Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 133 (2005) 217–236
www.elsevier.com/locate/entcs

Integration of Reliability and Performance Analyses for Active Network Services

María del Mar Gallardo, Jesús Martínez
Pedro Merino, Guillermo Rodríguez

Dpto. de Lenguajes y Ciencias de la Computacion
University of Malaga, 29071 Malaga, Spain

Abstract

Increasing the quality of software for new telecommunication services requires the joint use of different testing techniques. For instance, automatic verification and performance evaluation are necessary to ensure desired throughput and reliability. However, both kinds of analysis were traditionally performed without sharing a common description of the system, and much work and time was wasted constructing different specifications oriented to particular tools.
In recent years a lot of research has been carried out to design languages and tools to manage both functional and performance analysis with only one description, specially within communities devoted to process algebras and Petri nets. These homogeneous frameworks remove the inconveniences of maintaining a set of specifications for the same software.
In the paper, we explore an alternative approach to keep the specification of different aspects to be analyzed consistent. Taking into account the number and quality of existing tools for modelling and analyzing telecommunication software, we explore semi-automatic methods to integrate these tools in a way that is as transparent as possible for users. Ideally, the designer of new services will provides only one description of the software with its most familiar language, and a toolset will generate particular specifications to analyze each aspect of interest (currently, reliability and performance). Our proposal takes advantages of recent work on model-driven architecture (MDA) and XML for automating tool construction. Its applicability is shown in the context of developing new services with the active network paradigm, integrating the features of the model checker SPIN and the network simulator *ns2*.

Keywords: Software testing, specification, verification, performance analysis, MDA, XML

[1] Email: {gallardo,jmcruz,pedro}@lcc.uma.es, guille@iies.es

doi:10.1016/j.entcs.2004.08.066

1 Introduction

An important issue in the formal methods community is to avoid several (probably unrelated) descriptions of a given system, each one oriented to a given task (safety or reliability analysis, performance analysis, code generation, etc.). Using only one formal description for all development tasks seems to be a very ambitious trend; however, there are some languages and tools that successfully cover several of these tasks. This is the case of tools like Times [1] and UPPAAL [11] or formal specification languages like MoDeST [13], that in particular integrate time, schedulability or performance features with traditional reachability analysis. This approach of a *single language* (and a single description of the system) clearly keeps all the aspects to be analyzed consistent, but with the cost of specific algorithms and tools for each kind of property to be analyzed. It is also worth noting that the resulting language could be hard to use by non-experts in the field of formal methods.

One alternative approach to keep the piece of formal descriptions consistent and to avoid implementing new algorithms or adding complexity to the languages is the *tool integration* approach. In this large category we identify two main lines of work.

The first one consists of building environments to encapsulate tool functionalities, working with internal translators between source and the corresponding destination tool. This is the approach followed by the well-known ETI coordination platform [39].

The second method for integration is based on defining *intermediate representation languages* (usually, new formal methods) that group features which are common to the majority of tools and existing formal methods. These languages are not usually oriented to users but to tools that produce this internal representation from final user oriented languages (including both specification and programming languages). This method was initially followed in the projects SPECS [34] and SEDOS [14], and more recently it is again followed with projects such as CADP [16], Bandera [22], IF [5] and Veritech [36]. Their intermediate languages allow the exchange of information among tools for common tasks like model checking, automatic code and documentation generation, static analysis, syntax checking or reduction techniques.

Following this second approach, in this work we consider three main objectives:

- To integrate existing and efficient tools for analyzing protocols and software for telecommunication services.
- To keep only one description of the system to be analyzed, avoiding several (potentially) inconsistent specifications.

- To reduce the number of translators and manage their complexity and possible evolution with respect to the use of intermediate proprietary languages.

These objectives can be reached using standard representation languages and technologies, like the eXtended Markup Language (XML) [42] and the model-driven architecture (MDA) [29] to integrate tools. We have experience with using XML technologies to implement one of the hot topics in formal methods: abstract model checking [10]. The first successful story [19] was the use of XML as the internal representation to support the transformation of the system specifications for SPIN. In that way we extended the model checker SPIN to implement *data abstraction*, integrating abstraction and model checking features.

A second step [20] was the use of XML Model Interchange (XMI) to allow the interchange of data among tools for Statecharts. In that work, we proposed the use of XMI to create an abstraction plug-in for STATEMATE. In both cases, a major benefit is the possibility of completely reusing the model-checking tool, without modifying its internal code.

Following the XML evolution through XMI, the third natural step is to explore the features of the new OMG MDA-based standard called Meta Object Facility Specification (MOF). This recommendation defines an abstract language and a framework for specifying and managing models and metamodels. In the MOF context, a model is a representation of a system (e.g., a software design), containing so-called metadata. The metamodel is the description of the structure followed by models, defining their abstract syntax and static semantics. Therefore, MOF language allows the creation of metamodels following precise, well-defined and common meta-metamodel facilities. In the XML domain, DTDs or XML Schemas are also metamodels, both defining the structure of valid documents (models) that contain tagged data (metadata). Regarding XMI, this language is part of the MOF standard, and supplies the mechanisms to interchange metadata and metamodels using XML, as a way to integrate tools. In addition, MOF defines a way to manipulate XMI metadata using programming APIs in CORBA or Java. These libraries may be generated automatically taken a metamodel as reference.

In the paper, we propose the use of MDA/MOF and XML as a way to integrate existing tools for the analysis of complex systems. We present a methodology to obtain the suitable input for the tools to be integrated, avoiding the need for several hand-made specifications. Therefore, we may define a metamodel (MOF-based or described in XML Schema) for describing common features of specification and programming languages, including communication and synchronization for concurrency. Using this approach, we will consider this metamodel as our intermediate representation language, which

will benefit from XML technologies to help in the development of parsers or code generators. In order to test the viability of our approach, we have chosen the domain of telecommunication services, and particularly the active network paradigm. In this way, we are reusing and comparing experiences with the previous work in our group: a) in [35] performance/traffic analysis of active networks using *ns2*; b) in [18] we use SPIN to analyze reliability [23,9,31]. This work can be considered a step towards the creation of a more flexible interchange language to perform different kinds of analysis, using other existing tools with complementary features.

The paper is organized as follows. The following section describes our previous experiences with analyzing active networks, using different languages and tools to describe performance and reliability models. Section 3 introduces our study of MDA and XML to support the integration of the previous analysis, along with a methodology that will allow us to generate code for tools in a way as much transparent as possible for users. Section 4 gives an overview of related work and finally, Section 5 enumerates our conclusions and future work.

2 Modelling and Analyzing Software for Active Networks

Active networks [38,6,15] open network nodes to the user defined code. This approach offers flexibility to develop new telecommunication services without the slow standardization process usually required in standardization institutions (ITU, IETF, IEEE, ANSI, etc). However, opening the nodes implies ensuring that the user defined code can not damage the reliability [23,9,31] of network or degrade its performance. Therefore, there is a need for testing tools for developing new active services, assuring their quality before they are implemented in a real active platform. The integration of several tools oriented to specific kinds of properties or aspects is clearly a major desire of developers. This integration allows the designer to write the code or the specification only once, avoiding the definition of particular specifications depending on the tool.

This section contains an overview of the best known proposal for programming active networks, along with our previous work on analyzing functional behavior with model checking and performance with network simulators. Both works are used in the next section to explain our proposal for integration.

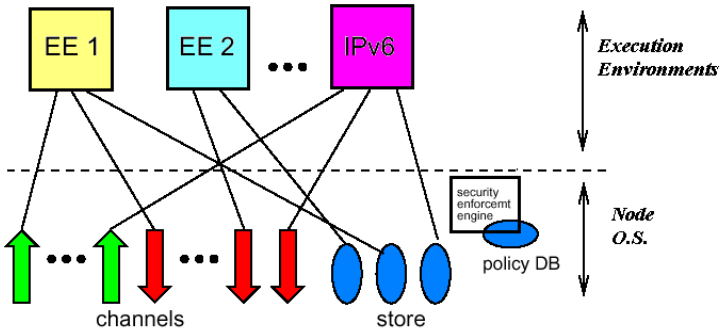

Fig. 1. Active Network Architecture

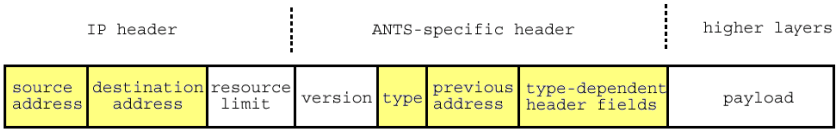

Fig. 2. The capsule format used by ANTS

Capsule Manipulation	Routing Control	Storage/Environment
getSrc()	sendTo(dest)	getAddr()
getDst()	sendCapsuleTo(dest,ref)	cacheGet(key)
setDst(dest)		cachePut(key,val)
getPrev()		cacheRemove(key)
newCapsule(ref,src)		

Table 1
Node API for capsules

2.1 Active Networks with the Capsule Approach

The functionality of an active network node is divided between the *Execution Environment* (EE), responsible for providing network abstractions, and the node operating system (NodeOS), which manages access to the network resources. As shown in fig. 1, this architecture allows for multiple EEs to be present on an active node. Each environment may provide a specific API to execute the code in the packets. The NodeOS is responsible for implementing the set of abstractions that will give access to the node resources. This access is protected by a security enforcement engine that requests the code's credentials before performing critical tasks. The operating system also gives access to the communication channels for sending and receiving packets, along with some storage facilities, usually consisting of some kind of soft-store cache.

Probably the most successful programming model, if we consider the applications developed with it, is the ANTS (Active Network Transport System)[41] model and its corresponding toolkit. With ANTS, protocols are automatically deployed by a mobile code technique at both the intermediate nodes and the end systems. Packets, called *capsules*, are processed according to their specific code. This code can use operations available in the node as four kinds of primitives: *capsule manipulation* (accessing the header and the payload), *control operations* (allowing the capsule to create new capsules and forward, copy and discard themselves), *environment access* (reading state information in the node, like its address) and *node storage* (manipulating the soft-store of application defined objects). All these primitives are summarized in table 1.

In ANTS, every capsule is identified with two values (see fig.2), the type of capsule and the protocol (the application). One protocol is supposed to be implemented with a number of instances of different kinds of capsules. The capsule is processed as follows. The code to process the capsule is defined by the sender and can not change within the network. The processing routine for the capsule has limited capabilities, since it is defined by distrusted users. The capsule is forwarded by non-active nodes (for instance, standard Internet router) using routing information, but the code is only executed at particular nodes. On receiving the capsule, active nodes execute its associated routine. The capsule itself decides whether it will continue to be forwarded to destination. This decision is usually included at the end of the processing code. The mechanism to transport the code depends on the real implementation. The proposal suggested for ANTS is to load the code on demand and to cache it to improve performance. By default, the soft-storage in the nodes is only shared by the capsules for the same protocol.

In practice, new services developed with ANTS are implemented in Java, and no previous analysis is done. Checking correctness of the services (for functional and timing properties) is usually done in the real network. Due to the lack of tools and methodologies for a priori analysis of active protocols, we have worked in the following two methods.

2.2 Reliability Analysis with the SPIN Model Checker

In [18], we developed a framework to describe and verify active services using SPIN, a very efficient and widely known tool (it received the ACM System Software Award for 2001, the same award given to well-known developments like TCP/IP, World-Wide Web, UNIX or Java).

Our contribution in that work consists of a) describing the fixed part of the PROMELA model representing the active node and b) a methodology to construct the PROMELA model for capsules, links and topology. In that way,

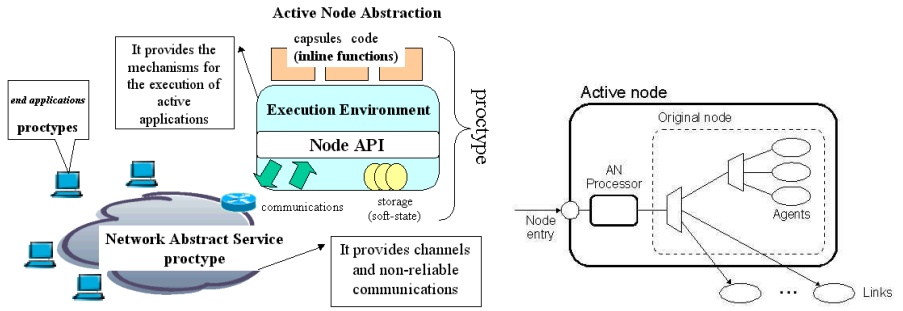

Fig. 3. Active node abstractions for SPIN (left) and *ns2* (right)

the designer only needs to append the code for the capsules. The architecture of each PROMELA model is represented in fig. 3 (left). The fixed part is composed of the Execution Environment (for each node) and the Network Abstract Service (which can be instantiated with different topologies). The user writes the PROMELA code for capsules and end applications (the ones running on user computers).

A typical sample application in active networks is the multicast service. When receivers want to be incorporated to a multicast session, they must send a *subscription* capsule to the sender. Along the way, active nodes append their identifiers to a list, also acting as fake receivers when propagating the capsule through the network. This operation creates a tree with the root being the sender and the leaves being the real receiver hosts. The PROMELA version of this capsule is described in fig. 4 (left), where the node updates the variable `distribution_group` in its cache memory, upon reception of this capsule. When a *data* capsule is received by an active node, it only has to resend it to every receiver contained in its distribution group.

In [18] we presented a complete specification for the RMANP protocol [8]. We also used temporal logic for very critical properties in this protocol. The whole code can be downloaded from `http://www.lcc.uma.es/gisum/active`. Our experience shows that the proposed scheme to use SPIN is suitable to analyze functional properties for active networks, but more automation is desirable to help in constructing the code for capsules.

2.3 Performance Analysis with the ns2 Network Simulator

The *ns2* network simulator [40] is a multi-protocol, object-oriented, programmable simulator which also includes a number of facilities for large-scale simulations and the capability to interface the simulator to a live network. Its architecture is designed in a way intended to promote extension by users. This ability allowed us to extend it for analyzing traffic generated by active services

```
/* Promela specification for SPIN

    capsule = EE global
    capsule.reverse = (not used in verification)
    MULTICAST (not used in verification)

    Initially sent towards the mcast sender
*/

inline capsuleSUBSCRIBE(){

short key,      /* index for soft-store */
      value,    /* the route */
      tmp;
atomic{
   key = capsule.group;
   value = cacheGet(key);
}

if
::(value == FREE) -> value = 0;
::else
fi;

/* are we at an intermediate node? */
atomic{
 tmp = 1 << getPrev();
 value = value | tmp;
 cachePut(key,value);
}

/*explicit discard of explicit forwarding */

sendTo(getDst());

return0:
key=0; value=0; tmp=0;

}
```

```
| # TCL specification for ns2
| #
| #   group = mcast group
| #   sender = mcast sender
| #   reverse = last visited node
| #
| # Initially sent towards the mcast sender
| #
| proc capsuleSUBSCRIBE { capsule ee } {
|   set group [lindex $capsule 0]
|   set sender [lindex $capsule 1]
|   set reverse [lindex $calsule 2]
|
|   # first, look up forwarding record or create one
|   set m [$ee cacheGet [key $group $sender]]
|   if { $m == "" } {
|     set m -1
|     $ee cachePut [key $group $sender] $m $MCAST_TTL
|   }
|
|   # are we at an intermediate node?
|   if { $reverse != "" } {
|     set nodes [lrange $m 1 [llength $m]]
|     if { [lsearch $nodes $reverse] == -1 } {
|       # the list does not contain our info?
|       lappend m $reverse
|       $ee cachePut [key $group $sender] $m $MCAST_TTL
|     }
|   }
|   set age [expr ([$ee getTime] - [lindex $m 0])]
|
|   # explicit discard or implicit forwarding
|   if { [lindex $m 0] != -1 && $age < $RATE } {
|     $ee discard
|   } else {
|     set m [lreplace $m 0 0 [$ee getTime]]
|     $ee cachePut [key $group $sender] $m $MCAST_TTL
|     $ee setData "$group $sender [$ee getAddr]"
|   # sendTo() is implicit
|   }
%| }
```

Fig. 4. Subscription capsule for SPIN (top) and *ns2* (bottom)

[35]. In [35] we presented how to extend the network simulator itself and also how to define the capsules for each new services. In particular, the simulator C++ code is extended to implement the active node behavior, following a scheme like the one in fig. 3 (right). This node offers an API similar to the one in table 1 for PROMELA.

The new API offered by the active node, following *ns2* methodology, should be accessed from OTcl scripts that represent the capsules. For instance, the code that simulates the *subscribe* capsule is described in fig. 4 (right).

With the new extension to *ns2* it is possible to obtain data on network traffic, response time and other metrics for performance. There are complete descriptions of realistic problems at `http://www.lcc.uma.es/gisum/active`), including the RMANP protocol (which is also specified in PROMELA as explained above). Like in the case of SPIN, more automation is desirable to construct the code for capsules and for the scenarios to be simulated.

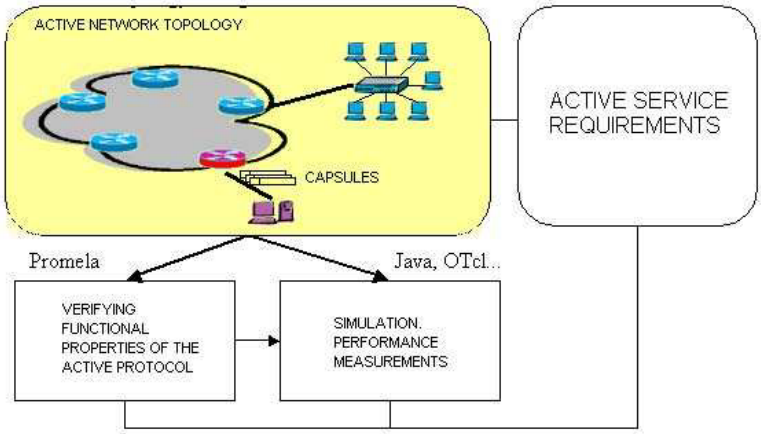

Fig. 5. Integrating *ns2* and SPIN

3 An MDA/XML approach to integrate SPIN and ns2

The MDA paradigm uses the well-known idea of separating the specification of the operation of a system from the implementation details, that is, from how that system will use the capabilities of a concrete platform or application to perform that operation. As mentioned before, the main concept used in MDA is the model, a specification of a system for certain purposes. The model-driven approach to system development uses models to direct its design, construction, maintenance, testing and modification.

Along with models, MDA also defines some important key elements, like the metamodelling and its viewpoints, along with some description-related standards like the meta object facility specification (MOF), the XML metadata interchange specification (XMI) and the object constraint language (OCL).

The metamodelling approach consists of using an abstract language and a visual framework to describe the abstract syntax of valid models, defining so-called metamodels, in a way similar to the BNF rules used to parse structured texts or documents, like a source code. In order to establish semantic rules, metamodels and models can be enforced by using the object constraint language.

The meta object facility (MOF) is the framework for creating metamodels, including standard technology mappings to describe MOF metamodels in XML formats for the interchange among tools. The XML Metadata Interchange (XMI) specification defines technology mapping from MOF metamodels to XML document type definitions (DTD) or XML Schema. The metamodeling architec-

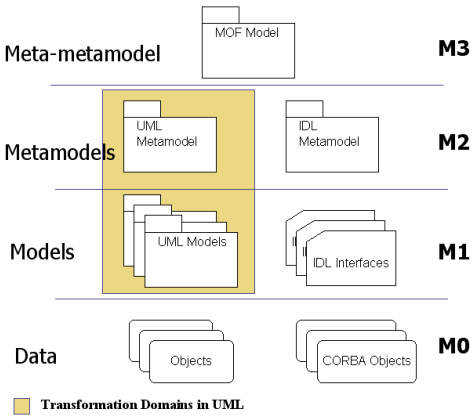

Fig. 6. MOF metamodeling architecture

ture depicted in fig. 6 is organized using different levels. The meta-metamodel level incorporates the abstract syntax suitable for describing metamodels. For example, the UML language conforms to its own metamodel, as shown in the figure.

The viewpoints are techniques for establishing simplified models using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system representation. From different viewpoints, MDA considers a platform independent viewpoint and a platform dependent one. Therefore, the platform independent viewpoint will share all common features of a model (so-called platform independent model or PIM) which are independent from the tool or platform in which the model will be finally implemented (these are platform specific models or PSMs). Usually, a PIM conforms to a specific domain metamodel, and a PSM conforms to a reduced metamodel which shares the core of the PIM metamodel along with some architectural concepts concerning the target platform. Therefore, a platform specific model is oriented to automatic code generation.

MDA-based technology will make it easier to work with a single model (from a PIM perspective) and find automatic ways to generate analysis models (PSMs) for tools that perform reliability and performance checks. In the following subsection, we propose an MDA-based methodology to carry out this task.

3.1 Developing active network services using MDA

We have identified some key points in order to use MDA and XML to perform our analysis tasks in the context of developing new services with the active

Metamodel	Network Topology	Capsule Behavior	Analysis Description
Type	MOF	XML Schema (future MOF)	MOF (in study)
Allow static analysis	not required	yes, αSPIN engine	
Parser generation	fully automatic (XMI-JMI)	fully automatic (SAX-DOM)	
Code extractors	PROMELA, OTcl, SDL, ANTS	PROMELA, OTcl, SDL	
User-friendly front-end	visual (in progress)	may use XSpin	

Table 2
Implementation stages of the methodology proposed

network paradigm:

- Define a metamodel to describe network topologies of active environments: active nodes, links, capsules, NodeOS APIs.
- Define a metamodel to describe the execution flow of capsules and active applications, introducing common elements of programming languages along with communication facilities.
- Describe the differences between a PIM active network model and its corresponding PSMs for both *ns2* and SPIN.
- Automatic code generation: proper guidance to mapping PSMs to the final code used as input of those analyzers.

Table 2 summarizes the important stages in the development of the final toolset. Each column represents a metamodel, one for topologies and capsules and a new one for analysis descriptions (now under study). In order to integrate a concrete tool, it will be necessary to implement some basic facilities for manipulating (reading/writing) metadata, along with proper code generators to create input in the format required by the tool. Following the proposed methodology, our metamodels benefit from fully automatic parser generation. XML metadata can also be analyzed using static analyzers' common features, like those included in αSPIN[19]. We also propose a framework that creates code generators. The existing PROMELA code generator is the reference to implement a new one for OTcl, although implementations for other languages are now being considered.

Although this work is focused on the study of techniques to integrate tools, we recognize the utility of developing a visual environment to create model instances of the proposed metamodels. This environment can be similar to any existing editor for UML (Statecharts and Activity Diagrams), but adapted to our analysis perspective.

3.1.1 A metamodel for active network topologies

From a metamodelling point of view, we have separated the description of the agents involved in an active network from the behavior of capsules and end-

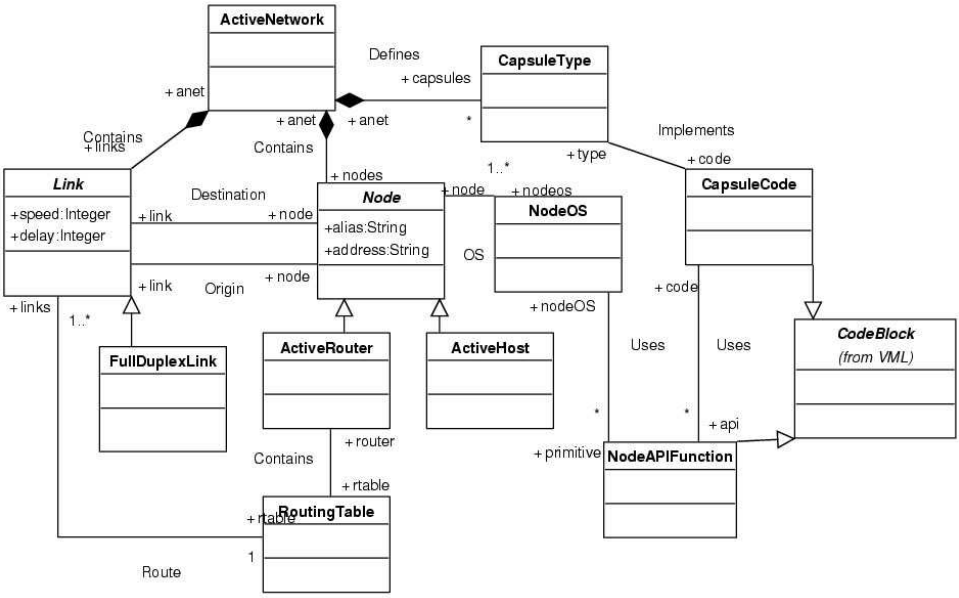

Fig. 7. MOF metamodel for active network topologies

applications. Using this approach, we work with two small metamodels that facilitate the way in which we will describe both scenarios and programmatic facilities. Therefore, topology models will conform to the MOF metamodel partially depicted in fig. 7. The figure shows that an active network is composed of links, nodes and capsules. The element node is specialized in the form of active routers, which will include routing information facilities of a routing table, and active hosts that will execute end applications. The active network will define an active service using instances of `CapsuleType`. The code associated with a concrete capsule type is incorporated using an instance of `CapsuleCode`, which inherits functionality and constraints corresponding to a flow diagram to represent source code in a platform-independent way. The same applies to the description of the node operating system facilities mentioned before in table 1.

3.1.2 A metamodel for capsules and end-application behavior

This is an important element in our methodology, because this metamodel has to describe models using traditional sentences (including control flow) from imperative languages as well as rules to express concurrency and interprocess communication facilities. Our starting point has been the DTD used to structure PROMELA models in XML[19]. Before jumping to the definition of

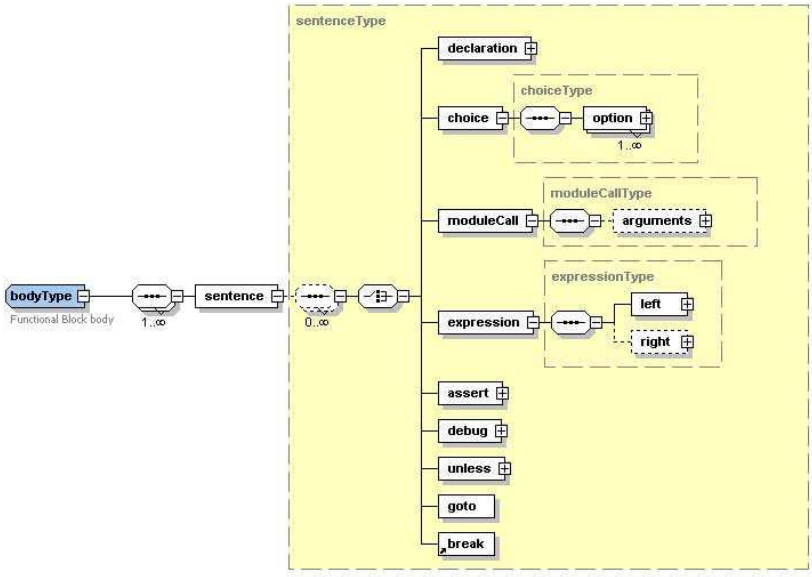

Fig. 8. XML Schema to describe functional block behavior (excerpts.)

a completely new MOF metamodel we have built a XML Schema with improvements on the existing PROMELA DTD, following the suggestions of the future XMI 2.0 standard [30]. This recent recommendation establishes mappings to port an XML Schema to a MOF metamodel and hopefully, this capability will be supported in the near future by new versions of popular CASE and XML tools.

Fig. 8 shows part of the redesigned metamodel. We have defined elements to describe software modules (parameterized code blocks) along with their possible interactions (function calls, message passing) and primitive and user-defined data types. The use of Schema complex types may be considered as an extension mechanism that allows the creation of new elements without modifying the original structure of the metamodel. For instance, the figure depicts the *code block body* type. It is composed of sentence elements which include expressions, variable declarations and module calls or control flow, among others. A very interesting feature of this metamodel is the treatment of expressions as trees, with left and right parts being also of type *expressionType*. Therefore, we may use XPath facilities to search into a model and perform any kind of static analysis and manipulation, as done in αSPIN[19]. It is expected that MOF models will have a way to be queried in the same manner that XPath is used with XML, and this is the objective of the OMG's forthcoming *Query-View-Transformation* language (QVT).

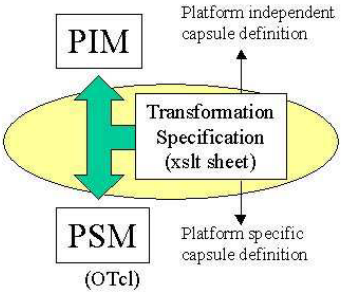

Fig. 9. MDA guidelines for model transformation, using XSL templates

The future adoption of a MOF-based metamodel will solve some expressiveness limitations of the XML Schema language, using OCL constraints to avoid any ambiguity in the definition of new model instances. Moreover, the previous validation of a model against OCL static semantics rules will reduce the complexity of our code generators.

3.1.3 From instances to the target platform

The metamodels described in the previous subsections structure the way in which models will be represented and interchanged among tools. The metadata embedded within topology and capsule model instances can be considered platform-independent. In order to prepare the code generation for the *ns2* simulator and SPIN, we have to perform a previous step to map the PIM to its corresponding PSM, because decoupling code generators from the application domain being considered will allow these to be completely reusable. The model transformation task should be fully automated, as recommended by MDA (see fig. 9), with the aid of some mapping rules or engine. Therefore, the QVT language would also define transformation specifications between models. These specifications are expected to guide the automatic generation of an engine to perform the transformation. Unfortunately, the adoption of a proper QVT language is in its early phase, so we use an approach based on XSL templates to generate final PSM models.

As mentioned before, a PSM model addresses some particularities needed by the destination platform. For example, the description of a capsule in *ns2*, like the one shown in fig. 4, will include the associated execution environment object reference as input argument. This is obviously a special feature of the implementation of capsules in the *ns2* active simulator [35], and will only be needed when generating the final OTcl code.

We are now developing and experimenting with more mappings (also em-

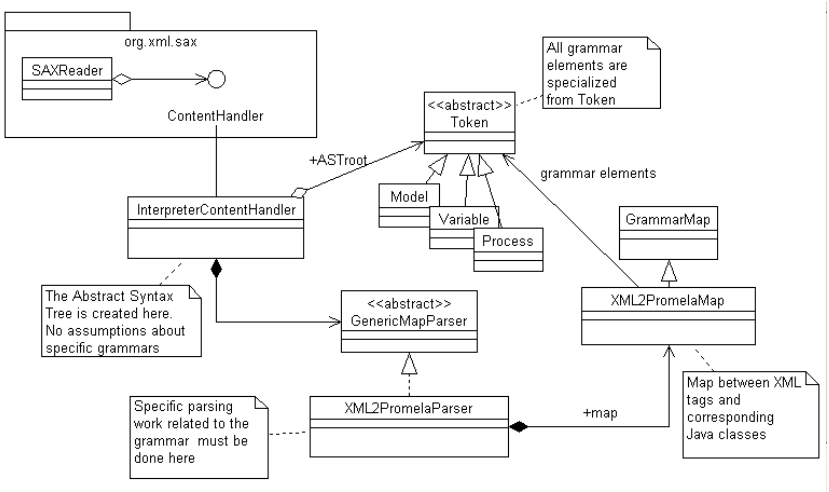

Fig. 10. The interpreter framework for automatic code generation

bedded into XSL), trying to reduce to a minimum the need for user code manipulation. For instance, an interesting verification-oriented feature will be the automatic reduction of an active network topology to one tractable by the model checking algorithms, in order to avoid the state explosion problem.

3.1.4 Generating the inputs for ns2 *and* SPIN

The use of XML descriptions to generate source code is very common due to the facilities and tools standardized by the world wide web consortium such as the SAX and DOM parsers, the XSL-transformation templates or the XPath language to search for elements and attributes. In our methodology, both topologies and capsule models are available in XML and can be manipulated in a flexible manner. In order to implement code generators, we have followed our previous experience in the `XML2Promela` module of αSPIN, reusing a *lightweight interpreter framework* to develop a `XML2OTcl` interpreter, which will create the OTcl code corresponding to *ns2* descriptions for capsules. The architecture of the interpreter framework is shown in fig. 10. It is worth noting that the framework is generic enough to allow the development of new code generators, extending `GrammarMap` and `GenericMapParser` classes. As part of recent ongoing work, we are experimenting with a new SDL [25] generator from behavior model instances.

4 Related work

The main contribution of the paper is the methodology to use standards defined by OMG to support the integration of tools for analysis, and its application to active networks. In this section, we discuss other works related to the main topics of the paper.

4.1 Integrating formal methods

In the context of formal methods, our proposal has the same motivation as projects like Sedos, SPECS, Bandera, CADP and IF. All these projects consider intermediate languages to produce suitable inputs for different tasks like model checking, code generation, static analysis, bisimulation, etc. They consider a wide range of user-level languages (to be translated into the internal one). For example, IF is able to process SDL and several variants of UML.

The main drawback of these projects is that they are based on closed intermediate proprietary languages, and they cannot directly exploit new developments in OMG technologies. Is is worth noting that many industrial tools now exploiting these technologies, even some of the companies related to StateCharts, SDL or UML (like Telelogic, I-logix, Rational), are now evolving to support MDA.

The ETI approach needs a separate discussion because it is not based on intermediate languages, but on translations among independent tools. It offers a coordination language and an open API to adapt the functionality of each tool to the platform.

Finally, mention must be made of the approaches based on a single language to describe all the aspects to be analyzed, like MoDest, UPPAAL and Times. In our opinion they are actually more concerned with the definition of new closed languages than with integrating existing formal methods. We think these powerful tools could be considered as elements to be integrated with the MDA approach, in such a way that we could consider them as specific platforms (PSMs) for specific tasks. In this sense, we plan to use UPPAAL or Times for the kinds of properties not covered by SPIN.

4.2 MDA for tool integration

A tool integration process implies a transformation between the source and target format. Regarding the use of MDA and related technologies, we have to mention the increasing interest for using UML (and MOF) to support these transformations in an automatic way. This is a main topic in the *Workshop on transformation in* UML. Therefore, recent works on integration focus on

model transformation, being mainly related with levels M2 and M1 defined in the MOF Metadata Architecture (fig. 6). The works in [12,21,32] follow a *metamodelling* approach. They change the origin and destination metamodels representing the model after transforming it. These proposals correspond to the M2 level. An alternative is the *model-to-model* approach, which considers transformations within the same metamodel context [33], i.e. performing model refinement, abstraction or refactoring. These proposals correspond to the M1 level depicted in the figure. Considering this classification, our proposal for obtaining PSM models corresponds to the second approach.

Despite these previous works, as MDA is a relatively recent proposal, as far as we know, it has not been employed yet to integrate model checking tools with other analysis techniques.

4.3 Analyzing Active Networks

We have chosen our own proposals in [18] and [35] to perform the integrated analysis of active networks. However there are other papers on these subjects.

The use of formal methods to ensure reliability for active networks based services has been considered in [3,27,4,37]. Network and protocol performance with network simulators has been considered in [2,15,28,26]. We think that our methodology is open and flexible enough to integrate the other works. In particular, we could change the PSMs to produce the specifications considered in those previous works.

5 Conclusions and Future work

The testing phase of a model for a software system involves a cycle in which some results of a test may suggest the introduction of changes to the original system, and then start the cycle again. These changes are usually carried out manually by the tester; for example, manipulating parts of the source code representing the system. The problem arises when using different testing platforms and different representations of the system modelled, as it is difficult to manage the changes needed for each model in order to assure the proper consistency.

We have proposed a methodology that uses standards defined by OMG and W3C to support the integration of tools for the analysis of complex systems, in such a way that the designer will only manage one description of the system. Therefore, our main goal is to define an interchange language to describe common aspects to specification and programming languages for concurrent systems. In particular we are applying MDA and XML concepts to a specific domain application area not usually covered by the MDA community: integrating

(formal methods-based) automatic verification with other testing techniques. We have noticed that the methodology is suitable to integrate reliability and performance analysis in the context of active networks, and it will simplify the development of parsers, code generators and filters. We are currently implementing a complete toolset to describe topologies and the code for the capsules (new results will be available at http://www.lcc.uma.es/gisum/active).

Our future work will extend the behavior metamodel to allow the integration with other analysis tools. The metamodel is flexible enough to incorporate new data types, sentences and expressions, without more repercussion into existing parsers and code generators. Moreover, we will have to create new mapping rules for the specific PSMs that will automate the generation of code for new verification tools. As mentioned before, we are also planning to create a new metamodel to interchange metadata regarding analysis tests and results.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, W. Yi. TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS 2003, Marseille, France, September 6-7, 2003.

[2] S. Bhattacharjee, K. L. Calvert, E. W. Zegura, Self-Organizing Wide-Area Network Caches, IEEE INFOCOM '98, San Francisco, CA, March, 1998.

[3] S. Bhattacharjee, K. Calvert, E. Zegura. Reasoning about active network protocols. *IEEE ICNP'98*, 1998.

[4] K. Bhargavan, C.A. Gunter, M. Kim, I. Lee, D. Obradovic, O. Sokolsky, and M. Viswanathan. Verisim: Formal Analysis of Network Simulations. *IEEE Transactions on Software Engineering*, 28(2), pp. 129-145, 2002.

[5] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier. IF: A Validation Environment for Timed Asynchronous Systems. In *Proceedings of CAV'00*, Springer-Verlag, LNCS 1855, pp. 543-547, July 2000

[6] K. L. Calvert, S. Bhattacharjee , E. Zegura, J. Sterbenz, Directions in Active Network Research, IEEE Communications Magazine 36(10), 1998, pp. 72-78.

[7] K. Chandy, J. Misra, *Parallel Program Design*. Addison-Wesley, 1998.

[8] M. Calderon, M. Sedano, A. Azcorra, C. Alonso. Active Network Support for Multicast Applications. *IEEE Network*, 1998, pp. 46-52.

[9] E.M. Clarke, et al. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4), 1996, pp. 626-643.

[10] D. Dams. Abstraction in Software Model Checking: Principles and Practice, in *9th Int. SPIN Workshop. Model Checking Software*, Springer LNCS 2318, pp. 14-21, 2002.

[11] A. David, G. Behrmann, K. G. Larsen and W. Yi. A Tool architecture for the next generation of UPPAAL. *Formal Methods at the Crossroads: from Panacea to Foundational Support, the proceedings of UNU/IIST 10th Anniversary Colloquium*, Springer LNCS 2757, 2003.

[12] E. Dominguez, A.Rubio, M. Zapata. Mapping models between different modelling languages. Workshop on Integration and Transformation of UML models.2000.

[13] P.R. D'Argenio, H. Hermanns, J.-P. Katoen, and R. Klaren. MoDeST - A Modelling and Description Language for Stochastic Timed Systems, Proc. of Process Algebra and Probabilistic Methods. *Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2001*, LNCS 2165, pp 87-104. 2001.

[14] M. Diaz, C. Vissers, J. Ansart. Sedos Software Environment for the Design of Open Distributed Systems. *The formal Description Technique LOTOS*. North-Holland. 1989.

[15] T. Faber, ACC: Using Active Networking to Enhance Feedback Congestion Control Mechanisms, IEEE Network, 12(3), pp. 61-65, 1998.

[16] J.C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier and M. Sighireanu: CADP: A Protocol Validation and Verification Toolbox,*Proceedings of the 8th Conference on Computer-Aided Verification*, Springer, vol. 1102, pp. 437-440, 1996.

[17] M.M. Gallardo, J. Martinez, P. Merino and E. Rosales, Using XML to implement Abstraction for Model Checking. In *Proc. of ACM Symposium on Applied Computing*, 2002.

[18] M.M. Gallardo, J. Martinez, P. Merino. Model Checking Active Networks with SPIN. *Computer Communications* (to appear).

[19] M.M. Gallardo, J. Martinez, P. Merino, E. Pimentel. aSPIN: A Tool for Abstract Model Checking. *International Journal on Software Tools for Technology Transfer*, 5 (2-3),pp. 165 - 184, 2004.

[20] M.M. Gallardo, J. Martinez, P. Merino, E. Pimentel. Abstracting UML behavioral diagrams for verification. Chapter in In Hongji Yang (ed.), *Software Evolution with UML and XML*, IDEA Group Publishing, 2004, to appear.

[21] A. Gerber, M. Lawley, K. Raymond, J. Steel, A. Wood. Transformation: The Missing Link of MDA. In Lecture notes in computer science (2505). Springer-Verlag, 2002.

[22] Hatcliff, J., Dwyer, M., Pasareanu, C., Robby. Foundations of the bandera abstraction tools. In *The essence of compution* (172-203). Springer Verlag, 2003.

[23] G.J. Holzmann. Design and Validation of Comp. Protocolos. Prentice-Hall, 1991.

[24] G.J. Holzmann. The Model Checker SPIN. *IEEE Trans. on SE*, 23(5), 1997.

[25] ITU-T. Z.100 - Specification and Description Language (SDL), 2000.

[26] S. Kasera et al. Scalable fair reliable multicast using active services. *IEEE Network Magazine*, 2000.

[27] C. Kong, D. Dieckman, P. Alexander. Formal Modeling of Active Network Nodes Using PVS. *Workshop on Formal Methods in Software Practice (FMSP-00)*, 2000.

[28] U. Legedza, D.J. Wetherall, J. V. Guttag. Improving the Performance of Distributed Applications Using Active Network Protocols. *Proc. IEEE INFOCOM'98*, 1998.

[29] Object Management Group. MDA guide version 1.0.1. omg/2003-06-01. June, 2003.

[30] Object Management Group. XML Metadata Interchange, version 2.0. May, 2003.

[31] D. Peled, *Software Reliability Methods*, Springer, 2001.

[32] M. Peltier, J. Bezivin, G. Guillaume. MTRANS: A general framework, based on XSLT, for model transformations. Workshop on Transformation in UML, 2001.

[33] D. Pollet, D. Vojtisek. OCL as a Core UML Transformation Language. Workshop on Integration and Transformation of UML models, 2002.

[34] R. Reed, W. Bouma, J.D. Evans, M. Dauphin , M. Michel eds. *The SPECS Consortium. Specification and Programming Environment for Communication Software*. North-Holland, 1993.

[35] G. Rodríguez, P. Merino, M. M. Gallardo. An extension of the ns simulator for active network research, *Computer Communications 25*, 2002, pp. 189-197.

[36] S. Katz: Faithful Translations among Models and Specifications, *Proc. of Formal Methods Europe*, 2001.

[37] M. Stehr, C. Talcott. PLAN in Maude. Specifying an Active Network Programming Language. *Electronic Notes in theoretical Computer Science* 71, 2002.

[38] D. Tennenhouse, D. Wetherall. Towards an active network architecture, *Computer Communication Review, 26,2*, 1996.

[39] The ETI Platform. Available at http://eti.cs.uni-dormund.de, 2004

[40] The Network Simulator ns-2. Available at http://www.isi.edu/nsnam/ns/ , 2004.

[41] D.J. Wetherall, J. V. Guttag, D. L. Tennenhouse. ANTS: Network Services without the Red Tape, *IEEE Computer*, 1999.

[42] W3Consortium. Extensible Markup Language (XML) 1.0 (Second Edition), available in: http://www.w3.org/XML/, (2000).