WWV 2005 Preliminary Version

Generating commercial web applications from precise requirements and formal specifications

David Crocker [1]

Escher Technologies Ltd.
Aldershot, United Kingdom

John H. Warren [2]

Precision Design Technology Ltd.
Maidenhead, United Kingdom

Abstract

We present a new model-based approach that we are using to build commercial web-based applications. The user requirements together with a data model are formally specified in a graphical notation using the CREATIV toolset. The specification may be checked by animation before being automatically translated to *Perfect* notation. The *Perfect Developer* toolset uses automated reasoning to generate formal proofs of correctness. It then generates C++ or Java code which, in conjunction with an application framework also written in *Perfect*, forms the complete application including the HTML user interface. The whole process provides a rapid turnaround from new requirements to a formally-verified application.

Key words: formal specification, formal verification, web applications

1 Introduction

A recent survey of 1027 information technology projects [1] found that only 12.7% were deemed successful. Poor requirements were considered to be a contributory factor in 76% of the failing projects. These figures indicate that a better approach to IT system development is needed and that any new approach must include better definition of requirements. We present such an approach.

[1] Email: dcrocker@eschertech.com
[2] Email: john.warren@precisiondesign.co.uk

2 Defining Requirements with CREATIV

One important specification dichotomy is that while user understanding requires the use of natural language, correctness requires the precision of a mathematical approach. The CREATIV approach offers an improved process for requirement specification that aids user understanding by using natural language while ensuring consistency and functional correctness by using concealed mathematical logic.

The underlying approach is model-based and axiomatic [2]. A proof in an axiomatic theory is a finite sequence of statements in the theory in which each statement either is an axiom or derives from an earlier statement by applying a rule of reasoning. The CREATIV reasoning process uses five axioms and one rule of inference, which reads informally as: IF a candidate entry satisfies its preconditions THEN add the entry to the specification AND include the consequential closure that results from its addition. The tool imposes some of the preconditions and the application itself imposes others. The reasoning occurs in a metamodel that represents the requirements process, represented in first order predicate calculus; the metamodel construction is also axiomatic, and uses identical axioms and rules of reasoning.

Starting from the axioms and the rule of inference, we can prove a succession of theorems at a very low level. There are several hundred such theorems, which are used as lemmas to support higher level proofs. We then present the reasoning system, specified in its own notation, for proof in exactly the same way. We can then present the application specification similarly. This layered organisation of the approach has an important advantage: we can express reasoning about specification behaviour (animation) as one more layer in the reasoning process and one that uses the same approach and rule of inference.

The tool can translate a correct specification into other languages, such as *Perfect* or Z, because there is one well-defined meaning associated with each theorem. When we translate into *Perfect*, which has its own more powerful theorem prover, we can also generate a large number of hypotheses, by rule, that express properties that the application code should possess. Users may also state further hypotheses, arising from their own knowledge. We expect all these hypotheses to be provable; proof failures almost always indicate important specification errors.

Translation to *Perfect* also allows immediate generation of Java or C++, so the production of web-based systems can be automated once the specification is complete. We guarantee that any proven property will be present in the generated system.

In order to construct a specification using this approach, it is necessary to construct the business model (diagram) and the supporting information (text). While analysts can enter information in any order, it is usual to develop a part of the diagram first, then to add the supporting text, using form-filling dialogues, and finally to check this partial specification or model. Analysts can

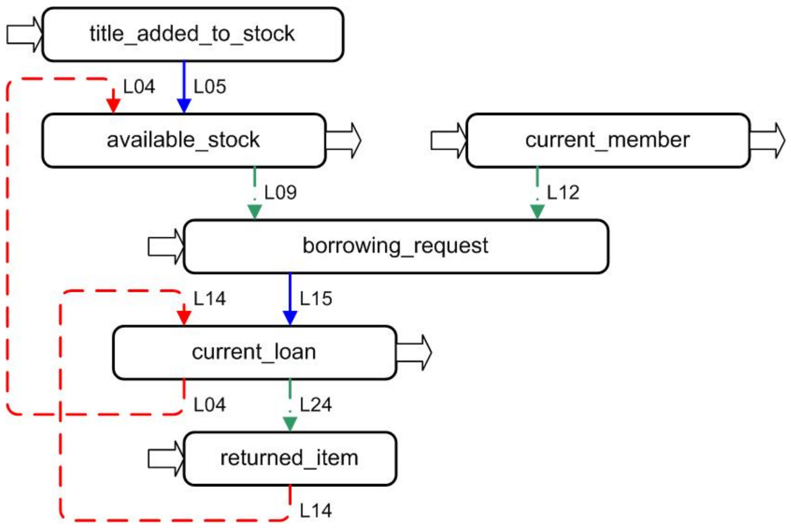

Fig. 1. Library model

then correct any errors and extend the partial model by repeating the above process. The CREATIV workbench supports this process and retains all the entered material in an underlying database. Analysts can make new entries in the database and change or delete existing entries. At user request, the workbench checks that all the entered facts are provable and self-consistent, by translating the database content into a formal notation, attempting mathematical proof, and reporting any proof failures. Since this is at user request, specifications can be temporarily inconsistent during their development; this makes those constructs that involve iteration and circularity easier to define.

We define a specification as an assembly of five structured collections of information; of these, three contain the data, functions, and events of the application. Of the other two, concepts conjoin data, function, logic, and event material to provide a view of the data within a class; constraints (or business rules) establish how data moves between processes within the system. Each constraint defines a data flow line, the specific data transfer (defined by one or more predicates), and the type of flow that occurs. Each destination concept includes a logical expression that must be satisfied before any flow can occur.

Each collection (data, function, event, concept, and constraint) contains one or more relational tabulations of facts relevant to the requirement specification. The entry of these facts constitutes the specification process. The CREATIV specification workbench guides users through this entry process with no reference to any mathematical notation. The totality of this information allows the derivation of a formal specification that will support detailed reasoning. In particular, we can predict the behaviour of the specification to arbitrary sequences of input data; this not only allows users to test the correctness of the specification but also provides acceptance test data and expected results.

The notation is similar to the Business Activity Diagram of UML but provides more detail. An example is shown in Fig. 1, which represents a lending library. The library acquires books (concepts that accept external events are marked with an input arrow), each entered as *title_added_to_stock*. For each such entry, the L05 constraint transfers selected attribute values to represent *available_stock*, that is, items available for loan. We similarly enter each *current_member*, who must be traceable if he is to borrow books, though in this case there is no consequential inference. The L12 broken line indicates read-only access to this data.

Any person can make a *borrowing_request* for any book but only members can borrow books (L12) and members can only borrow available books (L09). Requests by non-members are unsatisfiable, as are requests for non-present books. If the logic "L09 AND L12" is satisfied (both values are true) then the *borrowing_request* is accepted. An entry is added to *current_loan* to reflect the loan (L15) and as a further consequence, the item is deleted from *available_stock* (L04), indicating that the item is no longer available. The dashed line indicates a logical inversion.

Upon completion of the loan, the borrower returns the book and this event is entered in *returned_item*. Provided that the entry matches a borrowed item (as required by L24), the matching entry is deleted from *current_loan* (L14) and reinstated in *available_stock* (L04), since the deletion is now inverted. If we enter a small number of books and members, we can test the model specification by submitting borrowing request and returned item events (both sound and faulty).

This simple model is inadequate as a library model; but because it is easy to understand, it is easy to enquire about its deficiencies. For example: how do we limit the number of loans to one borrower? How do we represent members who leave the library?

The CREATIV tool can reason about the model to verify that the specification is self-consistent. Next, we can animate the specification in order to demonstrate the behaviour of the model to users of the proposed system, allowing specification correctness to be tested. It is entirely usual for initial versions of a specification to contain errors; the analyst corrects these until the users agree the correctness of the specified behaviour.

Documentation is generated automatically, by direct deduction from the specification facts. We can produce a structured English specification, UML documentation such as interaction diagrams, and various additional elements such as checklists and tabulations.

3 Generating and Verifying the Application with *Perfect Developer*

Perfect Developer [3], [4] is an object-oriented formal tool which provides for the definition of state-based specifications and related functional requirements.

It also has facilities for manual or automatic refinement to a lower-level specification, from which C++ or Java code can be generated. Included is an automated reasoning engine which attempts to prove that the specifications are complete and consistent, the specifications meet the requirements, and the refinements preserve the observable behaviour apart from resource usage.

The *Perfect* specifications generated by CREATIV are designed to work in conjunction with an application framework that has been written directly in *Perfect*. These specifications include a model of the relational database tables needed to represent the data, and a large number of operations specifying how the state is to be changed in response to various events. Also generated is a partial data dictionary, for use in generating HTML screens to interact with the user.

The application framework provides the base classes from which classes in the CREATIV-generated specification are derived. It automates the generation of HTML pages for user interaction and the parsing of returned form data. A small supporting library allows it to be built as a web application using the CGI interface. We have not yet made much effort to achieve full formal verification of the framework, so at present only 92% of the 1108 proof obligations that it gives rise to are discharged automatically. Most of the failures relate to proof obligations from library components used to perform file I/O and match regular expressions, rather than from the framework itself.

4 Case Study

As a commercial example of this approach, we have specified part of a web-enabled database system for a UK government department. We have currently specified about 10% of the system; this appears as 1340 specification clauses and translates at present to 35,799 lines of *Perfect*. The subsequent code generation produces 62,224 lines of Java. *Perfect Developer* also generates 9,819 proof obligations relating to the CREATIV-generated files, all of which are proven automatically. The entire generation process (axiomatic specification proof, translation to *Perfect*, and construction of Java code) takes less than 30 minutes on a laptop machine of modest speed (750 MHz). Complete proof by *Perfect Developer* of the generated obligations requires about 4 hours 20 minutes on the same machine (averaging about 1.6 seconds per proof).

5 Related Work

Automatic code generation for large parts of web applications from semi-formal or informal specifications (such as UML) is widely used. An approach to the construction of web applications from formal specifications is outlined in [5]. The use of formal specifications for testing web applications has been proposed by a number of authors including [6].

6 Conclusions and Further Work

We have shown that it is possible and practical formally to specify and verify a substantial part of a web-based commercial application, and to generate code from the specifications. We contend that the generated system is inherently immune to buffer overflow attacks. This will be proven formally when we achieve complete verification of the application framework and associated library.

There are some limitations to the present version of the workbench. Both reasoning systems are founded on predicate calculus and are unable to reason about temporal properties, non-functional properties or concurrency. CREATIV uses the relational model for information storage; this may not be the best representation for some applications.

In the future we intend to specify formally a subset of the W3C HTML 4.01 definition in *Perfect*. This will allow us to prove not only that the application always generates well-formed HTML pages, but also that the system is immune to cross-site scripting attacks.

References

[1] Taylor A, *IT Projects Sink or Swim.* In “BCS Review 2001”, British Computer Society. Available at `http://www.bcs.org/review/2001/articles/itservices/projects.htm`.

[2] Warren J.H and Oldman R.D, *A Rigorous Specification Technique for High Quality Software.* In “Proceedings of the Twelfth Safety-Critical Systems Symposium” (ed. F.Redmill and T.Anderson) 43-65, Springer-Verlag (London) (2004). ISBN 1-85233-800-8.

[3] Crocker D, *Safe Object-Oriented Software: The Verified Design-By-Contract Paradigm.* In “Proceedings of the Twelfth Safety-Critical Systems Symposium” (ed. F.Redmill and T.Anderson) 19-41, Springer-Verlag (London) (2004). ISBN 1-85233-800-8 (also available via `http://www.eschertech.com`).

[4] Crocker D and Carlton J, *A High Productivity Tool for Formally Verified Software Development.* To be published in the International Journal of Software Tools for Technology Transfer, Special Section on Formal Methods 2003.

[5] Fons J, Pelechano V et al., *Extending an OO Method to Develop Web Applications.* The Twelfth International World Wide Web Conference, Budapest, Hungary. At `http://www2003.org/cdrom/papers/poster/p329/p329-fons.htm`.

[6] Xiaoping Jia and Hongming Liu, *Rigorous and Automatic Testing of Web Applications.* At `http://jordan.cs.depaul.edu/research/web-test-paper.htm`.