Domain-specific Visual Languages for Model-driven Software Engineering

A THESIS PRESENTED
BY
JOHN C. GRUNDY
PHD (1994, UNIVERSITY OF AUCKLAND),
MSC (1991, UNIVERSITY OF AUCKLAND),
BSC(HONS) (1989, UNIVERSITY OF AUCKLAND)

IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF SCIENCE

UNIVERSITY OF AUCKLAND, AUCKLAND, NEW ZEALAND
AUGUST 2021

Abstract

THIS THESIS IS A COLLECTION OF MY ORIGINAL SCHOLARLY WORKS published in peer reviewed journals and conferences. The collection of articles investigate the intersection of the fields of Domain-Specific Visual Languages (DSVLs) and Model-driven Engineering (MDE). These combined allow software engineers – and in limited circumstances end users of software systems – to specify complex software system models at high levels of abstraction (using DSVLs), and then use these models to generate parts (or even all) of the modelled target software system code, configurations, user interfaces, data formats, test cases, and/or other implementation-level details (using MDE).

The first set of articles discuss the development of a range of novel DSVL and MDE supporting tools. The second set of articles show how these can be used to support software engineers to conduct requirements engineering and define software architectures for complex software systems. The third set of articles discuss support for software engineers in designing, implementing and testing software systems using DSVLs and MDE. The fourth set of articles present DSVL and MDE-based approaches to supporting software process management. The fifth set of articles present a variety of "human-centric" and collaborative approaches to supporting these tasks in DSVL-based tools. The sixth set of articles describe support for DSVL and MDE-based tools targted to "end users", allowing these non-technical end users to define and generate their own software solutions. I conclude with a recent article describing future directions for the field.

For Judith, Stephanie, Jessica, Joshua, Alexander and Hannah.

Acknowledgments

I WOULD LIKE TO GREATLY THANK my wife Judith for her many years of love for me and support for my professional work, but also for my family life which gives my work meaning. My children Stephanie, Jessica, Joshua, Alexander and Hannah I would like to thank for their love and support, and also their understanding when work demands, especially overseas and interstate travel, have taken me away from them and sometimes important things happening in their lives.

Special appreciation goes to my PhD supervisor and mentor Professor John Hosking, who I have continued to work and co-author with to this day, as evidenced by a number of co-authored works in this thesis. There are a great many others I would like to thank for their support of my work and our valuable collaborations. These include, but are certainly not limited to, Rick Mugridge, Robert Amor, Mohamed Abdelrazek, Iman Avazpour, Amani Ibrahim, Jean-Guy Schneider, Emilia Mendes, Hourieh Khalajzadeh, Tanjila Kanij, Rashina Hoda, Qiang He, Yun Yang, Feifei Chen, Norsaremah Salleh, Massila Kamalrudin, Svetha Venkatesh, Xin Xia, David Lo, Li Li, Scott Barnett, Rajesh Vasa, Ewan Tempero, and many others. Most of the research I have advanced has been done in collaboration with the great many students I have been fortunate enough to supervise, including most of the works collected here.

The significance of this body of work has been recognised by my award of a very prestigious Australian Research Council Australian Laureate Fellowship in 2019 for the project "Human-centric, Model-driven Software Engineering" – a direct result of this long programme of work on DSVLs and MDE. Prior recognition of my research leadership and contributions has included an Alfred Deakin Professorship (2017), and Fellow of Automated Software Engineering (2012), awarded by the Steering Committee of the Automated Software Engineering conference, where some of these works included in the thesis were published.

I would like thank the funding bodies for appreciating and supporting the work I do, including in particular the Foundation for Research, Science and Technology and the Australian Research Council, the majority of the works in this thesis supported by these two funders. I greatly appreciate the many companies I have worked with on a wide range of challenging, interesting and forward-looking R&D projects, several of which are also reported in chapters in this thesis. This includes work with Orion Health, XSOL, PRISM, CSIRO, CA Labs, Peace Software, Thales, NICTA, Sofismo, and others.

Finally I thank the many supervisors I have had for supporting my research endeavours, sometimes when they conflicted with other Department, Faculty and University needs, including Mark Apperley, John Hosking, Peter Brothers, Michael Davies, Leon Sterling, John Wilson, Peter Hodgson, Jon Whittle and Ann Nicholson.

Contents

1 Introduction

Software engineering depends on models of varying levels of abstraction e.g. software processes, requirements, architecture, UI and database design, code, test cases etc. Because of their complexity, software engineers have developed visual representations of these models, as diagrams, over the past several decades. Many of these diagrammatic visual models are general-purpose and fit for modelling various aspects of most software systems. Examples include flow diagrams [13], state transition diagrams [62], entity relationship diagrams [21], state charts [42], data structure diagrams [9], and various object-oriented modelling languages, such as the Unified Modelling Language [22]. While some of these techniques are limited to certain aspects of software system modelling, in general they are designed to model "any" (or most) kinds of software application for any (or most) kinds of software domains of application.

An alternative approach to these "general purpose" software visual modelling languages is to use what I term "domain specific visual languages" – or "DSVLs" – models suited to a (sometimes very) limited domain of software systems and for (sometimes very) limited modelling tasks. These DSVLs sacrifice generality but have one or more advantages over general purpose modelling languages for this specific domain. These include using higher levels of abstraction, fitting more closely to the modeller's cognitive model of their domain and its software, and/or use of iconic or visual structures familiar to the modeller in the domain of application. Different DSVLs might be used by all software engineers or by those working on specific phases of development or in specific problem domains. They might even be designed for use by non-technical "end users" of software systems to allow them to – under constrained conditions – model their own software.

Software models must typically be translated by hand into implementations using programming language source code and/or other implementation-level artefacts, such as data schema, user interface elements, business logic and processing code, configuration files, and so on. An alternative approach is to use "Model-Driven Engineering" – or "MDE" – where a **tool** automatically translates one or more high level models into lower-level models and/or implementation-level artefacts. Advantages of such MDE approaches can include much faster software implementation, improved quality of software, enabling less technically proficient developers to realise solutions, and even allowing end users to model and build their own software solutions [66]. MDE may use general-purpose models, textual domain-specific languages, or models created and visualised with domain-specific visual languages.

This thesis summarises my efforts over many years to develop domain-specific visual language (DSVL) for software engineering models and associated model-driven engineering (MDE) tools to turn these models into software solutions.

1.1 Visual Modelling Languages in Software Engineering

Visual Languages have been used for thousands of years to communicate complex ideas, or "models" of the world. They use a range of human capabilities and allow complex models to be presented and manipulated using metaphors close to a user's cognitive models. The idea is to provide a way to visualise and/or author complex models using human visual capabilities, compared to textual representations [60, 68, 59]. These include but are not limited to boxes and lines, information containment, colour and shading, formal or informal annotations, sketched or computer-constructed diagrams, iconic representations, and textual labels [12, 59].

Software Engineering has depended on using visual models to represent complex software system characteristics even before software engineering as a discipline in its own right began [59, 55]. Even a non-trivial software system is made up of many concepts, ranging from high level requirements describing the problem space; architectural abstractions describing both software and hardware components in the solution space; design-level data, interface, processing logic and other information about decisions made about implementing the software; various information about testing and deployment; and are often used to aid in software process implementation and project management. Such visual models can be constructed manually to aid development, might be reverse-engineered from existing code and other implemented system artefacts, or might be generated from other, higher level models [66, 6].

As software systems have grown ever more complex, new general-purpose visual modelling techniques have been developed to help software engineers to manage this complexity and model a very diverse range of systems. Entity-relationship diagrams are still a very commonly used approach to describe database structures [21]. Various forms of state transition diagrams and state charts model how software system state changes over time [62, 42]. The Business Process Modelling Language (BPML) [70] has been widely used to model general business processes, technical implementations of such processes as services, and service communication. The Unified Modelling Language (UML) [22] has been used to model software requirements, system designs, software designs, and various specialised aspects of systems. Computer Aided Design (CAD) tools have been deployed in many engineering domains and many share common representations of models [43].

Key advantages of such general purpose visual modelling approaches include:

- standardised modelling constructs, semantics and notations;
- notations familiar to a wide range of users in the general domain;
- 2D (and sometimes 3D) layouts and visual elements allowing wide range of model representations;
- tool support to build models, reverse engineer models from code, and support aspects of model-driven engineering; and
- useful for solving problems in a very wide variety of application domains.

1.2 Domain-Specific Visual Languages in Software Engineering

Such general-purpose modelling languages are not always the best choice to use to model a software system. They do not leverage particular domain-specific concepts, leading often to overly-complex models, cluttered visual formalisms, hard-to-read and hard-to-maintain diagrams [17, 67, 72, 37]. These problems are caused by using modelling and notational concepts designed for a very wide range of purposes. In order to describe a problem in a target domain, a large number of general-purpose abstractions may need to be assembled. No domain-specific concepts are built into the modelling language and hence models need to be composed to describe them. General purpose visual notations similarly need to be composed or augmented in often cumbersome ways to express domain-specific concepts e.g. with UML profiles and BPMN annotations. Usability can become a major problem of both the general purpose visual notation and its supporting tools [59, 1, 64]. Even domain-specific extensions to the general purpose modelling language, such as UML profiles and BPMN annotations, do not solve these issues, and often further complicate the visual models used [65, 64, 14].

Domain-specific Visual Languages (DSVLs) provide a powerful, human-centric approach to presenting and manipulating complex information [58, 41, 20, 51, 69]. DSVLs are designed for use in a specific domain with a domain-specific set of modelling concepts and a domain-specific set of visual notational elements [58, 51]. As domain concepts are built into the language, modelling problems in the domain – for which the DSVL is targeted – typically requires much smaller models and visual notational elements. Ideally the DSVL visual notation leverages both concepts and representations – "visual metaphors" – from its target domain of use. This makes the DSVL both more efficient for modelling in the target domain, but also better suits its target end users' modelling needs than a general purpose modelling language.

1.2.1 What are DSVLs?

There is no specific, generally accepted definition for a "DSVL". In fact, they go by a variety of names e.g. visual domain-specific language, domain-specific visual modelling language, or domain-specific visual model. I define a Domain-specific Visual Language (DSVL) – for the purpose of this thesis and based on a definition John Hosking and I devised many years ago – to be:

"a visual modelling language where the model and notation are customised for a particular problem domain"

In DSVLs we make a trade off between generality of the language – i.e. the range of problems that are able to be solved – and terseness of notation and closeness of mapping to the target problem domain – i.e. how specialised the visual notation is for use in a particular domain. By this I mean that we have a specialised visual modelling language only suitable for use in specific application domains, **but which is optimised for these domains**.

The critical features of a DSVL are thus:

- an underlying model – or "meta-model" – with constructs for modelling only within the target domain, not very general problem domains;
- visual notations ideally familiar to the DSVL target users in the specific target domain;
- the use of visual metaphors specific to the target domain;
- only useful (usually) for solving problems in the target domain; and
- sometimes is an existing, preferred modelling language for its users in the target domain.

To give tangible examples of such DSVLs, I briefly review a few below, with their key features, advantages – and some key limitations. Enterprise Modelling Language [52], Horus-HPC [3] and Statistical Design Language [50] are described in detailed chapters of their own later in this thesis.

Enterprise Modelling Language (EML)

EML (Enterprise Modelling Language) is a DSVL that we invented for modelling business processes and process structures [52]. It uses a novel tree-based metaphor to structure services within an organisation into a hierarchical form, similar to that used to group organisational structures. A novel overlay metaphor is used to describe process flow, linking services in the hierarchy together to form the process flow. Multiple overlays correspond to different processes. Process branching can be visualised by diverging or converging overlaps. Exceptions and error handling can be described with specialised overlays. A supporting tool, EMLTool [52], provides an authoring tool with novel fish-eye viewer and other large-scale DSVL support.

Figure 1.1 shows a student enrolment system's key services and processes modelled in EML. This example shows a tree structure being used to represent different components and work tasks implemented as discrete services in a student management system, overlaid with multiple "process flows". The tool, EMLTool, allows users to show and hide different process overlays, break large enterprise service hierarchies into multiple views, and provides large scale service representation via fish-eye views and alternative views of processes using a subset of BPML. Business Process Execution Language for Web Services (BPEL4WS) scripts are generated from the EML models to be "run" by a BPEL-based workflow management system.

EML does not support general software application modelling and only a subset of service orchestration. It is oriented towards knowledgeable domain experts who are non-technical uses of complex service-oriented applications and want and need the power to configure new process flows using pre-implemented and hosted services. While EML supports flexible and scalable modelling of these services, services themselves can not

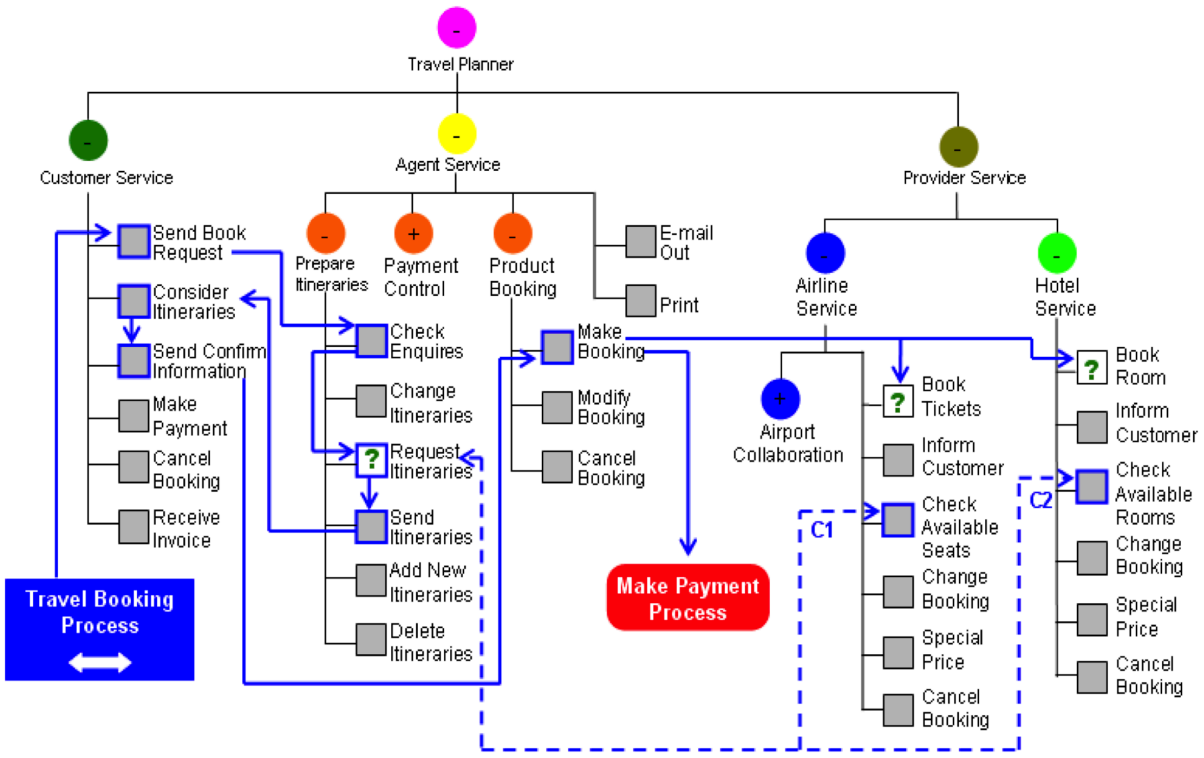


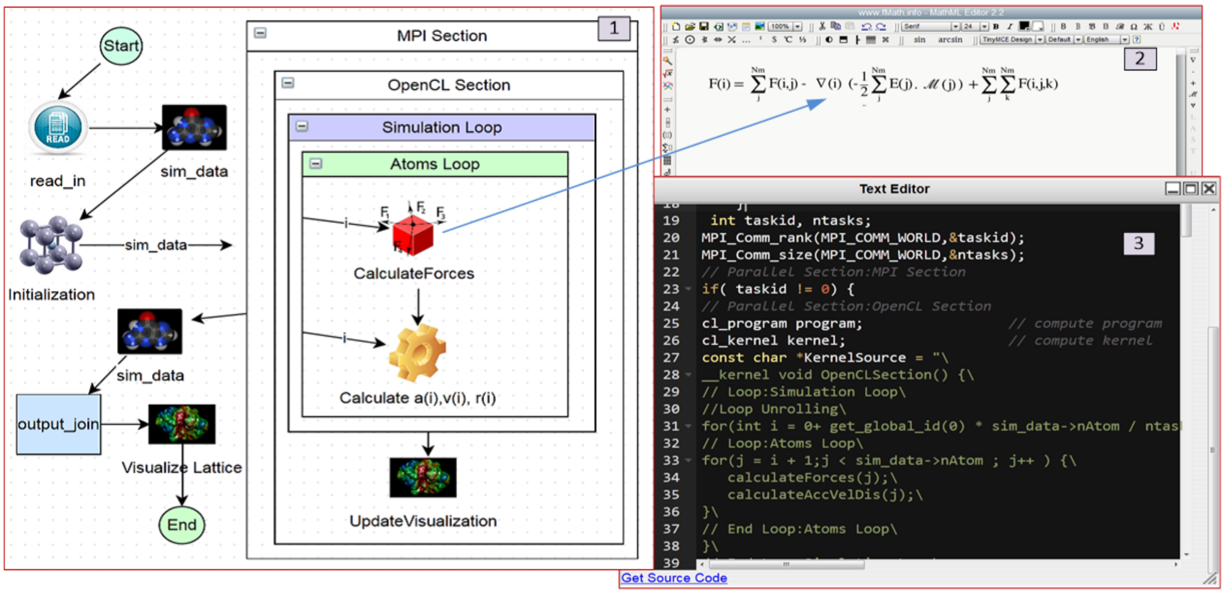
Figure 1.1: Enterprise Modelling Language (EML) showing a student enrolment process (from [52]).

be built, only aggregations and orchestrations. No data storage and user interface capabilities for the services are provided by EML itself. The tree-and-overlay metaphor is natural for these target end users but may be frustrating or counter-intuitive to others, including service implementers.

Horus-HPC

Horus-HPC (High Performance Computing) is a web-based IDE for developing parallel programs for complex scientific tasks to be run on GPUs, CPU grids or cloud computing platforms [3]. The target end user is a scientific programmer who is knowledgeable about the target scientific problem domain; has some coding skills; and some limited HPC design and development skills. The idea is to use a set of DSVLs at differing levels of abstraction to model target problem domain (e.g. molecular simulation or pulsar discovery algorithms), including scientific formulae; model sequential solutions; model - with the help of packaged patterns - HPC parallelised solutions; describe deployment onto a target HPC platform; and generate parts of a C or GPU kernel program code implementation; and then complete this code by hand. The tool is not designed for complete code or script generation but to be a quality and productivity improvement support platform. The DSVLs have the added advantage that the higher level ones are aimed at mirroring how scientific end users actually describe problems in their domain using whiteboards, formulae, pseudo-code, schematics etc.

Figure 1.2 shows an example of three Horus-HPC diagrams modelling a parallel program. The left hand side diagram shows a high level box-and-line representation of key HPC program components wired together. The top right formulae editor allows scientists to model mathematical aspects of their problems at a high level and link these abstractions both to box-and-line component realisations and code implementations of (parts of) their scientific problem. The C code editor in the bottom right shows a generated program from the component diagram with user additions of code. The message passing (OpenMPI) or GPU (OpenCL) code is generated from lower-level component diagrams that map the algorithm components onto HPC hardware implementation components for the chose implementation platform. Further views include data visualisation and modelling to show results of complex calculations as charts, scatter plots, etc,



$$F(i) = \sum_{j}^{Nm} F(i,j) - \nabla(i)\ (-\frac{1}{2}\sum_{j}^{Nm} E(j).\ \mathscr{M}(j)) + \sum_{j}^{Nm}\sum_{k}^{Nm} F(i,j,k)$$

```
 int taskid, ntasks;
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
// Parallel Section:MPI Section
if( taskid != 0) {
// Parallel Section:OpenCL Section
cl_program program;                    // compute program
cl_kernel kernel;                      // compute kernel
const char *KernelSource = "\
__kernel void OpenCLSection() {\
// Loop:Simulation Loop\
//Loop Unrolling\
for(int i = 0+ get_global_id(0) * sim_data->nAtom / ntas
// Loop:Atoms Loop\
for(j = i + 1;j < sim_data->nAtom ; j++ ) {\
   calculateForces(j);\
   calculateAccVelDis(j);\
}\
// End Loop:Atoms Loop\
}\
```

Figure 1.2: Horus-HPC being used to model a parallel program (from [3]).

and to model complex input and output data formats used to produce the generated C code implementation data structures.

While Horus-HPC provides a fully-fledged, cloud-based development environment for a very wide range of HPC applications and HPC hardware platforms, removing the 3GL C code editor from the IDE leaves it with special purpose modelling views that are not general purpose. These have been designed for scientist-programmers to model several levels of abstractions of their HPC programs. While their models allow us to generate significant amounts of HPC C code, they do not generate full implementations without direct textual C code editing.

Statistical Survey Design Language (SDL)

SDL (Statistical Design Language) is a DSVL that we invented for designing and enacting statistical surveys. It provides a suite of related visual modelling languages addressing different aspects of statistical survey design and implementation using the R statistical analysis tool [50]. Survey diagrams are used to represent a high level view of a survey including purpose, key population sources and key outcomes. Survey data diagrams are used to model the datasets used in the surveying including data sourcing and management. Task diagrams are used to model the key steps involved in running the survey including pre-survey activities and post-survey activities. Technique diagrams are used to model low-level statistical techniques to analyse parts of the survey data. Technique diagrams are used to generate R scripts and web services providing remote access to the encapsulated scripts. Services cane be orchestrated to realise the surveying.

Figure 1.3 shows an example of three SDL diagrams modelling a NZ Crime Victimisation Survey (from [50]). After creating the overall survey structure a statistician creates SDL views to specify the survey process and techniques to use in detail. Figure 1.3 (a) shows a hierarchical task diagram, specifying two data analysis tasks to be carried out on the survey data. During data collection, the main concern is to specify sampling techniques used in the survey process and types of statistical metadata related to collected data. Figure 1.3 (b) specifies the two sampling methods to be used in the survey. Here the sampling frame is stratified in two stages by the modified area unit (1) then household visits planned using patterned clustering (2). Data Frame and Sample data icons indicate data used in pre-testing and collection phases. Figure 1.3 (c) shows a survey technique diagram describing the data analysis operations implementing the tasks in Figure 1.3 (a). Two visualisation methods (boxplot and multivariate analysis) are used to assess whether there is evidence of a statistical association between data variables. It generates textual R script implementations of aggregate techniques and uses R scripts to implement its low-level technique components. Figure 1.3 (d) shows links

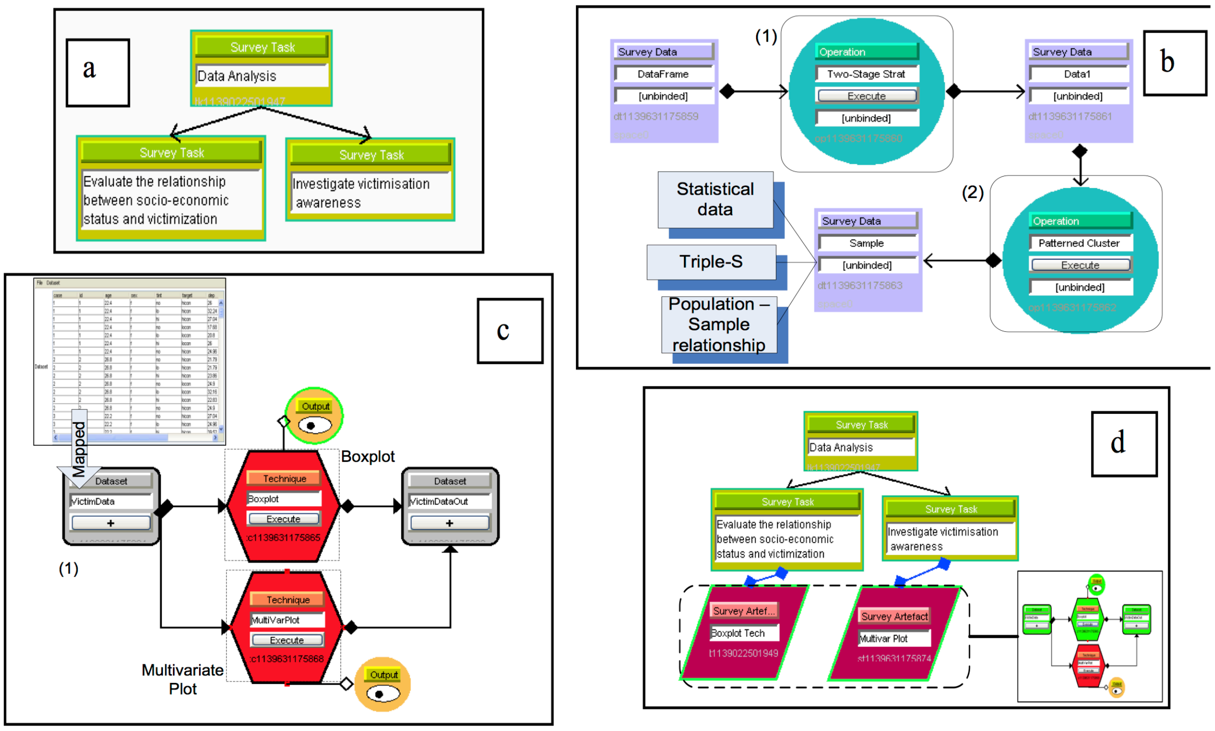


Figure 1.3: Statistical Design Language (SDL) being used to model (a) survey tasks; (b) survey sampling techniques; (c) an analytical technique; and (d) links between diagrams (from [50]).

between diagrams and artefacts.

SDL provides a powerful range of multi-level abstractions for statistical survey designers, statisticians being the target end users. This includes high level design, data specification, process flow, low-level statistical techniques, and a wide range of visualisations to show survey data analysis results. SDLTool also provides capabilities to generate reusable aggregate services that implement complex processes and techniques for reuse by other service-oriented applications. However, SDL can not express general purpose programming concepts and constructs and nor is it designed for general purpose business intelligence applications.

1.2.2 Why use DSVLs

Sometimes it is not obvious that a DVSL-based solution either exists or is necessarily a better solution than a textual Domain-Specific Language (DSL) or a general purpose modelling approach or programming language. Some indicators I have found useful when exploring the use of DSVLs for application to a new domain are outlined below. DSVLs are a useful modelling approach in situations where:

- a domain has a set of domain-specific concepts capturing rich properties about the domain that lend themselves to being captured in a meta-model;
- experts in the target domain have a set of notations or representations of the domain concepts they regularly use to describe aspects of the domain in designs, meetings, to explain models etc;
- models describing solutions in the domain can be readily constructed from these notations and underlying concepts far more readily and efficiently that if using more general purpose modelling languages and tools;
- domain-specific models can be used to synthesize general models e.g. code or parts of general models to realise solutions
- target end users find the domain specific modelling concepts, notations, tool support for authoring the models and support for generating solutions (or partial solutions) from the models more effective and efficient than using general-purpose modelling tools and approaches.

A good indication of a DSVL being potentially useful is when talking with software engineers or target end users – and/or visiting their work spaces or observing them interacting in meetings – a set of informal

DSVLs and their related domain-specific concepts are frequently used. Business process designers and indeed many business process improvement activities for many years have adopted BPMN and EML-style modelling constructs to design how the processes work, who enacts them, and related artefacts and data.

If a domain has a set of limited concepts that naturally provide an abstract description of a solution in that domain, this can be indicative of potential for DSVL (or DSL or combination) solution. These concepts are limited in scope i.e. are domain-specific, but provide powerful constructs for expressing complex ideas simply, consistent and elegantly. For example, the process stage, process flow, split flow, join, flow, actor and artefact concepts in business process modelling provide a very powerful, simple metaphor for describing a wide range of solutions. This leads to DSVLs like EML being able to model complex business process problems far more quickly and accurately and usefully than general-purpose approaches. The statistical surveying domain does not have an agreed DSVL or set of DSVLs, but we identified when talking with statisticians it does have well-defined concepts that we were able to map to SDL model concepts. From this we developed a set of DSVL elements to represent these domain-specific concepts and their inter-relationships.

While 3GL computer programming languages are very flexible and powerful, and it may be argued APIs and libraries provide sets of packaged abstractions for sub-domains, key disadvantages are the time it takes to construct solutions, repetitiveness, and lack of higher level abstractions than code constructs. DSVLs describing higher level domain elements can be enriched with properties to enable generation of whole or part 3GL programming language solutions. Enrichment may also be by relating one DSVL element to another e.g. a Horus-HPC parallel computation element in one view to a GPU grid compute element in another view, indicating how OpenCL kernel code is to be generated from the combined model.

1.2.3 Designing DSVLs

DSVL meta-models can be designed using conventional conceptual and data modelling techniques [51, 20]. Design concepts are identified from various information sources: domain experts; existing partial models (whether on paper or computer); existing 3GL or DSL solutions where meta-model elements and relationships are abstracted; or from databases, CSV and XML files, or other domain artefacts that capture part of the necessary modelling constructs for the domain. Often other solutions exist for the target domain, even DSVL based solutions, that provide most if not all of the information needed to define and construct the necessary meta-model for the new DSVL based solution [20].

For example, Horus-HPC's low-level models and parallel programming patterns are derived from the large body of work in this area over many years. However, its high level models we had to define after considerable work with scientists who develop their own bespoke HPC solutions for different domains [3]. EML uses BPMN and earlier business process modelling tool meta-model constructs to describe its overlay processes. SDL provides ways to package R scripts for reuse related to survey implementation, and structure data definitions and higher level survey process tasks and goals.

The most challenging – and often most creative – aspect of DSVL based solution design are the visual notations that provide much of the power and advantage of DSVL-based solutions [59]. A range of drivers influence how the visual notation is designed and how the overall solution will appear to target end users. These include but are not limited to:

- Who are the target end users of the DSVL? Are they familiar with visual oriented modelling approaches and support tools? Are they technically knowledgeable software engineers or non-technical domain experts or potentially both?
- What size and complexity of model will need to be represented?
- How big and complex will the resultant DSVL-based models get?
- if multiple DSVLs will be used to represent the problem domain, how do we link parts together across diagrams?
- What visual metaphors do the target end users work with now? Can we reuse and build on these in some way so the proposed DSVL-based solution will look familiar to them?
- Will DSVL models need to be shared amoung many people, live for a long time, be modified exten-

sively during their useful lifespan?
- Are there any existing DSVLs used for different problem domains that might translate well to the one under consideration? Can we learn from the successes (or failures) of these DSVLs in those other domains?

Practical considerations also need to be taken into account. Can the intended implementation platform for the DSVL tool actually support the range and complexity of the visual notations? Are the target end users going to be able to effectively and efficiently understand and use both the notations themselves and their editing tools? How do we handle challenges like version control, configuration management and collaborative editing of DSVL-based representations? Is the DSVL-based solution really better than using a conventional general purpose 3GL programming language (C, Java, Python etc), scripting language (R, Matlab, Perl etc). Is a special purpose DSL (textual) language more useful than a visual form?

These approaches are not always incompatible e.g. EMLTool supports both the EML DSVL and general purpose BPML; Horus-HPC includes a fully functional C code editor and extensive API library; and including components and scripts implemented in textual R code is supported by SDLTool.

Users of DSVL-based modelling solutions can be software developers e.g. HorusHPC, domain experts e.g. statisticians for SDL and business process experts for EML.Sometimes multiple user groups can be supported by the same tool/DSVL or subsets of the DSVL. Scientists can specify high level physical model principles in HorusHPC formula views and software developers translate these into detailed parallel code level DSVLs. Survey designers can specify goals and high level process tasks in SDL process diagrams, and expert statisticians and data scientists specify statistical technique details in Technique diagrams.

1.2.4 Realising DSVL-based Applications

Once a DSVL has been designed for a target domain we need to build a tool that supports the use of this DSVL. Given the complexity of such a task, a number of platforms have been developed to aid the creation of DSVL-based tools e.g. MetaEDIT+ [71], Eclipse GMF [24] and Microsoft DSL tools [16]. As can be seen from the representative DSVLs and their supporting tools in this chapter, a wide range of considerations need to be taken into account when building such a tool [63]:
- How large and complex with the DSVL tool likely be - how many DSVL diagram types, elements, and other features like code generation, data import or export, data visualisation etc? Different platforms provide different ranges of solutions.
- Can the tool platform handle the range of visual abstractions, appearances, composites and intended editing operations and interactions?
- Will complex data or code or scripts need to be imported? Generated and exported? Does the tool platform have the necessary capabilities?
- Will the tool interface need to be provided through a web browser or mobile phone?
- Are there approaches provided to supporting DSVL versioning, diffing, shared editing, large scale rendering etc?
- Will the DSVL-based tool need to be integrated closely with other tools / applications, and is there support for such integration in the DSVL tool platform?
- Can we impose various necessary constraints, checks and design critics on the DSVLs using the DSVL tool, in order to ensure models are correct, consistent and complete?

One of my main contributions over many years to making DSVL concepts realisable is the development of numerous supporting tools, both for the DSVL modelling and associated MDE-based support. These include but are not limited to MViews [38], JViews [27], JComposer [35], Pounamu [73], Marama [37], VikiBuilder [44], and Horus [3].

When developing prototype DSVLs and their support tools for research and practice, we want to evaluate both the support tool and its DSVL solution using a range of criteria [59, 23]. Learning from the results of these evaluations, we may want to refine the DSVL and/or its support tool to address issues our target end users have encountered and reported to us or that we have observed.

1.3 Model-Driven Engineering of Software

Doug Schmidt in a widely read introduction to a COMPUTER special issue on Model-driven Engineering describes MDE as:

"Model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively. " [66]

Similarly to some of the drivers behind Domain-Specific Visual Languages choice in some domains, MDE approaches to software engineering attempt to address long standing issues including:

- textual programming languages ("3GLs") are often too low-level to describe many abstractions in software engineering
- SE models are often too disconnected from 3GLs (program code) e.g. traditional analysis and design languages
- we often need high-level modelling languages to better express requirements, architectures, designs, tests etc through software engineering processes
- such SE models can be used to "construct" software directly i.e. translate from design-level elements to code-level elements via a "model transformation" approach – these models at various levels of abstraction can still be directly turned into/related to code constructs
- to do this we need to provide ways to build models, reason with models, translate models to(/from) code

Note that working with textual 3GL code is often still very useful/necessary even in systems where a lot of model-driven development support is used. Similarly, textual Domain-specific Languages (DSLs) can be used with visual DSVL models in domains where textual representations are more useful some of the time.

Many early MDE approaches originally focused on code generation. Increasingly, rather than generating just code from models, scripts or configurations or even other models are generated. For example, SDL-Tool generates R scripts from its survey technique diagrams, some of which may contain prepacked R script function calls themselves. EML generates BPEL4WS (Business Process Execution Language 4 Web Services) models that themselves can be run (enacted) via a workflow engine to orchestrate business process-implementing web services.

An example of such model-driven engineering from a textual Domain Specific Language is RAPPT [11]. Figure 1.4 outlines its process. RAPPT takes a textual design-level mobile App Description – sometimes called a Platform Independent Model (PIM) (it can also take a DSVL that represents limited, high-level parts of the textual app model). It has a model transformer that transforms this abstract design-level PIM App Description into a much more detailed code- and API-level Android Model for the Android platform mobile apps – sometimes called a Platform Specific Model (PSM). It then uses a further Code Generator component to translate the Android Model combined with a set of Code Templates into Android mobile app implementation artefacts – source code but also scripts, manifest, directories, XML and iconic elements. These can then be built by an Android development environment to implement the modelled mobile app. RAPPT allows generated Android code to be edited by the developer to add further features not supported by its code generated or to refine the skeleton mobile app implementation into a fully-fledged app product.

1.4 DSVLs and MDE for Software Engineering

Bringing these two approaches together – modelling complex software systems with DSVLs and using these DSVL-visualised models in MDE processes –- has been a key focus of a large part of my research work to date. Key requirements for such a combined approach, as outlined in Figure 1.5, include:

- a set of domain meta-model(s), model instances are used to model the problem domain;

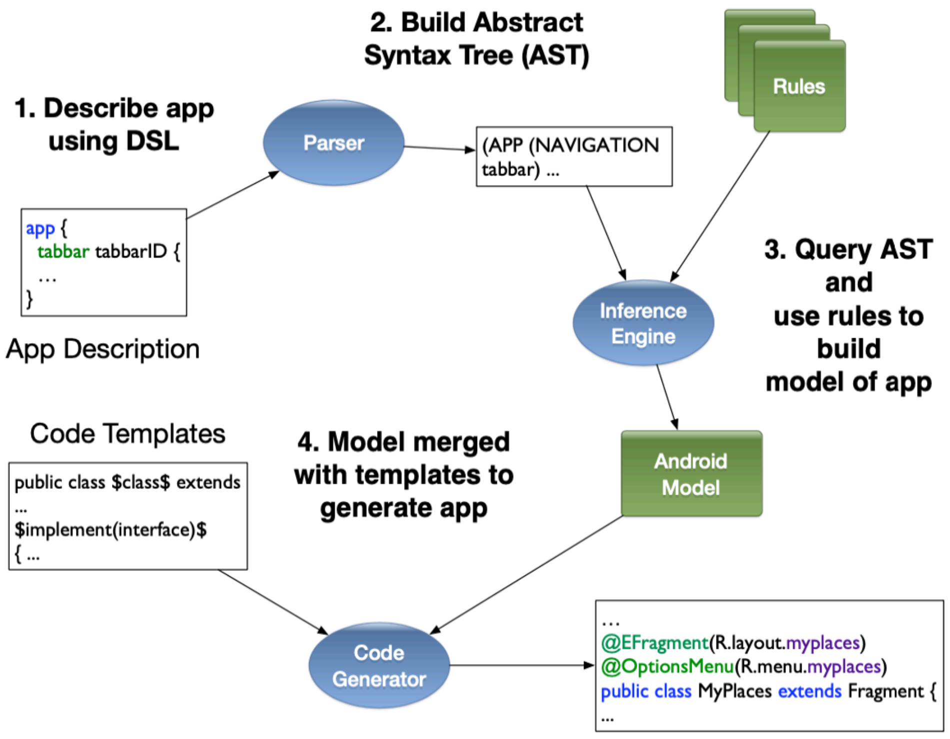

2. Build Abstract Syntax Tree (AST)
Rules
1. Describe app using DSL
Parser
(APP (NAVIGATION tabbar) ...
app {
tabbar tabbarID {
...
}
App Description
3. Query AST and use rules to build model of app
Inference Engine
Code Templates
4. Model merged with templates to generate app
Android Model
public class $class$ extends
...
$implement(interface)$
{ ...
Code Generator
...
@EFragment(R.layout.myplaces)
@OptionsMenu(R.menu.myplaces)
public class MyPlaces extends Fragment {
...

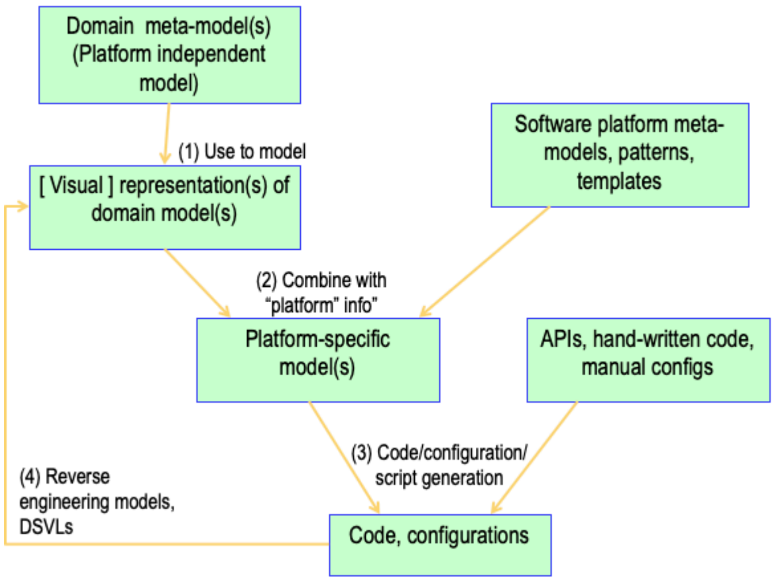
Figure 1.4: Example of MDE-based textual app model to android model to code generation process in RAPPT (from [11])

- a set of visualisation(s) of these domain model instances –- textual and graphical – allow software engineers and/or end users to construct and refine problem domain solutions;
- mappings between models allow a tool to transform higher level models to lower level ones, including to code, scripts, XML etc.;
- a high level model might be transformed into a lower level model e.g. a requirements-level problem space model to a design-level solution space model, or a platform independent model enriched to become a more detailed platform specific model;
- low-level models are transformed to implementation-level code, scripts, configurations, etc that can be compiled to implement the system, or may be interpreted by an engine to realise the software desired;
- editing tools for DSVL-based model visualisations;
- transformation support i.e. model->model, model -> code transformers;
- visualisation support of models is sometimes useful e.g. code/data -> model -> DSVL represenation;
- reasoning support e.g. analysis of models –- completeness, correctness, consistency; and
- model management support e.g. version control, diffing/merging, team collaboration support etc.

To illustrate the variety of ways this DSVL+MDE approach can be used, in the following subsections I illustrate a few example DSVL+MDE tools from my work. Some of these are targeted at supporting software engineers performing specific tasks. Some are targeted at supporting domain experts (non-technical end users). For each example I outline its problem domain; target user group(s); key meta-model elements; DSVL(s) used; transformation approach used; and target generated artefacts.

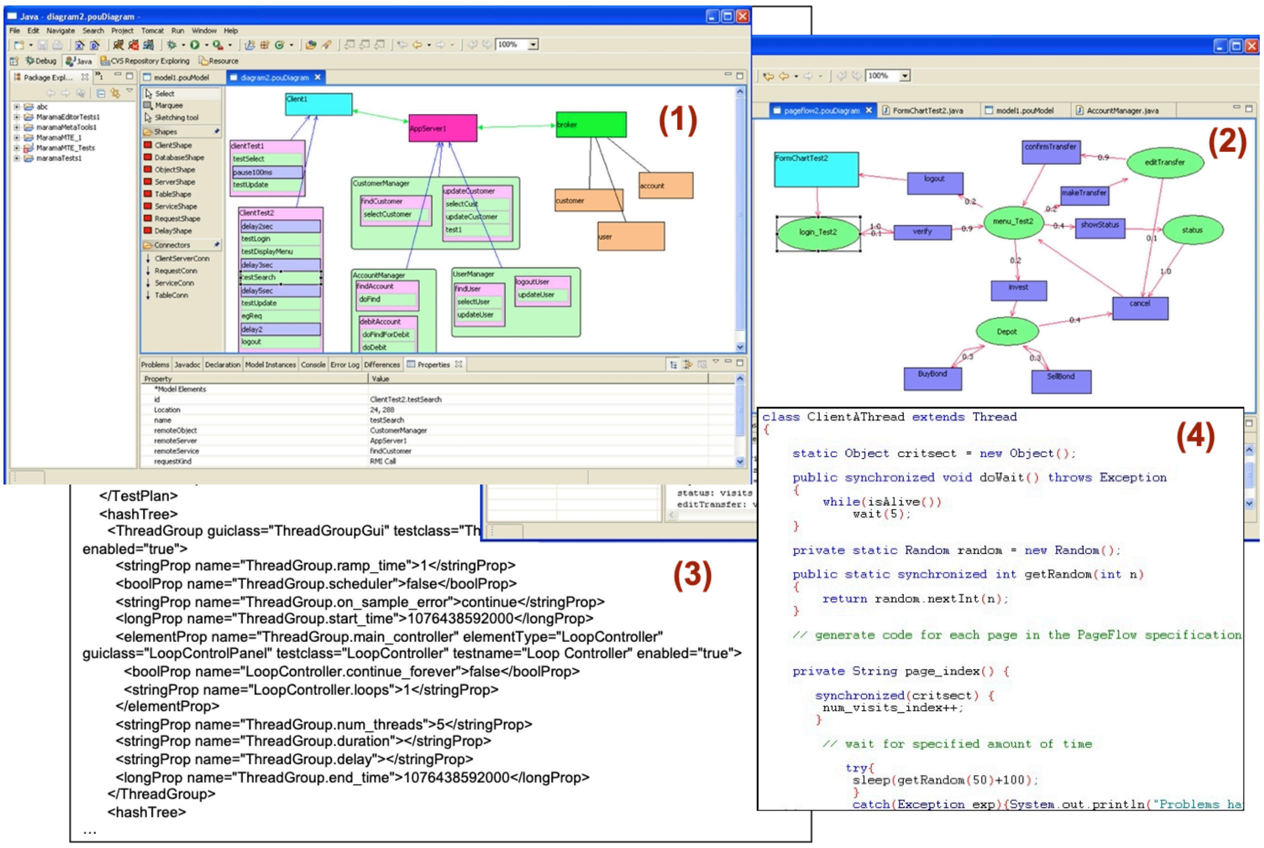
1.4.1 Performance Test-bed Generation Tools

I have led research into numerous performance engineering tools using DSVLs and MDE approaches. The initial ideas for this line of research came from my time in industry in the 1980s where I had to try and improve performance of complex database systems. This required writing many testing scripts – or "performance test-beds" – that was very tedious work. However, many of these test-bed scripts had great similarities and if a high level model of the target system structure could be defined, much if not all the performance testing scripts could be automatically generated from these models. Figure 1.6 shows one such tool in use,



Domain meta-model(s) (Platform independent model)
(1) Use to model
Software platform meta-models, patterns, templates
[Visual] representation(s) of domain model(s)
(2) Combine with "platform" info"
Platform-specific model(s)
APIs, hand-written code, manual configs
(3) Code/configuration/ script generation
(4) Reverse engineering models, DSVLs
Code, configurations

Figure 1.5: Outline of the DSVLs + MDE engineering process



```
</TestPlan>
<hashTree>
  <ThreadGroup guiclass="ThreadGroupGui" testclass="Th
enabled="true">
    <stringProp name="ThreadGroup.ramp_time">1</stringProp>
    <boolProp name="ThreadGroup.scheduler">false</boolProp>
    <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
    <longProp name="ThreadGroup.start_time">1076438592000</longProp>
    <elementProp name="ThreadGroup.main_controller" elementType="LoopController"
guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Controller" enabled="true">
      <boolProp name="LoopController.continue_forever">false</boolProp>
      <stringProp name="LoopController.loops">1</stringProp>
    </elementProp>
    <stringProp name="ThreadGroup.num_threads">5</stringProp>
    <stringProp name="ThreadGroup.duration"></stringProp>
    <stringProp name="ThreadGroup.delay"></stringProp>
    <longProp name="ThreadGroup.end_time">1076438592000</longProp>
  </ThreadGroup>
  <hashTree>
...
```

```
class ClientAThread extends Thread
{
    static Object critsect = new Object();

    public synchronized void doWait() throws Exception
    {
        while(isAlive())
            wait(5);
    }

    private static Random random = new Random();

    public static synchronized int getRandom(int n)
    {
        return random.nextInt(n);
    }

    // generate code for each page in the PageFlow specification

    private String page_index() {

       synchronized(critsect) {
        num_visits_index++;
       }

        // wait for specified amount of time

            try{
             sleep(getRandom(50)+100);
             }
             catch(Exception exp){System.out.println("Problems ha
```

Figure 1.6: Example of the MaramaMTE performance testbed generator in use (from [19])

MaramaMTE [19].

MaramaMTE has the following key features:

- Problem domain: generation of performance test bed code/scripts from high level modelling of software architectures of distributed systems
- Target user group(s): performance engineers (typically highly experienced software engineers)
- Key meta-model elements: architectural elements of distributed systems – clients, servers, databases, network connections, compute and data applications ; loading models for clients e.g. number users, number transactions per second, types of transactions etc
- DSVL(s) used: (1) architecture model with detailed characteristics of each element (representing key software components); and (2) stochastic form charts (representing probabilistic client loading models)
- Transformation approach used: XSLT scripts and Java code
- Target generated artefacts: client, server code, loading scripts, build scripts, deployment scripts

When using MaramaMTE, the performance engineer models their system's software architecture using architectural view DSVLs, as shown in Figure 1.6 (1). They model client loading models using stochastic form charts, as shown in Figure 1.6 (2). The XSLT-based code generators synthesize from these two DSVL models scripts e.g. for Apache JRunner (3) and Java client and server code e.g. Figure 1.6 (4). MaramaMTE generates real code that is compiled and run for clients and servers. The loading scripts have the clients run a very large number of transactions against the servers and MaramaMTE collects results for presentation to the performance engineers.

1.4.2 Data Transformation Tools

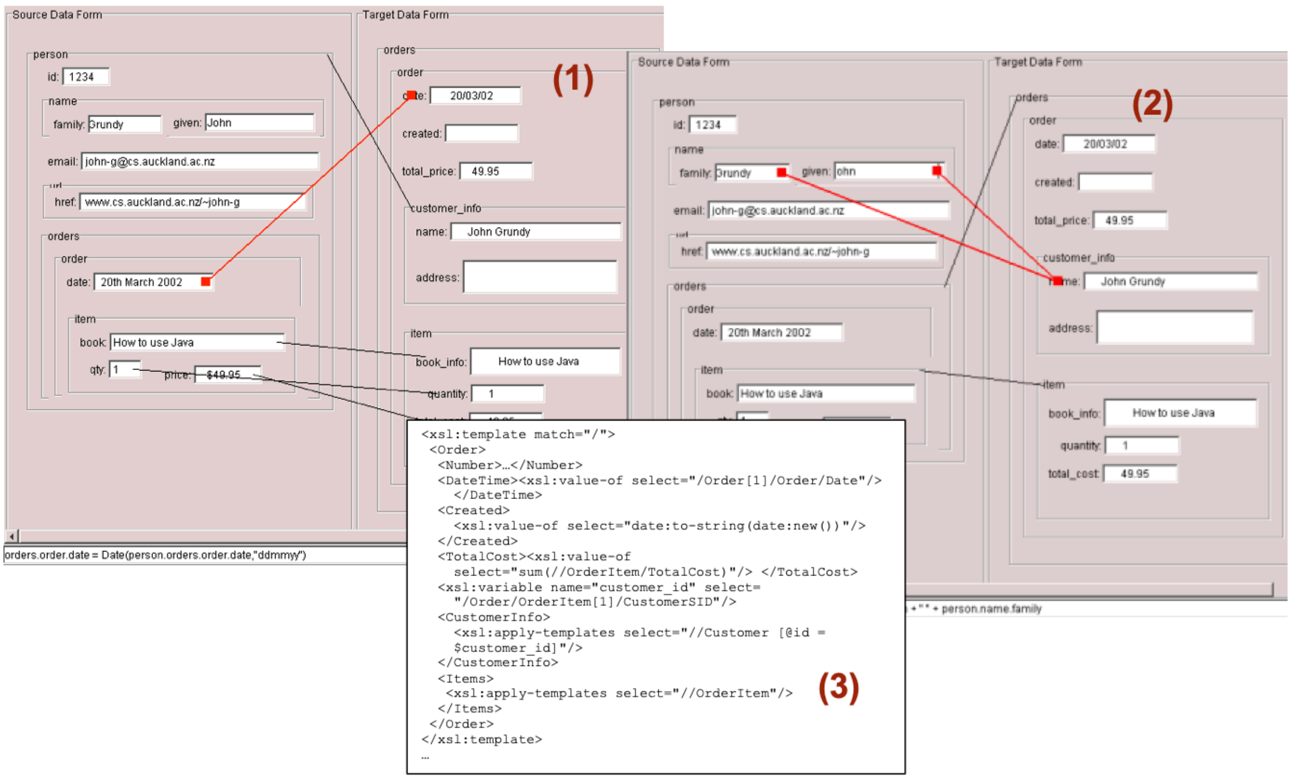
Integrating complex distributed systems by exchanging complex data is a very common need in many systems. Originally Electronic Data Interchange messages were used and the writing of EDI encoding and decoding software to support EDI message exchange between systems is a very challenging task. We developed a tool with Orion Health in a collaborative R&D project to facilitate the modelling of complex EDI messages for health system data exchange, the Orion Message Mapper [34], eventually commercialised as the Raphsody message mapping engine suite. This approach used hierarchical tree-based message mappings to specify correspondences between source EDI message elements and target EDI message elements, and formulae to translate source data to target data formats. While suitable for software engineers to use, the original target users of Orion Message Mapper, they are not very suitable for non-technical domain experts.

This led to a new approach I invented, and supervised a Masters student to prototype and evaluate, the Form-based Mapper [53]. This uses form visualisations of XML data models – meant to be analogous to the business forms traditionally used to exchange data between businesses e.g. fill out an order form, send to supplier, supplier copies data from form to another format, supplier processes order etc. Figure 1.7 shows an example of the Form-based Mapper in use.

Key features of the Form-based Mapper include:

- Problem domain: data exchange between complex business systems
- Target user group(s): non-technical domain experts
- Key meta-model elements: business form structures – represented as XML models
- DSVL(s) used: hierarchical "form" visualisation meant to resemble real-world paper and electronic forms
- Transformation approach used: XSLT scripts
- Target generated artefacts: XSLT scripts

The domain end user e.g. someone who is knowledgeable about the data to be exchanged between businesses, imports an XML schema and has it visualised as a form like layout, as in Figure 1.7 (1). The user may rearrange the format to look more like a real-world paper or electronic form. They then specify via drag and drop "correspondences" between form elements, as shown in Figure 1.7 (1) and (2). Some of these, like the correspondences in Figure 1.7 (1) can be simple one to one mappings with little or no data format



```
orders.order.date = Date(person.orders.order.date,"ddmmyy")
```

```
+ " " + person.name.family
```

```
<xsl:template match="/">
 <Order>
  <Number>…</Number>
  <DateTime><xsl:value-of select="/Order[1]/Order/Date"/>
    </DateTime>
  <Created>
    <xsl:value-of select="date:to-string(date:new())"/>
  </Created>
  <TotalCost><xsl:value-of
    select="sum(//OrderItem/TotalCost)"/> </TotalCost>
  <xsl:variable name="customer_id" select=
    "/Order/OrderItem[1]/CustomerSID"/>
  <CustomerInfo>
    <xsl:apply-templates select="//Customer [@id =
    $customer_id]"/>
  </CustomerInfo>
  <Items>
   <xsl:apply-templates select="//OrderItem"/>
  </Items>
 </Order>
</xsl:template>
…
```

Figure 1.7: Example of the Form-based mapping generator in use (from [53])

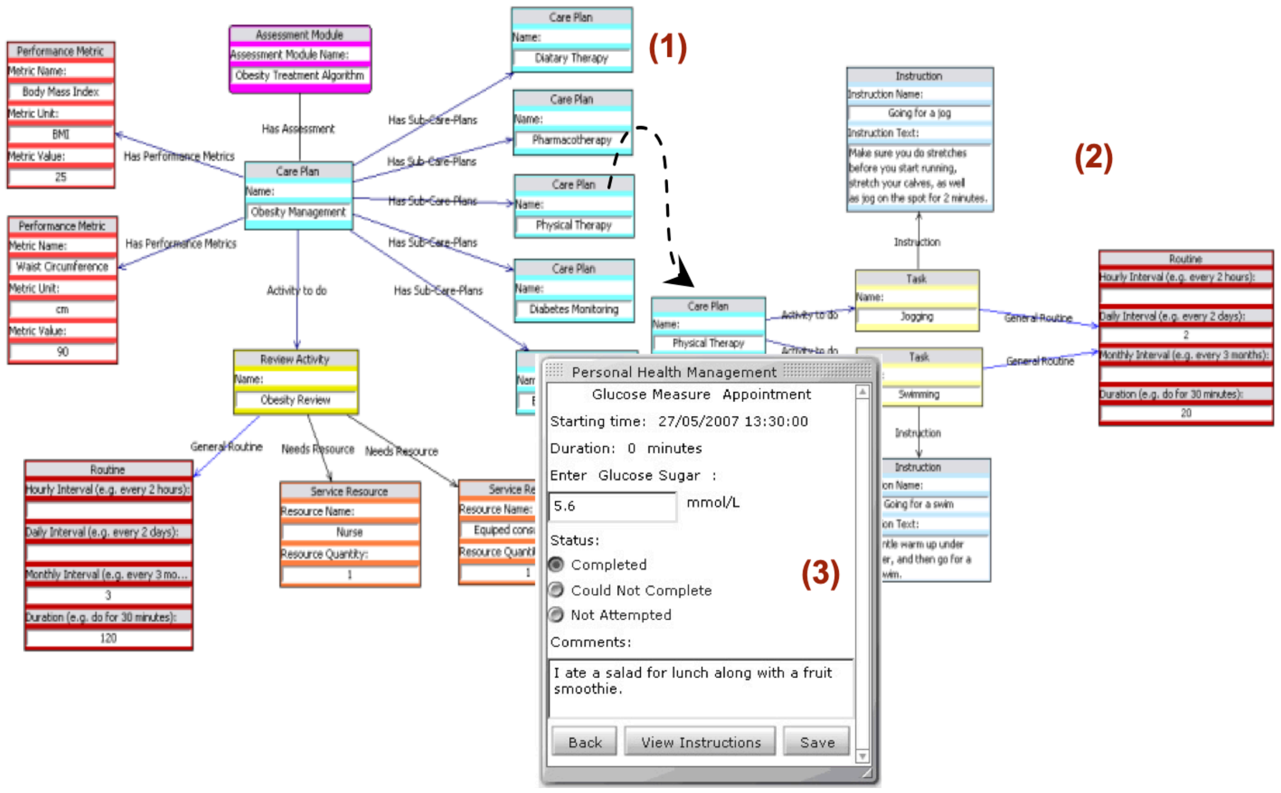
translation; some can be 1:many or many:many complex data transformations. A formulae builder is provided (examples shown bottom text field in Figure 1.7 (1) and (2)) allows specification of formula-based data transformations, meant to be like spreadsheet-like formulae. The Form-based Mapper then generates XSLT scripts to implement XML to XML data transformations based on the specified mappings, part of one shown in Figure 1.7 (3). One issue we found when evaluating the format translation formulae is these are difficult to end users to use, being themselves based on XSLT-based formulae.

1.4.3 Mobile App Generation Tools

Developing mobile apps has become very popular but is still predominantly limited to those with high development expertise. Despite the availability of a range of low-code/no-code MDE-based app generation and configuration tools, these have major limitations around flexibility, expressive power, and quality of generated mobile app [10]. Despite these limitations of MDE-based app generation approaches, eHealth apps are a promising area for modelling and generating fully functional apps. This is because eHealth apps in specific domains share many commonalities. We wanted to support public health clinicians in modelling and generating eHealth apps for chronic disease management e.g. diabetes, obesity, etc. These all use a similarly-structured "care plan" concept and mobile apps provide patients with self-management steps following a set of exercise, diet, pharmacology and monitoring interventions. We developed a clinician-oriented chronic disease management app modeller and generator for this domain [48]. An example of this in use is shown in Figure 1.8.

Key features of this Visual Care Plan Modeling Language (VCPML)-based eHealth app generator include:

- Problem domain: eHealth apps for chronic disease management
- Target user group(s): clinicians model app care plans and tailor to individual patient needs, patients use generated eHealth app
- Key meta-model elements: care plans and app interface components
- Visual Care Plan Modelling Language (VCPML) and a visual interface specification language



Personal Health Management

Glucose Measure Appointment

Starting time: 27/05/2007 13:30:00

Duration: 0 minutes

Enter Glucose Sugar :

5.6 mmol/L

Status:

[✓ Completed]

Could Not Complete

Not Attempted

Comments:

I ate a salad for lunch along with a fruit smoothie.

Figure 1.8: Example of the Visual Care Plan Modelling Language eHealth app generator in use (from [48])

- Transformation approach used: C implemented code generators
- Target generated artefacts: OpenLaszlo code which itself is then translated into Flash or Microsoft Mobile code to run on (old) phone platforms

Figure 1.8 (1) shows a VCPML diagram to support patients with obseity management. The idea for VCPML came from medical texts where these care plans were textually described, and we developed a meta-model and visual model to describe them. These can be hierarchical, with Figure 1.8 (2) showing a more detailed care plan relating to physical therapy after clicking on the icon in Figure 1.8(1). A further visual language (not shown) is used to describe the details of how to present care plan elements in a mobile phone GUI. An OpenLaszlo implementation of the mobile app is generated, itself then transformed into one or more specific mobile app implementations. The one shown in Figure 1.8 (3) is a Flash-based implementation running on a handset with a Flash player embedded. This is a 1000% code generation approach - unlike our RAPPT tool described previously, the generated code can not be changed. While in theory the VCPML approach to chronic disease management app generation was good, a number of limitations were encountered by users. These included confusing user interface specification language; lack of accounting for diverse users of the mobile apps in the generated apps – a one size fits all approach; and inability to change or augment generated app appearance and functionality.

1.4.4 DSVL Tool Generators – "DSVL Meta-tools"

As discussed previously, we need to implement sophisticated support tools in order to realise DSVL and MDE based approaches. We have found that such DSVL-based tools themselves are amenable to using DSVL+MDE approaches, given they have many commonalities, common meta-models and DSVLs can be developed to describe them, and much of their functionality can be generated, as either code or configurations. I have used DSVL and MDE techniques to specify and generate many DSVL-based tools - I call these meta-tools or meta-DSVL tools [35, 37]. One such example is the Vikibuilder, a DSVL and MDE-based tool for specifying Visual Wikis [44]. An example of VikiBuilder in use to specify a "Lost" TV series Visual

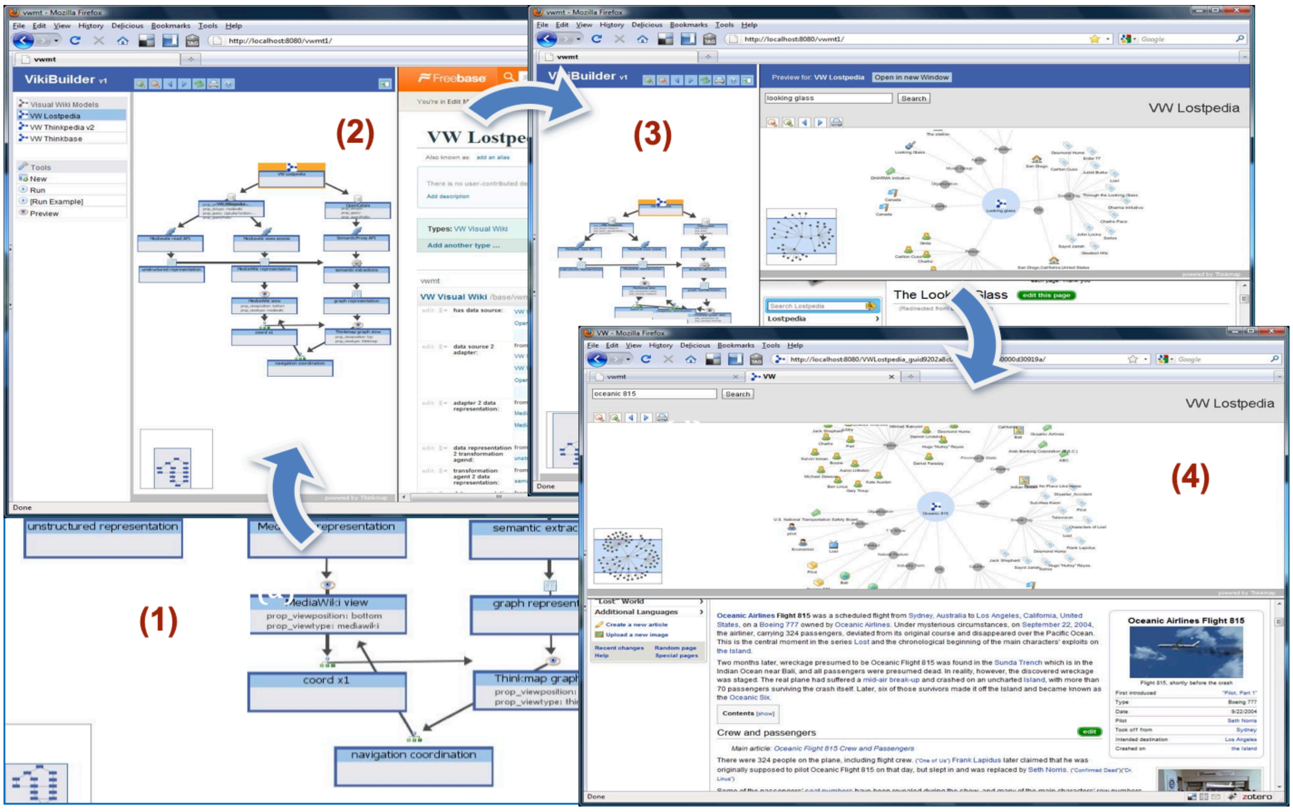


Figure 1.9: Example of the VikiBuilder Visual Wiki designer and generator in use (from [44])

Wiki is shown in Figure 1.9.

- Problem domain: Generating "visual wikis"
- Target user group(s): Visual wiki designers - not necessarily software engineers (unless they want to author their own Java plug-ins to extend the platform – see below).
- Key meta-model elements: key visual wiki elements e.g. data sources, data filters, data transformations, various visual wiki screen elements
- DSVL(s) used: a simple box and line visual wiki element composer, itself realised as a Visual Wiki (thus VikiBuilder is itself a "meta-Visual Wiki")
- Transformation approach used: Java code creating database content
- Target generated artefacts: creates a database containing configuration information for the new visual wiki, which is interpreted to create the new Visual Wiki on the Confluence platform ; additional hand-implemented Java code plug-ins can be added to enhance functionality

Figure 1.9 (1) shows part of the specification of a new Lostpedia Visual Wiki that is intended to provide an interactive, visual way of exploring the Lostpedia site. Various details about data sources, filters, transformations, aggregations, and visualisations are specified using form-based information associated with each visual wiki DSVL element, as shown Figure 1.9 (2). Note the VikiBuilder tool is itself a Visual Wiki built on top of the Confluence Enterprise Wiki platform with a set of Java plug-ins. The new visual Wiki can be tested and specifications, both DSVL elements and their properties updated interactively, shown in Figure 1.9 (3). Finally the complete Lostpedia Visual Wiki can be deployed for use, shown in Figure 1.9 (4).

The generator takes the specified visual wiki information and populates a database with essentially a set of detailed configuration model information, rather than generating code to implement the Visual Wiki. A single Visual Wiki code platform thus interprets different detailed specification models to produce quite different Visual Wikis. The VikiBuilder is thus a good example of model to model transformation using MDE, rather than model to code. In addition, software engineers can specify Java-based plugins to include in the new visual wiki that use a set of APIs to extend the platform capabilities, to include functionality not built into our original Visual Wiki platform. The VikiBuilder also thus illustrates semi-automated use of

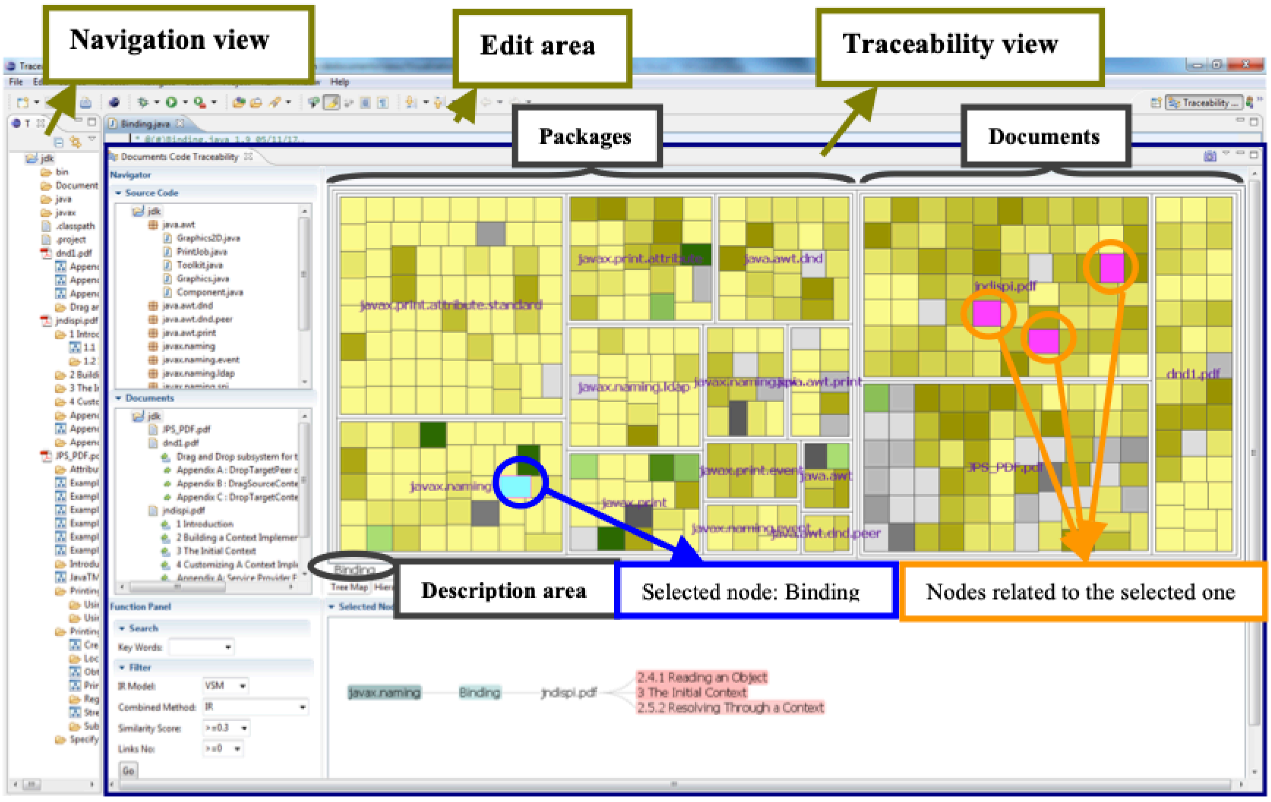


Figure 1.10: Example of the DCTracVis traceability link reverse engineering tool in use (from [15])

MDE to produce solutions – the DSVL allows modelling of a wide range of concepts, but ones not catered for it supports specifying Java plugins to use that are hand-coded by developers.

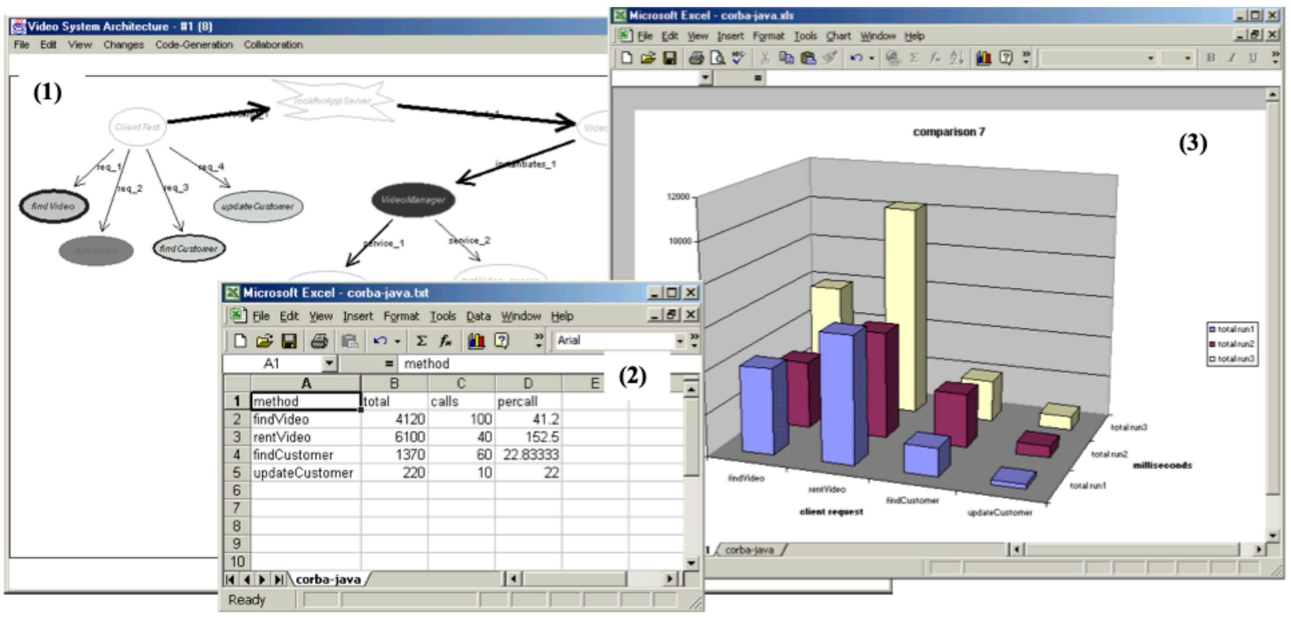
1.4.5 Reverse-engineering DSVL Models from Code

All of the examples I have shown so far of DSVLs and MDE use "forward engineering" – high level model to lower level model to code/script/configuration model. Sometimes it is useful to use "reverse engineering" where a high level DSVL-visualised model is extracted from lower level models, code, documentation, performance logs, etc. An example of such a tool is DCTracVis, a tool that we developed to address the problem of visualising a large number of reverse-engineered traceability links between code and documentation [15]. Figure 1.10 shows an example of DCTracVis in use.

Key features of DCTracVis include:

- Problem domain: want to visualise and explore interactively a large number of reverse-engineered traceability links between source code and documentation
- Target user group(s): software engineers
- Key meta-model elements: code abstract syntax tree, document paragraphs and words, and links between code tree elements and document words/phrases.
- DSVL(s) used: a heat map and tree visualisation
- Transformation approach used: Java-based reverse engineering tool to extract models from Java source code and PDF documents
- Target generated artefacts: higher level model of tracelinks between code elements and documentation elements

Figure 1.10 shows a complex Java program that has been analysed and its Packages (collections of Java classes) shown as group of heat map visualised items (left hand side "Packages" heat map). A set of PDF documents and section headings within the PDFs are show in the right hand side heat map ("Documents"). When the software engineer clicks on a node – a Java class – in the left hand side Packages heat map, elements



	A	B	C	D	E
1	method	total	calls	percall	
2	findVideo	4120	100	41.2	
3	rentVideo	6100	40	152.5	
4	findCustomer	1370	60	22.83333	
5	updateCustomer	220	10	22	

Figure 1.11: Example of the SoftArch/MTE performance engineering tool showing augmented DSVL with reverse engineered data (from [26])

corresponding to sections in the documentation about this class are highlighted in the right hand side Documents heat map. A tree visualisation of links is also shown at the bottom, here showing a traceability link from the selected javax.naming class to three sections in the jndspi.pdf documentation file explaining uses of this Java class. The developer can then go into the PDF at each indicated section to see potentially useful information about using this javax.naming class.

Tools that use DSVLs and MDE for forward-engineering can also use reverse-engineering and annotation of their DSVL diagrams to show e.g. run-time reverse engineered information. An example of this is used in our SoftArch/MTE, Marama/MTE and Cloud/MTE performance engineering tools [26, 19]. These extract low-level run-time performance data after running performance tests, abstract this data into high level performance summaries, and visualise these summaries by highlighting DSVL diagram icons. Figure 1.11 shows an example of this in Softarch/MTE [26]. A software architecture DSVL model (1) is highlighted with shading and line thickness to show – at a high architectural level – places in the architecture design causing potential performance bottle-necks. The data is abstracted from run-time captured low-level performance data, shown in (2), and can also be visualised in an alternative way using a bar chart, as shown in (3).

1.5 Overview of the papers in this Thesis

Figure 1.12 shows a summary of many of the tools and approaches described in the papers that make up this thesis, with an indicator to which part/chapter they appear in. Each tool/approach is briefly introduced and summarised in the following subsections. At the top of Figure 1.12 are several DSVL-based meta-tools and various extensions to support more human-centric and collaborative modelling with these DSVLs. A range of software engineer-supporting tools are shown below the timeline in Figure 1.12 in blue, supporting a range of requirements, architecture, design, coding, testing and process management tasks. A set of end-user oriented tools are shown in red at the bottom of Figure 1.12.

The collection of papers I include in Part 1 describe ways in which DSVL-based MDE tools can be described and implemented, including themselves using DSVL and MDE-based approaches. In Parts 2-4, a collection of papers describe why DSVLs can be a good choice for various aspects of software engineering, including requirements engineering, software architecture, design, testing and to support software process modelling and enactment. In Part 5, the selected papers describe a body of my work creating various "human-centric" and collaborative support facilities that can aid the use of DSVL-based MDE approaches in various

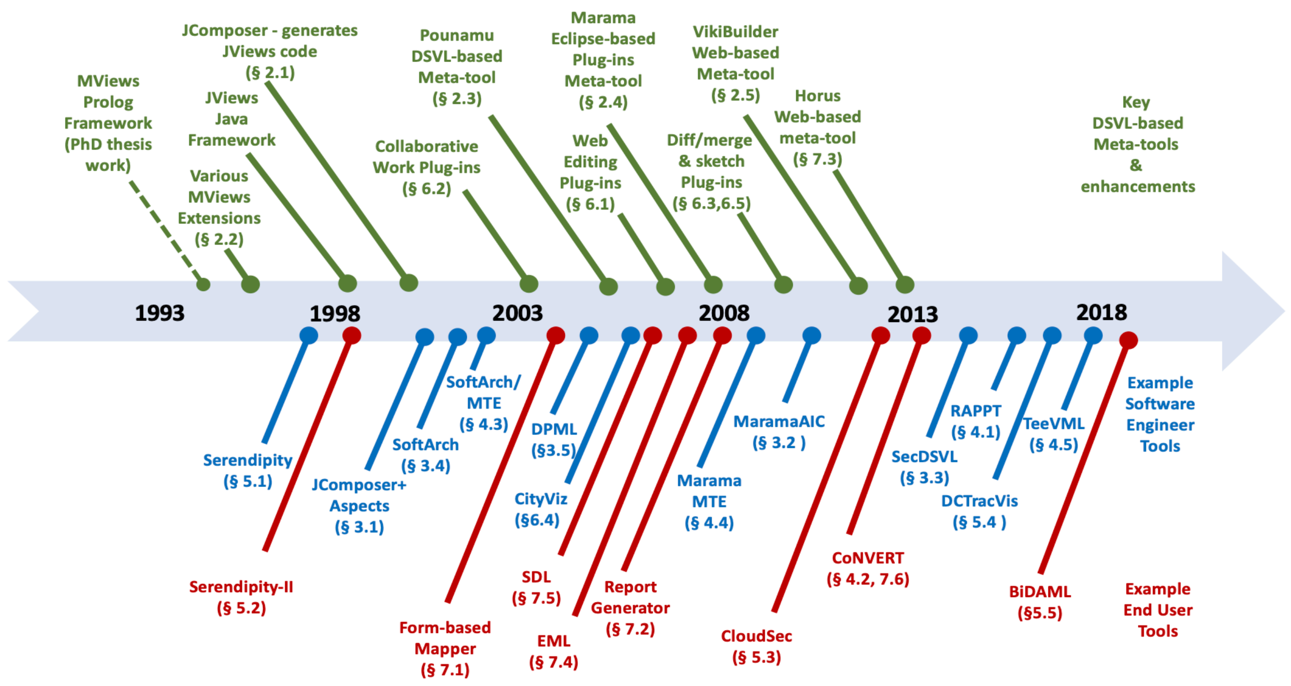

MViews Prolog Framework (PhD thesis work)
JComposer - generates JViews code (§ 2.1)
JViews Java Framework
Various MViews Extensions (§ 2.2)
Pounamu DSVL-based Meta-tool (§ 2.3)
Collaborative Work Plug-ins (§ 6.2)
Marama Eclipse-based Plug-ins Meta-tool (§ 2.4)
Web Editing Plug-ins (§ 6.1)
VikiBuilder Web-based Meta-tool (§ 2.5)
Diff/merge & sketch Plug-ins (§ 6.3,6.5)
Horus Web-based meta-tool (§ 7.3)
Key DSVL-based Meta-tools & enhancements
1993
1998
2003
2008
2013
2018
Serendipity (§ 5.1)
JComposer+ Aspects (§ 3.1)
SoftArch (§ 3.4)
SoftArch/ MTE (§ 4.3)
DPML (§3.5)
CityViz (§6.4)
Marama MTE (§ 4.4)
MaramaAIC (§ 3.2)
SecDSVL (§ 3.3)
RAPPT (§ 4.1)
DCTracVis (§ 5.4)
TeeVML (§ 4.5)
Example Software Engineer Tools
Serendipity-II (§ 5.2)
Form-based Mapper (§ 7.1)
SDL (§ 7.5)
EML (§ 7.4)
Report Generator (§ 7.2)
CloudSec (§ 5.3)
CoNVERT (§ 4.2, 7.6)
BiDAML (§5.5)
Example End User Tools

Figure 1.12: Approximate timeline of this thesis collected contributions

ways. In Part 6 I include several papers presenting reasons and examples of why DSVL-based MDE tools can also be used to support non-technical software end users to model and build their own software, in a wide variety of constrained domains. Finally, in Part 7 a recent paper outlines some future directions in supporting more human-centric model-driven software engineering using DSVLs.

1.5.1 Part 1 – Modelling tools and their development

In order to realise the approach of DSVLs and MDE to produce software systems, we need tools to model with DSVLs and generate other models/code/configurations etc. from these models with these DSVLs. The papers in this part of the thesis describe selected examples of a range of solutions we have produced to realise such tools. These examples include post-PhD work on a number MViews framework extensions, the JViews framework, the JComposer DSVL- and MDE-based JViews modeller and code generator, the Pounamu DSVL-based meta-tool, the Marama Eclipse IDE-based meta-tool, and the VikiBuilder Visual Wiki modeller and generator. Using these platforms my collaborators and I have realised dozens of innovative DSVL and MDE-based tools for both software engineers and end users in diverse application domains. These results have been published in well over 200 of my papers.

MViews (implemented in an OO Prolog) and JViews (implemented in Java) are Object-oriented (OO) frameworks for building DSVL-based tools, and provide some code and model generation support using MDE. However building DSVL-based tools with these frameworks requires textual-coding and specialising complex framework classes, time- consuming and only suitable for expert programmers to do. ***"Constructing component-based software engineering environments: issues and experiences"*** [35] describes JComposer, itself a JViews-based DSVL and MDE meta-tool for modelling and partially generating JViews-based DSVL and MDE tools. We describe the need for DSVL-based meta-tools like JComposer, its capabilities and evaluation of its support, including architectural support for distributed, collaborative work specifying and partially generating DSVL-based tools. JComposer supports modelling of partial DSVL tools. It then uses MDE to generate partial DSVL tool implementation code using the JViews Java class framework. These partial tool implementations are then completed by tool developers modifying and extending by hand the generated Java class code to complete the tool. JComposer results in much quicker/easier DSVL-based tool development in JViews than using JViews alone. JComposer, like JViews, was intended for software engineers to use, as it requires quite a lot of coding knowledge for its Java framework class specialisation and implementation.

Inconsistency between DSVL views frequently occurs when a designer modifies e.g. a high level process model or requirements model using a DSVL, but it is unclear how this change can/should be translated to a design-level DSVL model or code-level text. This becomes a more complex problem as more diagram (view) types are added, collaborative work between multiple designers is supported, and various kinds of models are integrated. ***"Inconsistency Management for Multi-view Software Development Environments"*** [32] describes a range of extensions made to the Prolog-based MViews platform (originally developed in my PhD) and its successor Java-based JViews framework to support inconsistency management in DSVL-based tools. We show a range of inconsistencies that can result and novel ways to manage them when trying to keep various general-purpose and domain-specific graphical representations of software engineering models consistent. MViews and JViews were both intended for software engineers to use, as they require a lot of coding knowledge for their OO framework class specialisation and implementation.

Pounamu: a meta-tool for exploratory domain-specific visual language tool development [73] describes Pounamu, the successor DSVL meta-tool to JComposer/JViews. Pounamu is a stand-alone, Java-implemented meta-tool with a greater range of DSVL-based meta-tools, and it generates DSVL tool specification files which are interpreted by Pouanmu itself to realise the target DSVL-based tool. Pounamu supports development of Java code-based plug-ins to extend the generated tool functionality in more seamless ways than JViews. A number of extensions to Pounamu support collaborative work and web- and mobile-based editing, described in later thesis chapters. We have used Pounamu to build a wide range of DSVL-based tools for software engineers and end users. Pounamu was intended for software engineers to use, as it requires some coding knowledge for its Java plug-ins, though we have also had some end users successfully use it to develop their own DSVL-based tools in limited ways.

Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications [37] describes Marama, the successor to Pounamu. Marama is a DSVL meta-tool realised by a set of Eclipse IDE-based plug-ins, rather than a stand-alone tool like Pounamu. Marama provides much more sophisticated DSVL-based tool specification than Pounamu or JComposer, and many more extensions via its own Eclipse plug-ins and also via many third-party Eclipse IDE plug-ins. This includes Eclipse-based code development IDEs, making Marama-based tools much more closely integrated with other development tools than in our previous meta-tool approaches. Marama even has a DSVL-based MDE code generator specification and generation tool – a DSVL-based meta-MDE tool [45]. It also has a range of novel DSVL-based design critic and constraint modelling and generation tools, these making complex DSVL-based tool functionality much easier to build than Pounamu and JViews/JComposer [2]. We have used Marama to build a wide range of DSVL-based tools for software engineers and end users. Marama was intended for software engineers to use, but we have had non-technical end users successfully use it to build basic DSVL-based tools.

Finally in this part I describe ***VikiBuilder: end-user specification and generation of Visual Wikis*** [44]. As previously summarised, VikiBuilder is a web-based, DSVL-based Visual Wiki specification and generation tool. It is itself a Visual Wiki and generates Visual Wiki configuration data via MDE from its DSVL-based models. Unlike the previous meta-tools, VikiBuilder was always intended for end user, non-technical user specification and generation of Visual Wikis. It can also be used by software engineers who can develop Java-based plug-ins to extend the Visual Wikis generated by VikiBuilder.

1.5.2 Part 2 – Requirements and Design support with DSVLs and MDE

We have used our meta-tools from the previous section, and others, to develop a wide range of tools to support software engineers during requirements engineering and software architecting tasks for complex software systems. An early example is a set of extensions to existing DSVLs – those of JComposer – described in ***Aspect-oriented Requirements Engineering for Component-based Software Systems*** [25]. In this work, I invented a set of novel requirements-level "aspect" annotations on high level components describing cross-cutting problem space concerns. These augmented JComposer requirements-level specifications and allow a software engineer to describe and reason about cross-cutting concerns in DSVL-based tools, as well as other

DSVL-described software requirements. We later added this concept to general purpose UML diagrams, used them to augment design level models, and used them to describe implemented software component capabilities to support run-time dynamic integration of component-based systems. JComposer uses MDE to take its aspect-augmented requirements DSVL models and generate design-level detailed aspect information (via model to model transformation).

MaramaAIC: Tool Support for Consistency Management and Validation of Requirements [46] describes MaramaAIC, a tool supporting requirments engineering using Essential Use Case (EUC)-based models and Essential User Interface (EUI) models. MaramaAIC provides a novel DSVL-based representation of these EUC based requirements models and uses MDE to generate EUCs from essential interaction models extracted from natural language text. It also generates example user interface mockups using MDE from its EUI based DSVL models. MaramaAIC was, as the name suggests, implemented using our Marama meta-tools.

Software security engineering is challenging. ***Adaptable, Model-driven Security Engineering for SaaS Cloud-based Applications*** [5] describes several DSVLs to support modelling different aspects of software security, including requirements-level and design-level characteristics. MDE-based tool support enables requirements level software security properties to be translated to solution space architecture and design level choices to realise security requirements. Further MDE support assists developers in encoding these DSVL-specified security solution decisions into software component code and configurations. This includes supporting run-time security property management. Finally, low-level run-time security monitoring data can be reverse-engineered and abstracted into design level DSVL-visualised information for system security managers.

SoftArch is a tool I developed to model a range of complex software architecture abstractions. **SoftArch: tool support for integrated software architecture development** [29] describes the DSVLs provided by Softarch to support a range of software architecture modelling at various levels of abstraction. MDE techniques are used to generate partial OO design models to exchange with other modelling tools. Reverse engineering is used to provide dynamic visualisation of running systems based on these models. Softarch DSVLs can be augmented by running system data to debug and understand how the systems work.

Design patterns are reusable solution approaches to tackling common design and programming problems. They are often described with UML-based design models. ***A Visual Language for Design Pattern Modelling and Instantiation*** [56] describes a novel DSVL, the Design Pattern Modelling Language (DPML), used for describing design patterns, and MDE techniques for realising these in programming code. A supporting tool, DPMLTool, was originally implemented with the JViews/JComposer meta-tool. A more advanced version, MaramaDPML, was subsequently reimplemented with the Marama meta-tool.

1.5.3 Part 3 – Development and Testing with DSVLs and MDE

We have invented many innovative DSVL- and MDE-based tools to support design, implementation and testing of software systems. ***Supporting Multi-View Development for Mobile Applications*** [10] describes RAPPT, a DSVL- and DSL-based mobile app code generation tool. Described previously, RAPPT supports combined visual DSVLs for high level mobile app modelling combined with more detailed textual DSL models to describe lower level app designs. These are then used to generate a fully functioning Android app including code, manifest, configuration and build files. The generated code is intended to be further hand-edited to polish and complete the app. RAPPT was intended for professional app developers to increase their productivity.

I have described many uses of model transformation, used in our DSVL meta-tools and in several example tools. Most of these are usually implemented as Java code, XSLT transformation scripts, or using other textual DSL code generator scripting languages. ***Specifying Model Transformations by Direct Manipulation using Concrete Visual Notations and Interactive Recommendations*** [7] describes CoNVErT, a DSVL-based model-to-model and model-to-visualisation mapping and generation tool. This paper describes the visual model-to-model specification aspects of CoNVERT – its domain-specific DSVLs used to visualise XML models, its model to model visual mapping specification DSVL, and its mapping generator that uses MDE to transform its models to a detailed XML model transformation implementation. CoNVERT was intended for non-technical end users, but can be used by software engineers too.

SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions [26] describes extensions to SoftArch including augmented software architecture DSVL diagrams and MDE to generate performance test-beds for these architectures to support large scale software performance engineering. As discussed previously, testing whether a planned complex software system will meet its performance targets is very challenging. In this paper we describe how we augmented the original Softarch DSVL-based software architecture models to add detailed implementation platform and client, server and database properties and templates to synthesize realistic client loading models and server and database models. We describe how we generate these detailed models using XSLT-based model transformation approaches. We also describe the reverse engineering and abstraction of detailed run-time performance data into summaries and display of these to the user by augmenting the architectral DSVL models. SoftArch/MTE was implemented with our JViews/JComposer meta-tool.

Previously I described and illustrated MaramaMTE, a successor to Softarch/MTE for distributed system test bed generation and performance engineering. ***Realistic Load Testing of Web Applications*** [19] describes using novel DSVL-based models in MaramaMTE – augmented architecture diagrams and augmented stochastic form charts – to model and generate via MDE web-based system performance test beds. Unlike the JViews-based Softarch/MTE, MaramaMTE is fully integrated into the Eclipse IDE as a toolset and makes use of third party Eclipse plug-ins to provide a much more integrated performance engineering toolset for software engineers.

Finally in this part I describe TeeVML, a performance emulation *environment* generator. Softarch/MTE and MaramaMTE described above generate fully functional models of complex distributed systems that are compiled and run with associated environment software (database servers, web servers etc) to performance test them. In contrast, TeeVML and its supporting tool generate "emulation environments" to test the behaviour and performance of real, very large scale software systems in mock emulation environments – essentially the opposite of MaramaMTE and Softarch/MTE. ***A Domain-Specific Visual Modeling Language for Testing Environment Emulation*** [54] describes this approach where a new DSVL-based high-level model of "system endpoints" – basically models of complex software system interfaces – are used to model and generate the complex environment a real-world software system would have to operate in. This software system is then run in this generated emulation environment, instead of having to hand-construct a (very) complex testing environment for it. TeeVML is implemented with the commercial MetaEDIT+ meta-tool.

1.5.4 Part 4 – Process and Project Management with DSVLs and MDE

Software process models can be very complex. In the 1990s there was a lot of interest in (semi-)automated tools to model and enact (run) process models to guide software development. ***Serendipity: integrated environment support for process modelling, enactment and work coordination*** [39] describes Serendipity, such a process-centred environment. A set of DSVLs are used by developers to model complex software processes. Serendipity then uses MDE to generate detailed software process models from these DSVLs that are then run – or enacted – to implement the specified software process and guide developers in following it. Serendipity provides a range of high-level and detailed software process descriptions, including "agents" that monitor software development tool activities to semi-automate complex software processes. Serendipity users are software engineers, particularly project leaders. Serendipity was implemented with the MViews meta-tool framework.

A decentralized architecture for software process modeling and enactment [36] describes the successor to Serendipity, Serendipity-II. This supports a more powerful visual editing tool, more advanced collaborative work and third party tool integration, and some improved DSVLs. Serendipity-II generates process models using MDE like Serendipity, but also generates Java code using MDE to implement a variety of tool integration support features. End users of Serendipity-II were originally intended to be software engineers, but it can also be used by non-technical project managers in other domains. Serendipity-II was implemented with the JComposer meta-tool and JViews framework.

Specifying complex software security requirements and behaviours is challenging, especially for end users whose security requirements may evolve over time. In ***Collaboration-Based Cloud Computing Security Management Framework*** [4] we present a combined DSVL- and DSL-based framework to support the process of specifying and enforcing end-to-end, complex, cloud-based Software as a Service (SaaS)-based application security requirements. From these models we generate detailed security properties for the target system using MDE. These models are then used at run-time to configure a running cloud-based SaaS application's run-time security enforcement approaches. End users are SaaS application owners, often non-technical end users. Software engineers can also use the toolset to model security requirements and have them enforced at run-time. An early version of our web-based Horus meta-tool [6] and a set of web forms are used to specify the security models.

DCTracVis: a system retrieving and visualizing traceability links between source code and documentation [15] describes the DCTracVis reverse engineering tool introduced previously. Unlike many of the systems presented in this thesis, DCTracVis abstracts higher-level models from low-level code – documentation links, reverse engineered using text processing algorithms. It then uses Heat Map- and Tree-based DSVL representations of these models to allow software engineers to browse between high-level links between documentation and code elements.

BiDaML is a tool for modelling complex data analytics applications, described in ***An End-to-End Model-based Approach to Support Big Data Analytics Development*** [47]. BiDaML uses several DSVLs ranging from high-level brainstorming diagrams to low-level technique diagrams to specify complex data analytics applications from varying perspectives. MDE-based generators produce detailed reports to guide data analytics teams, and partial Java and Python implementations of data analytics applications. End users are multi-disciplinary data analytics team members – domain experts, business analysts, software engineers, project managers and cloud platform experts. BiDaML is implemented with the commercial MetaEDIT+ meta-tool.

1.5.5 Part 5 – Human-centric DSVL Modelling and Collaboration

In this part of the thesis I describe several research works that look to support more "human-centric" modelling with DSVL-based MDE tools. By this I mean approaches to make the tools easier to use, support more "natural" modelling, and support multiple developers or end users working collaboratively together on models.

Experiences developing architectures for realising thin-client diagram editing tools [31] describes a variety of Pounamu extensions to enable users of Pounamu tools to edit DSVL diagrams using a web brower, mobile phone and even a 3D browser plug-in. The idea is that Pounamu is a Java-based application that needs to be installed on each users machine and regularly updated. Similarly, due to Pounamu's use of Java code plug-ins for much DSVL-based tool implementation, this makes sharing new DSVL tools and developing them collaboratively very challenging. Pounamu/Thin – the web/mobile-supporting extensions – instead host a single Pounamu instance on a server and provide web browser/mobile phone editing interfaces. Some of the web browser editors are quite sophisticated, using SVG and ECMA script to provide highly interactive diagramming very similar to the desktop Pouanu. Note also that this work was done in the mid-2000s, long before today's more powerful browser-based client capabilities were developed. Pounamu/Thin even allows Pounamu meta-tool specifications to be edited and thus new Pounamu-based DSVL tools to be designed using the browser-based interface.

In ***Engineering plug-in software components to support collaborative work*** [28] we describe a set of JViews-based plug-ins that provide a range of collaborative work facilities to JViews-based DSVL editing tools. A wide range of plug-ins are provided to support collaborative editing, awareness support, shared repositories, process-centred environment control and annotation of change histories, and so on. These plug-ins use the aspect-based extensions to JViews and JComposer to support very dynamic, run-time plug-in support.

A generic approach to supporting diagram differencing and merging for collaborative design [57] presents a set of Pounamu and early Marama plug-ins that support collaborative work via DSVL diagram comparison and merging support. When working with other designers on a shared DSVL designs, such support is essential to enable changes made simultaneously or asynrhconously to be compared, and selected changes to be "committed" to a shared, unified model. The techniques described in this paper are generic and work for any Pounamu or Marama DSVL-based tool.

DSVLs don't have to be just two dimensional, box-and-line diagrams. **A 3D Business Metaphor for Program Visualization** [61] proposes a highly novel DSVL for visualising project management information using a 3D "city" metaphor of buildings, streets and various annotations. Like DCTracVis, we reverse engineer detailed project management information, abstract it, then use the city metaphor to visualise the project management data. Forward engineering from these project management visualisations using MDE was proposed to allow restructuring of projects based on modifications made to the city-based DSVLs.

Finally, bringing much of the prior works in this part together, we describe "sketching-based" interfaces to DSVL tools in ***Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool*** [30]. This describes a set of Marama plug-ins enabling sketching-based input of sketched designs, much as one would using a whiteboard or piece of paper. The sketched diagrams are automatically formalised into underlying Marama DSVL models. These sketch-supporting plug-ins also work with Marama versions of our collaborative work supporting plug-ins to support distributed, collaborative, sketched-based DSVL tools! The techniques described in this paper are generic and work for any Marama DSVL-based tool.

1.5.6 Part 6 – End user Applications of DSVLs and MDE

While many of the tools we have produced are for software engineers to help in their work, we have also produced many DSVL- and MDE-basd tools for end users, usually focused on very specific domains of work. In this part I overview several papers that support a diverse range of end users and diverse range of end user application domains.

Domain-specific visual languages for specifying and generating data mapping systems [40] describes several tools aimed at supporting complex data integration, most aimed at supporting end users in different domains e.g. business analysts, construction engineers, and eHealth system integrators. Tools described include the Orion Message Mapper and Form-based Mapper described previously. Some of the data integration tools described were designed and built with MViews and JViews frameworks.

In ***A domain-specific visual language for report writing*** [18], we describe a tool developed with an industrial partner, PRISM, who build sophisticated software solutions for the commercial print industry. One of their software systems is a DSL-based report generation tool, to be used by print industry experts. This paper describes a DSVL-based report writing designer and generator. A DSVL is used to specify complex print industry report layout and content. MDE approaches are used to generate the textual DSL language, which is then complied and run by PRISM's existing software. The DSVL- and MDE-based approach makes authoring, modifying and understanding these complex professional print industry reports easier, faster and more maintainable. PRISM commercialised the prototype toolset. Our DSVL-based report designer was implemented with Microsoft Visual Studio's DSVL-based designers and code generators.

HorusHPC, described previously, is aimed at scientists wanting to parallelise software for high performance computing domains. **Supporting Scientists in Re-engineering Sequential Programs to Parallel Using Model-driven Engineering** [3] describes its DSVLs and web-based toolset. It uses MDE to generate lower-level models and GPU C code skeletons. Scientists then complete these code skeletons by hand. Partial HorusHPC models can also be reverse engineered from existing C code.

A visual language and environment for enterprise system modelling and automation [52] describes Enterprise Modelling Langugae (EML) and its support tool, described earlier in this chapter. EML and EMLTool were designed for enterprise service modelling and generation of service orchestration scripts.

EML is a DSVL using novel tree- and overlay-based metaphors. EMLTool supports creagting EML models, and uses these EML models to generate BPML4WS service orchestrations using Eclipse model transformer and code generation plug-ins. A third party BPEL analyser checks for problems in the specifications and uses a DSVL to highlight these to the user. EMLTool was realised using our Marama meta-toolset.

Described previously, Statistical Design Language (SDL) and its supporting tool, SDLTool, are aimed at professional and amateur statisticians, and provide multiple DSVLs for high-level to low-level statistical survey designs. These are described in ***A suite of visual languages for model-driven development of statistical surveys and services, Journal of Visual Languages and Computing*** [49]. MDE techniques are used to generate R scripts and web service implementations of specified survey technique descriptions for reuse. SDLTool was implemented using the Pounamu meta-tools, along with a number of specially designed Pounamu plug-ins.

Finally in this part of the thesis I describe the use of CoNVERT for information integration and complex visualisation, in the context of household travel data aggregation, harmonisation, integration and visualisation. The paper ***Engineering Complex Data Integration and Harmonization Systems*** [8] presents an industry collaboration with the AURIN project and Data61 to source, analyse, aggregate, harmonise, transform, integrate, and visualise complex household travel survey data from several Australian states. End users of the DSVL-based tool are domain experts in human geography-based survey data, government planning, and use of diverse government planning data.

1.5.7 Part 7 – Future directions

The single paper in this part of the thesis, ***Towards Human-Centric Model-Driven Software Engineering*** [33], outlines a new research programme for more human-centric, model-driven software engineering. This reseaerch includes integrating diverse human characteristics into requirements-level and design-level DSVLs and using these human characteristics during model-driven engineering. The MDE generators either generate different apps and web site pages tailored to different end users, or generate configuration data that can be used by the apps/web sites at run-time to tailor them to diverse end user human characteristics. This work builds heavily on the contributions of many of the papers presented earlier in the thesis. It attempts to address some of the limitations of these works, specifically the lack of modelling diverse end user characteristics, such as age, gender, language, culture, personality, emotions, etc in DSVLs, and the lack of using this information during MDE to produce software better suited to diverse end user needs.

1.6 Evidence of Impact

1.6.1 Citations

The research community has utilised the results from the papers in this thesis to inform many other research projects by many leading groups internationally. These are illustrated by both number of citations to many of the works, as well as citation of the works by many leading research teams. For example, several papers have over 200 citations (Google Scholar), or the paper in this thesis and its earlier conference version having together well over 200 citations. Examples include papers on Inconsistency management in MViews and Views [32], collaborative cloud security modelling end enactment [4], design pattern modelling and instantiation [56], collaborative DSVL diagram diffing and merging [57], and aspect-oriented requirements engineering [25]. Many other papers, or the papers in this thesis and earlier conference version, have well over 100 citations. Examples include JViews/JComposer, Pounamu and Marama meta-tools papers and their earlier conference versions [35, 73, 37], MaramaAIC essential use case DSVL-based tool [46], MaramaMTE web application testbed modeller and generator [19], Serendipity process-centred environment [39], Serendipity-II workflow modelling and enactment toolset [36], City metaphor project management information visualisation tool [61], and JViews-based collaborative editing plug-ins [28].

1.6.2 Industrial collaborations and Translation to practice

Many of the research works in this thesis have been carried out in collaboration with a wide range of industrial partners. In addition, a number of the meta-tools have been used on follow-on projects with these or other collaborators.

We have developed and used our data mapping tools, including the Orion Message Mapper, Form-based Mapper [40] and CoNVERT [7], with several companies interested in complex data integration problems. These include Orion Health, Peace Software, XSOL, First Data Utilities, NICTA, VicRoads and AURIN. The Orion Message mapper was a prototype for the very successful Raphsody message mapping and integration toolset produced and commercialised by Orion Heath.

We have used our process-modelling and workflow-modelling tools, including Serendipity and Serendipity-II [39, 36] derivatives of these, or our meta-tools Pounamu and Marama [73, 37], to implement similar prototype systems, for several domains. This includes modelling complex business process models with XSOL and Peace Software. The city visualisation DSVL and prototype tool were developed in collaboration with Peace Software's R&D team [61]. A summer student used Serendipity-II ideas to prototype a workflow tool with Peace Software to model and co-ordinate complex billing system processes. This was subsequently commercialised by Peace development teams.

We used our performance engineering tools, including Softarch/MTE and MaramaMTE [26, 19], on several industrial projects, several of them as confidential consulting projects on large scale enterprise system performance analysis and improvement. Some projects we are able to talk about include performance engineering of virtual database-based systems with XSOL, large scale clinical data repository engineering with Orion Health, and large scale database systems performance engineering with First Data Utilities. We collaborated with CSIRO on Softarch/MTE development [26]. This included using information from CSIRO collaborations with a range of large Australian corporates with complex enterprise system performance engineering needs. A patent for the principles underpinning Softarch/MTE was successfully applied for.

We collaborated with Swiss consulting company Sofismo AG to attempt a commercialisation of a derivative of the Marama meta-toolset [37]. This was to underpin Sofismo's complex system modelling and analysis work, carried out with a wide range of predominantly European corporates.

TeeVML development [54] was informed by our collaboration with CA Labs on generating complex enterprise system emulation environments. The idea was to augment CA Lab's commercial toolsets with support for very large scale testing environment modelling with TeeVML DSVL-based models, and then generate test bed emulation environments from these models using MDE techniques.

We have collaborated with several scientific and medical discovery teams using, among others, HorusHPC [3] and BiDaML [47]. These included the astrophysics team at Swinburne University of Technology on modelling complex radio telescope data process software for pulsar discovery, and the medical imaging team at the Alfred Hospital on MRI image processing for disease identification.

We collaborated with Thales on developing improved requirements engineering tools for complex air traffic control and related systems. This included modelling and semi-formalising textual requirements using DSVL models using MaramaAIC [46].

We have worked with several companies during the development and evaluation of BiDaML [47]. This included real estate cost estimation algorithms with ANZ bank, predicting congestion for Melbourne CBD using VicRoads data, and recently with eHealth application developers.

Our report writing DSVL-based tool was commercialised by PRISM [18]. This involved turning the initial prototype into an "industrialised" version, supporting larger scale reports, multiple user editing, and versioning.

For our Visual Wiki and VikiBuilder [44] work we successfully applied for a US patent for its underlying principles. We then secured very significant venture capital funding to commercialise this as the Mohio information visualisation platform.

1.6.3 Next Generation Education and Training

The impact I am most proud of from this body of work is the number and range of students, post-doctoral fellows and research assistants who I have worked with and educated in this domain. We used Pounamu and Marama tools for several years in our undergraduate final year software engineering course, many undergraduate Honors student projects, and in several graduate courses on domain-specific visual languages at the University of Auckland. While not having precise number of student teams or individual projects, well over 200 students used the tools to learn about DSVL and MDE principles and built their own DSVL- and MDE-tools.

Just contributing to the papers contained in this thesis alone, there are a total of 14 PhD students, 10 Masters by research students and 1 Honors student, and 10 post-doctoral fellows and 3 research assistants. Most of these students, research assistants and post-doctoral fellows have gone into industry positions with this knowledge and skills. Of the rest, 6 have academic positions and 4 have post-doctoral fellow positions as I write this. Including the many other derivative works using JViews/JComposer, Pounamu, Marama, and a few using third party frameworks and toolkits, there are a great many more students, research assistants and post-doctoral fellows who have benefited from learning in the environment and with the techniques and toolsets that we have created.

References

[1] Agarwal, R. & Sinha, A. P. (2003). Object-oriented modeling with uml: a study of developers' perceptions. *Communications of the ACM*, 46(9), 248–256.

[2] Ali, N. M. (2007). A generic visual critic authoring tool. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)* (pp. 260–261).: IEEE.

[3] Almorsy, M. & Grundy, J. (2015). Supporting scientists in re-engineering sequential programs to parallel using model-driven engineering. In *Software Engineering for High Performance Computing in Science (SE4HPCS), 2015 IEEE/ACM 1st International Workshop on* (pp. 1–8).: IEEE.

[4] Almorsy, M., Grundy, J., & Ibrahim, A. S. (2011). Collaboration-based cloud computing security management framework. In *2011 IEEE 4th International Conference on Cloud Computing* (pp. 364–371).: IEEE.

[5] Almorsy, M., Grundy, J., & Ibrahim, A. S. (2014a). Adaptable, model-driven security engineering for saas cloud-based applications. *Automated software engineering*, 21(2), 187–224.

[6] Almorsy, M., Grundy, J., & Rüegg, U. (2014b). Horuscml: Context-aware domain-specific visual languages designer. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 133–136).: IEEE.

[7] Avazpour, I., Grundy, J., & Grunske, L. (2015). Specifying model transformations by direct manipulation using concrete visual notations and interactive recommendations. *Journal of Visual Languages & Computing*, 28, 195–211.

[8] Avazpour, I., Grundy, J., & Zhu, L. (2019). Engineering complex data integration, harmonization and visualization systems. *Journal of Industrial Information Integration*, 16, 100103.

[9] Bachman, C. W. (1969). Data structure diagrams. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems*, 1(2), 4–10.

[10] Barnett, S., Avazpour, I., Vasa, R., & Grundy, J. (2019). Supporting multi-view development for mobile applications. *Journal of Computer Languages*, 51, 88–96.

[11] Barnett, S., Vasa, R., & Grundy, J. (2015). Bootstrapping mobile app development. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2 (pp. 657–660).: IEEE.

[12] Blackwell, A. F. (2001). Pictorial representation and metaphor in visual language design. *Journal of Visual Languages & Computing*, 12(3), 223–252.

[13] Böhm, C. & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5), 366–371.

[14] Chaudron, M. R., Heijstek, W., & Nugroho, A. (2012). How effective is uml modeling? *Software & Systems Modeling*, 11(4), 571–580.

[15] Chen, X., Hosking, J., Grundy, J., & Amor, R. (2018). Dctracvis: a system retrieving and visualizing traceability links between source code and documentation. *Automated Software Engineering*, 25(4), 703–741.

[16] Cook, S., Jones, G., Kent, S., & Wills, A. C. (2007). *Domain-specific development with visual studio dsl tools*. Pearson Education.

[17] Cox, P. T., Giles, F., & Pietrzykowski, T. (1989). Prograph: a step towards liberating programming from textual conditioning. In *Visual Languages, 1989., IEEE Workshop on* (pp. 150–156).: IEEE.

[18] Dantra, R., Grundy, J., & Hosking, J. (2009). A domain-specific visual language for report writing using microsoft dsl tools. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 15–22).: IEEE.

[19] Draheim, D., Grundy, J., Hosking, J., Lutteroth, C., & Weber, G. (2006). Realistic load testing of web applications. In *Conference on Software Maintenance and Reengineering (CSMR'06)* (pp. 11–pp).: IEEE.

[20] Esser, R. & Janneck, J. W. (2001). A framework for defining domain-specific visual languages. In *Workshop on Domain Specific Visual Languages, ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-2001)*.

[21] File, P., Castell, A., Marshall, I., Walker, I., & Williams, L. (1970). From requirements specification to entity-relationship diagrams using rules. *WIT Transactions on Information and Communication Technologies*, 7.

[22] Fowler, M. (2004). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

[23] Green, T. R. G. & Petre, M. (1996). Usability analysis of visual programming environments: a ?cognitive dimensions? framework. *Journal of Visual Languages & Computing*, 7(2), 131–174.

[24] Gronback, R. C. (2009). *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education.

[25] Grundy, J. (1999). Aspect-oriented requirements engineering for component-based software systems. In *Proceedings IEEE International Symposium on Requirements Engineering (Cat. No. PR00188)* (pp. 84–91).: IEEE.

[26] Grundy, J., Cai, Y., & Liu, A. (2005). Softarch/mte: Generating distributed system test-beds from high-level software architecture descriptions. *Automated Software Engineering*, 12(1), 5–39.

[27] Grundy, J. & Hosking, J. (2002a). Developing adaptable user interfaces for component-based systems. *Interacting with computers*, 14(3), 175–194.

[28] Grundy, J. & Hosking, J. (2002b). Engineering plug-in software components to support collaborative work. *Software: Practice and Experience*, 32(10), 983–1013.

[29] Grundy, J. & Hosking, J. (2003). Softarch: Tool support for integrated software architecture development. *International Journal of Software Engineering and Knowledge Engineering*, 13(02), 125–151.

[30] Grundy, J. & Hosking, J. (2007). Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 282–291).: IEEE.

[31] Grundy, J., Hosking, J., Cao, S., Zhao, D., Zhu, N., Tempero, E., & Stoeckle, H. (2007). Experiences developing architectures for realizing thin-client diagram editing tools. *Software: Practice and Experience*, 37(12), 1245–1283.

[32] Grundy, J., Hosking, J., & Mugridge, W. B. (1998a). Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering*, 24(11), 960–981.

[33] Grundy, J., Khalajzadeh, H., & Mcintosh, J. (2020). Towards human-centric model-driven software engineering. In *ENASE* (pp. 229–238).

[34] Grundy, J., Mugridge, R., Hosking, J., & Kendall, P. (2001). Generating edi message translations from visual specifications. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)* (pp. 35–42).: IEEE.

[35] Grundy, J., Mugridge, W., & Hosking, J. (2000). Constructing component-based software engineering environments: issues and experiences. *Information and Software Technology*, 42(2), 103–114.

[36] Grundy, J. C., Apperley, M. D., Hosking, J. G., & Mugridge, W. B. (1998b). A decentralized architecture for software process modeling and enactment. *IEEE Internet Computing*, 2(5), 53–62.

[37] Grundy, J. C., Hosking, J., Li, K. N., Ali, N. M., Huh, J., & Li, R. L. (2012). Generating domain-specific visual language tools from abstract visual specifications. *IEEE Transactions on Software Engineering*, 39(4), 487–515.

[38] Grundy, J. C. & Hosking, J. G. (1996). Constructing integrated software development environments with mviews. *International Journal of Applied Software Technology*, 2(3-4), 133–160.

[39] Grundy, J. C. & Hosking, J. G. (1998). Serendipity: integrated environment support for process modelling, enactment and work coordination. In *Process Technology* (pp. 27–60). Springer.

[40] Grundy, J. C., Hosking, J. G., Amor, R., Mugridge, W. B., & Li, Y. (2004). Domain-specific visual languages for specifying and generating data mapping systems. *Journal of Visual Languages & Computing*, 15(3-4), 243–263.

[41] Guerra, E., de Lara, J., Malizia, A., & Díaz, P. (2009). Supporting user-oriented analysis for multi-view domain-specific visual languages. *Information and Software Technology*, 51(4), 769–784.

[42] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3), 231–274.

[43] Henderson, K. (1999). *On line and on paper: Visual representations, visual culture, and computer graphics in design engineering*. JSTOR.

[44] Hirsch, C., Hosking, J., & Grundy, J. (2010). Vikibuilder: end-user specification and generation of visual wikis. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 13–22).

[45] Huh, J., Grundy, J., Hosking, J., Liu, K., & Amor, R. (2009). Integrated data mapping for a software meta-tool. In *2009 Australian Software Engineering Conference* (pp. 111–120).: IEEE.

[46] Kamalrudin, M., Hosking, J., & Grundy, J. (2017). Maramaaic: tool support for consistency management and validation of requirements. *Automated software engineering*, 24(1), 1–45.

[47] Khalajzadeh, H., Simmons, A. J., Abdelrazek, M., Grundy, J., Hosking, J., & He, Q. (2020). An end-to-end model-based approach to support big data analytics development. *Journal of Computer Languages*, 58, 100964.

[48] Khambati, A., Grundy, J., Warren, J., & Hosking, J. (2008). Model-driven development of mobile personal health care applications. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (pp. 467–470).: IEEE.

[49] Kim, C. H., Grundy, J., & Hosking, J. (2015). A suite of visual languages for model-driven development of statistical surveys and services. *Journal of Visual Languages & Computing*, 26, 99–125.

[50] Kim, C. H., Hosking, J., & Grundy, J. (2005). A suite of visual languages for statistical survey specification. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)* (pp. 19–26).: IEEE.

[51] Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., & Karsai, G. (2001). Composing domain-specific design environments. *Computer*, 34(11), 44–51.

[52] Li, L., Grundy, J., & Hosking, J. (2014). A visual language and environment for enterprise system modelling and automation. *Journal of Visual Languages & Computing*, 25(4), 253–277.

[53] Li, Y., Grundy, J., Amor, R., & Hosking, J. (2002). A data mapping specification environment using a concrete business form-based metaphor. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments* (pp. 158–166).: IEEE.

[54] Liu, J., Grundy, J., Avazpour, I., & Abdelrazek, M. (2016). A domain-specific visual modeling language for testing environment emulation. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 143–151).: IEEE.

[55] Ludewig, J. (2003). Models in software engineering–an introduction. *Software and Systems Modeling*, 2(1), 5–14.

[56] Maplesden, D., Hosking, J. G., & Grundy, J. C. (2001). A visual language for design pattern modelling and instantiation. In *HCC* (pp. 338–339).: Citeseer.

[57] Mehra, A., Grundy, J., & Hosking, J. (2005). A generic approach to supporting diagram differencing and merging for collaborative design. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 204–213).

[58] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4), 316–344.

[59] Moody, D. (2009). The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Softw. Eng.*, 35(6), 756–779.

[60] Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1), 97–123.

[61] Panas, T., Berrigan, R., & Grundy, J. (2003). A 3d metaphor for software production visualization. In *Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003.* (pp. 314–319).: IEEE.

[62] Parnas, D. L. (1969). On the use of transition diagrams in the design of a user interface for an interactive computer system. In *Proceedings of the 1969 24th national conference* (pp. 379–385).

[63] Pelechano, V., Albert, M., Muñoz, J., & Cetina, C. (2006). Building tools for model driven development. comparing microsoft dsl tools and eclipse modeling plug-ins. In *DSDM*.

[64] Petre, M. (2013). Uml in practice. In *2013 35th international conference on software engineering (icse)* (pp. 722–731).: IEEE.

[65] Planas, E. & Cabot, J. (2020). How are uml class diagrams built in practice? a usability study of two uml tools: Magicdraw and papyrus. *Computer Standards & Interfaces*, 67, 103363.

[66] Schmidt, D. C. (2006). Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2), 25.

[67] Shneiderman, B. (1993). 1.1 direct manipulation: a step beyond programming languages. *Sparks of innovation in human-computer interaction*, 17, 1993.

[68] Shu, N. C. (1988). *Visual programming*. Van Nostrand Reinhold New York.

[69] Sprinkle, J. & Karsai, G. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages & Computing*, 15(3), 291–307.

[70] Thiagarajan, R. K., Srivastava, A. K., Pujari, A. K., & Bulusu, V. K. (2002). Bpml: a process modeling language for dynamic business models. In *Advanced Issues of E-Commerce and Web-Based Information Systems, 2002.(WECWIS 2002). Proceedings. Fourth IEEE International Workshop on* (pp. 222–224).: IEEE.

[71] Tolvanen, J.-P. & Rossi, M. (2003). Metaedit+: defining and using domain-specific modeling languages and code generators. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 92–93).: ACM.

[72] Whitley, K. N. (1997). Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing*, 8(1), 109–142.

[73] Zhu, N., Grundy, J., Hosking, J., Liu, N., Cao, S., & Mehra, A. (2007). Pounamu: A meta-tool for exploratory domain-specific visual language tool development. *Journal of Systems and Software*, 80(8), 1390–1407.

2 DSVL Modelling Tool Development

2.1 Constructing component-based software engineering environments: issues and experiences

Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences, *Information and Software Technology*, Vol 42, No. 2, Elsevier, January 2000, Pages 103-114.
DOI: 10.1016/S0950-5849(99)00084-1

Abstract: Developing software engineering tools is a difficult task, and the environments in which these tools are deployed continually evolve as software developers' processes, tools and tool sets evolve. To more effectively develop such evolvable environments, we have been using component-based approaches to build and integrate a range of software development tools, including CASE and workflow tools, file servers and versioning systems, and a variety of reusable software agents. We describe the rationale for a component-based approach to developing such tools, the architecture and support tools we have used some resultant tools and tool facilities we have developed, and summarise the possible future research directions in this area.

My contribution: Developed initial ideas for the approach, co-led design of the approach, implemented most of the software, led evaluation of the platform, co-authored substantial parts of the paper, investigator for funding for the work from the Foundation for Research Science and Technology (FRST)

Author pre-print available at: PDF

2.2 Inconsistency Management for Multi-view Software Development Environments

Grundy, J.C., Hosking, J.G., Mugridge, W.B. Inconsistency Management for Multi-view Software Development Environments, *IEEE Transactions on Software Engineering*, Vol. 24, No. 11, 1998, pp. 960 - 981.
DOI: 10.1109/32.730545

Abstract: Developers need tool support to help manage the wide range of inconsistencies that occur during software development. Such tools need to provide developers with ways to define, detect, record, present, interact with, monitor and resolve complex inconsistencies between different views of software artifacts, different developers and different phases of software development. This paper describes our experience with building complex multiple-view software development tools that support diverse inconsistency management facilities. We describe software architectures we have developed, user interface techniques used in our multiple-view development tools, and discuss the effectiveness of our approaches compared to other architectural and HCI techniques.

My contribution: Led development of the key ideas, co-designed the approach, implemented most of the software, led evaluation of the platform, wrote most of the paper, one of the investigators on grant for funding for the work from the Foundation for Research Science and Technology (FRST)

Author pre-print available at: PDF

2.3 Pounamu: a meta-tool for exploratory domain-specific visual language tool development

Zhu, N., **Grundy, J.C.**, Hosking, J.G., Liu, N., Cao, S. and Mehra, A. Pounamu: a meta-tool for exploratory domain-specific visual language tool development, *Journal of Systems and Software*, Elsevier, vol. 80, no. 8, August 2007, Pages 1390-1407.
DOI: 10.1016/j.jss.2006.10.02

Abstract: Domain-specific visual language tools have become important in many domains of software engineering and end user development.However building such tools is very challenging with a need for multiple views of information and multi-user support, the ability for users to change tool diagram and meta-model specifications while in use, and a need for an open architecture for tool integration.We describe Pounamu, a meta-tool for realising such visual design environments. We describe the motivation for Pounamu, its architecture and implementation and illustrate examples of domain-specific visual language tools that we have developed with Pounamu.

My contribution: Co-developed initial ideas for the approach, co-led design of the approach, co-supervised research assistant, 1 PhD and two Masters students working on software, led evaluation of the platform, wrote substantial parts of the paper, co-lead investigator for funding for the work from Foundation for Research Science and Technology

Author pre-print available at: PDF

2.4 Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications

Grundy, J.C., Hosking, J.G., Li, N., Li, L., Ali, N.M., Huh, J. Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications, *IEEE Transactions on Software Engineering*, vol. 39, no. 4, April 2013, pp. 487 - 515

DOI: 10.1109/TSE.2012.33

Abstract: Domain-specific visual languages support high-level modeling for a wide range of application domains. However, building tools to support such languages is very challenging. We describe a set of key conceptual requirements for such tools and our approach to addressing these requirements, a set of visual language-based metatools. These support definition of metamodels, visual notations, views, modeling behaviors, design critics, and model transformations and provide a platform to realize target visual modeling tools. Extensions support collaborative work, human-centric tool interaction, and multiplatform deployment. We illustrate application of the metatoolset on tools developed with our approach. We describe tool developer and cognitive evaluations of our platform and our exemplar tools, and summarize key future research directions.

My contribution: Co-developed initial ideas for the approach, co-led design of the approach, wrote initial software for the approach, co-supervised research assistant and 3 PhD students working on project, oversaw evaluation of the platform, wrote substantial amounts of the paper, co-lead investigator for funding for the work from Foundation for Research Science and Technology

Author pre-print available at: PDF

2.5 VikiBuilder: end-user specification and generation of Visual Wikis

Hirsch, C., Hosking, J.G. and **Grundy, J.C.** VikiBuilder: end-user specification and generation of Visual Wikis, In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, 20-24 Sept 2010, ACM, pp 13–22.

DOI: 10.1145/1858996.1859002

Abstract: With the need to make sense out of large and constantly growing information spaces, tools to support information management are becoming increasingly valuable. In prior work we proposed the ”Visual Wiki” concept to describe and implement web-based information management applications. By focusing on the integration of two promising approaches, visualizations and collaboration tools, our Visual Wiki work explored synergies and demonstrated the value of the concept. Building on this, we introduce ”VikiBuilder”, a Visual Wiki meta-tool, which provides end-user supported modeling and automatic generation of Visual Wiki instances. We describe the design and implementation of the VikiBuilder including its architecture, a domain specific visual language for modeling Visual Wikis, and automatic generation of those. To demonstrate the utility of the tool, we have used it to construct a variety of different Visual Wikis. We describe the construction of Visual Wikis and discuss the strengths and weaknesses of our meta-tool approach.

My contribution: Developed initial idea for the approach, co-designed the approach, co-supervised PhD student working on project, co-authored significant amount of the paper, co-lead investigator for funding for the work from Foundation for Research Science and Technology and BuildIT.

Author pre-print available at: PDF

3 DSVLs and MDE for Software Requirements and Architectures

3.1 Aspect-oriented Requirements Engineering for Component-based Software Systems

Grundy, J.C. Aspect-oriented Requirements Engineering for Component-based Software Systems, *1999 IEEE Symposium on Requirements Engineering*, Limmerick, Ireland, 7-11 June, 1999, IEEE, pp 84 - 91.
DOI: 10.1109/ISRE.1999.777988

Abstract: Developing requirements for software components, and ensuring these requirements are met by component designs, is very challenging, as very often application domain and stakeholders are not fully known during component development. The author introduces a new methodology, aspect-oriented component engineering, that addresses some difficult issues of component requirements engineering by analysing and characterising components based on different aspects of the overall application a component addresses. He gives an overview of the aspect-oriented component requirements engineering process, focus on component requirements analysis specification and reasoning, and briefly discuss tool support.

My contribution: Sole author ; developed all ideas, software, wrote whole paper.

Author pre-print available at: PDF

3.2 MaramaAIC: Tool Support for Consistency Management and Validation of Requirements

Kamalrudin, M., **Grundy, J.C.**, Hosking, J.G., MaramaAIC: Tool Support for Consistency Management and Validation of Requirements, *Automated Software Engineering*, Springer, 2017, vol 24, no 1, pp. 1-45.
DOI: 10.1007/s10515-016-0192-z

Abstract: Requirements captured by requirements engineers (REs) are commonly inconsistent with their client's intended requirements and are often error prone. There is limited tool support providing end-to-end support between the REs and their client for the validation and improvement of these requirements. We have developed an automated tool called MaramaAIC (Automated Inconsistency Checker) to address these problems. MaramaAIC provides automated requirements traceability and visual support to identify and highlight inconsistency, incorrectness and incompleteness in captured requirements. MaramaAIC provides an end-to-end rapid prototyping approach together with a patterns library that helps to capture requirements and check the consistency of requirements that have been expressed in textual natural language requirements and then extracted to semi-formal abstract interactions, essential use cases (EUCs) and user interface prototype models. It helps engineers to validate the correctness and completeness of the EUCs modelled requirements by comparing them to "best-practice" templates and generates an abstract prototype in the form of essential user interface prototype models and concrete User Interface views in the form of HTML. We describe its design and implementation together with results of evaluating our tool's efficacy and performance, and user perception of the tool's usability and its strengths and weaknesses via a substantial usability study. We also present a qualitative study on the effectiveness of the tool's end-to-end rapid prototyping approach in improving dialogue between the RE and the client as well as improving the quality of the requirements.

My contribution: Contribution: Co-developed main idea for the approach, co-developed tool design, co-supervised PhD student, wrote substantial parts of paper, co-led investigator for funding for this project from FRST

Author pre-print available at: PDF

3.3 Adaptable, Model-driven Security Engineering for SaaS Cloud-based Applications

Almorsy, M., **Grundy, J.C.**, Ibrahim, A., Adaptable, Model-driven Security Engineering for SaaS Cloud-based Applications, *Automated Software Engineering*, vol. 21, no. 2, April 2014, Springer, pp. 187–224.
DOI: 10.1007/s10515-013-0133-z

Abstract: Software-as-a-service (SaaS) multi-tenancy in cloud-based applications helps service providers to save cost, improve resource utilization, and reduce service customization and maintenance time. This is achieved by sharing of resources and service instances among multiple "tenants" of the cloud-hosted application. However, supporting multi-tenancy adds more complexity to SaaS applications required capabilities. Security is one of these key requirements that must be addressed when engineering multi-tenant SaaS applications. The sharing of resources among tenants—i.e. multi-tenancy—increases tenants' concerns about the security of their cloud-hosted assets. Compounding this, existing traditional security engineering approaches do not fit well with the multi-tenancy application model where tenants and their security requirements often emerge after the applications and services were first developed. The resultant applications do not usually support diverse security capabilities based on different tenants' needs, some of which may change at run-time i.e. after cloud application deployment. We introduce a novel model-driven security engineering approach for multi-tenant, cloud-hosted SaaS applications. Our approach is based on externalizing security from the underlying SaaS application, allowing both application/service and security to evolve at runtime. Multiple security sets can be enforced on the same application instance based on different tenants' security requirements. We use abstract models to capture service provider and multiple tenants' security requirements and then generate security integration and configurations at runtime. We use dependency injection and dynamic weaving via Aspect-Oriented Programming (AOP) to integrate security within critical application/service entities at runtime. We explain our approach, architecture and implementation details, discuss a usage example, and present an evaluation of our approach on a set of open source web applications.

My contribution: Contribution: Developed initial ideas for the research, co-supervised the two PhD students, co-authored significant parts of paper

Author pre-print available at: PDF

3.4 SoftArch: tool support for integrated software architecture development

Grundy, J.C., and Hosking, J.G. SoftArch: tool support for integrated software architecture development, *International Journal of Software Engineering and Knowledge Engineering*, vol 13, no 2, April 2003, World Scientific, pp. 125-151.
DOI: 10.1142/S0218194003001238

Abstract: A good software architecture design is crucial in successfully realising an object-oriented analysis (OOA) specification with an object-oriented design (OOD) model that meets the specification's functional and non-functional requirements. Most CASE tools and software architecture design notations do not adequately support software architecture modelling and analysis, integration with OOA and OOD methods and tools, and high-level, dynamic architectural visualisations of running systems. We describe SoftArch, an environment that provides flexible software architecture modelling using a concept of successive refinement and an extensible architecture meta-model. SoftArch provides extensible analysis tools enabling developers to analyse their architecture model properties. Run-time visualisation of systems uses dynamic annotation and animation of high-level architectural modelling views. SoftArch is integrated with a component-based CASE tool and run-time monitoring tool, and has facilities for 3rd party tool integration through a common exchange format. This paper discusses the motivation for SoftArch, its modelling, analysis and dynamic visualisation capabilities, and its integration with various analysis, design and implementation tools.

My contribution: Developed initial ideas for this research, co-designed approach, wrote the software the approach based on, wrote majority of the paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

3.5 A Visual Language for Design Pattern Modelling and Instantiation

Maplesden, D., Hosking, J.G. and **Grundy, J.C.**, A Visual Language for Design Pattern Modelling and Instantiation, *Chapter 2 in Design Patterns Formalization Techniques*, Toufik Taibi (Ed), Idea Group Inc., Hershey, USA, March 2007, pp. 20-43. DOI: 10.4018/978-1-59904-219-0.ch002

Abstract: In this chapter we describe the Design pattern modeling language, a notation supporting the specification of Design pattern solutions and their instantiation into UML design models. DPML uses a simple set of visual abstractions and readily lends itself to tool support. DPML Design pattern solution specifications are used to construct visual, formal specifications of Design patterns. DPML instantiation diagrams are used to link a Design pattern solution specification to instances of a UML model, indicating the roles played by different UML elements in the generic Design pattern solution. A prototype tool is described, together with an evaluation of the language and tool.

My contribution: Co-designed approach, wrote some of the software the approach based on, co-supervised Masters student, co-authored significant parts of the paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

4 Software Development and Testing using DSVLs and MDE

4.1 Supporting Multi-View Development for Mobile Applications

Barnett, S., Avazpour, I., Vasa, R., **Grundy, J.C.** Supporting Multi-View Development for Mobile Applications, *Journal of Computer Languages*, Volume 51, April 2019, Elsevier, Pages 88-96
DOI: 10.1016/j.cola.2019.02.001

Abstract: Interest in mobile application development has significantly increased. The need for rapid, iterative development coupled with the diversity of platforms, technologies and frameworks impacts on the productivity of developers. In this paper we propose a new approach and tool support, Rapid APPlication Tool (RAPPT), that enables rapid development of mobile applications. It employs Domain Specific Visual Languages and Modeling techniques to help developers define the characteristics of their applications using high level visual notations. Our approach also provides multiple views of the application to help developers have a better understanding of the different aspects of their application. Our user evaluation of RAPPT demonstrates positive feedback ranging from expert to novice developers.

My contribution: Co-developed main ideas for the research, co-supervised PhD student, wrote substantial part of the paper

Author pre-print available at: PDF

4.2 Specifying Model Transformations by Direct Manipulation using Concrete Visual Notations and Interactive Recommendations

Avazpour, I., **Grundy, J.C.**, Grunske, L. Specifying Model Transformations by Direct Manipulation using Concrete Visual Notations and Interactive Recommendations, *Journal of Visual Languages and Computing*, Volume 28, June 2015, Elsevier, pp 195–211.

DOI: 10.1016/j.jvlc.2015.02.005

Abstract: Model transformations are a crucial part of Model-Driven Engineering (MDE) technologies but are usually hard to specify and maintain for many engineers. Most current approaches use meta-model-driven transformation specification via textual scripting languages. These are often hard to specify, understand and maintain. We present a novel approach that instead allows domain experts to discover and specify transformation correspondences using concrete visualizations of example source and target models. From these example model correspondences, complex model transformation implementations are automatically generated. We also introduce a recommender system that helps domain experts and novice users find possible correspondences between large source and target model visualization elements. Correspondences are then specified by directly interacting with suggested recommendations or drag and drop of visual notational elements of source and target visualizations. We have implemented this approach in our prototype tool-set, CONVErT, and applied it to a variety of model transformation examples. Our evaluation of this approach includes a detailed user study of our tool and a quantitative analysis of the recommender system.

My contribution: Developed initial ideas for this research, co-designed approach, co-supervised PhD student, wrote substantial parts of paper, investigator for funding for this project from ARC

Author pre-print available at: PDF

4.3 SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions

Grundy, J.C., Cai, Y. and Liu, A. SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions, *Automated Software Engineering*, Kluwer Academic Publishers, vol. 12, no. 1, January 2005, pp. 5-39
DOI: 10.1023/B:AUSE.0000049207.62380.74

Abstract: Most distributed system specifications have performance benchmark requirements, for example the number of particular kinds of transactions per second required to be supported by the system. However, determining the likely eventual performance of complex distributed system architectures during their development is very challenging. We describe SoftArch/MTE, a software tool that allows software architects to sketch an outline of their proposed system architecture at a high level of abstraction. These descriptions include client requests, servers, server objects and object services, database servers and tables, and particular choices of middleware and database technologies. A fully-working implementation of this system is then automatically generated from this high-level architectural description. This implementation is deployed on multiple client and server machines and performance tests are then automatically run for this generated code. Performance test results are recorded, sent back to the SoftArch/MTE environment and are then displayed to the architect using graphs or by annotating the original high-level architectural diagrams. Architects may change performance parameters and architecture characteristics, comparing multiple test run results to determine the most suitable abstractions to refine to detailed designs for actual system implementation. Further tests may be run on refined architecture descriptions at any stage during system development. We demonstrate the utility of our approach and prototype tool, and the accuracy of our generated performance test-beds, for validating architectural choices during early system development.

My contribution: Developed initial ideas for this research, co-designed approach, wrote some of the software the approach based on, co-supervised Masters student, wrote majority of the paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

4.4 Realistic Load Testing of Web Applications

Draheim, D., **Grundy, J.C.**, Hosking, J.G., Lutteroth, C. and Weber, G. Realistic Load Testing of Web Applications, In *Proceedings of the 10th European Conference on Software Maintenance and Re-engineering*, Berlin, 22-24 March 2006, pp 57-70.

DOI: 10.1109/CSMR.2006.43

Abstract: We present a new approach for performing load testing of web applications by simulating realistic user behaviour with stochastic form-oriented analysis models. Realism in the simulation of user behaviour is necessary in order to achieve valid testing results. In contrast to many other user models, web site navigation and time delay are modelled stochastically. The models can be constructed from sample data and can take into account effects of session history on user behaviour and the existence of different categories of users. The approach is implemented in an existing architecture modelling and performance evaluation tool and is integrated with existing methods for forward and reverse engineering.

My contribution: Developed some of the key initial ideas for this research, co-designed approach, wrote the software the approach based on, wrote substantial parts of the paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

4.5 A Domain-Specific Visual Modeling Language for Testing Environment Emulation

Liu, J., **Grundy, J.C.**, Avazpour, I., Abdelrazek, M. A Domain-Specific Visual Modeling Language for Testing Environment Emulation, *2016 IEEE Symposium on Visual Languages and Human-Centric Computing*, Cambridge, UK, Sept 4-8 2016, IEEE Press, pp. 143 - 151.
DOI: 10.1109/VLHCC.2016.7739677

Abstract: Software integration testing plays an increasingly important role as the software industry has experienced a major change from isolated applications to highly distributed computing environments. Conducting integration testing is a challenging task because it is often very difficult to replicate a real enterprise environment. Emulating testing environment is one of the key solutions to this problem. However, existing specification-based emulation techniques require manual coding of their message processing engines, therefore incurring high development cost. In this paper, we present a suite of domain-specific visual modeling languages to describe emulated testing enviroements at a high abstraction level. Our solution allows domain experts to model a testing environment from abstract interface layers. These layer models are then transformed to runtime environment for application testing. Our user study shows that our visual languages are easy to use, yet with sufficient expressive power to model complex testing applications.

My contribution: Developed initial idea for the research, co-developed tool design, co-supervised PhD student, wrote substantial parts of paper, investigator on funding of the project from the Australian Research Council

Author pre-print available at: PDF

5 Software Process Management with DSVLs and MDE

5.1 Serendipity: integrated environment support for process modelling, enactment and work coordination

Grundy, J.C. and Hosking, J.G. Serendipity: integrated environment support for process modelling, enactment and work coordination, *Automated Software Engineering*, Vol. 5, No. 1, January 1998, Kluwer Academic Publishers, pp. 27-60.

DOI: 10.1023/A:1008606308460

Abstract: Large cooperative work systems require work coordination, context awareness and process modelling and enactment mechanisms to be effective. Support for process modelling and work coordination in such systems also needs to support informal aspects of work which are difficult to codify. Computer-Supported Cooperative Work (CSCW) facilities, such as inter-person communication and collaborative editing, also need to be well-integrated into both process-modelling tools and tools used to perform work. Serendipity is an environment which provides high-level, visual process modelling and event-handling languages, and diverse CSCW capabilities, and which can be integrated with a range of tools to coordinate cooperative work. This paper describes Serendipity's visual languages, support environment, architecture, and implementation, together with experience using the environment and integrating it with other environments.

My contribution: Developed initial ideas for the research, did majority of tool design, implemented and evaluated tool, wrote majority of the paper, investigator for funding for the project from FRST

Author pre-print available at: PDF

5.2 A decentralized architecture for software process modeling and enactment

Grundy, J.C. Hosking, J.G., Mugridge, W.B., Apperley, M.D. A decentralised architecture for software process modelling and enactment, IEEE Internet Computing, Vol. 2, No. 5, September/October 1998, IEEE CS Press, pp. 53-62.

DOI: 10.1109/4236.722231

Abstract: Many development teams, especially distributed teams, require process support to adequately coordinate their complex, distributed work practices. Process modeling and enactment tools have been developed to meet this requirement. The authors discuss the Serendipity-II process management environment which supports distributed process modeling and enactment for distributed software development projects. Serendipity-II is based on a decentralized architecture and uses Internet communication facilities.

My contribution: Developed initial ideas for the research, did majority of tool design, implemented and evaluated tool, wrote majority of the paper, investigator for funding for the project from FRST

Author pre-print available at: PDF

5.3 Collaboration-Based Cloud Computing Security Management Framework

Almorsy, M., **Grundy, J.C.** and Ibrahim, A. Collaboration-Based Cloud Computing Security Management Framework, In *Proceedings of 2011 IEEE International Conference on Cloud Computing (CLOUD 2011)*, Washington DC, USA on 4 July – 9 July, 2011, IEEE, pp. 364-371.
DOI: 10.1109/CLOUD.2011.9

Abstract: Although the cloud computing model is considered to be a very promising internet-based computing platform, it results in a loss of security control over the cloud-hosted assets. This is due to the outsourcing of enterprise IT assets hosted on third-party cloud computing platforms. Moreover, the lack of security constraints in the Service Level Agreements between the cloud providers and consumers results in a loss of trust as well. Obtaining a security certificate such as ISO 27000 or NIST-FISMA would help cloud providers improve consumers trust in their cloud platforms' security. However, such standards are still far from covering the full complexity of the cloud computing model. We introduce a new cloud security management framework based on aligning the FISMA standard to fit with the cloud computing model, enabling cloud providers and consumers to be security certified. Our framework is based on improving collaboration between cloud providers, service providers and service consumers in managing the security of the cloud platform and the hosted services. It is built on top of a number of security standards that assist in automating the security management process. We have developed a proof of concept of our framework using. NET and deployed it on a test bed cloud platform. We evaluated the framework by managing the security of a multi-tenant SaaS application exemplar.

My contribution: Developed initial ideas for the research, co-supervised the two PhD students, wrote substantial parts of paper

Author pre-print available at: PDF

5.4 DCTracVis: a system retrieving and visualizing traceability links between source code and documentation

Chen, X., Hosking, J.G., **Grundy, J.C.**, Amor, R., DCTracVis: a system retrieving and visualizing traceability links between source code and documentation, *Automated Software Engineering*, vol 25, no 4, 2018, Springer, pp. 703–741

DOI: 10.1007/s10515-018-0243-8

Abstract: It is well recognized that traceability links between software artifacts provide crucial support in comprehension, efficient development, and effective management of a software system. However, automated traceability systems to date have been faced with two major open research challenges: how to extract traceability links with both high precision and high recall, and how to efficiently visualize links for complex systems because of scalability and visual clutter issues. To overcome the two challenges, we designed and developed a traceability system, DCTracVis. This system employs an approach that combines three supporting techniques, regular expressions, key phrases, and clustering, with information retrieval (IR) models to improve the performance of automated traceability recovery between documents and source code. This combination approach takes advantage of the strengths of the three techniques to ameliorate limitations of IR models. Our experimental results show that our approach improves the performance of IR models, increases the precision of retrieved links, and recovers more correct links than IR alone. After having retrieved high-quality traceability links, DCTracVis then utilizes a new approach that combines treemap and hierarchical tree techniques to reduce visual clutter and to allow the visualization of the global structure of traces and a detailed overview of each trace, while still being highly scalable and interactive. Usability evaluation results show that our approach can effectively and efficiently help software developers comprehend, browse, and maintain large numbers of links.

My contribution: Developed initial ideas for this research, co-designed approach, co-supervised PhD student, co-authored significant parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

5.5 An End-to-End Model-based Approach to Support Big Data Analytics Development

Khalajzadeh, H., Simmons, A., Abdelrazek, M., **Grundy, J.C.**, Hosking, J.G., He, Q., An End-to-End Model-based Approach to Support Big Data Analytics Development, *Journal of Computer Languages*, Volume 58, June 2020, Elsevier

DOI: 10.1016/j.cola.2020.100964

Abstract: We present BiDaML 2.0, an integrated suite of visual languages and supporting tool to help multidisciplinary teams with the design of big data analytics solutions. BiDaML tool support provides a platform for efficiently producing BiDaML diagrams and facilitating their design, creation, report and code generation. We evaluated BiDaML using two types of evaluations, a theoretical analysis using the "physics of notations", and an empirical study with 1) a group of 12 target end-users and 2) five individual end-users. Participants mostly agreed that BiDaML was straightforward to understand/learn, and prefer BiDaML for supporting complex data analytics solution modeling than other modeling languages.

My contribution: Developed initial ideas for this research, co-designed approach, co-supervised the two post-doctoral fellows, co-authored significant parts of paper, lead investigator for funding for this project from ARC

Author pre-print available at: PDF

6 Human-centric DSVL Modelling and Collaboration

6.1 Experiences developing architectures for realising thin-client diagram editing tools

Grundy, J.C., Hosking, J.G., Cao, S., Zhao, D., Zhu, N., Tempero, E. and Stoeckle, H. Experiences developing architectures for realising thin-client diagram editing tools, *Software – Practice and Experience*, vol. 37, no.12, Wiley, October 2007, pp. 1245-1283

DOI: 10.1002/spe.803

Abstract: Diagram-centric applications such as software design tools, project planning tools and business process modelling tools are usually 'thick-client' applications running as stand-alone desktop applications. There are several advantages to providing such design tools as Web-based or even PDA- and mobile-phone-based applications. These include ease of access and upgrade, provision of collaborative work support and Web-based integration with other applications. However, building such thin-client diagram editing tools is very challenging. We have developed several thin-client diagram editing applications realized as a set of plug-in extensions to a meta-tool for visual design environment development. In this paper, we discuss key user interaction and software architecture issues, illustrate examples of interacting with our thin-client diagram editing tools, describe our design and implementation approaches, and present the results of several different evaluations of the resultant applications. Our experiences will be useful for those interested in developing their own thin-client diagram editing architectures and applications.

My contribution: Co-developed key ideas for this research, co-designed approach, co-supervised the two Masters students and the research assistant, wrote majority of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

6.2 Engineering plug-in software components to support collaborative work

Grundy, J.C. and Hosking, J.G. Engineering plug-in software components to support collaborative work, *Software - Practice and Experience*, Vol. 32, No. 10, August 2002, Wiley, 983-1013
DOI: 10.1002/spe.472

Abstract: Many software applications require co-operative work support, including collaborative editing, group awareness, versioning, messaging and automated notification and co-ordination agents. Most approaches hard-code such facilities into applications, with fixed functionality and limited ability to reuse groupware implementations. We describe our recent work in seamlessly adding such capabilities to component-based applications via a set of collaborative work-supporting plug-in software components. We describe a variety of applications of this technique, along with descriptions of the novel architecture, user interface adaptation and implementation techniques for the collaborative work-supporting components that we have developed. We report on our experiences to date with this method of supporting collaborative work enhancement of component-based systems, and discuss the advantages of our approach over conventional techniques.

My contribution: Developed initial ideas for this research, co-designed approach, wrote and evaluated the software, wrote majority of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

6.3 A generic approach to supporting diagram differencing and merging for collaborative design

Mehra, A., **Grundy, J.C.** and Hosking, J.G. A generic approach to supporting diagram differencing and merging for collaborative design, In *Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering*, Long Beach, California, Nov 7-11 2005, IEEE Press, pp. 204-213
DOI: 10.1145/1101908.1101940

Abstract: Differentiation tools enable team members to compare two or more text files, e.g. code or documentation, after change. Although a number of general-purpose differentiation tools exist for comparing text documents very few tools exist for comparing diagrams. We describe a new approach for realising visual differentiation in CASE tools via a set of plug-in components. We have added diagram version control, visual differentiation and merging support as component-based plug-ins to the Pounamu meta-CASE tool. The approach is generic across a wide variety of diagram types and has also been deployed with an Eclipse diagramming plug-in. We describe our approach's architecture, key design and implementation issues, illustrate feasibility of our approach via implementation of it as plug-in components and evaluate its effectiveness.

My contribution: Co-developed key ideas for this research, co-designed approach, co-supervised Masters student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

6.4 A 3D Business Metaphor for Program Visualization

Panas, T., Berrigan, R. and **Grundy, J.C.** A 3D Business Metaphor for Program Visualization, In *Proceedings of the 2003 Conference on Information Visualisation*, London, 16-18 July 2003, IEEE, pp. 314-319.
DOI: 10.1109/IV.2003.1217996

Abstract: Software development is difficult because software is complex, the software production process is complex and understanding of software systems is a challenge. We propose a 3D visual approach to depict software production cost related program information to support software maintenance. The information helps us to reduce software maintenance costs, to plan the use of personnel wisely, to appoint experts efficiently and to detect system problems early.

My contribution: Co-designed approach, supervised the visiting PhD student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

6.5 Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool

Grundy, J.C. and Hosking, J.G. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool, In *Proceedings of the 2007 IEEE/ACM International Conference on Software Engineering (ICSE'07)*, Minneapolis, USA, May 2007, IEEE CS Press, pp. 282-291.

DOI: 10.1109/ICSE.2007.81

Abstract: Software engineers often use hand-drawn diagrams as preliminary design artefacts and as annotations during reviews. We describe the addition of sketching support to a domain-specific visual language meta-tool enabling a wide range of diagram-based design tools to leverage this human-centric interaction support. Our approach allows visual design tools generated from high-level specifications to incorporate a range of sketching-based functionality including both eager and lazy recognition, moving from sketch to formalized content and back and using sketches for secondary annotation and collaborative design review. We illustrate the use of our sketching extension for an example domain-specific visual design tool and describe the architecture and implementation of the extension as a plug-in for our Eclipse-based meta-tool.

My contribution: Developed initial ideas for this research, co-designed approach, wrote and evaluated the software, wrote majority of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

7
End-User Applications of DSVLs and MDE

7.1 Domain-specific visual languages for specifying and generating data mapping systems

Grundy, J.C., Hosking, J.G., Amor, R., Mugridge, W.B., Li, M. Domain-specific visual languages for specifying and generating data mapping systems, *Journal of Visual Languages and Computing*, vol. 15, no. 3-4, June-August 2004, Elsevier, pp 243-263,
DOI: 10.1016/j.jvlc.2004.01.003

Abstract: Many application domains, including enterprise systems integration, health informatics and construction IT, require complex data to be transformed from one format to another. We have developed several tools to support specification and generation of such data mappings using domain-specific visual languages. We describe motivation for this work, challenges in developing visual mapping metaphors for different target users and problem domains, and illustrate using examples from several of our developed systems. We compare cognitive dimension-based evaluations of the different approaches and summarise the lessons we have learned.

My contribution: Co-developed initial ideas for much of this research, co-designed approaches, developed and evaluated some of the software, co-supervised Masters student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

7.2 A domain-specific visual language for report writing

Dantra, R., **Grundy, J.C.** and Hosking, J.G. A domain-specific visual language for report writing, In *Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing*, Cornwallis, Oregon, USA, Sept 20-24 2009, IEEE CS Press, pp 15-22.
DOI: 10.1109/VLHCC.2009.5295308

Abstract: Many domain specific textual languages have been developed for generating complex reports. These are challenging for novice users to learn, understand and use. We describe our work developing the prototype of a new visual language tool for a company to augment their textual report writing language. We describe key motivations for our visual language tool solution, its architecture, design and development using Microsoft DSL tools, and its evaluation by end-users.

My contribution: Co-developed key ideas for this research, co-designed approach, co-supervised Masters student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST and Technology NZ

Author pre-print available at: PDF

7.3 Supporting Scientists in Re-engineering Sequential Programs to Parallel Using Model-driven Engineering

Almorsy, M. and **Grundy, J.C.** Supporting Scientists in Re-engineering Sequential Programs to Parallel Using Model-driven Engineering, *1st ICSE Workshop on Software Engineering for High Performance Computing in Science (SE4HPCS 2015)*, Florence, Italy, May 19 2015, pp. 1-8.
DOI: 10.1109/SE4HPCS.2015.8

Abstract: Developing complex computational-intensive and data-intensive scientific applications requires effective utilization of the computational power of the available computing platforms including grids, clouds, clusters, multi-core and many-core processors, and graphical processing units (GPUs). However, scientists who need to leverage such platforms are usually not parallel or distributed programming experts. Thus, they face numerous challenges when implementing and porting their software-based experimental tools to such platforms. In this paper, we introduce a sequential-to-parallel engineering approach to help scientists in engineering their scientific applications. Our approach is based on capturing sequential program details, planned parallelization aspects, and program deployment details using a set of domain-specific visual languages (DSVLs). Then, using code generation, we generate the corresponding parallel program using necessary parallel and distributed programming models (MPI, Open CL, or Open MP). We summarize three case studies (matrix multiplication, N-Body simulation, and digital signal processing) to evaluate our approach.

My contribution: Came up with initial ideas for the research, co-designed the solution, supervised the post-doc who implemented solution, lead investigator on grant funding the project from the ARC

Author pre-print available at: PDF

7.4 A visual language and environment for enterprise system modelling and automation

Li, L, **Grundy, J.C.**, Hosking, J.G. A visual language and environment for enterprise system modelling and automation,*Journal of Visual Languages and Computing*, vol. 25, no. 4, April 2014, Elsevier, pp. 253-277
DOI: 10.1016/j.jvlc.2014.03.004

Abstract: Objective: We want to support enterprise service modelling and generation using a more end user-friendly metaphor than current approaches, which fail to scale to large organisations with key issues of "cobweb" and "labyrinth" problems and large numbers of hidden dependencies. Method: We present and evaluate an integrated visual approach for business process modelling using a novel tree-based overlay structure that effectively mitigate complexity problems. A tree-overlay based visual notation (EML) and its integrated support environment (MaramaEML) supplement and integrate with existing solutions. Complex business architectures are represented as service trees and business processes are modelled as process overlay sequences on the service trees. Results: MaramaEML integrates EML and BPMN to provide complementary, high-level business service modelling and supports automatic BPEL code generation from the graphical representations to realise web services implementing the specified processes. It facilitates generated service validation using an integrated LTSA checker and provides a distortion-based fisheye and zooming function to enhance complex diagram navigation. Evaluations of EML show its effectiveness. Conclusions: We have successfully developed and evaluated a novel tree-based metaphor for business process modelling and enterprise service generation. Practice implications: a more user-friendly modelling approach and support tool for business end users.

My contribution: Co-developed key ideas for this research, co-designed approach, co-supervised PhD student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

7.5 A suite of visual languages for model-driven development of statistical surveys and services

Kim, C.H., **Grundy, J.C.**, Hosking, J.G. A suite of visual languages for model-driven development of statistical surveys and services, *Journal of Visual Languages and Computing*, Elsevier, Vol 26, Feb 2015, pp 99–125

DOI: 10.1016/j.jvlc.2014.11.005

Abstract: Objective: To provide statistician end users with a visual language environment for complex statistical survey design and implementation. Methods: We have developed, in conjunction with professional statisticians, the Statistical Design Language (SDL), an integrated suite of visual languages aimed at supporting the process of designing statistical surveys, and its support environment, SDLTool. SDL comprises five diagrammatic notations: survey diagrams, data diagrams, technique diagrams, task diagrams and process diagrams. SDLTool provides an integrated environment supporting design, coordination, execution, sharing and publication of complex statistical survey techniques as web services. SDLTool allows association of model components with survey artefacts, including data sets, metadata, and statistical package analysis scripts, with the ability to execute elements of the survey design model to implement survey analysis. Results: We describe three evaluations of SDL and SDLTool: use of the notation by expert statistician to design and execute surveys; useability evaluation of the environment; and assessment of several generated statistical analysis web services. Conclusion: We have shown the effectiveness of SDLTool for supporting statistical survey design and implementation. Practice implications: We have developed a more effective approach to supporting statisticians in their survey design work.

My contribution: Co-developed key ideas for this research, co-designed approach, co-supervised Masters student, wrote substantial parts of paper, co-lead investigator for funding for this project from FRST

Author pre-print available at: PDF

7.6 Engineering Complex Data Integration and Harmonization Systems

Avazpour, I., **Grundy, J.C.**, Zhu, L., Engineering Complex Data Integration and Harmonization Systems, *Journal of Industrial Information Integration*, vol 16, Elsevier, Dec 2019
DOI: 10.1016/j.jii.2019.08.001

Abstract: Complex data transformation, aggregation and visualization problems are becoming increasingly common. These are needed in order to support improved business intelligence and end-user access to data. However, most such applications present very challenging software engineering problems including noisy data, diverse data formats and APIs, challenging data modeling and increasing demand for sophisticated visualization support. This paper describes a data integration, harmonization and visualization process and framework that we have been developing. We discuss our approach used to tackle complex data aggregation and harmonization problems and we demonstrate a set of information visualizations that can be developed from the harmonized data to make it usable for its target audience. We use a case study of Household Travel Survey data mapping, harmonization, aggregation and visualization to illustrate our approach. We summarize a set of lessons that we have learned from this industry-based software engineering experience. We hope these will be useful for others embarking on challenging data harmonization and integration problems. We also identify several key directions and needs for future research and practical support in this area.

My contribution: Developed many of the key research ideas, co-authored significant parts of the paper, co-investigator for funding for this project from ARC and co-investigator for funding from AURIN

Author pre-print available at: PDF

8 Future Directions

8.1 Towards Human-Centric Model-Driven Software Engineering

Grundy J.C., Khalajzadeh, H., McIntosh, J., Towards Human-Centric Model-Driven Software Engineering, *15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE2020)*, 5-6 May 2020, Prague, Chez Republic, SitePress, pp. 229-238.
DOI: 10.5220/0009806002290238

Abstract: Many current software systems suffer from a lack of consideration of the human differences between end users. This includes age, gender, language, culture, emotions, personality, education, physical and mental challenges, and so on. We describe our work looking to consider these characteristics by incorporation of human centric-issues throughout the model-driven engineering process lifecycle. We propose the use of the co-creational ”living lab” model to better collect human-centric issues in the software requirements. We focus on modelling these human-centric factors using domain-specific visual languages, themselves human-centric modelling artefacts. We describe work to incorporate these human-centric issues into model-driven engineering design models, and to support both code generation and run-time adaptation to different user human factors. We discuss continuous evaluation of such human-centric issues in the produced software and feedback of user reported defects to requirements a nd model refinement.

My contribution: Developed all of the key research ideas, wrote majority of the paper, sole investigator for funding for this project from the ARC

Author pre-print available at: PDF

THIS THESIS WAS TYPESET using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface.A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.