Foreword

There's no question that use of MDA and other Model-Driven Software Development (MDSD) techniques is spreading rapidly. In mid-2002 BZ Research found that only 7% of software developers were using code generated from models in their applications (and even then always with some hand-tweaking). By mid-2008 a commissioned study conducted by Forrester Consulting on behalf of Unisys showed that 22% of developers claimed to be using MDD to drive development through some level of code generation.

There are several reasons for this strong growth. An obvious one is the increased productivity achieved when writing new software applications. Multiple studies over several years have shown productivity gains of around 35% when creating new code, compared to using traditional coding techniques. In spite of having to learn many new skills, software development teams nevertheless often report improved productivity from their very first MDSD project.

It's significant that these productivity gains also carry over into software maintenance. Model-driven software production would be little use if it made maintenance harder, since maintaining existing software today consumes up to 90% of all IT spending. However, laboratory studies show similar productivity improvements when maintaining model-based applications, compared to maintaining those where no precise model is available. In fact, the most spectacular MDSD benefits are often realised when moving an application to a new platform or a major revision of an exiting platform. Making changes to a code generator and re-generating a software application from source models can achieve in hours what might take months if the application code has to be modified by hand.

Another compelling reason for using MDA and other MDSD techniques is the quality of the resulting software. Buggy code that doesn’t meet the its specifications costs more to own than high quality software, either because of the cost of finding and fixing the bugs, or the cost of the work-arounds required to compensate for them. Here again, empirical results are encouraging, with software generated from models having about one third the defect rate of hand-written code. As more software is found in safety-critical contexts, these improvements in code quality could literally become a matter of life and death.

It is against this background of the increasing use, importance and proven benefits of MDSD and Model-Driven Domain Analysis that Janis Osis and Erika Asnina have gathered an impressive list of international authors to describe significant new contributions to their theoretical foundations and survey how they are being applied. In this diverse collection of papers you will find contributions to several different aspects of these rapidly-growing fields. The thread that links them all, however, is a drive towards rigour and precision. While software development has traditionally been more of an empirical art than an engineering discipline, this is slowly changing as software users demand from software engineers the same levels of system reliability and predictability that are achieved by engineers in other disciplines. Several contributions to this book stem from the way that software is today increasingly
embedded within complex systems that have mechanical, electronic and software components; those who build these systems quite rightly demand that the software components are built to the same levels of rigour as the non-software parts. Other contributions focus on the important but often-neglected topic of software configuration, recognising that complex software systems are often assembled from hundreds or even thousands of distinct software components, which must work together predictably and without conflict. In this, as in other aspects of Model-Driven Engineering, papers in this collection give valuable insights into the problems being addressed and make useful contributions to advancing current practice.

The closely-linked fields of Model-Driven Domain Analysis and Model-Driven Software Development are advancing rapidly as practitioners and theoreticians work together to create new techniques and extend our understanding of how and why existing methods work (and also why they sometimes fail). This impressive collection of papers makes a valuable contribution to both fields.

Andrew Watson
Vice-President and Technical Director
Object Management Group™
Cambridge, UK

Andrew Watson has overall responsibility for OMG’s technology adoption process, and also chairs the Architecture Board, the group of distinguished technical contributors from OMG member organisations which oversees the technical consistency of OMG’s specifications. In previous lives Andrew researched distributed object type systems, wrote Lisp compilers and helped improve software engineering practices at large multi-nationals (but not all at the same time). He spends his spare time skiing down, gliding around or walking up mountains.