JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 2, No. 1, January-February 2003

How You Could Use NEPTUNE in the Modelling Process

Agusti Canals, **Yannick Cassaing**, **Antoine Jammes, Laurent Pomiès, Etienne Roblet,** CS Communication & Systèmes, France

Abstract

The European Community plays an active role in giving concrete expression to the new developments taking place in modelling languages and in UML (Unified Modelling Language) [1] in particular. These developments often make use of norms and specifications that are mainly carried out by the OMG (Object Management Group).

The main objective of the European NEPTUNE project led by CS (Nice Environment with a Process and Tools Using Norms and Example) is to develop both a method and tools (in addition to the existing software development tools) that support the use of the UML notation. This method has emerged from considerable experience gained in the industrial environment. It will apply to a variety of application fields, including software engineering, business process, and knowledge management [4].

The newly developed tools will enable UML models to be statically checked for their coherence and consistency. They will also enable professional documentation resulting from the transformation of models [5].

The method and tools developed make the application of the UML standard easier, and they promote its use in a large number of business fields so that the UML standard might be further improved with the aim to participate effectively in the work of the OMG.

This paper will present the NEPTUNE method and tools. Then, lessons learned from this project will be reported, outlining the benefits and drawbacks of this technology as experienced by the development team. A conclusion will offer suggestions for future improvements and provide an overview of the next actions related to NEPTUNE deployment.

1 INTRODUCTION

NEPTUNE project (http://neptune.irit.fr) is part of the IST program. It is coached by CS, in collaboration with the Universitat polytechnica de Catalunya (UPC, Barcelona, Spain), the Institut de Recherche Informatique de Toulouse (IRIT, Toulouse, France), University Babes-Bolyai (UBB, Cluj, Romania), GTD company (GTD, Barcelona, Spain) and Novasys company (NOVASYS, Montréal , Canada).

Cite this article as follows: Canals et al.: *How You Could Use NEPTUNE in the Modelling Process*, in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pages 69-83. http://www.jot.fm/issues/issue_2003_01/article1

The whole NEPTUNE software is a java development. It aims to be Windows and UNIX compatible.

Below is a synthetic view of NEPTUNE architecture and the different activities handled by each of the consortium partners:

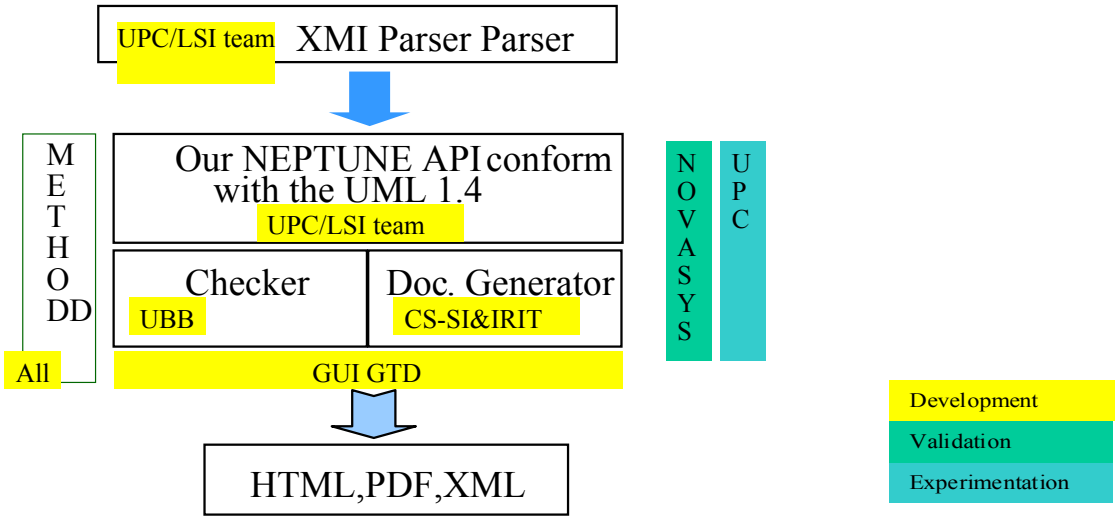


Fig. 1: Neptune architecture

In the next paragraphs, we will first give a synthesis of NEPTUNE method, already detailed and presented in its first version called “Use of UML/CS-SI development process" [2]. This method can be seen as an instance of the analysis and a design part of the unified process [3].

Then, after a presentation of the technologies involved in NEPTUNE, we will give the basics of the checker concepts and then focus on the document generator which will be presented in detail.

2 NEPTUNE METHOD PRESENTATION

Within the modelling process, the use of NEPTUNE technology starts with a conception based on the UML/NEPTUNE method. The guidelines provided by this method have to be followed to guarantee an efficient use of the various NEPTUNE tools.

This process is divided into phases, each phase being composed of activities. Below the various phases are presented in sequential order for easy understanding, although the process is naturally iterative and incremental.

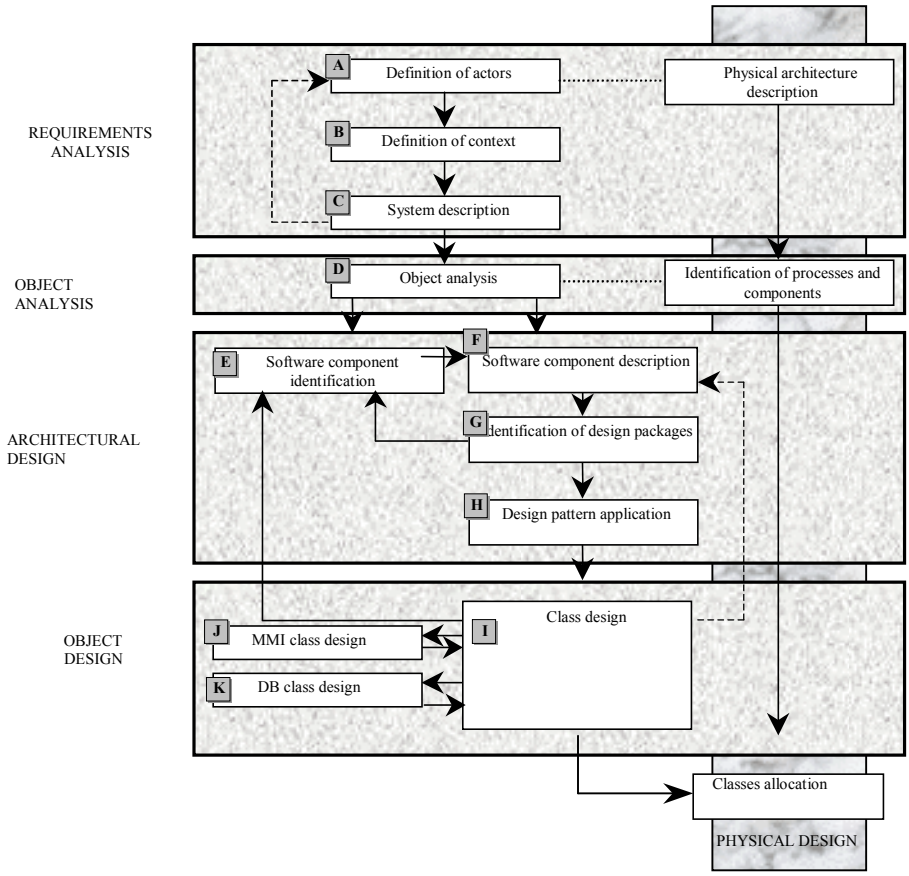


Fig. 2: Neptune method

Requirements Analysis

The aim of this phase is to turn the user needs into UML formalism.
It is composed of three steps or activities:

Activity A: Consists of defining the actors and external entities. Two use case diagrams will come out of this step: One for the actors and one for the external entites.

Activity B: Aims at defining the passive context or, in other words, the collaboration between the external entities and the modelled system. This step is based upon the production of two interaction diagrams representing both dynamic and static data flows.

Activity C: Consists of defining the use cases, which means:

- Build a use case general view
- Describe the use cases (collaboration between the actors and the modelled system) using interaction, activities, and class diagrams.

Finally, it is recommended to create a data flow general view diagram that shows all the interactions between the system and all the actors and external entities.

Object Analysis

The object analysis aims at producing a first logical organisation (packages) of the classes found during the requirements analysis.
The object analysis can be seen as one main activity:

Activity D:

- For each Use Case, produce a class diagram in which all classes involved in the Use Case appear.
- Organise the classes identified during the requirements analysis into a set of packages. For each package, create a class diagram.

Architectural Design

The main objective of this phase is to produce a first architectural view of the modelled system. In this phase, the analysis packages are turned into logical components.

Note: The first logical organisation could be modified here if the logical organisation does not respect the design constraints.

Four activities are identified:

Activity E: A package diagram is first defined in order to show all dependencies between all atomic packages. Then the packages' interface classes can be identified and presented in a package collaboration diagram.

Activity F: For each package, the classes can be completed by adding methods and/or attributes. At this point it can be useful to add activity or state diagrams describing any collaboration between the classes.

Activities G and H: If needed, modification of the package diagrams by adding design components or applying design patterns.

Object Design

In this phase, the objective is to produce the detailed architectural organisation (complete classes, attributes, methods, diagrams etc; find new classes, new packages, etc.) which will finalise the package description started in activity F.

This phase is composed of the following activities:

Activity I: Upgrade the contents of each existing diagram.

Activity J: Create the man-machine interface packages and description diagrams.

Activity K: Create the database interface packages and description diagrams.

Physical Design

The main goal of this phase is to produce the physical architecture and to identify the logical associations between the hardware and the software components. The steps are sequential:

- Creation of the hardware components in a deployment diagram.
- Processes identification and attachment to the nodes of the deployment diagram. Identification of the collaboration between the processes.
- Components identification (from our design or from existing frame works) and attachment to the processes previously defined . Allocation of the classes to the components (one or more classes by component).

Links between the method and NEPTUNE tools usage

To support the approach described above, the method is completed with a specific instrumentation for Rational Rose software. This instrumentation allows the user to create a first browser architecture and to complete it during the next steps of the modelling process.

The method also embeds a set of methodological rules described all along the process. These methodological rules will later be checked by the checker. Moreover, NEPTUNE users can also define their own methodological rules and design them with the rule designer feature of NEPTUNE. If they do so, these personal rules will be taken into account by the checker.

The recommendations given in the process regarding the use of specific UML diagrams and the use of XML format for the associated documentation have been chosen so that they can be of great help in standard documentation generation. For example, in software engineering, the document generator makes the generation of a validation plan easy if the guidelines have been followed during the modelling process.

To summarise: Following the modelling approach, with the methodological rules and the recommendations aforementioned, will make the use of both checker and documentation generator very efficient.

3 NEPTUNE TOOLS PRESENTATION

Technologies involved

The different technologies on which the NEPTUNE tools rely:

OCL (**O**bject **C**onstraint **L**anguage) is a part of the UML specification whose purpose is to express properties over a model that cannot be specified graphically. The language is

mainly based on first order logic and model navigation expressions. OCL can be used in several contexts [7], for example within a class to express class invariants or within an operation to express pre and post conditions. For the NEPTUNE checker rules design, we have enriched the standard OCL with several features like the capability to compute transitive closures of navigation expressions or to specify temporal properties of dynamic diagrams.

XML (e**X**tensible **M**arkup **L**anguage) is a standard, simple, self-describing way of encoding both text and data, designed for easy exchange between various hardwares and exploitation systems. XML documents describe the information they contain. XML is a markup language, using tags, and it is extensible. This means that the actual tagsets are not pre-defined, but that they are generally defined for a specific application or by an associated stylesheet called DTD (Document Type Definition). This format of documentation is recommended for documenting the UML model elements. It is also used during the documentation generation process.

XMI (**X**ML **M**eta **I**nterchange) specifies an information interchange format that is intended to give developers working with UML the ability to exchange any model in a standardized way, thus bringing consistency and compatibility to applications created in collaborative environments. XMI definition includes a specific XML-DTD. The choice of XMI as the input format of NEPTUNE (either Checker or document generator) makes the tool set compatible with any UML IDE featuring XMI export.

XSL (e**X**stensible **S**tylesheet **L**anguage) is a general-purpose language to define the presentation and formatting of XML data. The need for XSL arises on account of the fact that XML merely describes data; it includes no formatting instructions or display specifications. One ore more stylesheets (transformations) can be written using XSL in order to make practical use of the information encoded within an XML document. By separating presentation semantics from data, it becomes possible to display multiple views of the same data and to export the same information to a variety of different formats. NEPTUNE document generator implements this technology.

Checker

In this chapter we present a quick overview of the NEPTUNE checker so that the reader has a general idea of the NEPTUNE package from every angle.

The main objective of this tool is to CHECK the UML models [6] to improve the quality of the software, and find some problems, errors, etc. before coding. These checks are static (e.g. each package in the logical view of the design must have an interface class). The language used for writing the rules is OCL.

The Checker inputs are: The UML model (to be checked) in XMI format, the current UML META model (at present 1.4) and a set of OCL rules. These inputs are loaded through the NEPTUNE MMI before checking.

The Checker output is a formal report with warnings and errors.

There are three kinds of rules, the WFR (Wellformedness rules), the NEPTUNE methodological rules (described by the process) and other standard or methodological rules. With this tool the users can use a set of pre-defined rules provided with the NEPTUNE package and they can also create their own rules to check their specific process.

Note: it is possible to create specific rules in OCL language through the NEPTUNE MMI.

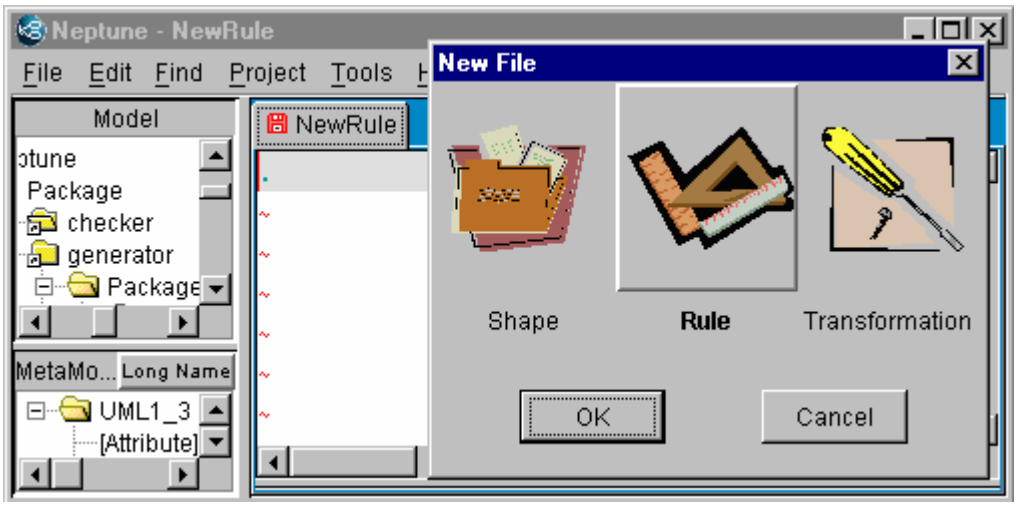


Fig. 3: OCL rule creation MMI

To get more information on the checker, please read the chapter "The NEPTUNE Technology to Verify and to Document Software Components" that will be published before the end of this year in the book “Business Component-Based Software Engineering” (Kluwer International Series in Engineering and Computer Science, 705).

Document generator

The main purpose of the document generator is to produce some professional documentation that results from the exploitation of UML model sub parts. This documentation is an end-user-oriented documentation, which takes into account the professional expertise of the reader. It is the result of transformations applied on UML elements available in the actual model. The transformations turn the information expressed with the UML formalism into textual easy-to-read information. It is important to say that there is no need to be UML skilled to understand the produced documentation. That makes the tool not specifically dedicated to software engineering, but also to other business fields like knowledge management or business process.

1. Inputs

XMI UML model: The XMI format is the mandatory format of the UML model to be documented. Once the model is loaded, its content is displayed in the model browser, making it available for the documentation designer.

UML metamodel: The UML metamodel is systematically loaded during the NEPTUNE sessions and displayed in the metamodel browser. Thus, for systematic treatments (over all actors for example), the documentation designer will not have to select all actors one by one in the model, but only the actor concept in the metamodel browser.

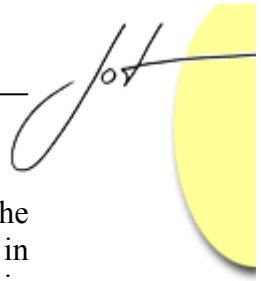
XML External documentation: The NEPTUNE document generator gives the opportunity to make use of external documentation. While designing the final documentation, the user can make references to this external documentation, whose contents will then appear inside the produced documentation, bringing in additional information.

2. Key words

XSL elementary transformation: These transformations, written in XSL, are able to perform the extraction of information from the model and to define its presentation. They can be seen as a way to generate different views of information. The documentation designers will pick some of the available transformations while designing the targeted documentation.

Documentary element: A documentary element can be defined at the metamodel level (meta documentary element) or at the model level (documentary element). At the metamodel level it is composed of an UML metamodel element (mandatory), an XSL elementary transformation to apply on this element (mandatory). It can also embed an icon to represent the UML elements involved, and links to external documentation or to other metamodel elements. At the model level each transformation is defined for one or more specific elements of the model, where each information will be considered as local. Each documentary element will constitute a raw brick of a higher level concept in NEPTUNE, namely the shape.

Shape: The shape is a frame containing the structure of the documentation the user will produce, together with some documentary elements. The shape can be defined at metamodel level – the generic shape – in order to provide generic models of documents, and at model level – the user shape – in order to be fully adapted for the user's needs. A user shape may be defined from a generic shape and extended by the user. The generic shapes will lead to the generation of standard documents, often dedicated to a particular business.



UDD: The User Document Definition (UDD) is an XML-format file, generated by the NEPTUNE Document Generator core according to the documentation needs specified in the shape. It will then be processed though XSL transformation (part of the software) in order to produce an XSL stylesheet, called XMI Processing Stylesheet.

XMI Processing Stylesheet: As mentionned above, this XSL stylesheet is the result of UDD processing. It will be applied to the XMI file to generate the final XML documentation.

Final document: Ultimately, the aforementioned XML documentation can be processed to generate a file in a particular format (pdf, html,etc.) called final document.

3. Documenting an application

Let's suppose that the documentation designer has already loaded its model that is now displayed in the model browser.

The first thing to do is to define the aspect of the documentation expected as output of the generator. This can be done through the selection of a standard shape, or through the creation of a new one called user shape. The shape is built up in the design zone and can be stored at any time. A shape is composed of both structure and content information.

Regarding the rendering of the structure (titles, sub-titles), a style zone offers a selection of fonts, polices, and sizes, etc.

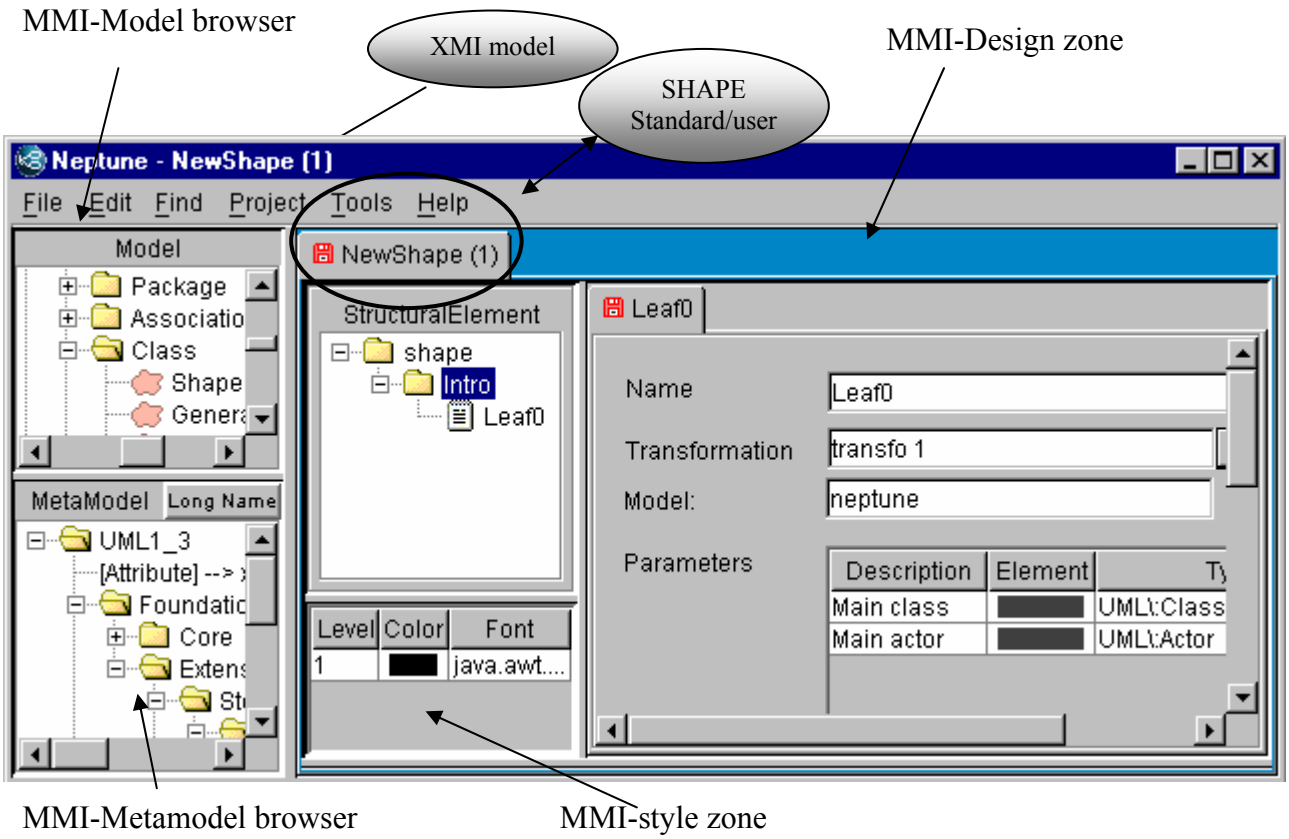


Fig. 4: standard MMI view 1

Concerning the content, each paragraph of the final document is represented in the shape by a leaf, or documentary element. The design of each documentary element starts with the selection of one or more model elements in the model browser. For systematic treatment (over all actors for example), the documentation designer will use the UML browser in which all UML notions appear.

These elements have now to be associated to an XSL elementary transformation. A set of transformations is available in the tool. As it is not easy to imagine what the rendering of a transformation can be, we have also added an example view of the output produced for each of the standard transformations.

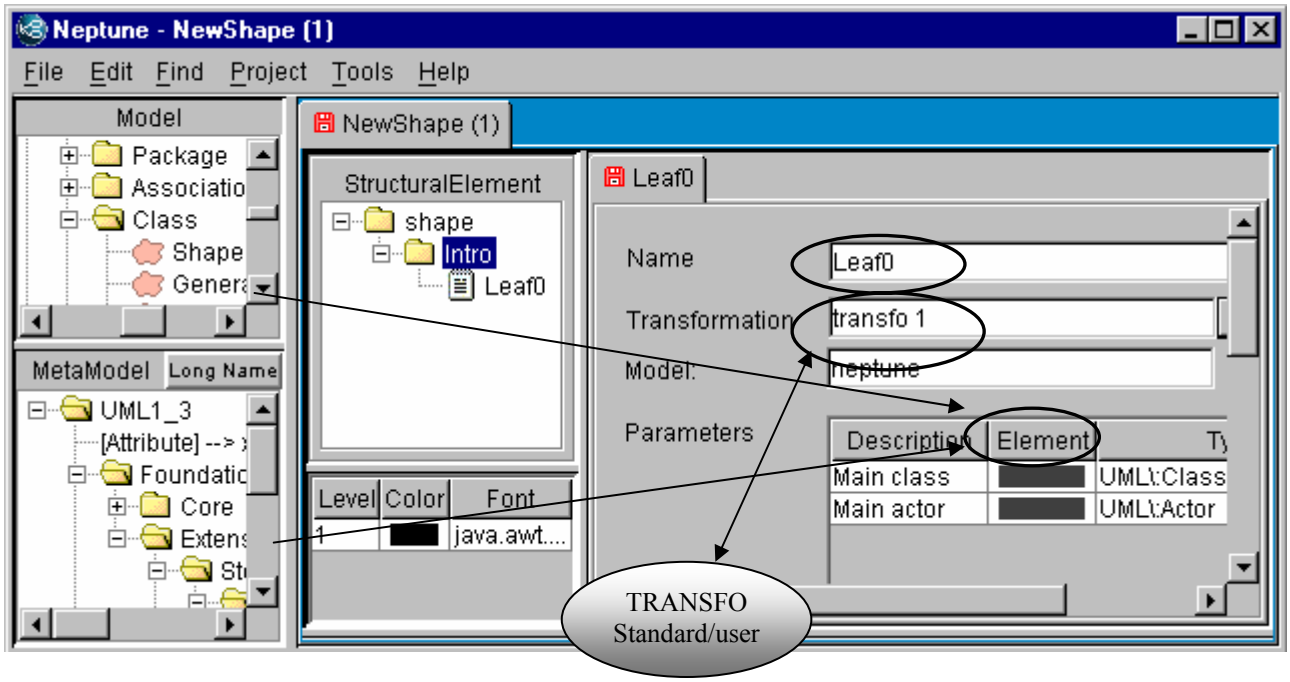


Fig. 5: standard MMI view 2

Moreover, if this set of transformations does not exactly fit the users' needs, it is also possible to write their own transformations.

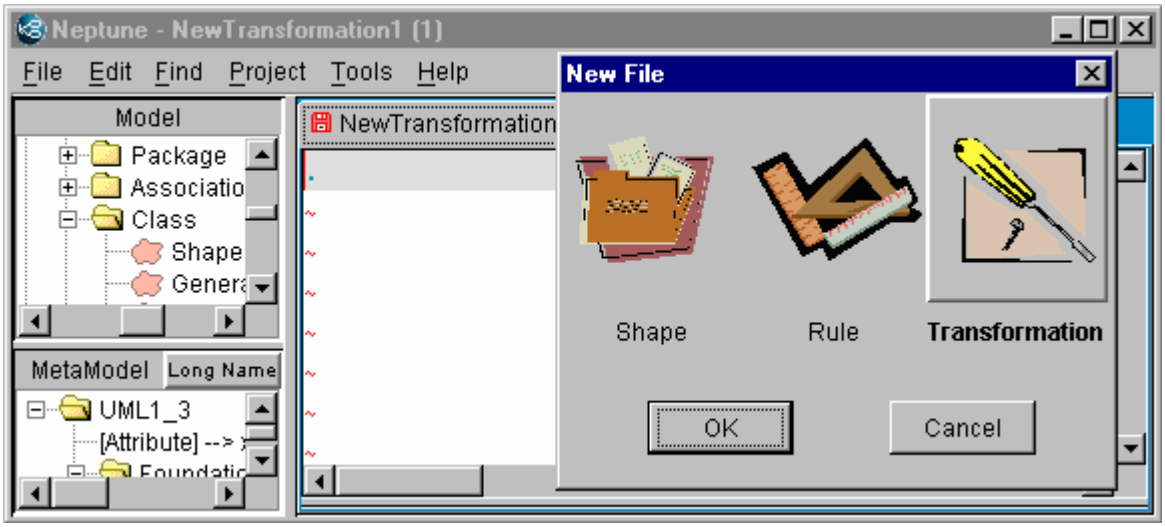


Fig. 6: XSL transformation creation MMI

An additional feature available while designing the shape is the ability to deal with external documentation. In other words, the NEPTUNE users can add several references to external documentation to a shape, thus enriching the information extracted from the

model. Of course, the only transformations handling these external pieces of documentation are the ones that have been designed to do so.

Once the shape is selected or customised, the next step consists of feeding the generator with both the shape and the UML model to be documented. The first output of the generator is an XML file in which all information specified in the shape has been interpreted and organised.

The NEPTUNE document generator has a last feature: it is the format transformer. Depending on the choice of the documentation designer, this feature is able to turn the XML document previously produced into another format. It can be either RTF, PDF, or HTML.

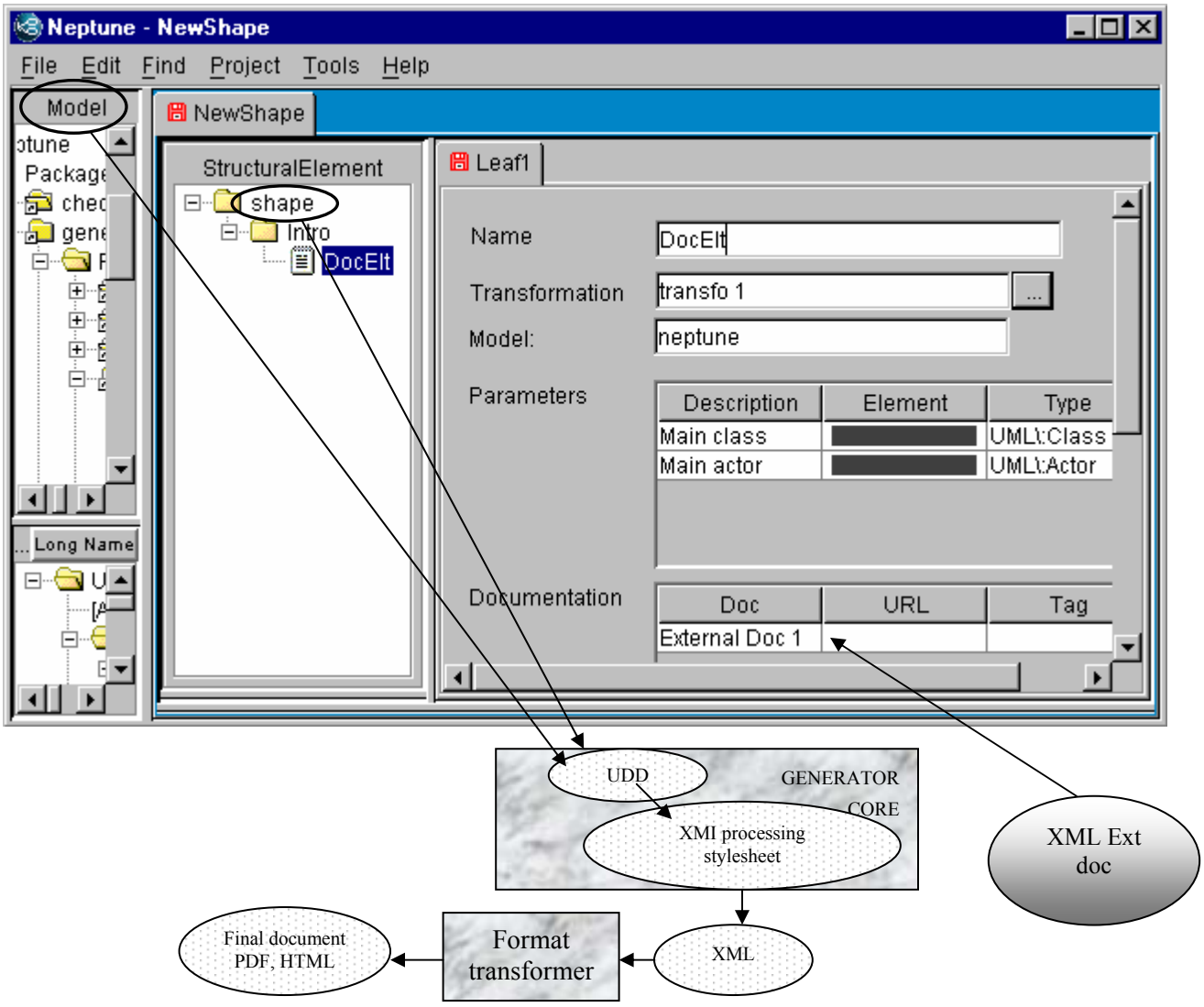


Fig. 7: standard MMI view 3

4 LESSONS AND BENEFITS

Lessons

Today, even if NEPTUNE tools are still in a validation phase, we can already say that the technologies involved in the development seem to be pretty well adapted to the expected objectives.

On the other hand, in spite of the design tools provided by NEPTUNE for OCL rules and XSL transformations creation, the efficient manipulation of the two languages takes a while, and is thus not easy for beginners.

Benefits

The most striking benefits of the NEPTUNE tools are their ability to save time:

- During the integration, the validation and the maintenance phases, most of the errors that might appear in these final development steps have already been identified earlier in the modelling process thanks to the checker.
- The document generator is obviously efficient and saves time during the documentation phase of a project.

Another benefit of the document generator is its ability to make the communication easier between customers and providers: even during the modelling process, it is always possible to generate end-user documentation, providing an easy way to review the work in progress.

5 CONCLUSION

Today, a trial version of the software is available for download on the web site.

Among the activities with whom the team is now busy, there is the writing of a book in which all the NEPTUNE concepts will be developped. The book is scheduled to be published by mid 2003. We also organise a workshop called “UML: model checking” (see http://neptune.irit.fr for details). We plan this workshop to be the first occurrence of a classic UML year event.

The NEPTUNE project will end on the 31st of January 2003 and has already produced significant results (mature method and operational tools). We strongly believe that it is important to keep on working on the same path and hope to be one of the key players of the sixth european Frame Programme for Research and Development (sixth PCRD).

REFERENCES

[1] Grady Booch, James Rumbaugh and Ivar Jacobson: *The Unified Modelling Language User Guide*; edited by Addison-Wesley, 1998.

[2] Agusti Canals: *Use of "UML/CS-SI" development process"*; ICSSEA'99; Journal of Object Oriented Programming, May 2001.

[3] Grady Booch, James Rumbaugh and Ivar Jacobson: *The Unified Software Development Process*; edited by Addison-Wesley, 1998.

[4] Agusti Canals, Séverine Carles and Thierry Lacomme: *Full ADA software projects integrating UML and HOOD*; DASIA'2000.

[5] Pierre Bazex, Jean-Paul Bodeveix, Louis Feraud, Thierry Millan, Christian Percebois, (IRIT): *Data Design and Transformation;* SCI 2000.

[6] Christophe Lecamus: *Vérification de la coherence de modelisations UML*; CNAM – mémoire de diplôme d'ingénieur 2001.

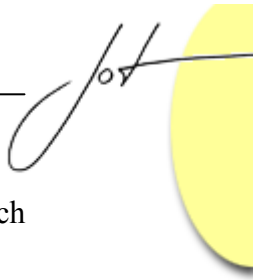
[7] Dan Chiorean: *Using OCL Beyond Specification*; Fourth International Conference on the UML "Modeling Languages, Concepts and Tools", Toronto, Canada, October 2001.

About the authors

Agusti Canals (agusti.canals@c-s.fr) is a software engineer (Université Paul SABATIER, Toulouse) and has been working at CS since 1981. Now project manager and senior software engineering consultant, he has already presented papers on HOOD, Ada, UML and object business patterns. He also currently teaches software engineering in different training structures, like Ecole Centrale Paris or Université Paul Sabatier (Toulouse).

Yannick Cassaing (yannick.cassaing@c-s.fr) is a software engineer (DEA in Object Oriented Databases). He graduated from the Université Paul SABATIER, Toulouse in 1990. He has now 12 years of experience in software development of projects using various Object Oriented techniques. He has been working at CS since 1997.

Antoine Jammes (antoine.jammes@c-s.fr) is a software engineer (Master). He graduated from the Université de Valenciennes in 1999 (UVHC). He also studied ecosystems and



population biology (maitrise) at Universite Paul Sabatier (UPS). He joined CS in March 2001.

Laurent Pomies (laurent.pomies@c-s.fr) is a software engineer. He graduated from the Ecole Nationale Supérieure de Physique de Marseille (1997). He spent three years in automotive industry (VDO CC, Siemens automotive), as an embedded software developer and architect before joining CS in august 2001.

Etienne Roblet (etienne.roblet@c-s.fr) is a software engineer graduated from the "Institut d'Informatique d'Entreprise" (Evry) in 1999. He also obtained a DEA (Diplôme d'études approfondies) at the Paul Sabatier University, Toulouse, in 2000. His skills include several programming languages (Java, C, Delphi), distributed objects techniques (CORBA) and object modelling languages (UML). He joined CS in August 2000.