Use of a CASE tool for developing ATLAS DAQ software

R. Jones, L. Mapelli[a], A. Patel[b]
CERN, Geneva, Switzerland

N. Parrington
University of Sunderland, Sunderland, UK

The ATLAS collaboration has defined a requirements document [1] for a software process and supporting environment, for the development and modification of all ATLAS software products. CASE tools have been identified as an appropriate technology for supporting many aspects of this process.

In order to asses the viability of this CASE based approach the ATLAS DAQ project [2] has evaluated the StP/OMT CASE tool, which supports a repository based customisable code generator, and extended its code generation capabilities to fulfil the specific needs for the project. Code generators have been developed for the static and dynamic models with a view to supporting evolving standards (the Unified Modelling Language (UML)). This paper describes how the various code elements are generated and the integration of StP/OMT into the development environment.

1. Overview of StP CASE tool

StP/OMT [3] is a commercial CASE tool sold by AONIX. StP is primarily a design tool that supports the OMT method by J Rambaugh, [4] providing a UNIX based software development environment for creating graphical representations of systems using the OMT notation. All design information is stored in the underlying SYBASE repository, these data can then be used via the Query and Reporting System (QRS) for code generation, document generation and design checking. All designs are entered into a *system* which is the primary unit for partitioning. Designs are entered into StP via:

- *Requirements Table Editor* - Captures high-level rules and requirements.
- *Use Case Editor* - StP/OMT integrates Jacobson Use Cases [5] to model system-level concepts and requirements from the users point of view.
- *Object Model Editor* - Expresses classes, their attributes and operations, and the relationships between classes.
- *Dynamic Model Editor* - Models class behaviour, events and timing.
- *Class Table Editor* - Provides a textual expression of important class attribute and operation detail not visible in a graphical model.
- *State Table Editor* - Similar in format to the class table editor, but used to provide detail for states.

In addition StP provides facilities for:

- *Diagram Checking* - Checking can be performed for individual diagrams or system wide.
- *Report Generation* - Class metrics and Use case reports can be generated. Reports are implemented via Query and Reporting Language (QRL) scripts, the user can create his/her own script to combine diagrams, tables and text into a complex document. To this end a report generator has been developed for ATLAS [6].
- *Code Generation* - Scripts are provided for code generation in languages including C++, Smalltalk, Java, ADA etc. See 2 Code Generation
- *Printing* - Diagrams and tables can be printed in FrameMaker, Interleaf or PostScript.
- *Integration with third party tools* - StP has been successfully integrated with the SNIFF+ [7] programming environment (support for compilers configuration tools etc.) for C and C++. SNIFF (via its parser) can be used to reverse engineer existing code into StP, it is also possible to navigate from methods, attributes etc. to the SNIFF code editor. AONIX also provide a testing tool StP/T [8], which is integrated with StP, and generates test cases for StP models. StP/T has been integrated and evaluated [9].
- *Multi-user support* - Through locking, StP prevents users from modifying the same files at the same time.

---

a. Spokesperson.
b. On leave from School of Computing and Information Systems, University of Sunderland, UK

- *Version Control* - StP is integrated with Source Code Control System (SCCS) to manage versioning of designs.

Although support for the implementation phase of the software life-cycle is limited, StP's core architecture allows the various code generators to be customized/extended.

2. Code Generation

All the data entered into StP diagrams and tables are held in the StP *Repository*. The StP *Query and Reporting System* (QRS)[10] allows information to be extracted from the Repository for the purpose of code generation, document generation, or design checking. This is done via scripts written in the *Query and Reporting Language* (QRL)[10]. There are limits to the amount of code that can be generated, since there must be a direct correspondence between design elements and designs do not cover detailed algorithms. However, OMT models are rich in content, providing sufficient detail for the development of a variety of code generators.

2.1 StP Repository

The StP SYBASE Repository is a central store for project information. The Repository stores and retrieves information based on the *Persistent Data Model* (PDM)[11] which is a scheme that hierarchically organises objects (from diagrams and tables) in the Repository. Users and applications can gain access to the repository via an application programmers interface or the *Object Management System* (OMS) query language.

2.2 OMS query

All PDM types have attributes and associations to other PDM types. The OMS query language is based on PDM types and uses a syntax that is based on the type's attributes and relationships between types. Using the OMS query language applications can retrieve any data in the StP Repository. OMS queries can be embedded in QRL scripts or called directly using the *Repository Browser*.

2.3 The Repository Browser

Using the Repository Browser [12] requires familiarity with the StP PDM and the OMS query language. The Repository browser allows the user to view the Repository via OMS queries. This is useful for:

- Verifying the contents of the Repository.
- Performing low-level debugging.
- View (but not edit) project specific data.
- Learn how Repository data is stored, which can be helpful in writing QRL scripts.

Repository data appears in a table editor where operations such as sorting, navigation and hypertext-like browsing can be performed.

2.4 QRL Scripts

The StP Query and Reporting Language (QRL) is a high-level C like script language. StP provides a *Script Manager* which is the main graphical user interface for the query and reporting system (QRS). QRL also provides OMS interface functions that can be used in QRL scripts to return information about Repository objects. QRL also supports the types described in the PDM for this purpose.

3. Code generators developed

A number of code generators have been developed to support the selected software packages [13] for the prototype -1 project:

3.1 C++ with Rogue Wave

StP already generates C++ from the object model, hence a QRL script exists for C++ code generation. This script has been extended to generate code specifically for the Rogue Wave (RW) [14] class library, as well as additional C++ features. It includes support for multiple associations using RW lists, inheritance from the RW base class, include files, default constructors/destructors, all virtual functions necessary for persistence, etc. The code generator has been used to develop a prototype persistent application of the Run Control Parameters database.

3.2 *OKS schema definition generator*

The DAQ configuration databases will be implemented using a two-tier approach; a light-weight in-memory persistent object manager to support the real-time requirements and a full ODBMS as a backup for long term data management. For the object manager a package called OKS [15] has been developed on top of Rogue Wave's Tools.h++ C++ class library. The OKS system provides a tabular GUI and a C++ API for DB schema definition. The OKS schema definition generator takes as input a schema definition from StP's object diagrammer and generates the necessary C++ API calls to build an equivalent schema in OKS.

3.3 *FSM (Finite State Machine) generator*

AONIX do not deliver any code generation scripts for the dynamic model in StP. In order to investigate the possibility of code generation for the Dynamic model .fsm (implementation language for FSM[16]) was generated. This was a language used during the RD13 phase of the project to describe dynamic behaviour for the DAQ system. FSM's are simple; no use is made of concurrency, nesting, guard conditions and no attributes are passed. The evaluation was a success and usable code was generated.

3.4 *CHSM (Concurrent, Hierarchical, Finite State Machine) generator*

Due to the limitations of FSM, the CHSM [17] language system was chosen [18] as the means by which behaviour will be expressed for the DAQ. CHSMs are a realization of Harel statecharts [19] just as FSMs are a realization of state-transition diagrams. The CHSM code generator uses both the State Diagrammer and State Table Editor as input. The CHSM generator supports nesting, concurrency, default entry/exit actions, inheritance, states and transitions.

4. Assessment of the StP CASE tool

The main strength of StP, as it is delivered from AONIX is as a design tool, providing a platform for requirements discovery, analysis, design and partial implementation of Object-Oriented applications. However, the implementation phase can and has been extended successfully. StP provides very good support for the OMT method and integrates Jacobson Use Cases with OMT models for capturing system requirements and processes. This should allow an easy transition to the forthcoming UML [20] which provides the best hope for standardisation in the OO method arena.

Via the QRS and the central data repository, StP provides code generation, document generation and diagram checking through QRL scripts and the Script Manager. This architecture has proved to be very flexible and one of the best features of the tool, allowing us to develop code and document generators specific to the project. The Repository Browser proved to be a very effective tool for viewing the repository data and its structure, which is useful for writing QRL scripts for code generation.

When writing code generation scripts, one has to take care to delete object references that are held in the StP repository otherwise these objects appear in the generated code as if they were full objects present in the repository. It is also possible to corrupt the StP repository e.g. if an object is renamed without using the 'global object re-name facility' that is provided, then the object is duplicated with the new name. In this situation there is no easy way to delete the old object reference and one has to delete and then re-build the repository.

The QRL language, although it is structured rather than Object-Oriented, provides good support for PDM types as well as functions to access the repository. There is also support for abstract types; lists, bags and functions to manipulate them as well as support for strings and string manipulation functions. All values are passed by reference and there are also a limited number of system functions. A debugger is provided, however it is primitive without stop, step, etc. Hence one has to rely more on warning and error messages generated by the QRL interpreter.

StP is well integrated with third party tools; the integration of StP with the SNIFF+ programming environment made reverse engineering possible as well as providing facilities not normally found in a design tool (intelligent editors, makefiles support, compilers, etc.). The fully integrated test tool StP/T automatically generates test cases directly from analysis and design models, test execution however, requires additional work.

Designs can be checked for correctness and consistency within and across all models and views, enabling design flaws to be detected in the early stages of development.

If code is generated to the same file more than once some parts of the file are preserved as part of incremental code generation. This is done via special StP tags placed in the generated code. If the user introduces additions to

the code outside of the tags, then updates the design and re-generates the code, the changes will be retained providing they were made outside of the tagged area.
From the administration point of view there have been a number of problems that arise mainly from the integration with the SYBASE repository. The size of the repository is fixed, as is the size of individual systems inside the repository. If the repository or a system becomes full, its size then has to be increased manually. The installation was also found not to be straight forward with a lot of low-level problems. In short one can say that the link to SYBASE is not transparent and the administrator requires a knowledge of SYBASE to maintain StP.

5. Conclusions and Future

The StP architecture has allowed us to generate code and documentation tailored to our specific requirements. The initial investment in developing code generators has encouraged developers to use StP and should pay off later on in the development cycle, as well as allowing us to focus our efforts on better designs. The integration of StP/OMT into our development environment has been very smooth, largely due to StP's support for configuration management and document generation in FrameMaker [21]. The StP development cycle fits well with the ATLAS DAQ schedule for the development of the Prototype -1, therefore the adoption of StP/OMT has provided a disciplined approach to development. Although StP is primarily a design tool its integration with the T test case generator and the SNIFF+ programming environment has allowed us to test, reverse engineer and also deploy products. Our experience has been very positive and we believe that automating software production from designs in StP will ultimately save time and effort, giving fewer coding errors and allowing developers to spend more time on formal designs and documentation. We will continue to use StP to develop the prototype -1 and develop code generators to automate software production in ATLAS DAQ.

6. References

[1] ATLAS software development environment working group, ATLAS Software Development Environment User Requirements Document, http://ptsun00.cern.ch/~ruggier/AtlasSE/URD/URD.1.0/html/.

[2] L. Mapelli et al, The ATLAS DAQ and Event Filter Prototype -1 project, published in these proceedings.

[3] Information on StP/OMT is available on WWW at: http://www.ide.com/Products/StP/StP_OMT.html.

[4] James Rambaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, Object Oriented Modeling and Design, Prentice Hall 1991.

[5] Ivar Jacobson, et al, Object Oriented Software Engineering, A Use Case Driven Approach, Addison-Wesley, New York, NY, 1992.

[6] Steve Fisher, Design Report, Rutherford Appleton Laboratory (RAL), http://hepunx.rl.ac.uk/atlas/oo/tools/stp/design.html.

[7] SNIFF+, TakeFive Software, http://www.cern.ch/PTTOOLS/Sniff/.

[8] StP/T User Guide, AONIX.

[9] ATLAS DAQ Technical Note 24, StP/T Evaluation Report, http://atddoc/Atlas/Notes/024/Note024-1.html.

[10] StP Core, User Manual, Query and Reporting System, AONIX.

[11] StP Core, User Manual, Object Management System, AONIX.

[12] StP Core, User Manual, Fundamentals of StP, AONIX.

[13] R. Jones et al, Software Technologies for a Prototype ATLAS DAQ, published in these proceedings.

[14] Tools.h++ foundation class library for C++ programming User Guide, Rogue Wave Software.

[15] Igor Soloviev, OKS documentation Set, http://atddoc.cern.ch/Atlas/Notes/033/Note033-1.html.

[16] RD13 Technical Note 5, RunControl for RD13,http://rd13doc.cern.ch/public/doc/Note5/Note5.html.

[17] Paul Lucas, An Object Oriented Language for Implementing Concurrent, Hierarchical, Finite State Machines, MSc. Thesis.

[18] ATLAS DAQ technical note 12, Evaluation of the CHSM language system, http://atddoc/Atlas/Notes/012/Note012-1.html.

[19] David Harel, et al. On the Formal Semantics of statecharts, Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, IEEE Press, NY, 1987.

[20] Grady Booch, James Rambaugh, Ivar Jacobson, The Unified Modeling Language for Object-Oriented Development. Rational Software Corporation.

[21] FrameMaker, Adobe Systems inc.