.