Designing precise and flexible graphical modelling languages for software development

Stephen John Cook

Designing precise and flexible graphical modelling languages for software development

Stephen John Cook M00549888

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the award of the degree of PhD by public works

February 2017

Abstract

Designing precise and flexible graphical modelling languages for software development

Stephen John Cook

Model-driven approaches to software development involve building computerized models of software and the environment in which it is intended to operate.

This thesis offers a selection of the author's work over the last three decades that addresses the design of precise and flexible *graphical modelling languages* for use in model-driven software development. The primary contributions of this work are:

- *Syntropy*: the first published object-oriented analysis and design (OOAD) method to fully integrate formal and graphical modelling techniques.
- The creation of the Object Constraint Language (OCL) and its integration into the Unified Modeling Language (UML) specification.
- The identification of requirements and mechanisms for increasing the flexibility of the UML specification.
- The design and implementation of tools for implementing graphical Domain Specific Languages (DSLs).

The starting point was the author's experience with formal specification techniques contrasted with the lack of precision of published object-oriented analysis and design methods. This led to a desire to fully integrate these two topics – formal specification and object-orientation - into a coherent discipline. The Syntropy approach, created in 1994 by this author and John Daniels, was the first published complete attempt to do this.

Much of the author's subsequent published work concerns the Unified Modeling Language (UML). UML represented a welcome unification of earlier OOAD approaches, but suffered badly from inflexibility and lack of precision. A significant part of the work included in this thesis addresses the drawbacks of the UML and proposes improvements to the precision of its definition, including through the invention of Object Constraint Language (OCL) and its incorporation into the UML specification, and the consideration of UML as source material for the definition of Domain Specific Languages (DSLs). Several of the author's published works in this thesis concern mechanisms for the creation of DSLs, both within a UML framework and separately.

Acknowledgements

I would like to thank all of the co-authors, collaborators and colleagues who have helped me, worked with me and encouraged me over the past thirty years. There are far too many to list individually without the risk of leaving somebody out.

I thank Professors Balbir Barn and Tony Clark for agreeing to supervise me and for their suggestions and critical comments on various drafts of this work.

This work is dedicated to my wife Hazel, my children Imogen, Laurence and Oliver, and to the late Stuart Kent.

List of submitted publications

PW1: S. Cook, “Languages and object-oriented programming,” Software Engineering Journal, vol. 1, no. 2, pp. 73–80, 1986. © Reproduced by permission of the Institution of Engineering & Technology.

PW2: S. Cook, “Modelling Generic User-Interfaces with Functional Programs,” in Proceedings of the HCI’86 Conference on People and Computers II, 1986, pp. 369–385. © 1986 Cambridge University Press. Reprinted with permission.

PW3: C. T. Burton, S. J. Cook, S. Gikas, J. R. Rowson, and S. T. Sommerville, “Specifying the Apple Macintosh™ Toolbox event manager,” Formal Aspects of Computing., vol. 1, no. 1, pp. 147–171, 1989. © 1989 BCS. Reprinted with permission of Springer.

PW4: S. Cook and J. Daniels, Designing Object Systems: object-oriented modelling with Syntropy. Prentice-Hall 1994. © Steve Cook and John Daniels, 1994. Available under the Creative Commons Attribution No Derivatives license http://creativecommons.org/licenses/by-nd/3.0/.

PW5: J. Warmer, J. Hogg, S. Cook, and B. Selic, “Experience with Formal Specification of CMM and UML,” in Object-Oriented Technologys, Springer Berlin Heidelberg, 1998, pp. 216–220. © 1998 Springer. Reprinted with permission.

PW6: A. Kleppe, J. Warmer, and S. Cook, “Informal formality? The Object Constraint Language and its application in the UML metamodel,” in «UML’»98 - The Unified Modeling Language: Beyond the Notation, Springer Berlin Heidelberg, 1998, pp. 148–161. © Springer-Verlag Berlin Heidelberg 1999. Reprinted with permission.

PW7: S. Cook, A. Kleppe, R. Mitchell, J. Warmer, and A. Wills, “Defining the context of OCL expressions,” in «UML»’99 - The Unified Modeling Language: Beyond the Standard, Springer Berlin Heidelberg, 1999, pp. 372–383. © Springer-Verlag Berlin Heidelberg 1999. Reprinted with permission.

PW8: S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. C. Wills, “The Amsterdam manifesto on OCL,” in Modeling with the OCL, T. Clark and J. Warmer, Eds. 2002, pp. 115–149. ©Springer-Verlag Berlin Heidelberg 2002. Reprinted with permission.

PW9: S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. C. Wills, “Defining UML family members using prefaces,” Proceedings of Technology of Object-Oriented Languages and Systems. TOOLS 32, 1999. © 1999 IEEE Computer Society. Reprinted with permission.

PW10: S. Cook, “The UML family: Profiles, prefaces and packages,” in ≪ UML≫ 2000 - The Unified Modeling Language: Advancing the Standard, Springer Berlin Heidelberg, 2000, pp. 255–264. ©Springer-Verlag Berlin Heidelberg 2000. Reprinted with permission.

PW11: S. Cook, “Domain-specific modeling and model driven architecture,” In The MDA Journal: Model Driven Architecture Straight from the Masters, ed. D.S.Frankel, J.Parodi and R.Soley, Meghan Kiffer Press, 2004. Also available online at http://www.bptrends.com/publicationfiles/01-04%20COL%20Dom%20Spec%20Modeling%20Frankel-Cook.pdf. © 2004 Steve Cook.

PW12: J. Greenfield, K. Short, S. Cook and S. Kent, “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools,” Wiley, 2004, pp. 279–336. © 2004 by Wiley Publishing, Inc., Indianapolis, Indiana. Reprinted with permission.

PW13: S. Cook, “Separating concerns with domain specific languages,” in Modular Programming Languages, Springer Berlin Heidelberg, 2006, pp. 1–3. © Springer-Verlag Berlin Heidelberg 2006. Reprinted with permission.

PW14: S. Cook, G. Jones, S. Kent, and A. C. Wills, Domain-specific Development with Visual Studio DSL Tools. © Addison Wesley, 2007. Chapter 1 reprinted by permission of Pearson Education, Inc., New York, New York.

PW15: S. Cook and S. Kent, “The Domain-Specific IDE,” CEPIS Upgrade, vol. IX, no. 2, pp. 17–21, 2008. © Novática 2008. Reprinted with permission.

PW16: S. Cook, “Object technology--a grand narrative?” in ECOOP 2006 - Object-Oriented Programming, Springer Berlin Heidelberg, 2006, pp. 174–179. © Springer-Verlag Berlin Heidelberg 2006. Reprinted with permission.

PW17: S. Cook, “Looking back at UML,” Software and Syst. Modeling, vol. 11, no. 4, pp. 471–480, 2012. © Springer-Verlag 2012. Reprinted with permission.

Declaration

The solely-authored submitted publications and the context statement are all the product of original work by the candidate. In the case of the jointly-authored submitted publications the candidate's contribution is accurately described as follows. None of the submitted publications has been or is being submitted for any other degree of a University or other degree-awarding body.

PW3: C. T. Burton, S. J. Cook, S. Gikas, J. R. Rowson, and S. T. Sommerville, "Specifying the Apple Macintosh™ Toolbox event manager"

This journal article was the result of a substantial piece of work carried out within the Alvey HI 059 project for which the candidate and Jon Rowson were the lead investigators. The candidate chose the case study. Technical work and writing was done in close collaboration in a workshop style by all five authors. Steve Sommerville edited the final document.

PW4: S. Cook and J. Daniels, Designing Object Systems: object-oriented modelling with Syntropy.

The candidate specified the formal language, which was based on Z, and articulated the key modelling perspectives. The book was written in close and detailed collaboration, with each author contributing about 50% of the text and contributing to every chapter. The candidate's primary focus was in the definition of the formal language and the detailed semantics of the modelling perspectives, including the details of how subtyping interacts with statecharts. John Daniels' focus tended to be more methodological.

PW5: J. Warmer, J. Hogg, S. Cook, and B. Selic, "Experience with Formal Specification of CMM and UML"

The CMM specification itself was edited by the candidate, who oversaw the specification and introduction of OCL and wrote the static semantics. This workshop paper is a summary of CMM; its text was written primarily by Jos Warmer, who was the main contributor to the job of integrating OCL into the merged UML 1 specification.

PW6: A. Kleppe, J. Warmer, and S. Cook, "Informal formality? The Object Constraint Language and its application in the UML metamodel"

This conference paper summarizes the way that OCL was used in the UML 1.0 specification. The example and the tutorial material on OCL come from Kleppe and Warmer. The candidate's contribution was leadership of the original definition of OCL and its incorporation into CMM, which provided the pattern for its use in UML.

PW7: S. Cook, A. Kleppe, R. Mitchell, J. Warmer, and A. Wills, "Defining the context of OCL expressions"

This set of authors constituted an informal working group who worked together on OCL semantics and applications. In the case of this conference paper the candidate was the primary author, and defined the problem and solution. The other authors provided technical feeback.

PW8: S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. C. Wills, “The Amsterdam manifesto on OCL”

This book chapter reported on a workshop by the informal OCL semantics research group which discussed and proposed solutions to several distinct problems identified in the OCL specification. The authors are listed in alphabetical order and each contributed ideas and fragments of text. The paper appeared in at least two revisions, each one being primarily edited by Rumpe.

PW9: S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. C. Wills, “Defining UML family members using prefaces”

The terminology “prefaces” originated from discussions between the candidate and Richard Mitchell. This conference paper was a collaborative effort by the informal OCL semantics research group. The primary authors were the candidate and Mitchell.

PW12: J. Greenfield and K. Short, “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools”

Chapters 8 “Language Anatomy” and 9 “Families of Languages” and Appendix B “The Unified Modeling Language” were co-written by the candidate and Stuart Kent in close collaboration. The chapters were divided into sections and each section was written by one of the authors and edited by both. The appendix was written by the candidate and edited by both authors. Only these two chapters and appendix are submitted as part of this thesis.

PW14: S. Cook, G. Jones, S. Kent, and A. C. Wills, Domain-specific Development with Visual Studio DSL Tools.

The candidate wrote chapters 1, 3, and 6 of this book which describes a Microsoft product developed by a team led by the candidate. Only chapter 1, solely written by the candidate, is submitted as part of this thesis.

PW15: S. Cook and S. Kent, “The Domain-Specific IDE”

This article in a special issue of the Upgrade journal of the Council of European Professional Informatics Societies (CEPIS) on Model-Driven Software Development is a report on work done in close collaboration between the candidate and Stuart Kent, written by the candidate.

Glossary

API	Application Programming Interface
BNF	Backus-Naur Form
BPMN	Business Process Modelling Notation
CASE	Computer Aided Software Engineering
CASL	Common Algebraic Specification Language
CFG	Context-Free Grammar
CMM	Core Meta-Model
DSL	Domain-Specific Language
DSML	Domain-Specific Modelling Language
EBNF	Extended Backus-Naur Form
ECMFA	European Conference on Modelling Foundations and Applications
ECOOP	European Conference on Object-Oriented Programming
FUML	Foundational Subset For Executable UML Models
IDE	Interactive Development Environment
IDEF	Integrated Computer-Aided Manufacturing (ICAM) Definition
JSD	Jackson System Development
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MOF	Meta Object Facility
NIEM-UML	UML Profile For National Information Exchange Model
OCL	Object Constraint Language
OMG	Object Management Group
OMT	Object-Modelling Technique
OO	Object-oriented, object-orientation
OOAD	Object-oriented analysis and design
OOPSLA	Object-oriented programming systems, languages and applications
RDF	Resource Description Framework
RFI	Request for Information
RFP	Request for Proposals
ROOM	Real-Time Object-Oriented Modelling
SDL	Specification and Description Language
SoSyM	International Journal on Software and Systems Modeling
SysML	Systems Modelling Language
TUM	Technische Universität München
UIMS	User Interface Management System
UML	Unified Modelling Language
WIMP	Windows, Icons, Menus and Pointing devices
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Contents

Context Statement

Model driven approaches to software development have been widely promoted and researched over the past thirty years under a number of slogans and acronyms: Model Driven Software Development (MDSD) [1], Model Driven Development (MDD) [2], Model Driven Engineering (MDE) [3] and Model Driven Architecture (MDA) [4]. Today there are numerous research initiatives dedicated to these topics, including the conferences *Models* since 1998 [5] and *ECMFA* since 2005 [6], the International Journal on Software and Systems Modeling (SoSyM) since 2002 [7], and many others. This entire body of effort establishes the overall landscape for the author's work submitted here.

The word "model" is worth explaining. Michael Jackson's book "Problem frames: analysing and structuring software development problems" [8] distinguishes two kinds of models: *analytic* and *analogic* (here he follows Ackoff [9]). An *analytic* model is a description of a system which is used to analyse that system; an example cited by Jackson is a model of an economy in the form of differential equations. An *analogic* model is a second system that has similar properties to the modelled system, for example toy aircraft on a large map used to represent the state of an ongoing battle. Jackson finds the difference between these types of model crucial in software development, and chooses to avoid the word model altogether for *analytic* models, preferring instead simply to say "description". However, in the mind of this author the discipline of model-driven development specifically constitutes the building of *analytic* models of software systems and the worlds in which they are embedded, so that a software developer can reason precisely about these systems by referring to a suitable abstraction without having to deal with all of the details of the modelled system. In this thesis the word *model* means a mathematical or logical construct that accurately represents aspects of a modelled system and which can be used to deduce (either directly or through a process of logical inference) conclusions about the properties and behaviour of the modelled system. A *graphical* model is one that can be rendered as one or more two-dimensional diagrams containing shapes and connections between them; and a *graphical modelling language* provides precise syntactic and semantic rules for creating and manipulating graphical models.

There are 17 publications included in this thesis, including articles from refereed journals and conferences, magazine articles, book chapters and a complete book. These were published between 1986 and 2012 and represent an evolving body of thinking about the precision and flexibility of graphical modelling languages, including several attempts to introduce that thinking

into widespread practice both through the publications themselves as well as through the author's involvement in the development of the international standard Unified Modeling Language (UML) and several software development tools.

The author is credited in many other publications which are not included in this thesis, either because they cover unrelated topics e.g. [10] [11] [12] [13], they are related but do not add materially to content already included e.g. [14] [15] [16] [17] [18], or they cover related topics but this author was not a primary contributor e.g. [19] [20] [21].

The 17 included publications are organized into four groups covering foundations, language precision, language flexibility, and a summary of work to date. Section V of this context statement offers a summary of the overall technical, managerial and political contributions of this work, while Section VI provides conclusions about the submitted work and comments on possible future work. References for citations in the context statement are provided. For each publication, the number of current citations according to Google Scholar is indicated. Finally, the publications themselves are presented, as copies bound into a separate volume.

I. Foundations

The first group comprises three papers that offer significant individual contributions in two areas foundational to the overall narrative of the thesis: object-orientation and formal methods. These papers explore these two areas in some depth but do not in themselves combine the themes of the thesis into a coherent whole.

Object-orientation. In the 1980s when personal computers with bitmapped displays were first emerging into widespread use, prevailing principles for organizing computer programs such as structured programming [22] were proving inadequate to the task of organizing programs that manipulated the new generation of interactive "WIMP" user interfaces (Windows, Icons, Menus and Pointing devices). New principles emerged, most notably object-orientation, constituted of a set of basic concepts: classes, types and instances, polymorphism, separation of interface and implementation, class inheritance and type substitution [23]. It took a number of years for these principles to become mainstream for the specification and design of computer programs.

Formal methods. The idea of creating a specification for a program separately from the program itself has existed since Ada Lovelace worked with Charles Babbage in the mid-19th century. Since the 1970s a large body of work has studied techniques for doing this *precisely*, mainly using mathematical formulae based on set theory, algebra and logic. Creating a formal specification allows the requirements for a program to be understood and refined prior to (or at least separately from) creating the program itself, and enables the program to be checked against its specification, by means either of formal proof or structured testing.

PW1: Languages and object-oriented programming

(23 citations)

This 1986 article attempts to provide an in-depth explanation of object-oriented principles at a time when these principles were not widely known or understood, certainly in the UK. The article explains about types and instances, generic types, abstract data types and polymorphism, which it explains under the heading of message/object programming. The article seeks to draw a distinction between data abstraction and object orientation, and it does this by setting up three criteria for recognizing object-orientation:

1. Message names are bound to methods at run-time depending on the receiver of the message;
2. The language includes an explicit class hierarchy;
3. Both control and data structures are uniformly represented as objects.

In hindsight, only the first of these is truly essential for object-orientation. Whether phrased in terms of messages and methods, invocation of procedures or functions, or otherwise, the real test of an object-oriented approach is that the method (procedure) that is invoked depends upon the individual object that receives the invocation at run-time. In class-based languages the chosen method depends on the class of the object, but some languages such as JavaScript [24] allow methods to be defined on individual objects: JavaScript is a *prototype-based* language, rather than a *class-based* language. Prototype-based object-oriented languages were just emerging in 1986 [25] and it seems that when the article was written this author was unaware of them. Prototype-based languages typically rely on a system of delegation rather than inheritance for sharing implementations and state between objects. In JavaScript, for example, every object is associated on creation with a prototype. Trying to retrieve a property value from an object, if the object lacks that property, will cause JavaScript to look for (*delegate* to) the property in the object's prototype. If the prototype is lacking the property, the search continues in *its* prototype, and the delegation process recursively continues, terminating at the Object.prototype object. The existence of prototype-based object-oriented languages contradicts criterion 2: the need for an explicit class hierarchy.

The author's observation that both control and data structures should be uniformly represented as objects represents an interesting goal, very strongly influenced by Smalltalk. However, later languages that are generally considered to be object-oriented such as Java did not attain this goal, although the more recent introduction of lambda expressions [26] into popular languages including Java [27] and C# [28] shows movement towards it.

The article compares four programming languages: Simula [29], Smalltalk [30], Clu [31] and Ada [32]. It concludes that of these Smalltalk is the only truly object-oriented language and Simula goes most of the way, whereas Clu and Ada do not. Clu and Ada can better be categorized as *interface-based* languages or data abstraction languages, providing the capability to separate interface from implementation which was also novel at that time.

In retrospect, the article appears somewhat idiosyncratic. It defines and recommends object-orientation, but its characterization is restrictive, although it does correctly identify the essential point, namely run-time binding of invocation to method depending on the recipient. This essential point is the basis of the definition of the Objective-C language, an object-oriented extension to the C programming language [33] invented by Brad Cox in the early 1980s [34] and still in use by Apple for their operating systems and APIs [35].

The article mentions single and multiple inheritance, an important point of debate at that time, but does not discuss benefits or disadvantages of either. It draws attention to a relationship between type inheritance hierarchies and generic types, but does not delve into the detail of this relationship, which has been subsequently studied in depth e.g. Bertrand Meyer's OOPSLA'86 paper [36].

The article's attempt to blur the distinction between programming language and operating system seems somewhat naïve with three decades of hindsight, because it misunderstands – or at least under-estimates - the value chain and security issues involved in delivering a high-performance system of interactive applications on a modern personal computer or device. Distributed systems, component-based development and service-oriented architecture were all in the future at the time this paper was written. In 2016 a computing system is necessarily composed of loosely-coupled components or services, where each component or service provides value to others, the whole being the result of the activity of numerous people and organizations. Usually the components and services are organized into layers, to avoid cyclic dependencies, where each component must protect itself against misbehaviour by its neighbours and users. There are many reasons, including performance, expressive power, verifiability, skills availability and taste, why the developer of a component may wish to choose a particular programming language or paradigm, and so the interface between components should not force this choice upon them.

Nevertheless, this article makes two important contributions. Firstly, it identifies the importance of objects and makes a reasonable attempt to define them, and secondly it points out that objects are pervasive in the sense that all aspects of an interactive computer system may be conceived in terms of them. Neither of these points are at all controversial in 2016, but in 1986 they were novel, and it was the software industry's recognition of these points that led to the development of object-oriented design languages over the following decade.

PW2: Modelling Generic User-Interfaces with Functional Programs
(6 citations)

This conference paper, which is a shortened version of a Queen Mary College internal report, introduced modelling into the author's vocabulary. Modelling means here the creation of an abstract description of a system rather than a programmed implementation; such a separation is a basic theme of this thesis.

At the time this paper was written, WIMP (Windows, Icons, Menus and Pointing devices) interfaces were still relatively novel. This author had recently been experimenting with

programming an early bitmapped workstation – the PERQ [11] – for which programming the user interface entailed a great deal of low-level platform-specific coding. The complexity of WIMP interfaces meant that a large portion of the overall effort in programming an interactive application on a computer with bitmapped screen and pointing device went into creating the user interface. To help with this, much research was being done into User Interface Management Systems (UIMS) and other techniques for formalizing the user interface independently from the application, working out how best to separate concerns such as presentation, dialogue management, user feedback, and interfacing to the application itself [37], [38], [39].

This paper falls within the same research area. It sets out to use a combination of object-oriented and functional programming principles to describe user interface aspects of a system conceptualized as a function. The intuition is that an interactive personal computer responds to a sequence of input events such as key presses and mouse gestures by generating a sequence of pictures. This is formalized as a function from the input so far to the current picture, and the meat of the paper describes how that function may be described in terms of subsidiary functions such as Display, InputHandler, Window and so forth.

The author's declared aspiration here is that such an abstract description might be amenable to formal proofs about certain properties of the modelled system, for example the reachability or otherwise of certain states. No such proofs are attempted, and indeed it is not clear that the functional programming approach is particularly amenable to such proofs: an axiomatic approach such as that in [PW3] would make a better foundation.

One of the key points made by this paper is the need for a concept of type substitutability in order to make the description extensible and enable the separation of user interface from application. The model exploits the concept of inheritance introduced by Cardelli in his 1984 paper "A Semantics of Multiple Inheritance" [40]. The paper extends Cardelli's language by adding pattern matching, recursive equations, "where" clauses and type inference. These additions to Cardelli's language certainly increased its expressive power, but it is not obvious that the result is sound; considerable additional work would be needed to prove that each expression in the language is correctly typed, and such work was not done in this paper.

Had the paper been written a few years later, it could have used an existing functional language such as Haskell [41]. Later again the Common Algebraic Specification Language (CASL) [42] could have provided the ability to construct the structures of this description in the form of a set of logical axioms and analyse their logical consequences by automatically generating examples and

counter-examples. Using CASL or the logical specification language Alloy [43] would certainly have been a better approach to satisfying the proof aspirations expressed for this paper.

PW3: Specifying the Apple Macintosh™ Toolbox event manager

(12 citations)

The third and final publication in this Foundations group represents another approach to using a formal specification language to create precise descriptions of abstractions involved in an interactive user interface. This work makes use of an existing specification language (Larch [44]) and an existing software system (the Macintosh toolbox) [45]. The work was done collaboratively over a significant period by a research group co-led by the author and funded by the Alvey research initiative [46], and represented a considerable learning experience for all participants, both in terms of the languages and systems studied, and in terms of the methodological issues arising from building a substantial formal specification.

The article makes a case for formal specification primarily by observing that the English documentation of the software is in many cases ambiguous or incomplete. One way to address this is to write experimental programs and tests to elucidate the exact behaviour of the software; the role of the formal specification is then to capture accurately the insights arising from such experiments.

Section 3.2 of the article discusses the choice of specification styles between a *constructive* approach, in which a specific abstract structure would be chosen to represent the toolbox state, and an *axiomatic* approach, in which a set of logical axioms are given which any toolbox-representing abstraction must satisfy, but which do not commit themselves to any particular abstraction. The axiomatic approach is chosen, and a sequence of specifications relating to the toolbox event queue is developed expressed using axioms such as the following:

Next(add(q,e)) = **if** isEmpty(q) **then** e **else** Next (q)

These axioms are presented in the form of logical equations, and their consequences are to be deduced by the normal rules of equational logic. The semantic domain for these equations is to be thought of as a many-sorted algebra, where the sorts involved in the axiom above would be Queue, Element and Boolean. None of the sorts are explicitly declared in these axioms. The authors have taken the liberty of using λ-expressions, higher-order functions and where-clauses to make the axioms more compact and expressive.

These specifications are developed in sequence, incrementally adding features by modifying and adding axioms: Basic queue; queue with masking; priority queue with masking; priority queue/stack with masking; removing discarded events. The end result is then transformed into Larch traits, and this shows the limited expressive power of Larch, because many of the conveniences introduced for expressive power have to be removed, and all operations have to be explicitly typed. Having transformed the specification into Larch, several of the types are explicitly declared to correspond to the actual toolbox: for example, event types {null, update, autokey, mouse, keyboard, activate, deactivate}. The Larch specification is fleshed out quite fully for the behaviour of the event queue, and finally used to specify the Pascal toolbox function GetNextEvent.

As a result of this work the authors drew a number of conclusions about how to create algebraic specifications, and about Larch itself. In the former category the team tried and discarded several different approaches to developing the specification before landing on the approach described in the paper. Only after several iterations and readings of the documentation did it become clear that the key semantic element was the behaviour of the event queue with different kinds of events; once this had been identified, the equational formulation enabled the team to make good progress towards defining the essential abstractions. Finally, the specification had to be transformed into Larch and bound to the practicalities of the toolbox. Because Larch does not support higher-order functions the resulting specification was more verbose, and we found that the definition of Larch/Pascal needed to be extended with an “expects” construct to handle the concept of global variable (the singular event queue) which was an essential feature of the toolbox.

Conclusion

This first group of papers contributed towards the exposition of object-orientation and formal methods, which are fundamental to all of the work presented later in this thesis. At the conclusion of this work the author had learnt much about the technicalities of object-oriented design and programming, and separately about the practicalities of formal specification using a variety of different techniques: functional, logical and algebraic. What came next were attempts to combine these elements into a single paradigm for specifying software systems. To that end, section II of this thesis explores the integration of formal methods into diagrammatic object-oriented specification languages.

II. Language precision

The years 1990-1994 saw the emergence of a significant number of dissimilar object-oriented analysis and design methods, each one represented by one or more books and usually commercially associated with CASE (Computer Aided Software Engineering) tools. These included methods by Booch [47], Coad and Yourdon [48], Jacobson [49], Martin and Odell [50], Reenskaug [51], Rumbaugh [52], Shlaer and Mellor [53], and others. These typically introduced a diagrammatic notation, explained by means of examples, intended to enable a designer to create a high-level depiction of a software system prior to its detailed implementation, thus helping to bridge the gap between articulating the system's requirements and designing its implementation. Many of these methods became commercially available in the form of CASE tools that provided graphical editors for the notations coupled with various forms of validation and code generation capabilities: Rational Rose for the Booch method, Select SDL for Rumbaugh's Object Modeling Technique, ObjectOry for Jacobson's approach, and many others. In what follows these pre-1994 methods will be called *first-generation* object-oriented analysis and design methods.

In 1994 the Object Management Group (OMG) set out on a process to standardize object-oriented analysis and design notations, culminating with the publication of the Unified Modeling Language (UML) specification in 1997. The details of this process are covered in depth in [PW17]; it involved studying the first-generation methods, creating a Request for Proposals that solicited submissions from the authors of these methods and others, and finally consolidating these submissions into the published UML specification. This author was a primary contributor to all of the major releases of UML.

None of the first-generation object-oriented methods contained significant formal content and their use did not involve the formal manipulation of expressions representing mathematical or logical entities. In this sense they were not *precise*: their interpretation was informal, and they offered no system for deduction or verification apart from whatever was provided by specific CASE tool implementations.

The group of publications presented in this part of the thesis addresses this imprecision in first-generation graphical modelling languages by adding a logical constituent, based on the author's experience with formal methods. The first and primary contribution is the Syntropy method introduced in 1994; subsequent contributions chart the introduction of ideas derived from Syntropy into the Unified Modeling Language (UML) and Object Constraint Language (OCL).

PW4: Designing Object Systems: object-oriented modelling with Syntropy
(612 citations)

This author, working with colleague John Daniels and others, became increasingly dissatisfied with first-generation object-oriented methods when putting them to practical use in a commercial environment. There were two key points of dissatisfaction. Firstly, the methods were inherently imprecise in their definition, and the actual meaning of models would only emerge through the actual behaviour of a CASE implementation; there were no accurate ways to reason about the meaning of a model apart from implementing it. This imprecision was particularly frustrating to the author given his experience of formal specification. Secondly, the words *analysis* and *design* as promoted by these methods often appeared to offer a distinction without a difference. Analysis is supposed to focus on functional requirements while design is supposed to focus more on implementation concerns. However, making this distinction in practice is very hard to sustain without a much better description of the difference. First-generation methods tended to subscribe to one of two competing ideologies: the *elaborational* approach vs the *translational* approach [54]. In the elaborational approach, most typically exemplified by Booch [47], the path from analysis to design involves adding and adjusting details (elaborating) until the model represents the desired implementation sufficiently accurately. Here, the distinction between analysis and design is fundamentally ambiguous. By contrast, the translational approach, of which the archetype was Shlaer and Mellor's Recursive Design [53], proposes that analysis will produce a complete, accurate model of the behaviour of the desired software, whereas design involves the creation of a translation which will generate the implementation from the analysis model. However, it is reasonable to question whether "a complete, accurate model of the behaviour of the desired software" may be said to constitute an analysis, rather than a design.

In the light of these major concerns with first-generation OO methods, the author and John Daniels saw an opportunity to create a new approach that tackled them, and wrote the book "Designing Object Systems" to meet this opportunity. The book sets out a complete discipline, called *Syntropy*, for modelling a system within a software development process using object-oriented ideas combined with formal methods, which made clear distinctions between different modelling viewpoints.

Syntropy was one of the two first published attempts to define a graphical object-oriented modelling language incorporating the precision of formal specification languages, the other being Fusion [55] (see below). Syntropy did this by making systematic use of *navigation expressions* [56] to integrate a type model with formal constraints. Navigation expressions appeared a few times informally in the popular book by Rumbaugh et al [52], but Syntropy was the first published

formalism to define and use them systematically. Section 3.2 of the book gives a comprehensive definition of navigation expressions, which provided the foundation for the approach that appeared later in OCL. To express logical constraints, Syntropy adopted ideas and notations from the specification language Z [57].

Syntropy avoids the terminology of analysis vs design altogether. Instead, it identifies three distinct modelling viewpoints: *essential*, *specification,* and *implementation*. The essential model is a behavioural model of the subject domain of a system, which models the subject domain as a set of state machines communicating by means of shared events. In principle it can be used to model any subject domain and makes no commitments about whether software is involved. The specification model identifies a boundary between a software system and its environment, and models that boundary again in terms of communicating state machines, this time specifying which events are inputs to the software and which are outputs. The implementation model specifies the mechanisms by which the software responds to its input events, once again using state machines, this time communicating by sending and receiving messages.

Syntropy is one of very few methods to make a clear distinction between modelling the subject domain and modelling the software itself [58]. Jackson System Development (JSD) [59] made a similar distinction: Jackson has throughout his career demonstrated refreshing and unusual clarity on points such as this.

The book is divided into 5 parts:

1. **Systems models and views**. A single chapter sets out the philosophy and overview of the approach.
2. **Modelling the world**. Chapters 2-5 explain how to model state and behaviour by means of communicating state machines using the essential model interpretation, with many examples. Chapter 2 makes the important distinction between object type and object class; an object type defines the features in common to a set of similar objects which *conform* to the type, while an object class is a description of implementation details. Chapter 4 introduces the idea of an "initial object" making the scope of a model well-defined. This idea reappears in the discussion of OCL `allInstances` in [PW8].
3. **Models of software**. Chapter 6 explains the specification model and chapter 7 the implementation model. Chapter 8 goes into considerable detail about the complexities of sub-typing of statecharts in the three different modelling viewpoints. To the best of this author's knowledge, this material remains one of the most detailed expositions of this problem area in the literature. Chapter 9 addresses concurrency, and provides a fairly

detailed description of how to manage type-conformance in situations where multiple clients may concurrently access the same object.

4. **System architecture**. Chapters 10-12 explain how to scope and encapsulate models, partition them into domains and viewpoints, and transition between the three interpretations.
5. **The development process**. The final chapter 13 briefly discusses how to manage a software development process incorporating the ideas set out in the book.

Syntropy takes a very state-centric view of modelling. Every object type is described using a combination of a statechart [60] and logic, and the behaviour of a system is to be thought of as a set of communicating state machines. This state-centric view is in considerable contrast to the functional and algebraic methods put forward in the author's earlier works [PW2] and [PW3]. Syntropy would not be a good formalism for (say) specifying the factorial function: although it would be in principle possible to define a factorial-calculator object and the various events that occurred as that object recursively used itself to calculate the answer, the result would be baroque and unhelpful. Nevertheless, Syntropy did prove to have a significant level of applicability to many real-world problems and was used in many training and consulting engagements by its authors and their colleagues between 1990 and 1994. It was also quite widely cited, and its attempts at formality were attractive to researchers looking to improve on the precision of object-oriented representations, e.g. [61].

Another method that emerged at the same time as Syntropy was Fusion [55]. Fusion aspired to integrate the most valuable aspects of first-generation OO methods into a coherent approach (hence its name), and made some use of formal methods, although nowhere near as much as Syntropy. For Fusion, operation pre- and post-conditions were logically formalized using a structured English approach, and object lifecycles were formalized using lifecycle expressions, which used a kind of Extended Backus-Naur Form (EBNF) [62] to specify possible sequences of operations for an object.

An important method based on UML that took formalism seriously was Catalysis by D'Souza and Wills [63], published in 1999 after several years of development. Catalysis had its roots in Alan Wills' PhD thesis which added formality to object-oriented programming [64], and also in Desmond D'Souza's earlier work on applying and improving OMT and Fusion [65]. Catalysis adopted UML notation and put it to service in a complete and rigorous method that offered a wealth of ideas across all stages of a software development lifecycle, from business modelling through requirements specification, component design, object design and architecture. Catalysis

is unique amongst the UML family of languages (see [PW10]) in its insistence on precision and formality across all aspects of the lifecycle and all kinds of models: static models, behaviour models and interaction models. Catalysis cites Syntropy and Fusion as major influences, and Alan Wills became a close colleague and co-author with this author for many years [PW7, PW8, PW9, PW14].

Syntropy was also influential in the development of component-based development, particularly as realized in CASE tools produced by Texas Instruments. In a personal communication to the author, Keith Short says the following:

> *"Between 1985 and 1997 Texas Instruments, in conjunction with James Martin Associates, led the field in establishing Computer Aided Software Engineering (CASE) products and methodologies. When sold to Sterling Software in 1997, TI (then owner of James Martin Associates) was the seventh largest software company in the world, with a turnover of $800M per year, and over a thousand customers worldwide representing many diverse enterprises, military groups and government departments. The core CASE approach championed by James Martin himself, and later employees of TI, was based on Information Engineering (IE), a comprehensive methodology for building computer systems that matched business objectives and system user requirements.*
>
> *…*
>
> *As Architect for the Information Engineering Facility (TI's CASE product, later renamed to Composer), and later as Research Director for TI's CASE Research Laboratory, I was at the forefront of TI's effort to combine IE with OO software development. It was during this period of research into a workable combined approach to what eventually became published as Component Based Development, that my team and I discovered the work of Steve Cook and John Daniels published in 1994 in the book "Designing Object Systems, Object Oriented Modelling with Syntropy". Cook and Daniels' work was groundbreaking, being the first fully fleshed-out approach to combining design in-the-large (software components that could match business tasks) and programming in-the-small (OO design of classes of which components are composed). Cook and Daniels' approach was also novel in using well-formed logic expressions to describe component and class behaviors, providing an opportunity for machine-testing and verification of component and class implementations. This was of great interest to us at TI, as we were working to extend the specification of business tasks, and to exploit further opportunities for program generation. In many ways, Syntropy represented the first published effort to provide the glue combining IE and OO."* [66]

The authors' aspirations to create one or more CASE tools based directly on Syntropy were never realized. Syntropy was commercially overtaken by the initiative by the Object Management Group (OMG) to create a standardized object-oriented modelling language which started in 1994, more-or-less contemporaneously with the publication of Syntropy. This initiative, documented in [PW17], led to the creation of UML, and some of the ideas from Syntropy – most notably the incorporation of logical constraints using navigation expressions – found their way into UML and OCL, as explained in the remaining publications in this group. UML was initially extremely successful and for all intents and purposes superseded all other existing object-oriented methods.

Later work by John Daniels with John Cheesman pulled together ideas from Syntropy, Catalysis, and Sterling Software's "Software Adviser" method into a component-based development method using UML notation [67].

PW5: Experience with Formal Specification of CMM and UML

(10 citations)

In 1994 this author left his consulting company Object Designers Ltd to join IBM's new European Object Technology Practice. There, Syntropy became a fundamental influence in a commercial project run in the IBM Insurance Solutions Development Center in La Hulpe, Belgium, by a team largely recruited by this author and comprising Aldo Eisma, Anna Karatza, Mark Skipper, Georges-Pierre Reich, and Jos Warmer. The goal of this project was to build a software development tool for modelling insurance policies and processes using object-oriented concepts, heavily based on Syntropy. However unlike in Syntropy there was to be no use of unfamiliar mathematical symbols; instead it used a library of operations with names rendered using only ASCII characters. These operations acted on single objects and on the collections of objects resulting from navigation expressions; the collection operations were inspired by operations defined in the Smalltalk language [30]. This project was where Object Constraint Language (OCL) was developed.

Like Syntropy, this project succumbed commercially to the OMG's initiative to create the standardized object-oriented modelling language UML. The project was never published or turned into a product or service; instead IBM decided to base its insurance modelling work on the emerging UML standard. The energies of this author were hence redirected into influencing the UML standard on behalf of IBM in order to improve the precision and flexibility of the UML definition. IBM teamed up with the company ObjecTime who were at that time vendors of a tool for the Real-Time Object-Oriented Modelling (ROOM) method [68], and a 194-page submission to the OMG was developed with this author and Bran Selic as the lead authors. This submission is for copyright reasons not in the public domain, although it can be accessed by OMG members at http://www.omg.org/cgi-bin/doc?ad/97-01-18. The document contains a complete definition of OCL, the Core Meta-Model (CMM) intended to be a basis for building general-purpose and domain-specific modelling languages, and a technique called *schemes* for customizing CMM to produce specialized languages.

The IBM/ObjectTime submission was just one of several submissions for the definition of UML. The other main submission was the so-called "three amigos" document from Rational, which represented a unification of the ideas of Booch, Rumbaugh and Jacobsen. Via a political process

described in [PW17], elements of the IBM/ObjectTime submission were incorporated into the eventual published UML 1 standard, including the entirety of OCL.

A key idea in CMM was to satisfy a requirement for flexibility. CMM was defined as a framework within which modelling languages could be built through schemes, somewhat analogous to the later UML profiles. UML in contrast was defined as a complete language and as a result makes many detailed design commitments that have subsequently proved problematic as UML has been applied in various domains. Part III of this thesis discusses this point in depth.

This workshop paper [PW5] was written by authors of the IBM/ObjectTime submission, and summarizes experiences with using OCL within the definition of UML and its predecessor CMM. It is the first mention of OCL in the published literature. Its primary author is Jos Warmer, who was assigned by IBM to integrate OCL into the UML submission once it had been decided to merge ideas from the IBM/ObjectTime submission into the final joint proposal.

An interesting online weblog post by Jean Bézivin about CMM and the incorporation of OCL into UML [69] describes this author as the "éminence grise" of the IBM contribution to UML 1 and explains how OCL was incorporated from CMM into UML while the contribution of schemes was lost.

[PW5] claims that a lack of a formal mathematical base for OCL is a "non-problem". In hindsight this is wrong. Logical issues in the OCL definition have certainly caused problems, and indeed one of the publications in this thesis [PW8] was an initiative by this author and colleagues to resolve some of these problems. There are many papers in the literature dealing with OCL issues, and ongoing work exists to this day: there has been a series of OCL workshops connected with the MODELS conference series for the past 15 years [70]. Work remains to be done: at the time of writing in 2016 there are 206 open issues recorded against the OCL definition[1].

PW6: Informal formality? The Object Constraint Language and its application in the UML metamodel

(37 citations)

This conference paper is the first reasonably comprehensive publication of OCL in the literature. It explains the language with examples, sets out the main characteristics of the language, and gives examples of the use of OCL to specify well-formedness rules in the UML metamodel. It coincided with the publication by Warmer and Kleppe of their popular book on OCL [71], to which this author contributed a foreword.

[1] http://issues.omg.org/issues/lists/ocl2-rtf?view=OPEN

The paper points out that tools are essential in order to verify the correctness of OCL expressions. At the time of writing in 1998, the only tool available was a simple parser, which found errors in more than 50% of the hand-written OCL. A later paper by Richters and Gogolla [72] using more sophisticated OCL tools to check UML 1.3 again found errors in more than 50% of the OCL expressions. In fact, it was not until the development of UML 2.5 from 2012-2015 that good enough tools were available to check the syntactic and semantic correctness of the OCL used to define the UML metamodel. Today in 2016 there is an online validator available at http://validator.omg.org/se-interop/tools/validator that will check UML models and metamodels for their validity under OCL constraints.

PW7: Defining the context of OCL expressions

(18 citations)

When UML version 1.1 was formally published by the OMG [73], the definition of OCL was included as a chapter in the UML specification document. Once OCL was formally incorporated into UML and reached a wide audience some of its deficiencies became more apparent. At that time an informal grouping of researchers emerged with a shared interest in identifying and articulating these deficiencies. The group comprised this author, Anneke Kleppe, Richard Mitchell, Bernhard Rumpe, Jos Warmer and Alan Wills. One of the deficiencies identified early by the group was a lack of precise definition for how the free variables in an OCL expression are bound in the environment defined by a UML model. This problem is also identified in section 3.11 of [PW8], below.

A proposed solution for this problem is presented in [PW7] for which this author was the lead writer. The solution proposes a system of global and local declarations that represent how elements from the enclosing UML model appear in the environment of an OCL expression. Global declarations are inferred for every package from the contents of the package and everything visible from it. Local declarations are inferred based on the placement of the OCL expression within the model. The treatment offers a use of OCL at the meta-level for specifying these declarations. It is not noted in the paper but we may note here that this is a circular definition: the meta-level definitions would rely on a context defined by the definitions themselves. In a practical implementation this would pose a bootstrapping problem.

When UML 2.0 was published in 2005, the specification for OCL was separated and became an independent specification managed and versioned in its own right, with the starting version being 2.0 in order to align with UML. Clause 12 of the formal specification for OCL 2.0 published by the

OMG in 2006 [74] and subsequently revised (currently at version 2.4) specifies mechanisms similar to those set out in [PW7].

PW8: The Amsterdam manifesto on OCL

(67 citations)

In 1998 the informal research group listed above met in Amsterdam for two days to discuss and clarify various issues in the definition of OCL. At that stage OCL was becoming pervasively used among researchers into object-oriented design and modelling: the Warmer and Kleppe book written in 1998 and published in 1999 [71] received more than 200 citations by the end of 2000. The discussions and conclusions of the Amsterdam workshop were encapsulated in this "manifesto" document which was edited by Bernhardt Rumpe and first published in 1999 as report TUM-I9925 of the Technische Universität München, and finally published in a Springer "State-of-the-Art Survey" volume edited by Tony Clark and Jos Warmer in 2002. All participants in the workshop are listed as authors of the manifesto.

The manifesto uses an example originated by Warmer and Kleppe to explain OCL in terms of business rules that would apply to a hotel. After an introduction to the language and a description of its status, the discussion is grouped into bug fixes, clarifications, extensions and applications.

An important problem identified in section 2.2 is the definition of `OclAny`. In the original OCL definition, `OclAny` was the common supertype of all types, which meant that `OclAny` was the supertype of `Set(OclAny)`, which in turn enabled the encoding of Russell's paradox [75]. This problem was initially observed in a paper by Mark Bickford and David Guaspari which was not published but is available to OMG members at http://www.omg.org/members/cgi-bin/doc?ad/98-10-01.pdf.

The essence of the problem is this: define OCL expressions R and P as follows.

```
R: OclAny.allInstances->select(x:Set(OclAny) | not x->includes(x))
P: R->includes(R)
```

According to the type rules for OCL, both expressions are well-typed; and it follows from the usual rules of set membership that `P = not P`.

One way of addressing this problem would be simply to declare that an expression such as R is undefined semantically, even though it is well-defined syntactically. This is a dubious principle for language design, so the problem was remedied in the definition of OCL 1.3 by removing `Set(OclAny)` from the subtypes of `OclAny`. However, this is a special case; the overall problem of semantic consistency has not been resolved. Consider the following:

```
context T inv: not self.oclIsKindOf(T)
```

This simple well-typed OCL expression is a contradiction semantically. Later work on OCL such as [76] has produced much more detail on this and many other issues.

Another "bug" identified in section 2.3 is the operation `allInstances()`, which is supposed to deliver the set of instances of a given type within some scope, although the scope is not well-defined: indeed, within the UML specification the scope of a model is explicitly undefined. Section 6.3 of UML 2.5 states "The concept of an execution scope is not further defined within UML semantics, because exactly to what it corresponds varies depending on the domain of discourse" [77]. The manifesto declares that "`allInstances()`" is dangerous, and recommends instead a style of modelling related to the "initial object" introduced in Syntropy [PW4].

The manifesto attempts to clarify the meaning of undefined, proposing a three-valued logic (Kleene logic) for logical expressions, which has subsequently been incorporated into the formal OCL specification. It also clarifies recursion, with a discussion about how recursive equations may in general have many solutions, and how it is normal for the semantics of programs to take the least fixed point solution if one exists. This discussion has not significantly impacted the OCL specification, which doesn't acknowledge this issue; OCL users normally give OCL an operational semantics, essentially treating any recursive constraint as a functional program.

The manifesto suggests a number of extensions to the language, some of which were introduced into OCL as a response to various versions of the manifesto: `oclIsNew`, `isUnique`, *let* expressions, and some of which were not. Finally, the manifesto discusses the application of OCL to defining behaviour in various styles, including the Syntropy style, and also the definition of an OCL metamodel and how it might be used to customize UML.

Conclusion

Syntropy was the first significant published attempt to combine a graphical object-oriented design language with a formal specification language. It showed how to add logical statements to class diagrams and statecharts using navigation expressions and collection operations, and how to interpret the resulting entities as models of the world and as specifications of software. As UML was first standardized, Syntropy ideas were incorporated, most notably in the creation of OCL, its use in the UML definition, and its subsequent criticism and evolution.

There is a considerable gap between conclusions about UML and OCL drawn within the academic literature and what actually happens in the evolution of the standardized technologies. Improvements to the standardized technologies take place slowly and are driven by processes

(see [PW17]) that take little account of what is published in the academic literature unless researchers actively engage with the OMG's processes. For example, some of the suggestions in the Amsterdam manifesto [PW8] found their way into the OCL specification by means of the OMG's issue resolution process. This was largely because a key participant in the workshop (Warmer) was also an influential participant in the relevant OMG Revision Task Force at the appropriate time.

III. Language flexibility

This group of papers addresses the flexibility of graphical modelling languages by proposing techniques for language customization, both within the context of UML and separately from it with Domain Specific Languages (DSLs).

As observed earlier [PW5], CMM was designed with flexibility as a key requirement, and with the concept of modelling *scheme* central to its architecture, where a scheme provided a way to assemble modelling language components into a language to address a particular subject domain. The UML 1.0 initiative did not adopt this requirement, and instead produced a single language which was in effect a political compromise between all of the parties to the UML submission. In 1999, the OMG issued a Request for Information [78] asking vendors and users of UML whether a major revision of UML was required. 23 responses were received [PW17] and analysed; the consequence was the issuance in 2000 of two Requests for Proposal, one for UML 2.0 Infrastructure [79] and another for UML 2.0 Superstructure [80].

The UML 2.0 Infrastructure RFP contains the following requirement about extensibility:

> *"Many of the users and vendors who are defining profiles are encountering problems with the semantic restrictions imposed by the lightweight extension mechanisms. Consequently, there is widespread support for including a first-class extensibility mechanism in the next major revision. This will allow users and vendors to define their own metaclasses, providing further support for specifying a "family of languages" based on UML.*
>
> *Further, certain metamodel elements (e.g., generalization) have been given semantics that are implementation-dependent on the C++ programming language. Consequentially, it is problematic to specialize the semantics of these model elements in a profile to meet the needs of other domains."*

Unfortunately, this extensibility requirement was not at all met by the UML 2.0 specification that was finally published in 2005. This turned out to be an architectural mess which was only somewhat resolved ten years later by UML 2.5, published in 2015 after a lengthy simplification process [81]. Many examples of this inflexibility are given in the group of papers in this section of the thesis; a simple example is the definition of *visibility*, which in UML 2.0 is implementation-dependent on the Java programming language and is problematic to apply to other domains. Papers [PW9] and [PW10] were written while the requirements for UML 2.0 were being defined; the influence of these papers on that process may be seen in the use of the phrase "family of languages" in the quotation above.

The appearance of UML 2.0 which failed to meet its extensibility or simplicity objectives was an important incentive for the separate emergence within the software industry of Domain Specific

Languages (DSLs): custom languages designed for creating models within a particular domain. This author was a significant contributor to this emerging trend, as shown in [PW11] to [PW15] in this group of publications.

PW9: Defining UML family members using prefaces

(61 citations)

This conference paper was written by the same group of researchers described in [PW7] above and was published in 1999, two years after the release of UML 1.1. The detailed history of publication of UML is described in [PW17]. In 1999 UML was the topic of many popular books as well as many academic publications, as the software development industry was beginning to get to grips with UML in order to decide what value it would have in practical software development processes. Practitioners were beginning to recognize that UML is in practice a family of languages, where each family member has different rules for the syntax and semantics. In recognition of this, the OMG was itself issuing Requests for Proposal for UML *profiles*, where each RFP contained a working definition of what was meant by profile. The origins of the word *profile* are described in Jean Bézivin's online weblog [82] which explains the background of these RFPs in work by Ed Seidewitz and Dave Frankel.

The intent of [PW9] is to go further than the OMG's working definition of profile, and provide a comprehensive description, with some examples, of the kinds of customization that UML users would want in practice. The term *preface* is used to avoid confusion, although ultimately the intention of profiles and prefaces would be very similar. Prefaces are reminiscent of the concept of schemes from CMM, but much more elaborated and detailed. Section 2 of the paper intentionally defines UML as a *family* of languages rather than a single language, and motivates this description by reference to profile development within the OMG, publications from authors including the "three amigos" and other researchers, and the practicalities of code generation in various different environments.

The paper explains some of the customization that a preface might accomplish using an example of how a class diagram might relate to a statechart diagram. The paper claims that UML does not insist on a particular interpretation of this relationship. In hindsight this claim may be misleading: UML 1.3 does in fact associate a Transition with an Event (although optionally), where an Event may include CallEvents, ChangeEvents, SignalEvents and TimeEvents, and where CallEvents are directly associated with Operations. Thus although UML may not *insist* upon a particular relationship between statecharts and classes it is certainly highly suggestive of a preferred relationship, and this presumption is a significant barrier to flexibility. Indeed, UML 2 later

explicitly introduced a second interpretation called “protocol state machines”; in the view of this author this was exactly the wrong direction of travel, the preferred approach being to remove any such presumption from the basic language and to defer details of relationships between state machines and classes to a preface or some other customization mechanism. This example hits at the heart of the flexibility problems with UML which eventually led to the emergence of domain-specific language tools.

Relationships between concepts are just one kind of information that might be contained in a preface. Section 4 of the paper lists many others, including customized meanings of specific notations, additional constraints, framing rules, programming-language specific support, communication paradigms and persistence mechanisms. Section 5 proposes that prefaces may be structured using packages, and that customizations may occur at different levels in the value chain: standards, toolsmiths, companies and individual projects. The paper also discusses how semantics of prefaces may be described, how overlaps and inconsistencies may be handled, and how code generation may be combined with prefaces to produce executable systems.

This paper unfortunately does not evaluate or analyse UML in terms of its flexibility. A naïve reader might assume that UML as defined, coupled with an implementation of prefaces as described in the paper, would offer a flexible route towards customized domain-specific modelling languages. Such is not the case, as is shown in the next item.

PW10: The UML family: Profiles, prefaces and packages

(39 citations)

This invited conference paper covers somewhat similar ground to [PW9] but focuses much more on the state of the UML definition itself, and points out ways in which the UML definition does not lend itself as a basis for variation. The paper was published when UML 2 was being envisioned, and was ancillary to this author’s ultimately unsuccessful campaign to formulate UML 2 as a family of variants, rather than a single language. It makes a critique of the UML specification document, noting that material called “semantics” is informal and erratic, that the language has no consistent level of abstraction, and that the metamodel contains unnecessary redundancy but is incomplete.

The paper overviews existing and proposed approaches for UML variation: profiles, metamodel extensions, and prefaces (citing and quoting [PW9]). It endorses and summarizes a feasibility study commissioned and funded by this author (representing IBM) and carried out by the Precise UML Group [21]. The purpose of the feasibility study was to propose a new architecture for UML 2 based on the following three requirements (quoted verbatim from [21]):

1. *It should be precise to the degree that conformance can be checked systematically, without argument, preferably automatically, and that self-consistency of the definition can be established.*
2. *It should be comprehensive, covering syntax, both concrete and abstract, and semantics. On the other hand, redundant and overlapping concepts should be kept to a minimum.*
3. *It should accept that UML is a family of languages, providing mechanisms that allow profiles and language extensions to be defined in a controlled and managed way, and which makes the relationships of profiles and extensions to existing language fragments explicit and unambiguous.*

These requirements are as compelling today as they were then. Unfortunately the working group constituted to formulate UML 2 never accepted the idea of UML as a family of languages, mostly ignored the work in [21] and other important contributions (see [PW17]), and in consequence in the mind of this author and many others missed a vital opportunity.

PW11: Domain-specific modeling and model driven architecture
(121 citations)

In 2003 this author changed employer, moving from IBM to Microsoft. Part of the reason for this move was dissatisfaction and frustration with the end result of the UML 2 development process, to which this author was a key contributor and representative of IBM's interests. The essence of this dissatisfaction, as mentioned above and discussed further in [PW17], was that UML 2 completely missed the opportunity to put flexibility at the heart of its architecture.

Working with new colleagues at Microsoft provided this author with an opportunity to push forward model-driven development approaches without a significant corporate commitment to UML. This magazine article, published online in 2004, is a position paper setting out a strategy for the role of models in software development based on domain-specific languages (DSLs), coupled with frameworks, patterns, tools and development practices integrated into software product lines. This approach was to be promoted by Microsoft under the slogan of Software Factories, and described in detail in the book from which the chapters [PW12] are drawn. It was at this point in the history of model-driven engineering that industry interest in model-driven development was shifting from generic approaches, such as UML and MDA, to domain-specific approaches based on product-line engineering by analogy with mass-production, continuous improvement and mass-customization approaches in more mature industries. This article caught the spirit of that shift: it triggered significant controversy [83], but is also rather widely cited including in many academic research publications, e.g. [84], [85], [86]. Many of these citations refer to it as a source of definition for domain-specific modelling; for example Lopes et al [84] quote [PW11] thus:

> "*DSLs are defined by Steve Cook as languages that instead of being focused on a particular technological problem such as programming, data interchange or configuration, are designed so that they can more directly represent the problem domain which is being addressed*".

The article points out that there are several important components of a complete software development approach: models, frameworks, patterns, code, tools and processes. It criticizes MDA primarily on the grounds that the MDA distinction between platform-independent models and platform-specific models is not often an important distinction in a software project, and leads to problems of performance and platform integration. The article criticizes UML on the grounds of its lack of flexibility, and in particular how the mapping between UML and .Net (an important Microsoft application framework) is problematical, leading Microsoft to adopt UML conventions but to eschew "legal" UML.

Having introduced the concept of domain-specific languages (DSLs) and the consequential need for languages and tools to implement them, the article goes on to find fault with MOF as a language for defining modelling languages, on the grounds of the narrow scope of MOF and the problems with XMI (XML Metadata Interchange) as an interoperability format. The article points out that generating XML schemas from metamodels, as is done with MOF and XMI, causes combinatorial versioning problems as the metamodel, MOF and XMI are all subject to revision. Instead, each domain-specific modelling language should define its own purpose-built domain-specific XML schema following widespread industry practice.

PW12: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools

(1319 citations)

2004 saw the publication of this book setting out a vision for the industrialization of software development through the use of patterns, models, frameworks and tools. The book sold widely, was translated into several languages, and won a Jolt Productivity award in 2005. The lead authors were Jack Greenfield and Keith Short. This author and Stuart Kent were invited to contribute two chapters focused on the technicalities of modelling language definition and implementation, and an appendix offering a detailed critique of UML and MDA.

Chapter 8: Language Anatomy concerns how domain-specific modelling languages are defined. It explains the various components of a language definition: abstract syntax, semantics, concrete syntax and serialization syntax. It uses a simple language called Our Simple Language (OSL) as an example to illustrate the various aspects of language definition.

Abstract syntax is defined using context-free grammar (CFG) and metamodel and the two approaches are compared and contrasted. The discussion points out that when languages combine graphical and textual elements, a combination of metamodels and CFGs is required. OCL is used to specify well-formedness rules, and a system of annotations designed by this author based on an extended OCL is used to define concrete syntax. The Serialization Syntax section explains XML-based serialization and contrasts purpose-built schemas with generated schemas, pointing out that generated schemas have the consequence that metamodel changes will invalidate existing serialized documents and require migration tools, whereas with purpose-built schemas a metamodel change only requires a change to the mapping between metamodel and schema.

The final Semantics section of chapter 8 was written by Kent and discusses translational semantics and trace-based semantics.

Chapter 9: Families of Languages builds on the material in Chapter 8 and describes tools for creating DSLs and their components, and how these components may be assembled and configured to create families of languages. This chapter introduces a DSL for creating DSLs, and points out that this can be used to define and bootstrap itself. It introduces language components as units of language reuse, with templates and "glue models" to hold them together. It then introduces a Tool Factory Architecture for the creation of custom tools for domain specific languages (including itself). This is essentially the beginnings of the project, led by this author, which designed and implemented the actual DSL Tools product (see PW14) (although regrettably some components described in this chapter – the Pattern Engine, Animator/Interpreter and Model By Example components - never saw the light of day).

The ideas presented in Chapter 9 correspond to a style of development which was labelled "Language-Oriented Programming" and "Language Workbenches" by Martin Fowler in an influential on-line article http://www.martinfowler.com/articles/languageWorkbench.html and also his 2010 book on Domain Specific Languages [87]. Fowler cites the work described here as an important example of this style of development.

Appendix B of the book offers a fairly detailed critique of the UML and MDA initiatives. It covers similar ground to PW11 but in more technical detail, with historical material similar to that covered in PW17.

PW13: Separating concerns with domain specific languages
(1 citation)

This brief article is the written counterpart of an invited keynote speech. It summarizes the case for domain-specific languages, motivating it by the inherent heterogeneity of contemporary software development tasks.

PW14: Domain-specific Development with Visual Studio DSL Tools
(279 citations)

Publication PW14 is an extract from a book written to accompany a product – "DSL Tools" - developed by Microsoft Corporation as a plugin for Microsoft Visual Studio. This author was the lead architect and designer of DSL Tools. The book was written by the core members of the development team and published in the Microsoft .Net Development series. The writing style is informal and tutorial, according to general guidelines for this kind of book.

Chapter 1 entitled "Domain-Specific Development", written by this author, is included in this thesis. The chapter sets out the philosophy and overall design of the DSL Tools product; the remainder of the book describes the product in detail. The chapter starts by describing how a custom language can be defined to solve a class of problems. The solution to a class of problems has a fixed part and a variable part, where the fixed part represents the commonalities amongst the class, and the variable part determines the differences between solutions. In domain-specific development the variable part is represented by the domain-specific model and plugged into the fixed part to constitute the complete solution.

Two real commercial examples are included from early customers of the product:

- Himalia[2] build user interfaces based on Windows Presentation Foundation by integrating the output from three models: a Navigation model, a Use Case model and a User Profile model.
- Ordina[3] in their Microsoft Development Center developed a software factory which uses a Web Scenario model, a Data Contract model, a Service model and a Business Class model.

Several other examples are proposed but not explained in detail: software defined circuitry; embedded systems constructed from state machines, device interfaces, and software process

[2] https://visualstudiogallery.msdn.microsoft.com/8A37F5B7-1AF8-4699-BD6A-2BB3317E5825
[3] https://www.ordina.com/

customization. Illustrated by these examples, the benefits of domain-specific development are enumerated and the balance of benefits against costs discussed.

The chapter continues on the theme of domain specific languages. Three approaches to defining textual DSLs are proposed: BNF definitions and a parser-generator, embedded DSLs, and XML schemas. Graphical DSLs are then introduced and a set of graphical conventions, derived from UML, are proposed. Each of the aspects of a DSL is described: notation, domain model, model validation, generation, serialization and integration. The DSL tools product integrates these aspects into a single user experience for creating customized graphical DSLs.

A closely-related product, the "Application Designer" which is a DSL for modelling endpoints in distributed systems, provides the example for a detailed walkthrough of how a simple model of stock price calculation can be used to generate a complete skeleton implementation of a web service to implement the model, using .Net artifacts. Maintenance of this implementation is illustrated by the ability to refactor a name over multiple technologies.

The *customization pit* is a refinement of the basic idea of domain-specific development. The customization pit represents a customized solution consisting only of a fixed part and a variable part, where the fixed part represents the commonalities amongst a class of problems and the variable part representing the differences. Instead of a hard boundary between these parts, developers would prefer a *customization staircase*, where solutions within the direct target boundary are simple, but solutions outside that boundary increase gradually in difficulty with distance. Various techniques to implement the staircase are proposed, including the use of multiple DSLs, the generation of code explicitly designed to be extended, and modification of the code generator itself.

The chapter concludes with an overview of UML which deconstructs it by presenting it as a set of conventions which can be used to build DSLs. An important advantage of doing this is the creation of a model API that represents the target domain, rather than representing the UML metamodel: this considerably simplifies the work of a developer using this API.

PW15: The Domain-Specific IDE

(9 citations)

This paper makes a case for domain-specific tools and processes as well as domain-specific languages. It criticizes UML for lack of flexibility (an omnipresent theme in this group of publications) and discusses software tools that integrate domain-specific languages, commands, user-interfaces and processes. The term domain-specific IDE is preferred to the term Software Factory, the latter term being rather overloaded. Barriers to agility are discussed, including the

ability of domain-specific models to avoid premature commitment to design decisions. Language evolution towards multi-paradigm programming languages is briefly discussed. Code generation, validation, debugging and testing are also briefly discussed, and the paper explains the idea of modelling all of that in a meta-tool environment.

Conclusion

The recognition that flexibility is a crucial aspect of the design of an object-oriented specification language originated with the CMM concept of *schemes* [PW5]. Flexibility goes hand-in-hand with precision: an imprecise language is inherently flexible in the weak sense that its meaning is ambiguous, but once the meaning of a software design language is precise then mechanisms for explicitly introducing flexibility become essential.

The first two of the papers in this group were published during the period when UML 2 was being envisaged, and were intended to influence that process so that UML 2 would explicitly incorporate mechanisms for supporting flexibility, such as *prefaces*. Unfortunately, this did not happen and in fact UML 2 was certainly no more flexible than UML 1 and probably less so, the limited concept of *profile* remaining the only way to customize the language while the language became more complicated. Perhaps in consequence of this limitation, Domain Specific Languages emerged, often loosely based on UML considered as a set of conventions.

Microsoft's DSL Tools product, for which this author was the lead architect, was a widely-adopted commercial implementation of Domain Specific Languages, available as a free download to customers of Microsoft Visual Studio. DSL Tools showed that such a language consists of more than just a meta-model, and that domain specificity applies to more than just the language, reaching out into all of the tools and processes used to develop software in a particular domain. Its design also provided features for resolving the "customization pit" problem.

IV. Summary

This final group contains two papers, both invited, that offer a retrospective on the development of the technologies discussed in this thesis together with some pointers towards future development.

PW16: Object technology--a grand narrative?

(4 citations)

This was an invited contribution to a retrospective session in the ECOOP 2006 conference. It goes through the history of object technology, and is critical about many of the historical claims made for object technology which have turned out to be naïve or mistaken. It sets out some of the contemporary challenges for programming: language design, library design, service orientation, data access and modelling. It finishes by rejecting the idea that there could be any one paradigm or philosophy that will solve all problems of software development: instead there will be a continuing proliferation of new representations and a need to fit them together.

PW17: Looking back at UML

(7 citations)

This "Expert's Voice" paper describes the development of UML from the beginning up to the time of writing in 2012. It covers precursors - OO and diagrammatic languages – and comprehensively describes the process of standardization and standard revision.

An important theme is the "meta-muddle", which is illustrated most vividly in the first four versions of UML 2.x. One example of the muddle is the split of the UML definition into superstructure and infrastructure, where the superstructure contains a modified copy of the infrastructure while the infrastructure contains the same set of concepts defined in slightly different ways. Another is the use of Package Merge in the UML specification which produces a document in which the metaclass Classifier is defined in 7 places in UML 2.4 Superstructure. Another is the fact that primitive types such as Integer are defined redundantly in several different places.

A second theme addresses UML profiles and their inconsistencies and deficiencies, including ambiguity of definition and lack of integration with OCL.

Section 7 discusses "What UML is for". It presents Martin Fowler's UmlMode characterization [88] into UmlAsSketch, UmlAsBluePrint and UmlAsProgrammingLanguage. Executable UML is a long-standing standardization initiative in the UmlAsProgrammingLanguage category which started with the Action Semantics addendum to UML 1.4 published in 2001 and continues today with

initiatives including FUML [89] and related initiatives. There is a specialized market for UmlAsProgrammingLanguage and it is supported by various open-source and commercial tools. UmlAsSketch, where UML provides conventions for drawing diagrams for communicating during software development is uncontentious, widespread and popular, and directly supported by diagramming tools that make no pretence to be software development environments e.g. http://www.softwarestencils.com/uml/. UmlAsBluePrint is the most controversial category, because this is the principle behind Model Driven Architecture, where UML's lack of flexibility causes most problems and where DSLs have emerged to compete and co-operate with UML.

An important issue for UML is interoperability and initiatives to improve it. The systems modelling language SysML is a derivative of UML [90]. SysML has made considerable headway in government and defence industries and such industries tend to mandate tool interoperability as a condition of purchase. This has triggered a renewed focus at the OMG on UML interoperability with the creation of a Model Interchange Working Group that has been studying and resolving interoperability issues with UML and SysML. One of their main conclusions was that UML interoperability is inhibited by ambiguity and imprecision in the UML specification [91].

This author was the instigator of a process under the aegis of the OMG aimed at improving the state of UML. This process started in 2008 with the issuance of a Request for Information (RFI) about the future development of UML [92]. The feedback from this RFI led to a strategy for the simplification of UML and improvement of its precision; this author was co-chair of the task forces that delivered UML 2.4 and 2.5 in line with this strategy. As a consequence UML 2.5 finally managed to deliver a version of UML from which the redundant definitions had been removed and in which almost all of the constraints were expressed in syntactically correct OCL [77]. This was a major undertaking: the report from the 2.5 task force amounted to 1115 pages – significantly larger than the UML specification itself.

Although UML 2.5 is a great improvement over its predecessors in terms of simplified and precise specification, it does little or nothing to address the problem of flexibility. That problem remains postponed to the future. For UmlAsSketch, nothing needs to be done: UML's diagrammatic conventions are well established and well known. For UmlAsProgrammingLanguage, work continues in defining executable subsets of UML and giving them operational semantics. For UmlAsBlueprint – the use of UML models as sources for model-driven engineering – the flexibility problem and the dilemma between UML and DSLs remains. The solution would be some form of unbundling into a family of interrelated configurable modelling languages or theories; whether

this kind of progress would be economically feasible within the OMG's policies and procedures is unknown.

Conclusion

Object-orientation, often seen during the 1980s as an all-embracing paradigm for software development, has become just one of the essentials in the toolbox of the software engineer.

Currently much of the energy devoted to UML improvement at the OMG is going into the definition of precise executable subsets of UML: FUML™ [89], Precise Semantics of UML Composite Structures [93], and a Request for Proposals for Precise Semantics of UML State Machines [94]. The end result of this work will be a standard subset of UML which is effectively a graphical object-oriented programming language: precise indeed, but with minimal flexibility. This precise executable UML will most likely fulfil a specialized role in the programming of embedded systems.

The absence of flexibility in today's UML implies that it is unlikely to continue to play an effective role as a flexible specification – "blueprinting" – language for systems implemented using modern languages for programming and data representation. Modern programming languages such as F# and Scala are multi-paradigm [95], and it seems appropriate that design and specification languages should be the same. Attempts to create UML profiles for different purposes including SysML [90], NIEM-UML [96] and others, have illustrated how UML places unnecessary restrictions which inhibit the expressive power and usability of its profiles. One simple example among many is the inability of UML to model a stand-alone property, without it being a property of something: this makes mappings of UML to modern data languages including XSD and RDF cumbersome and problematic.

V. Summary of Contributions

The overall goal of this work was not only to create original technical contributions to the landscape of modelling languages, but also to disseminate those contributions in the software engineering field, both through publications as exhibited here as well as through managerial and political activities aimed at the development, promotion and use of products and standards. This section summarizes the overall original contributions of this work under these headings.

Technical contributions

The core technical contributions for this work are the ideas set out in section 3.2 of Syntropy [PW4] for integrating formal predicates into graphical object-oriented design languages: the concept of a namespace associated with a class, with navigation expressions defined in a namespace, the name "self" referring to a contextual instance, and logical predicates written against those navigation expressions. Semantics of these expressions are given in terms of finite collections (sets, bags, and sequences) of objects and values. These ideas were adopted by OCL, altered to avoid the use of non-ASCII mathematical symbols, and as OCL was incorporated into UML the ideas became omnipresent in work on the semantics of UML and its derivatives.

Significant technical contributions from Syntropy also include the identification of modelling perspectives, cited in various publications including Michael Jackson's work on Problem Frames [8], and a detailed consideration of how subtyping applies to statecharts.

In the world of UML, this author created original contributions in how to add flexibility to the UML specification, through the identification of possible points of variability in that specification, and proposing mechanisms for instantiating those variabilities. This work followed through into the detailed definition and implementation of mechanisms for defining domain-specific languages as explained in section III.

Managerial contributions

Syntropy was created using funding and resources from a boutique consulting company, Object Designers Ltd, founded by this author together with John Daniels. Much of the content was used by the staff of the company delivering the material in the form of training courses and consulting engagements.

This author was headhunted by IBM in 1994 to become the technical leader of their emerging European Object Technology Practice, where Syntropy became a pivotal influence on the development of an internal tool for modelling insurance policies and processes, as described earlier in the summary for [PW5], for which this author provided technical leadership. This project

created OCL. On the appearance of UML 0.8 in 1995, IBM decided to base the insurance work on the emerging standard, anticipating being able to capitalize on a variety of UML tools. This author then became IBM's manager and lead representative for the UML standardization process, writing a complete proposal together with Bran Selic, with OCL forming part of the proposal, and then following through the development of later versions of UML up to and including UML 2.0.

Once incorporated into UML, OCL became widely used in academia. A search in "Google Scholar" for "OCL Object Constraint Language" yields 22600 results; for comparison, a similar search for "UML Unified Modeling Language" yields 121000 results. The "OCL Portal" maintained by the Technical University of Dresden[4] lists 18 OCL tools, dated from 2005 to 2012, including simple parsers, a commercial business rules tool, and an interactive proof checker.

Headhunted again in 2003, this time by Microsoft, this author created and led a team of six based in Cambridge, UK, with the remit to develop a tool for implementing Domain Specific Languages. The "DSL Tools" were developed and made generally available as part of the Visual Studio SDK and remain available to this date[5]. The book [PW14] explains the product in detail.

In 2008 Microsoft decided to re-embrace UML and to release a UML tool suite as part of the Visual Studio suite; the "DSL tools" provided the platform for the creation of this UML tool. This author became the architect of this tool suite, as well as becoming Microsoft's representative to the Object Management Group in efforts to improve the UML definition. In this capacity the author was the chair of the task forces that delivered UML 2.4 and UML 2.5, as well as serving on the OMG Architecture Board and Board of Directors.

Political contributions

"Political" here refers to activities performed in the interests of gaining strategic power in the process of translating technical ideas from the laboratory into widely-available products and standards. In relation to this thesis, these activities primarily involved engaging with the Object Management Group (OMG) in various capacities as summarised below, and explained more in [PW17].

The emergence of UML 0.8 provided the first political opportunity. To influence the standard, it was essential to engage in the process. This author convinced IBM executives to sponsor the writing of a submission, and to give permission for the use of OCL in it. The proposal was presented at various OMG meetings, and a consortium of submitting companies formed. This

[4] http://st.inf.tu-dresden.de/oclportal/
[5] https://msdn.microsoft.com/en-us/library/bb126259.aspx

author presented at many OMG meetings, and was instrumental in ensuring that OCL was an integral part of the final UML 1.0 compromise submission that was finally accepted. Subsequently, this author represented IBM at the OMG throughout the process that led to UML 2.0, and was a primary author of the UML 2.0 standard.

One of the author's objectives for UML 2.0 was to increase the language's flexibility and make it better-defined. To assist with this, the author persuaded IBM executives to fund the Precise UML Group to make a feasibility study [21]. However, this effort had a disappointingly small impact on the UML 2.0 that was finally published, because vested interests were too powerful to be overcome by technical quality judgements.

As noted above, Microsoft re-embraced UML in 2008. There were, however, misgivings about the quality of the standard. This author was assigned to represent Microsoft at the OMG in order to drive through improvements, and initiated a Request for Information process that led to the requirements for UML 2.5. As noted in the summary of [PW17], UML 2.5 was a great improvement on its predecessors in terms of clarity and simplicity of specification, but did little or nothing to address the problem of flexibility.

Support for UML 2.5 is claimed at the time of writing by most available UML tools, including No Magic's MagicDraw, Sparx Systems Enterprise Architect, and Eclipse Model Development Tools (MDT). Many other tools including Eclipse UML Designer, Papyrus, Rational Software Architect, and Altova's UModel are directly based on Eclipse MDT and either support UML 2.5 today or are likely to intercept it in due course.

So overall, this political engagement had successes and failures. The major successes were the incorporation of OCL into UML in the first place, and the major simplification represented by UML 2.5. The main failure was in implanting flexibility as a core feature of the UML architecture, despite many efforts to do so. It is very difficult to implant flexibility after the fact; the problem was inherent in the original definition of UML, and despite many efforts, it remains.

VI. Overall Conclusion

The use of graphical models in software development is widespread. UML is the best known graphical modelling language in terms of the number of publications about it, but there are many others, including Entity-Relationship modelling [97], Business Process Modelling Notation (BPMN) [98], EXPRESS information modelling [99], Fundamental Modelling Concepts [100], Integrated Computer-Aided Manufacturing (ICAM) Definition (IDEF) [101], Object Role Modelling [102], Specification and Description Language (SDL) [103], Petri nets [104], and flowcharts [105]. Many of these are precursors of UML, some are derivatives of it, and some are its competitors; several of these languages are primarily used in particular domains, for example SDL in telecommunications and IDEF in integrated manufacturing. This thesis argues that precision and flexibility are both highly desirable characteristics of any such language, and the body of work described here has contributed materially to promoting this argument in academia and industry, particularly focused on UML and also on the development of domain-specific languages.

Since its first appearance in 1997, UML has evolved through two major revisions, the second of which first added a great deal of unnecessary complexity and then attempted to simplify it again. The introduction of OCL into the UML world, based on work by this author and colleagues, made a large contribution to the precision of the UML specification. This author also presented arguments for flexibility using the slogan "family of languages", but these arguments did not manage to sufficiently influence the evolution of UML itself.

The software industry's experience with UML has not been uniformly positive. Alex Bell's ACM Queue paper "Death by UML Fever" [106] identifies 15 ways in which UML can be seriously and misguidedly misapplied in an organization with often devastating results on software development programs.

Marian Petre's 2013 paper [107] presents a study of interviews of 50 professional software engineers in 50 companies and identifies the following patterns of UML use:

Category of UML Use	**Instances of Declared Current Use**
no UML	35
retrofit	1
automated code generation	3
selective	11
wholehearted	0

If these results are taken as representative, we see that the majority of practitioners (70%) do not use UML. Of those that do, 73% use it "selectively", as a tool for thinking or communication; these

users typically use only some of the UML diagrams, and tend to adapt them for the task at hand. Nobody uses UML wholeheartedly (i.e. with organization-wide commitment to UML) at the time of the study, although some of the interviewees had experienced past attempts to do that which had not generally succeeded. Much of the feedback on UML describes it as too ideological and too complicated.

Hutchinson et al's 2011 paper [108] studies only MDE practitioners and considers how they use various modelling technologies. Of these practitioners, almost 85% use UML and almost 40% use a DSL. Use of UML is again found to be selective, often in conjunction with non-UML languages. Conclusions about the productivity, costs and benefits of MDE are mixed and complex, although there are plenty of reports of increased productivity, quality and consistency.

As these papers indicate, UML is looking increasingly obsolete amongst today's heterogeneous world of multi-paradigm programming languages and data representations. As UML ages, the need for a precisely defined and flexible multi-paradigm alternative that combines the benefits of UML and DSLs seems increasingly persuasive. There exist many ad-hoc attempts to combine UML and DSLs, for example a plugin for the tool ArgoUML that adds security-related features to UML [109], the open-source Papyrus UML tool that provides both UML and domain-specific modelling capabilities [110], and domain-specific UML profiles such as the UML Profile for BPMN Processes [111]. None of these attempts, however, provide any principled language architecture that systematically identifies or represents similarities and differences between UML elements and DSL elements; they all rely on one-off mappings between the UML and DSL worlds.

Further work is needed to define what any such multi-paradigm alternative might be or how it might come to be made commercially available. It should retain the value of the conventions that UML has made widely-understood while avoiding the pitfalls of inflexibility and complication. It is hoped that the work presented in this thesis would provide useful background to such an endeavour.

VII. References for Context Statement

[1] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2006.

[2] S. J. Mellor, A. N. Clark, and T. Futagami, "Model-driven development - Guest editor's introduction," *Software, IEEE*, vol. 20, no. 5, pp. 14–18, Sep. 2003.

[3] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer (Long. Beach. Calif).*, vol. 39, no. February, pp. 25–31, 2006.

[4] D. S. Frankel, *Model driven architecture: applying MDA to enterprise computing*. Wiley publishing, Inc. USA, 2003.

[5] J. Bézivin and P.-A. Muller, *«UML»98: Beyond the Notation*. Mulhouse: Springer-Verlag, 1998.

[6] A. Hartman and D. Kreische, *Model Driven Architecture – Foundations and Applications: First European Conference, ECMDA-FA 2005, Nuremberg, Germany, November 7-10, 2005. Proceedings*, vol. 3748. Springer-Verlag, 2005.

[7] "Editorial: First Issue of the International Journal on Software and Systems Modeling," *Softw. Syst. Model.*, vol. 1, no. 1, pp. 1–4, 2002.

[8] M. Jackson, *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[9] R. L. Ackoff, "Scientific method: Optimizing applied research decisions," 1962.

[10] G. Birch and S. Cook, "Discrete event simulation in Smalltalk/V Windows," in *IEE Colloquium on Object-Oriented Simulation and Control*, 1991.

[11] S. Cook, "Playing cards on the PERQ: An algorithm for overlapping rectangles," *Softw. Pract. Exp.*, vol. 13, no. 11, pp. 1043–1053, 1983.

[12] S. J. Cook and S. Abramsky, "Pascal-m in Office Information Systems," *Proc. Second Int. Work. Off. Inf. Syst.*, 1981.

[13] S. Cook, G. Birch, A. Murphy, and J. Woolsey, "Modelling groupware in the electronic office," *Int. J. Man. Mach. Stud.*, vol. 34, no. 3, pp. 369–393, 1991.

[14] S. Cook, "Object-oriented programming in the UK," *Comput. Bull.*, vol. 3, no. 4, pp. 22–23, 1987.

[15] S. Cook, "Introducing object-oriented systems," in *IEE Colloquium on Applications of Object-Oriented Programming*, 1989.

[16] S. Cook, "Object-oriented languages compared," in *Proceedings of the fourth international conference on Technology of object-oriented languages and systems*, 1991, p. 397.

[17] S. Cook, "The World isn't Software," *J. Object-Oriented Program.*, vol. 5, no. 9, 1994.

[18] S. Cook and J. Daniels, "Essential techniques for object-oriented design," in *Object development methods*, 1994, pp. 45–66.

[19] E. Blake and S. Cook, "On including part hierarchies in object-oriented languages, with an implementation in Smalltalk," in *ECOOP'87 European Conference on Object-Oriented Programming*, 1987, pp. 41–50.

[20] I. Jacobson and S. Cook, “The Road Ahead for UML,” *Dr Dobbs J.*, vol. May 12, 2010.

[21] T. Clark, A. Evans, S. Kent, S. Brodsky, and S. Cook, “A feasibility study in rearchitecting UML as a family of languages using a precise OO meta-modeling approach,” pUML Group and IBM, available from www.puml.org, 2000.

[22] O. J. Dahl, E. Dikstra, and C. A. R. Hoare, *Structured Programming*. London, UK: Academic Press Ltd., 1972.

[23] P. Wegner, “Dimensions of Object-based Language Design,” *SIGPLAN Not.*, vol. 22, no. 12, pp. 168–182, Dec. 1987.

[24] D. Crockford, *JavaScript: The Good Parts*. O’Reilly Media, Inc., 2008.

[25] A. H. Borning, “Classes Versus Prototypes in Object-oriented Languages,” in *Proceedings of 1986 ACM Fall Joint Computer Conference*, 1986, pp. 36–40.

[26] J. McCarthy, “Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I,” *Commun. ACM*, vol. 3, no. 4, pp. 184–195, Apr. 1960.

[27] “Lambda Expressions in Java 8.” [Online]. Available: http://www.drdobbs.com/jvm/lambda-expressions-in-java-8/240166764.

[28] “Lambda Expressions (C# Programming Guide).” [Online]. Available: https://msdn.microsoft.com/en-gb/library/bb397687.aspx.

[29] O.-J. Dahl and K. Nygaard, “SIMULA: an ALGOL-based simulation language,” *Commun. ACM*, vol. 9, no. 9, pp. 671–678, 1966.

[30] A. Goldberg and D. Robson, *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.

[31] B. Liskov, A. Snyder, R. Atkinson, and C. Schaffert, “Abstraction mechanisms in CLU,” *Commun. ACM*, vol. 20, no. 8, pp. 564–576, 1977.

[32] J. D. Ichbiah, R. Firth, P. N. Hilfinger, O. Roubine, M. Woodger, J. G. P. Barnes, J.-R. Abrial, J.-L. Gailly, J.-C. Heliard, H. F. Ledgard, and others, *Reference manual for the Ada programming language*. Castle House Publications Limited, 1983.

[33] B. W. Kernighan and D. M. Ritchie, *The C Programming Language (2nd Edition)*. Prentice-Hall, Englewood Cliffs, 1988.

[34] B. J. Cox, *Object Oriented Programming: an Evolutionary Approach*. Addison Wesley, 1986.

[35] “iOS Developer Library.” [Online]. Available: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html.

[36] B. Meyer, “Genericity Versus Inheritance,” in *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, 1986, pp. 391–405.

[37] S. E. Hudson, “UIMS Support for Direct Manipulation Interfaces,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 2, pp. 120–124, Apr. 1987.

[38] R. J. K. Jacob, “Using formal specifications in the design of a human-computer interface,” *Commun. ACM*, vol. 26, no. 4, pp. 259–264, 1983.

[39] R. J. K. Jacob, “A specification language for direct-manipulation user interfaces,” *ACM Trans. Graph.*, vol. 5, no. 4, pp. 283–317, 1986.

[40] L. Cardelli, “Semantics of Data Types,” in *International Symposium Sophia-Antipolis*, 1984, pp. 51–67.

[41] P. Hudak, J. Hughes, S. Peyton Jones, and P. Wadler, “A History of Haskell: Being Lazy with Class,” in *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, 2007, pp. 12–55.

[42] M. Bidoit and P. D. Mosses, *CASL User Manual: Introduction to Using the Common Algebraic Specification Language*. Springer Berlin Heidelberg, 2003.

[43] D. Jackson, “Alloy: A Lightweight Object Modelling Notation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, Apr. 2002.

[44] J. V Guttag, J. J. Horning, and J. M. Wing, “The Larch family of specification languages,” *IEEE Softw.*, vol. 2, no. 5, p. 24, 1985.

[45] C. Rose and B. Hacker, *Inside Macintosh*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[46] B. Oakley and K. Owen, *Alvey: Britain’s Strategic Computing Initiative*. Cambridge, MA, USA: MIT Press, 1989.

[47] G. Booch, *Object oriented design with applications.* Benjamin/Cummings Publishing, Redwood City, 1991.

[48] P. Coad and E. Yourdon, *Object-oriented design*, vol. 92. Yourdon Press, Englewood Cliffs, 1991.

[49] I. Jacobson, *Object oriented software engineering: a use case driven approach*. Addison-Wesley, 1992.

[50] J. Martin and J. J. Odell, *Object-oriented analysis and design*. Prentice-Hall, Inc., 1992.

[51] T. Reenskaug, P. Wold, O. A. Lehne, and others, *Working with objects: the OOram software engineering method*. Manning Greenwich, 1996.

[52] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. E. Lorensen, and others, *Object-oriented modeling and design*. Prentice-hall Englewood Cliffs, NJ, 1991.

[53] S. Shlaer and S. J. Mellor, *Object lifecycles: modeling the world in states*. Yourdon Press, 1992.

[54] I. Graham, A. O’Callaghan, and A. C. Wills, *Object-oriented methods: principles & practice*, vol. 6. Addison-Wesley Harlow, UK, 2001.

[55] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes, *Object-oriented development: the Fusion method*. Prentice-Hall, 1994.

[56] A. Hamie, J. Howse, and S. Kent, “Fundamental Approaches to Software Engineering,” in *First International Conference, FASE’98 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS’98 Lisbon, Portugal*, 1998, pp. 123–137.

[57] J. Woodcock and J. Davies, *Using Z: Specification, Refinement, and Proof*. Prentice Hall International, 1996.

[58] R. Wieringa, “A Survey of Structured and Object-oriented Software Specification Methods and Techniques,” *ACM Comput. Surv.*, vol. 30, no. 4, pp. 459–527, Dec. 1998.

[59] M. A. Jackson, *System Development*. Prentice-Hall, 1983.

[60] D. Harel, “Statecharts: a visual formalism for complex systems,” *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, Jun. 1987.

[61] J. C. Bicarregui, K. C. Lano, and T. S. E. Maibaum, “Towards a Compositional Interpretation of Object Diagrams,” *IFIP TC2 WG2.1 Int. Work. Algorithmic Lang. Calc.*, pp. 187–211, 1997.

[62] D. E. Knuth, “Backus normal form vs. Backus Naur form,” *Commun. ACM*, vol. 7, no. 12, pp. 735–736, 1964.

[63] D. D’Souza and A. C. Wills, *Catalysis: Objects, Frameworks, and Components in UML*. Addison-Wesley, 1998.

[64] A. C. Wills, “Formal Methods applied to Object-Oriented Programming,” 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.9639&rep=rep1&type=pdf .

[65] D. D’Souza and A. Wills, “Extending Fusion: practical rigor and refinement,” in *Object-oriented development at work*, Prentice-Hall, 1995, pp. 314–359.

[66] K. Short, “Personal communication.” 2016.

[67] J. Cheesman and J. Daniels, *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[68] B. Selic, “Real-Time Object-Oriented Modeling (ROOM),” in *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS ’96)*, 1996, pp. 214–217.

[69] J. Bézivin, “Niçoise salad, Salade niçoise or Insalata nizzarda,” 2010. [Online]. Available: https://modelseverywhere.wordpress.com/2010/11/12/bits-of-history/.

[70] “15th International Workshop on OCL and Textual Modeling.” [Online]. Available: https://ocl2015.lri.fr/.

[71] J. B. Warmer and A. G. Kleppe, *The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 1998.

[72] M. Richters and M. Gogolla, “Validating UML Models and OCL Constraints,” in *≪UML≫ 2000 : Advancing the Standard*, A. Evans, S. Kent, and B. Selic, Eds. Springer Berlin Heidelberg, 2000, pp. 265–277.

[73] OMG, “Unified Modeling Language™ (UML®) v1.1,” 1997. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/97-08-02.pdf.

[74] OMG, “Object Constraint Language,” 2006. [Online]. Available: http://www.omg.org/spec/OCL/.

[75] J. Van Heijenoort, *From Frege to Gödel: a source book in mathematical logic, 1879-1931*, vol. 9. Harvard University Press, 1967.

[76] M. Richters and M. Gogolla, “OCL: Syntax, semantics, and tools,” in *Object Modeling with the OCL*, Springer, 2002, pp. 42–68.

[77] OMG, “Unified Modeling Language™ (UML®) Version 2.5,” 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/.

[78] OMG, “UML 2.0 Request for Information,” 1999. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/99-08-08.pdf.

[79] OMG, “UML 2.0 Infrastructure RFP,” 2000. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/00-09-01.pdf.

[80] OMG, “UML 2.0 Superstructure RFP,” 2000. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/00-09-02.pdf.

[81] OMG, “UML Specification Simplification Request for Proposal,” 2009. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/09-12-10.pdf.

[82] J. Bézivin, “Bits of History (SPEM and UML Profiles),” 2010. [Online]. Available: https://modelseverywhere.wordpress.com/2010/11/17/bits-of-history-spem-and-uml-profiles/.

[83] M. Guttman, “A Response to Steve Cook,” *The MDA Journal: Model Driven Architecture Straight from the Masters*, 2004. [Online]. Available: http://www.bptrends.com/bpt/wp-content/publicationfiles/02-04 COL Resp to Cook Frankel-Guttman2.pdf.

[84] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault, “Mapping specification in MDA: From theory to practice,” in *Interoperability of Enterprise Software and Applications*, Springer, 2006, pp. 253–264.

[85] O. Avila-García, A. E. García, and E. V. S. Rebull, “Using software product lines to manage model families in model-driven engineering,” *Proc. 2007 ACM Symp. Appl. Comput. - SAC ’07*, pp. 1006–1011, 2007.

[86] X. Amatriain, “A domain-specific metamodel for multimedia processing systems,” *IEEE Trans. Multimed.*, vol. 9, no. 6, pp. 1284–1298, 2007.

[87] M. Fowler, *Domain-Specific Languages*. Pearson Education, 2010.

[88] M. Fowler, “UmlMode,” 2003.

[89] OMG, “Semantics Of A Foundational Subset For Executable UML Models,” 2011. [Online]. Available: http://www.omg.org/spec/FUML/.

[90] OMG, “Systems Modeling Language (SysML®),” 2007. [Online]. Available: http://www.omg.org/spec/SysML/.

[91] OMG, “Model Interchange Working Group Presentation,” 2011. [Online]. Available: http://www.omgwiki.org/model-interchange/doku.php?id=miwg_update.

[92] OMG, “Future Development of UML Request for Information,” 2008. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/08-12-12.pdf.

[93] OMG, “Precise Semantics Of UML Composite Structures™,” 2015. [Online]. Available: http://www.omg.org/spec/PSCS/.

[94] OMG, “Precise Semantics of UML State Machine RFP,” 2015. [Online]. Available: http://www.omg.org/cgi-bin/doc.cgi?ad/2015-3-2.

[95] P. Van Roy, “Programming paradigms for dummies: What every programmer should know,” *New Comput. Paradig. Comput. Music*, vol. 104, 2009.

[96] OMG, “UML Profile For National Information Exchange Model (NIEM™),” 2014. [Online]. Available: http://www.omg.org/spec/NIEM-UML/.

[97] P. P.-S. Chen, “The entity-relationship model: toward a unified view of data,” *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[98] S. A. White, “Introduction to BPMN,” *IBM Coop.*, vol. 2, no. 0, p. 0, 2004.

[99] D. A. Schenck and P. R. Wilson, *Information modeling: the EXPRESS way*. Oxford University Press, Inc., 1994.

[100] P. Tabeling, B. Gröne, and A. Knöpfel, “Fundamental Modeling Concepts-Effective Communication of IT Systems.” John Wiley & Sons, Ltd, 2006.

[101] C. Menzel and R. J. Mayer, “The IDEF family of languages,” in *Handbook on architectures of information systems*, Springer, 1998, pp. 209–241.

[102] T. A. Halpin and H. A. Proper, “Subtyping and polymorphism in object-role modelling,” *Data Knowl. Eng.*, vol. 15, no. 3, pp. 251–281, 1995.

[103] A. Rockström and R. Saracco, “SDL--CCITT specification and description language,” *Commun. IEEE Trans.*, vol. 30, no. 6, pp. 1310–1318, 1982.

[104] W. Reisig, *Petri nets: an introduction*. Springer-Verlag, 2012.

[105] B. Shneiderman, R. Mayer, D. McKay, and P. Heller, “Experimental investigations of the utility of detailed flowcharts in programming,” *Commun. ACM*, vol. 20, no. 6, pp. 373–381, 1977.

[106] A. E. Bell, “Death by UML fever,” *ACM Queue*, vol. 2, no. 1, pp. 72–80, 2004.

[107] M. Petre, “UML in Practice,” in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 722–731.

[108] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, “Empirical assessment of MDE in industry,” *2011 33rd Int. Conf. Softw. Eng.*, pp. 471–480, 2011.

[109] A. D. Brucker and J. Doser, “Metamodel-based UML notations for domain-specific languages,” in *4th International Workshop on Software Language Engineering (ATEM 2007)*, 2007.

[110] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, “Papyrus: A UML2 tool for domain-specific language modeling,” in *Model-Based Engineering of Embedded Real-Time Systems*, Springer, 2010, pp. 361–368.

[111] OMG, “UML Profile For BPMN Processes™,” 2014. [Online]. Available: http://www.omg.org/spec/BPMNProfile/.