Running and Debugging UML Models

Miguel Pinto Luz
IST / INESC, Lisboa, Portugal
miguelluz@acm.org

Alberto Rodrigues da Silva
IST / INESC, Lisboa, Portugal
alberto.silva@acm.org

Abstract

Software development evolution is a history of permanent seeks for raising the abstraction level to new limits overcoming new frontiers. Executable UML (xUML) comes this way as the expectation to achieve the next level in abstraction, offering the capability of deploying a xUML model in a variety of software environments and platforms without any changes. This paper comes as a first expedition inside xUML, exploring the main aspects of its specification including the action languages support and the fundamental MDA compliance. We also explore the model debugging capabilities as a premature means of conceptual fail discovery. In this paper is presented a new xUML tool called XIS-xModels that gives Microsoft Visio new capabilities of running and debugging xUML models.
Keywords: UML, executable UML, Model Debugging, Action Language.

1 INTRODUCTION

Processes, meetings, models, documents, or even code are superfluous artifacts for organizations: all they need is well design and fast implemented working system, that moves them towards a new software development paradigm, based on high level executable models. According this new paradigm it should be possible to reduce development time and costs, and to bring new product quality warranties, only reachable by executing, debugging and testing early design stages (models).

Executable UML (xUML) [Sulistyo 2002, Mellor 2002] together with its Action Language specification [Carter 2002, BridgePoint 2003, ObjectSwitch 2003, OMG 2003] and the model compiler concept adds to the traditional UML the execution and debugging capabilities.

1.1 xUML Overview

xUML [Sulistyo 2002, Mellor 2002]involves an UML profile and the capacity for describing a system's data and behavior. A xUML model doesn't include any coding decisions and can be deployed in various software environments without involving any changes (concept strongly coupled with the platform independent model of model driven architectures).

An xUML model mainly comprises UML class diagrams, UML state charts and a set of procedures, where each procedure is a set of actions. Other UML diagrams can also be used to support the definition of xUML models.

xUML relies on the Precise Action Semantics for UML [OMG 2003], adopted by OMG, to describe actions and sequence of actions. An *action* is the fundamental unit of behavior specification [Sunyé 2001]. An action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty. In addition, some actions modify the state of the system in which the action executes.

Classes as well as associations have actions for creating and deleting instances. Each instance generally has a lifetime. Its behavior is a progression through various stages over time. This is known as the lifecycle of the object. A *state machine* (or *state chart*) formalizes a lifecycle of an object in terms of states, events, and transitions. Each state on the state chart diagram has an associated procedure, comprehending events, calls or actions taking place in that particular state. The general behavior of the system is sequenced by signals between state machines, therefore objects communicate as a result of state machine sending signals each other.

1.2 xUML based on MDA

One important pillar supporting the Model Driven Architecture (MDA) [OMG 2003] initiative announced by the OMG in early 2001, is the Executable UML. MDA depends on three basic notions: (1) the notion of Computational-Independent Model (CIM), a model that describes a business context and business scenarios; (2) the notion of Platform-Independent Model (PIM), a model that does not rely on any implementation technologies; and (3) the notion of Platform-Specific Model (PSM), a model that contains within it the necessary details of the implementation, so that code can be generated from it. The main promises of Model Driven Development (MDD) were [The middleware Company 2003]: (1) shorter time-to-market; (2) increased reuse; (3) fewer defects; (4) systems are easier to understand; (5) good system documentation; and (6) ability to inexpensively explore architecture and design alternatives.

1.3 The Problem and Motivation

UML is a rich language that can be used for problem conceptualization, software system design, as well as implementation. UML also covers a wide range of issues from use cases and scenarios, to state behavior and operation declaration. However, UML currently uses un-interpreted strings to capture the description of the behavior of actions and operations [Glinz 2001, Stevens 2003]. To provide sharing of action semantics and operation behavior between UML modelers and UML tools, there needs to be a way to define this behavior precisely, and in a standardized language. Further, action specifications should not be overly constrained by programming-language specific details.

Designers create application models, which are independent of the implementation technology. Developers look at the models and write the production code, adding design detail. But once something is in a machine representation we shouldn't have to manually reinterpret it into another language. The designers' models don't have to be complete, correct, or consistent. Keeping the models in sync with the code is an intimidating task introducing development errors and obscures design errors.

There is missing important aspects, such as the decrease of the potentially large gap between model change and model execution. This temporal gap brings to the impossibility field, issues such as: (1) exploration and simulation of new models; or (2) ability to inexpensively explore architecture and design alternatives.

Designers should be able to create xUML models and to compile the models to produce a running system. Developer by other hand should be able to map model elements, to match specific platform aspects. It is also important to get a substantial reduction of the temporal gap between model changes and model execution.

The most exciting vision would be the implementation of an efficient UML Virtual Machine, with the ability of running every model designers wish.

1.4 Paper Organization

Section 2 briefly describes xUML, its tools and characteristics. Section 3 presents important model debugging issues. In Section 4 is presented the XIS-xModels tool together with its development process and architecture (Visio based, UML support and VBA dependent). Section 5 presents the main conclusions and the work to be developed in a medium term.

2 EXECUTABLE UML

We assume that any language that can be executed is a programming language, independently of its abstraction level. Therefore, xUML should be considered a (graphical) programming language. A xUML model completely specifies the semantics of a single subject matter, and in that sense, it is indeed a *de facto* "program" for that subject matter.

xUML is also a UML profile with defined semantics for both the components and the interactions between them during execution. UML was originally designed to sketch the

structure of object-oriented software systems. Consequently, it contains elements designed to map closely to software concepts such as class, task, parameter and so on. Given this sketch of the software, a designer/programmer can then add and fill more particular details in the selected programming language to complete the system. With this approach in mind, there is little rationale in defining the details of actions when they might just as well be programmed directly.

Consequently, xUML uses a subset of UML, as suggested in Figure 2. On the other hand, to be useful, this subset of UML must have a way to specify actions at a higher level of abstraction; it is done, for example, with traditional programming languages.

There are many executable profiles that could be defined to select a set of meaningful components and how they interact during execution. The first profile defined for xUML is one based on state charts and asynchronous distributed behavior [Mellor 2002]. Each object has a state machine that executes independently of all others. It is in just one state at a time, possibly executing many actions. It can invoke data access methods (that do not change state) synchronously, and it can send asynchronous signals to communicate and synchronize with other objects. Both these elements and interactions between them have a precise definition at run-time sufficiently well defined, to be executable. A xUML model is therefore independent of software organization and implementation decisions: it can be considered a computational independent (CIM) or software platform independent (PIM).

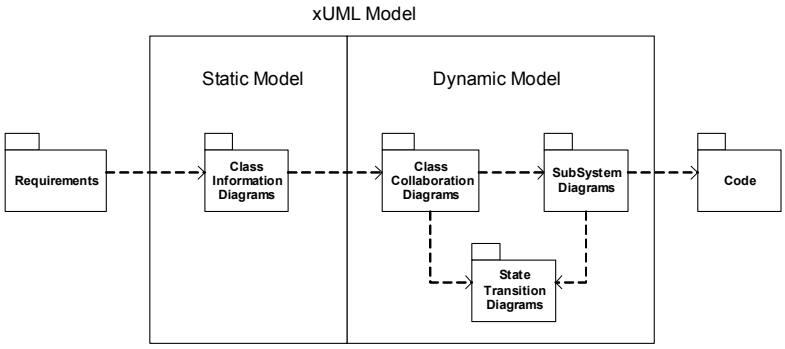

Figure 1 - xUML model components.

xUML [Mellor 2002] seeks to define, precisely, the execution semantics of the relevant elements of the UML. The benefits of this are many, namely the xUML models:

1. Can be unambiguously interpreted, which at least helps the human reader;
2. Can be executed, which, given a suitable environment, means that they can be validated early in the development lifecycle;
3. Can be translated to target code achieving near 100% code generation;
4. Can potentially be translated to silicon;

So xUML becomes a very high level language. The xUML can also be used to model the structure of a solution quite independently of the physical conditions. In this context, a class is an aggregator of information and behavior which could be rendered in any form, perhaps as a class in an object-oriented programming language; as a block of data and a set of functions in a traditional programming language; or even as silicon chip.

2.1 xUML based Tools

There are not so many research groups and companies that suggest and develop an executable UML tool. However three of them are: (1) iUML (intelligent UML) from Kennedy Carter; (2) Kabira from Kabira Design Center; and (3) BridgePoint Development Suite from Project Technology that have automated support for executable and translatable UML. Unfortunately, all of them are not full free distributed. One of the executable UML tools that limited distributed is iUMLite from Kennedy Carter, as a free version of iUML. Another important project in this

filed is the JaVis from the Department of Mathematics and Computer Science of the University of Paderborn [Mehner 2002]. The JaVis environment is used for visualizing and debugging concurrent Java programs. The information about a running program is collected by tracing it. UML is used for the visualization of traces. Traces are automatically analyzed for deadlocks. The tracing is implemented using the Java Debug Interface (JDI) of the Java Platform Debugger Architecture. The visualization is integrated into the Together CASE tool.

3 THE XIS-XMODELS PROJECT

3.1 Overview

As we have seen, iUML and the other xUML tools are standalone applications that deal with xUML and his action language. Our proposed approach is different and takes advantage of Microsoft Visio's [Microsoft 2004] visualization and model manipulation capabilities, together with the programmatic power of Visual Basic for Applications [Microsoft Visual Basic for Applications 2004]. Therefore we introduce here an addin solution ease to work by all MS Office users, which is called "XIS-xModels".

XIS-xModels is a VBA application that acts as a native Visio's feature and uses theirs capabilities of data storing together with code compilation and execution.

In this way, our xUML model aggregates (1) domain models (i.e. class diagrams); (2) object life cycles (i.e. state machines diagrams); and (3) behavior and action sequences represented by VB code associated with each state or transition.

3.2 XIS-xModels in the Context of the ProjectIT Project

XIS-xModels is a running sub-project inside the INESC-ID's Information Systems Group, called ProjectIT [ProjectIT 2004]. A set of good best practices are incentived by ProjectIT: (1) plan before do anything (e.g. good estimation of time and resources); (2) get the user/customer involved; (3) reuse, reuse every time you can (e.g re-using patterns, frameworks, architectures and knowledge); (4) avoid programming, try to model and use generative code approaches; and (5) keep it simple.

ProjectIT comprehends five cooperative projects: ProjectIT-Workbench, ProjectIT-Time, ProjectIT-Requirements, ProjectIT-Tests and ProjectIT-MDD

Each project is very important for the whole, but in the context of this paper we will only introduce the last one. ProjectIT-MDD (MDD stands for "model driven development") is a synonymous for the XIS project [XIS 2004, Silva 2003, Luz 2003, Silva 2003] and comprehends three important components: (1) the *XIS/UML profile*; (2) the *XIS-xModels tool*; and (3) the *XIS-dev tool*. XIS-dev is tightly coupled with IDE environments and code generations techniques, and XIS-xModels (presented in this paper) is a step higher in abstraction, giving emphasis to model architectures, execution, animation and simulation, being tightly connected with CASE environments.

3.3 Microsoft Visio

Visio, a part of the Microsoft Office suite, is a popular drawing tool for a variety of diagrams such as flowcharts, block diagrams, building plans, maps, etc. As an Office tool it shares all programmability advantages, such as: a huge number of end users; many developers; a consistent programming language (VBA); a consistent development environment (VBE); and components and code often shared across applications

Visio is also an automation server, exposing its object model for other software to use (e.g. VBA addin). The object model encompasses almost all data in Visio, including canvas and shape information, ShapeSheets, menus and dialogs, giving client applications full control over all aspects of Visio. They are able to modify existing elements in any way, as well as add new ones, or simply access information about the canvas' current contents.

Automation clients can be written in any language that supports automation (or COM), including Visual Basic for Applications (VBA), Visual Basic (VB), C, C++ and C#. Visio includes a VBA development environment and VBA macros can be embedded directly into Visio documents, thus simplifying both development and deployment.

3.4 Visual Basic for Applications (VBA)

Visual Basic for Applications (VBA) is a powerful but easy-to-use programming language available in Microsoft Office Applications. For scientific and engineering purposes it is similar to BASIC and FORTRAN. VBA is actually a shortened version of the Visual Basic programming language (since version 5). The language also has other features that make it useful for automating certain tasks, including such things as buttons and menus.

VBA commands are entered into two types of procedures. The first type of procedure is sometimes called “macro” procedures. Macros generally perform actions on worksheets and ShapeSheets but do not return values. Macros are generally executed through menus or buttons. Function procedures, on the other hand, returns a value (or values) to a spreadsheet or ShapeSheet or another procedure (although it may also be used to manipulate ShapeSheets as well).

3.5 XIS-xModels Architecture

This section defines the architecture for the XIS-xModels tool. This section also contains detailed pictorial representations of the architecture software and how it interfaces with other components. The XIS-xModels software architecture high-level block diagram is shown in the Figure 5. The software architecture is specifically designed to separate communication issues from model operations.

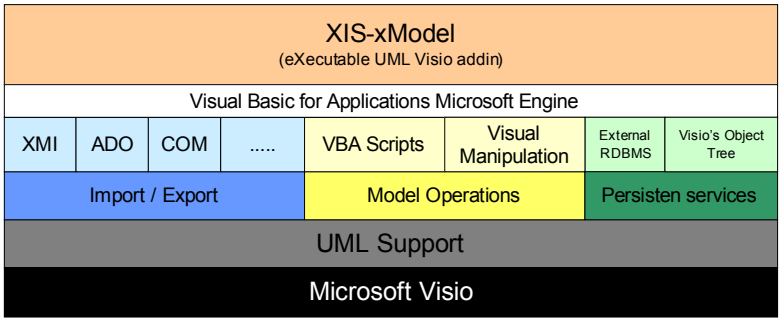

Figure 2 - XIS-xModels Architecture: a layer-based vision.

The architecture components involve the following aspects:

1. XIS-xModels addin: This is implemented as a VBA application stored as a Visio’s addin and can be accessed through the Visio’s toolbars.
2. UML Support: This component is offered by default on Visio. However, the specific xUML support is helped by the xUML XIS-xModels stencil (see below).
3. Model Operations (VBA Scripts and Visual Manipulation): These components are the way by which we can manipulate and read models and theirs semantics.
4. Import / Export and External Access
    a. XMI: XML Metadata Interchange provides a standard way to exchange UML model metadata between different modeling tools.
    b. ADO: The data-access component of the Microsoft .NET Framework that works with any component on any platform that understands XML.
    c. COM: Component Object Model is a software architecture that allows applications to be built from binary software components. COM is the

underlying architecture that forms the foundation for higher-level software services.

5. VBA Engine: The VBA compiler is actually a syntax checker. This means that, unlike compilers used by other languages, the VBA compiler doesn't produce a freestanding module that you can execute outside the Office environment. VBA uses the compiler to look for errors that prevent the program from running properly. The compiler runs constantly, so VBA finds some mistakes almost immediately after we make them.
6. Persistent Information Support: can be offered by the visio's own object tree that stores all information about models and objects, or by an external RDBMS such as SQL Server or Oracle 9i.

5.3.1 XIS-xModels Stencil

The XIS-xModels stencil is a collection of xUML specific artifacts, such as states with associated VBA code or classes with associated state machines. New stereotypes will be created together with new symbols according the XIS/UML profile [Silva 2003, Luz 2003] for representing these new instances.

3.6 xUML Based Development Process

The xUML process is a rigorous object-oriented system development process, based upon experience with executable modeling, that has been applied to many pioneering projects in sectors such as telecommunications, automotive, aerospace and defense [Sulistyo 2002, Mellor 2002, Jacobs 2003]. It is based upon a set of principles , such as:

1. It is based on precise and complete analysis models that can be subjected to rigorous testing by simulation. The analysis model is complete when it delivers the expected results for all test cases.
2. It is based on simple notations, using a subset of UML organized into a coherent set of layers. This means that normal human beings, such as customers, engineers and managers can read, understand and provide helpful feedback for the emerging analysis models.
3. It promotes an understandable and repeatable partitioning strategy, based, for example, upon the idea of separation of concerns.
4. It provides a small but sufficient set of techniques to address all subject matters, including "design", in a uniform way.
5. It promotes usable analysis models, which can be used by system designers and developers without the need for unreliable "interpretation" of the meaning of the models.
6. It promotes implementation by translation, in which the entire system can, if desired, be automatically generated from the analysis models, using a set of rigorously specified rules that deliver a system with the required characteristics.
7. It promotes large-scale reuse, in which entire sets of objects are reused as a single component.

According this xUML based process, problems are broken up into domains and connected by bridges.

Each domain has a class model, which contains a set of state charts that define the behavior of the involved classes. Execution rules define the behavior of the execution model. With this, documentation can be linked into the model. Use cases provide the generic support for documentation. Sequence diagrams define the bridges between the domains and the interaction between objects. There are four layers: domains, classes, operations and states.

On the other hand, there are three complementary phases in the xUML based process: (1) preparation or inception phase; (2) modeling or elaboration phase; and (3) construction phase.

The *preparation phase* does not produce any executable models but provides the foundation for the next phase; its main objective is to establish requirements and scope. The *modeling or elaboration phase*, involves the following activities:

1. Define the top-level domain diagram, if necessary. This activity involves define the top-level packages of the system, as well as define their main interactions and dependencies.
2. For each domain involved: (1) produce the class diagram; (2) produce state-chart models, for each class; (3) update the class diagram as required; (4) add operation to the classes; (5) specify the operations and state actions, according a specific action language; (6) specify initial conditions and test methods with that action language; (7) specify the provided and required interface of the domain; (8) build the domain model; and (9) simulate the scenario through the domain.
3. Specify the bridge which connect the different domains
4. Build the multi domain model
5. Simulate the scenario through all involved domains

Finally the *construction phase* can involve manual or automatic translation from models and abstract design rules to more specific application models, documentation, or even software code.

3.7 XIS-xModels Development Process

The XIS-xModels development process is based on the xUML process referred in the previous section, but adapted to XIS project owns specifications. XIS-xModels development process can be divided in four main tasks: (1) partition into domains (Packaging); (2) capture system requirements (Use Cases); (3) specify classes (Class Diagrams); and (4) specify system behavior (State Machines + VBA scripting)

After splitting the problem into different domains, the design of use cases is the next task , which involves identifying all the requirements together with system actors. This procedure will help us on the third task where we must find appropriate classes that accurately represent the real abstractions according the problem domain. We will not only have a solid foundation on which to build the system, but also excellent prospects for reuse by systems that will be designed and built over time.

With all the classes identified we are now capable of drawing the domain model (class diagram). For each class there is (if needed) a state machine representing its object lifecycle. To implement this lifecycles we take valuable information from the use cases and also, if needed, from sequence diagrams.

VBA code can be added in all tasks, in particular can be associated to classes operations or state actions specification.

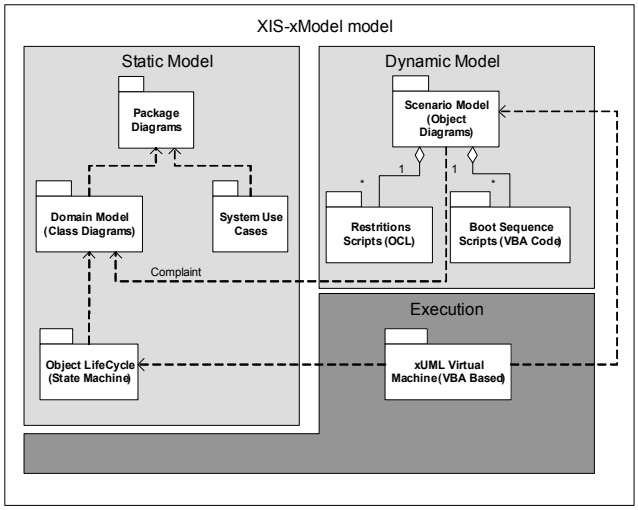

Figure 6 - Model organization for testing scenarios for simulation and animation.

A complete XIS-xModels model involves, as depicted in Figure 7, a package diagram, a domain model, object lifecycles, operation / action specification, plus a scenario model (which includes restrictions and boot sequence scripts). A dynamic model comprehends testing scenarios together with an execution environment. A scenario is modeled by an object diagram based on the entities defined in the domain model. To enrich and turn a scenario executable, a boot sequence is needed, being attached to the respective scenario giving it direct execution instructions (e.g. "call object 1 and invoke its operation 3"). Restrictions are as well added to a scenario when a particular behavior can not be described by the boot sequence. The execution is supported by a xUML virtual machine based on the Microsoft Visio's VBA engine. This VM is responsible for: (1) the connection between scenarios and the needed object lifecycles; (2) triggering procedures (temporal property); and (3) the user output, in charge of giving results to the end user.

Another important feature of this tool is the *model animation*, which involves showing at *modeling or animation time* (real time), the exact position of the execution in different models (e.g. when a particular object is invoked it will be highlighted together with its state machine). This characteristic is very useful, for example, in educational environments, giving students new ways to learn and understand the OO paradigm and its complexities.

3.8 XIS-xModels Action Specification vs. Action Languages

Action semantics gives a better definition on the behavior of model execution, but there are still gaps left: i.e. the system- dependence of computational actions and the time for a change to take place. It is still not standardized, and currently there are very limited materials on this subject.

A couple of action languages have been in use in system development for a number of years. All are Action Languages targeted at object modeling techniques and have primitives built in to support, for example, the creation and deletion of objects and links as well as the sending of signals and the invocation of operations. Some of action languages examples are:

1. Action Specification Language (ASL) [Carter 2002]: A public domain language of which there have been several implementations.
2. BridgePoint Action Language (AL) [BridgePoint 2003]: An action language supported exclusively by the BridgePoint modelling tool.
3. Kabira Action Semantics (Kabira AS) [ObjectSwitch 2003]: An action language for the ObjectSwitch middle-tier server suite.

Actions have no default orderings (like sequential execution as in traditional programming languages); actions that are not implicitly ordered by data flow or explicitly ordered by control flow can be executed either parallelly or in an arbitrary order.

XIS-xModels proposal uses VBA to specify procedures and sets of actions. Therefore, it is necessary a comparison between existing xUML tools and ours, as summarized in Table 1. All actions, procedures and scripts (boot sequence or restrictions) in XIS-xModels are implemented in VBA code, being this way necessary, to adopt some restrictions to the use of all VBA capabilities. Designer must be aware of the abstraction level they are working, not introducing hardware of platform specific components. In traditional action languages this problem doesn't appear because the particular structure of those languages are completely different comparing with common programming languages.

Table 1 - xUML tools comparison chart.

Tools / **Features**	**iUML**	**BridgePoint**	**XIS-xModels**
Standalone	Yes	Yes	No. Runs as an Microsoft Visio's addin
Drawing Tool	Given by the Tool. Very limited and not user friendly.	Given by the Tool.	Very powerful drawing capabilities given by the popular Visio tool.
UML support	Given by the tool	Given by the tool	Given by Visio
Action Language	ASL	AL	Subset of VBA (with restrictions)
Timing Capabilities (OS or Hardware simulation)	Automatic implemented in the xUML engine	Automatic implemented in the xUML engine	Given by VBA compiler and execution engine. Can be modified, introducing time delay in VBA code, to simulate hardware or OS timings.
Testing Scenarios Spec.	Using ASL	Using AL	Using Visio Models together with VBA scripting.

4 CONCLUSIONS AND FUTURE WORK

We have introduced and discussed new technology for executing, testing and debugging UML models, during early stages of the software development process.

Based on the idea of the executable UML (xUML), involving a new UML profile, to better represent sequences of actions based on state charts, we have introduced a new development process.

The key benefits of the process presented on this paper are: (1) models can be validated early by simulation and or animation, which can reduce risk and cuts development costs; (2) models promote expertise in separation of concerns (domains); (3) models can be translated to implementation artifacts (e.g. source code, binary code) through generative programming techniques; and (4) the delivered software has a uniform quality across the entire system, and so, makes the system easier to understand and reduces maintenance costs.

The xUML process requires an early study of how to exploit the implementation technology to deliver the required system performance and reliability. This allows concurrent analysis and design and addresses technical risks and costs early on the process, and so shortens development time and eliminates redundancy in analysis and design models.

We have also introduced an *architectural overview* explaining all the components and tools in the game involved, such as the model compiler, model checker or the simulator tool.

This paper also presents our project, the XIS-xModels tool and approach, which is based on Microsoft Visio's platform together with the VBA language. The XIS-xModels tool can be viewed as a way of making "Microsoft Visio an MDA complaint tool".

As future work we see an emergent research field offering scientist in this particular area an increased collection of choices. We can identify important questions to be addressed in the future, such as: (1) standardization of action semantic languages; (2) refinement of simulation tools; and, in our opinion the most important, (3) implementation of more effective model

compilers with a rich set of transformation capabilities, early analysis error detection, and model optimization tools.

The XIS-xModels tool is being implemented and optimized for productivity and quality parameters, bridging it with the XIS project [XIS 2004].

It is our belief that xUML will have an important impact on how software development is made. By providing techniques which do not only facilitate design by synthesis, but also provide powerful means to debug requirements and designs in early stages we will be able to contribute in the future with tools, techniques and best practices which would be useful in design of large scale software systems.

5 REFERENCES

Sulistyo, S., Najib, W., Executable UML, Dept of Information and Communications Technology Agder University College, Norway (2002)

Glinz, M., Problems and Deficiencies of UML as a Requirements Specification Language, (2001)

Gallardo, M., Merino, P., Pimentel, E., Debugging UML Designs with Model Checking, Journal of Object Technology, Vol. 1, No. 2, July-August (2002)

Mellor, S., Balcer, M., Executable UML: A Foundation for Model-Driven Architecture, Addison-Wesley, (2002)

Carter, K., The Action Specification Language Reference Manual, www.kc.com. (2002)

BridgePoint, modelling tool Action Language www.projtech.com. (2003)

ObjectSwitch, middle-tier server suite Action Language www.kabira.com. (2003)

Kobryn, C., Visual Requirements-Driven Development with UML 2.0, (2003)

Schumann, J., Automatic Debugging Support for UML Designs, (2001)

Stevens, P., Playing games with UML tools, (2002)

Mehner, K., JaVis: A UML-Based Visualization and Debugging Environment for Concurrent Java Programs, (2002)

OMG, Action Semantics, Response to OMG RFP ad/98-11-01, (2003)

Jacobs, T., Musial, B., Interactive Visual Debugging with UML, (2003)

Wedin, E., Open Code Translation from Open Code Translation from Executable UML Executable UML Models, (2002)

Stevens, P., Advanced Tools for UML: now and in the future, (2003)

Chonoles, M.J., UML Tools Catalog, Advanced Concepts Center, (2003)

OMG, Model Driven Architecture, www.omg.org/mda (2003)

XIS: eXtensible Interactive Systems, INESC-ID Lisbon, http://berlin.inesc-id.pt/projects/xis/, (2004)

ProjectIT, INESC-ID Lisbon, http://berlin.inesc-id.pt/projects/ (2004)

Silva, A., “The XIS Approach and Principles”, Proceedings of the 29th EUROMICRO Conference, IEEE Computer Society (2003)

Luz, M., Alberto Rodrigues da Silva, Software Development Guided by Models - The XIS UML Profile, Proceedings of the ICEIS'2003 Conference, (2003)

Silva, A., Gonçalo Lemos, Tiago Matias, Marco Costa. “The XIS Generative Programming Techniques”, Proceedings of the 27th COMPSAC Conference, EUA, IEEE Computer Society (2003)

Sunyé, G., Pennaneac'h, F., Ho, W., Le Guennec, A., Jézéquel, J., Using UML Action Semantics for Executable Modeling and Beyond, IRISA, Campus de Beaulieu, (2001)

The middleware Company, Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach, (2003)

iLogix, Rhapsody - UML Application Development Platform for Pervasive Computing,(2002)

Microsoft Visio, www.microsoft.com/office/visio, (2004)

Microsoft Visual Basic for Applications, msdn.microsoft.com/vba/, (2004)