Complementing Metaheuristic Search with Higher Abstraction Techniques

Frank R. Burton and Simon Poulding
Department of Computer Science
University of York
YO10 5GH
UK
{frank,smp}@cs.york.ac.uk

***Abstract*—Search-Based Software Engineering and Model-Driven Engineering are both innovative approaches to software engineering. The premise of Search-Based Software Engineering is to reformulate engineering tasks as optimisation problems that can be solved using metaheuristic search techniques. Model-Driven Engineering aims to apply greater levels of abstraction to software engineering problems. In this paper, it is argued that these two approaches are complementary and that both research fields can make further progress by applying techniques from the other. We suggest ways in which synergies between the fields can be exploited.**

I. Introduction

Search-Based Software Engineering (SBSE) considers software engineering tasks to be optimisation problems [1], [2]. By applying automated and efficient metaheuristic search algorithms to the optimisation problem, SBSE is capable of solving engineering problems that may be intractable or too costly to solve by other means [3].

Model-Driven Engineering (MDE) aims to raise the abstraction level used in software engineering to models. The use of the higher abstraction is intended to permit software engineering to work on software development projects that have much greater complexity [4], [5].

In this paper, we argue that these two approaches to improving software engineering can benefit by applying techniques from the other, and provide second examples of how this synergy can be exploited. The first is the use of Domain Specific Modelling Languages (DSMLs) from MDE to create better models of software engineering problems and their solutions to which search may be applied. The second example is the use of metaheuristic search to help perform model-to-model transformations when going from the DSMLs containing the problem domain concepts to the DSMLs containing the implementation concepts.

II. Model-Driven Engineering

Model-Driven Engineering (MDE) aims to deal with complexity in software engineering through use of abstract descriptions of phenomena of interest –viz., models [4], [5]. As a brief introduction, we are going to explain MDE from two perspectives: the *technique* perspective and the *methodology* perspective; the first focuses on techniques applied in the use of MDE, whereas the second focuses on process (e.g., how techniques are orchestrated).

The first technique that many users of MDE will encounter is Domain Specific Modelling Languages (DSMLs). A DSML is a dedicated modelling language that captures the concepts particular to some domain. This concept is similar to a Domain Specific Language (DSL) and by using a text-to-model tool such as XText [6] it is possible to create a corresponding DSL for any DSML. Each DSML is defined by a meta-model, which captures the language's concept and relationships (this is analogous to an EBNF in the definition of textual languages). The meta-model expresses these domain concepts in an object-oriented style, in terms of the allowed classes and allowed relationships between the classes. Models created solely from the classes and relationships found within a meta-model are said to conform to that meta-model [7].

Another common technique used in MDE is model-to-model (M2M) transformation. These specify how to transform models that conform to one meta-model into new models that conform to a different meta-model[1]. The transformations are specified in terms of the meta-models and not the models; therefore it is required that the models to conform to the meta-models. Other common techniques include model-to-text transformation, automatic GUI generation [8], model merging tools and model validation tools.

The methodology perspective on MDE is best explained via an example. In Figure 1 we show a hypothetical MDE toolchain for translating C++ source code into Java source code. The toolchain is made up of several different MDE tools working together to perform the translation. The toolchain firstly relies on the C++ & Java meta-models. These meta-models need to be written to capture the concepts and relationships between concepts found in their respective language. In this case the meta-models could simply capture the abstract syntax for each language. The majority of the hard work in creating this toolchain is most likely to be found in creating the model-to-model transformation between the two meta-models since the transformation needs to map every concept in the C++ language to a concept within the Java language. Finally,

[1]This is, technically, a specific yet common instance of M2M; in general an M2M transformation may take many input models and produce many output models.

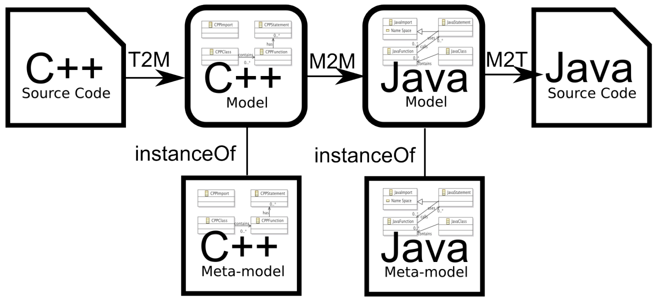

Fig. 1. C++ To Java MDE Toolchain

a text-to-model transformation is needed to read C++ source code into a model conforming to the C++ meta-model and a model-to-text transformation is needed to write out the content of the Java model to Java source code.

Normally, the intention is not to translate between two general purpose languages. The intention is that the first language in the toolchain is a dedicated DSML for the problem using the concepts found in the problem space and these will later on be translated by a model-to-model transformation to concepts that can be implemented.

III. Search-Based Software Engineering

To apply Search-Based Software Engineering (SBSE) to a software engineering problem requires three components: a representation for candidate solutions to the problem, a fitness metric that evaluates how 'good' a candidate solution is, and an optimisation method. The latter is typically a metaheuristic search algorithm: local search methods such as hill climbing and simulated annealing, or population-based methods such as genetic algorithms and ant colony optimisation. The search method uses the fitness values of 'current' candidate solutions to guide the derivation of even fitter candidates until a suitable solution to the problem is found.

Theory currently offers little guidance as to the choice of representation, fitness metric, and search method. Therefore such choices are often made empirically on a problem-by-problem basis.

A particular concern is the representation should be capable of encapsulating all the relevant features of possible solutions to the problem under consideration, but at the same time, the representation should remain amenable to search. For example, many search methods are often more effective when small changes in the search representation correspond to small changes in the semantic interpretation of that representation—a property known as locality [9]. Similarly, the fitness of a candidate solution is assessed in the context of a specific problem instance, and this requires a mechanism for representing the often complex problem instance appropriately.

We argue that MDE can assist with the choice of suitable problem and solution representations. Models enable the representation of highly structured, complex problems and solutions in a principled and consistent manner. MDE toolsets can facilitate the manipulation of the representation during the search process and provide mechanisms for users (and researchers) to interact with, and visualise, candidate solutions. By applying the representation at a higher level of abstraction, models may enable to empirical evaluation and selection of a suitable representation for a wider range of related problem domains, rather than on a problem-by-problem basis.

IV. Using Domain Specific Modelling Languages as a Problem Representation

A basic technique in MDE is the creation of DSMLs to represent the concepts and relationships found within a domain. A useful synergy between the two research fields is using DSMLs to create the problem representations of software engineering problems.

The advantage of this approach is that DSMLs are designed specifically for this purpose and are potentially able to capture the internal structure of software engineering problems better than the existing techniques within SBSE. This is best evidence by demonstration; we are going to show this on a software engineering problem taken from the SBSE literature: the Next Release Problem [10].

A typical problem formulation of the Next Release Problem given by Durillo et al [11] is as follows. There is a set C of customers each with a value to company c_i and a set R of requirements each with a cost r_j to implement. Each customer has a value of importance on each requirement that can be stored in a matrix v_{ij}. Requirements can have dependencies on other requirements. These requirement dependencies can be stored in a directed acyclic graph, however this part of the problem is normally dropped in SBSE work as a simplifying assumption. The solution to the problem is a binary decision vector saying if each requirement is to be implemented.

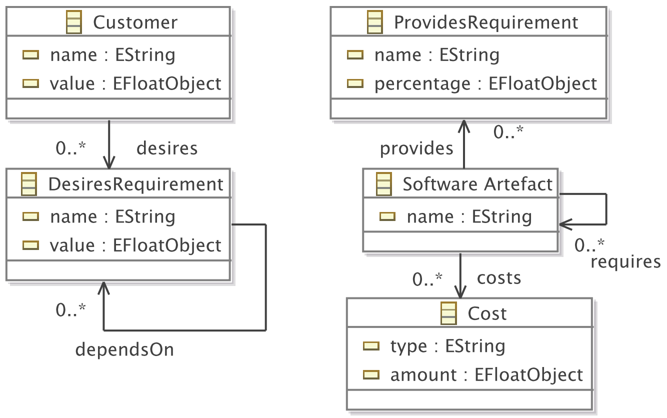

Fig. 2. Next Release Problem DSML

Now to contrast this with using DSMLs, a meta-model for the Next Release Problem is given in Figure 2. Each customer has a name and a value to the company and each customer desires requirements. This is represented by a relationship rather than a matrix. Each desired requirement has a name and a value to the customer and can depend on other requirements being fulfilled first. A simple observation made when modelling the domain is that each requirement is implemented by some software artefact that represents some amount of work being done to the software code base. Each software artefact has costs to implement that can be of various types including financial, man-hours, certifications costs, etc.

A solution to the Next Release Problem is given by another DSML defined as follows:

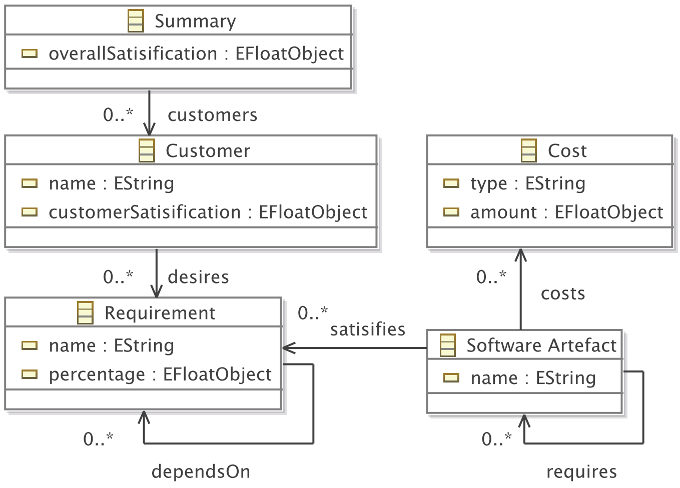

Fig. 3. Next Release Problem Solution DSML

This DSML contains the overall satisfaction for the company, each individual customers satisfaction, the wanted requirements and the chosen software artefacts. It also contains the dependencies between the requirements and software artefacts showing the solution's internal structure. The two problem representations are for the same problem and are subjectively similar. The first is defined in terms of vectors and matrices and the second is defined in terms of metamodels for DSMLs. The main question is how more expressive is the DSML representation over the vector and matrices representation. The first improvement in expressiveness is that partial requirement fulfilment (a challenge stated in [12]) can be supported by simply adding a single attribute called *Percentage* to the *ProvidesRequirement* class. This is much easier to do than in the vector and matrix case since the problem domain has being properly modelled. In the DSML representation, different software artefacts can produce the same requirements, this corresponds to supporting trade-offs in the software architecture. A more emergent property from using the DSML representation is that satisfying one requirement can change the costs of satisfying other requirements. This is because the software artefact used to satisfy a requirement can have dependent software artefacts that represent common work shared with other software artefacts and these can be used in satisfying other requirements at reduced cost. Requirement dependencies from a MDE perspective are a simple model query and therefore do not need to be removed as a simplifying assumption.

The point of this is that a simple DSML representation can produce a more expressive problem representation for a software engineering problem then can be done normally. Of course, with the use a lot of vectors and matrices the expressiveness of the DSML could be reproduced however unlike the DSML representation it would be highly complicated, tedious and error prone to do.

The other major benefit from the use of DSMLs comes from being able to exploit MDE tool support. The two DSMLs shown here with a small amount of extra annotation can be used to automatically generate GUI displays using a tool such as EuGENia [8]. In the NRP case, this means that the solutions can be visualised as directed acyclic graphs, containing both the fulfilled requirements and the selected software artefacts, giving valuable feedback on the solutions to the stakeholders.

Lastly, to show that search can be applied to this type of DSML representation, our previous work [13] applies MDE to the Multi-objective Next Release Problem [14] and all the DSMLs used here can be converted by simple model-to-model transformations into DSMLs used in that work.

V. Model-to-Model Transformations That Inherently Involve Search

An objective of MDE is that it will allow the stakeholders of a problem to define their problem using the domain concepts they naturally describe their problem in rather than using the implementation specific concepts found in a potential solution.

This can be done in MDE by providing DSMLs using the stakeholder domain concepts and by providing DSMLs using the implementation specific concepts. Then providing model-to-model transformations to convert the stakeholders domain concepts into the concepts found in the specific implementation technology. This shortens the distance between the problem and the implementation domain [15].

Changing concepts in the problem domain to concepts in the implementation domain is normally done by directly mapping concepts from one domain onto concepts in the other domain. However, if the problem domain is sufficiently complicated and abstract enough, it can easily become the case that such a direct mapping is no longer possible. A practical example of this can be found by looking into work by Thompson et al [16]. In their work they are attempting to design a mobile phone application that performs certain functions whilst meeting both performance and power constants. They have to make multiple design choices in going from their problem to their implementation domain such as the software architecture design, the choice of networking protocols and accuracy vs performance trade-offs in their implementing functions [16]. There is clearly no single mapping in their work from problem to implementation domain and not all mappings are viable due to the performance and power constants on the mobile phone hardware.

To demonstrate this point lets look at a basic example of how a problem level DSML can require search when transformed into an implementation level DSML. In Figure 4, an example DSML is given that describes an architecture containing connected nodes with processing power and data storage. This can be assumed to represent the layout of either a data centre or an embedded device. The DSML also describes a set of software processes to be allocated to nodes with the restriction that the nodes they are allocated to are all within a certain latency from each other.

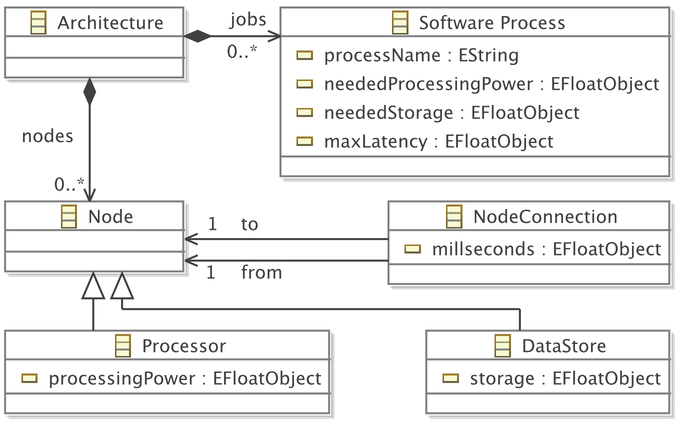
Architecture
jobs
0..*
Software Process
processName : EString
neededProcessingPower : EFloatObject
neededStorage : EFloatObject
maxLatency : EFloatObject
nodes
0..*
Node
1 to
NodeConnection
millseconds : EFloatObject
1 from
Processor
processingPower : EFloatObject
DataStore
storage : EFloatObject

Fig. 4. Problem Level DSML

The implementation level DSML in Figure 5 only needs to know the nodes to allocate to each of the software processes.

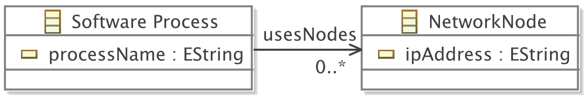

Fig. 5. Implementation Level DSML

Performing a model-to-model transformation from the problem level DSML to the implementation level DSML is no longer a case of simply directly mapping concepts in the problem level DSML to the concepts in the implementation level DSML. Depending on the model created from the problem level DSML there can be multiple or no corresponding models within the implementation level DSML that satisfy the problem level DSML. Each possible model in the implementation level DSML needs to be checked that it does not allocate less processing power or less data storage to a software process than is needed and all the allocation of nodes are within the required latency of the software process.

An effective way to solve problems like this is an open research problem. Hopefully, a technique can be developed for dealing with these types of model-to-model transformation where there are no direct mappings between the DSMLs but instead the problem is to search for a model in the implementation level DSML that meets the constraints and requirements specified by the problem stakeholders in the problem level DSML. This would almost certainly require the use of metaheuristic search techniques.

VI. Conclusions

In this position paper, we have argued that there are multiple places where SBSE and MDE can benefit from using techniques from the other. This includes the use of DSMLs to create better problem representations for software engineering problems and the open research area of using metaheuristic search to tackle complex model-to-model transformations where direct mappings cannot be applied.

Acknowledgements

This research was supported by MooD International and the EPSRC under the Large Scale Complex IT System project, EP/F501374/1.

References

[1] M. Harman and B. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.
[2] J. Clark, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proc. — Software*, vol. 150, no. 3, pp. 161–175, 2003.
[3] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering*. IEEE Computer Society, 2007, pp. 342–357.
[4] D. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, p. 25, 2006.
[5] J. Bzivin, "Model driven engineering: An emerging technical space," in *GTTSE 2006*, pp. 36–64.
[6] S. Efftinge and M. Völter, "oAW xText: A framework for textual DSLs," in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006.
[7] J. Bézivin, "In search of a basic principle for model driven engineering," *Novatica Journal, Special Issue*, vol. 5, no. 2, pp. 21–24, 2004.
[8] D. Kolovos, L. Rose, S. Abid, R. Paige, F. Polack, and G. Botterweck, "Taming emf and gmf using model transformation," *Model Driven Engineering Languages and Systems*, pp. 211–225, 2010.
[9] F. Rothlauf, *Representations for genetic and evolutionary algorithms*. Springer, 2006.
[10] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The Next Release Problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.
[11] J. Durillo, Y. Zhang, E. Alba, and A. Nebro, "A study of the multi-objective next release problem," in *Search Based Software Engineering, 1st Int. Symp. on*, 2009, pp. 49–58.
[12] Y. Zhang, A. Finkelstein, and M. Harman, "Search based requirements optimisation: Existing work and challenges," *Requirements Engineering: Foundation for Software Quality*, pp. 88–94, 2008.
[13] F. Burton, R. Paige, L. Rose, D. Kolovos, S. Poulding, and S. Smith, "Solving acquisition problems using model-driven engineering," *Modelling Foundations and Applications*, pp. 428–443, 2012.
[14] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1129–1137.
[15] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.
[16] C. Thompson, J. White, B. Dougherty, and D. Schmidt, "Optimizing mobile application performance with model–driven engineering," *Software Technologies for Embedded and Ubiquitous Systems*, pp. 36–46, 2009.