ARTICLE IN PRESS

Science of Computer Programming ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

# Model-driven engineering of information systems: 10 years and 1000 versions

Jim Davies *, Jeremy Gibbons, James Welch, Edward Crichton

*Department of Computer Science, University of Oxford, Oxford OX1 3QD, UK*

ARTICLE INFO

*Article history:*
Received 9 March 2012
Received in revised form 26 November 2012
Accepted 7 February 2013
Available online xxxx

*Keywords:*
Model-driven
Evolution
Data migration
Formal methods
Agile
Databases

ABSTRACT

This paper reports upon ten years of experience in the development and application of model-driven technology. The technology in question was inspired by work on formal methods: in particular, by the B toolkit. It was used in the development of a number of information systems, all of which were successfully deployed in real world situations. The paper reports upon three systems: one that informed the design of the technology, one that was used by an internal customer, and one that is currently in use outside the development organisation. It records a number of lessons regarding the application of model-driven techniques.



## 1. Introduction

The Object Management Group's concept of *model driven architecture* (MDA) [1] has been explained as the use of modelling languages 'as programming languages rather than merely as design languages [2]'. The broader concept of *model-driven engineering* is a natural extension of this: using models to support automated analysis of a design, as the basis for automatic test generation, and as a source for automatically-generated documentation. It may be seen also as a natural extension of the concept of *formal methods*: languages and tools with a precise, mathematical foundation. If models are being used to drive processes of implementation and development, in the sense of being used to produce structures or behaviours automatically, then they must admit a precise interpretation. Furthermore, as they describe features or requirements at a higher level of abstraction than the programming notations they augment or replace, they are more amenable to mathematical characterisation.

In this paper, we will report upon our experience of a particular, original approach to model-driven engineering: one that was inspired directly by formal languages and tools. We will introduce and explain the *Booster* technology, the systems that we have built using it, and the lessons that we learned along the way. This explanation will focus upon three systems: one that supports our own activities, another developed for an internal customer, and a third that is presently in use outside our organisation. The lessons that we present here are informed also by the other systems we have built, and should be equally applicable to other developments using model-driven techniques.

The paper starts with an introduction to the Booster toolkit: the modelling language and the code generation process. In Section 3, we discuss the first system built using Booster: a system whose development and requirements for support

* Corresponding author.
*E-mail addresses:* jimdavies@mac.com, Jim.Davies@cs.ox.ac.uk (J. Davies), Jeremy.Gibbons@cs.ox.ac.uk (J. Gibbons), James.Welch@cs.ox.ac.uk (J. Welch), Edward.Crichton@cs.ox.ac.uk (E. Crichton).

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2013.02.002

Please cite this article in press as: J. Davies, et al., Model-driven engineering of information systems: 10 years and 1000 versions, Science of Computer Programming (2013), doi:10.1016/j.scico.2013.02.002

had a profound influence upon the design of the toolkit. We present some lessons in domain-specific development, data migration, and in software engineering process. In Section 4, we discuss a system built for an internal customer, supported in operation for two years, and then replaced by a supposedly-identical product built using conventional techniques on a standard platform. We present some lessons in the business and organisational context of model-driven engineering.

In Section 5, we discuss the design and operation of a system that is presently in use, for a purpose that is both important and sensitive, by a partner organisation. This system was built on the experience of the others, and we expect it to evolve and continue in use for many years. We present some lessons in dealing with legacy data, in scalability, and in migrating to other technology platforms. The paper ends with a brief summary of the experiences and lessons, an explanation of related work, and an account of prospects for further development.

## 2. Booster

In 1998, following a competitive tendering and procurement process, the University of Oxford asked B-Core (UK) Limited to develop a new information system. The system would manage key aspects of the University's professional programmes in software engineering: programme finances, student registration, course attendance, on-line materials, submissions and examinations. Its development was to be managed for the University by Jim Woodcock and Jim Davies, both experienced users of formal methods: in particular, the Z notation [3]. B-Core (UK) was a company founded upon the industrial application of formal methods: in particular, the B-Toolkit [4].

Although the contract made no mention of formal methods, both sides were happy at the idea of using these techniques in the design process: indeed, given the level of expertise, and the shared, stated belief that the application of formal techniques could greatly reduce costs and increase reliability, not to do so might have seemed surprising to colleagues and customers alike. Both parties were aware that these techniques were originally developed for use in large, industrial projects, and that their application to the development of a 'simple database' could easily be counter-productive. Thus the initial plan was merely to provide a formal description of key features of the design alongside the conventional, written functional specification.

Soon after the specification, and the document, had been agreed with the developers, it became apparent that a system built exactly in accordance with it would fail to meet the users' needs and expectations. With each question from the developers, with each step of the implementation, it became apparent that the specification did not capture every aspect of the original intention, that there was scope for misinterpretation, and that some of the design decisions, so precisely expressed in the Z, were wrong. This can be explained in part by the emerging complexity of the proposed system – no longer a 'simple database' – and in part by the changing nature of the requirements: there were new regulations, new policies, and new structures to be taken into account.

It is indicative also of an inherent difficulty in the application of formal techniques: one that can be found also in the application of model-driven engineering. A precise description of intended structure and behaviour may contain a great deal of information: more than can be comfortably understood or managed by the unaided mind. Without tools for animating or otherwise validating the Z document, it served as a sketch of intentions, but nothing more; indeed, the same purpose would have been served by an object model written in a less formal language. A more concise, more abstract description might have allowed more in the way of manual exploration, and hence a more reliable, manual interpretation of the sets and symbols defined, but would have left many of the key features unspecified.

Nevertheless, the document was updated, repeatedly, to reflect changes in requirements and understanding. After several iterations, the developers observed that they might usefully treat the design document as source code for the system. This observation was based in part upon Ib Holm Sørensen's experience on the IBM CICS Project [5], where many of the Z specifications produced were deterministic at the chosen level of abstraction: that is, the value of each attribute or output included in the description would be entirely determined after each operation. This was captured as an informal principle:

**Principle of Functional Specification.** In a useful, formal specification, the value of each state variable should be uniquely determined after every operation.

If a variable is worth declaring, at a particular level of abstraction, then you ought to know precisely what happens to it. We leave it to the reader to suggest exceptions to this rule; we observe simply that this was proving to be the case.

At the same time, the Z notation was proving less than ideal for the description of the proposed design. We were describing the state of the system as a collection of classes, representing information held on students, courses, assignments, or invoices. As might be expected, many properties of interest concerned the relationship between attributes of different classes, causing us to adopt an idiomatic form of Z in which the model would include given sets of references to each class, and a generic dereferencing function. Furthermore, there was no mechanism for presenting each operation in the context of the 'owning' class. This led to considerable amounts of additional clutter in a model intended for manual inspection.

The Object-Z [6] notation was considered as an alternative, but a lack of tool support for presentation – drawing boxes around boxes of any significant size or number did not appear to be a scalable approach without some form of 'folding editor' – or analysis led to the decision to define a new object-based notation that could be supported by the existing B-Toolkit. Descriptions written in this notation would be translated automatically into the Abstract Machine Notation (AMN), and then refined automatically to produce an implementation of the core database functionality, based upon the toolkit's C libraries of datatypes and functions. Sørensen named this the B Object Oriented STate-based Refinement approach, or *Booster*.

Please cite this article in press as: J. Davies, et al., Model-driven engineering of information systems: 10 years and 1000 versions, Science of Computer Programming (2013), doi:10.1016/j.scico.2013.02.002