Revealing Complexity through Domain-Specific Modelling and Analysis

Richard F. Paige[1], Phillip J. Brooke[2], Xiaocheng Ge[1],
Christopher D.S. Power[1], Frank R. Burton[1], and Simon Poulding[1]

[1] Department of Computer Science, University of York, York, YO10 5GH, UK
{paige,xchge,cpower,frank,smp}@york.ac.uk
[2] School of Computing, Teesside University, Middlesbrough, TS1 3BA, UK
pjb@scm.tees.ac.uk

Abstract. Complex systems exhibit emergent behaviour. The explanations for this explicit emergent behaviour are often difficult to identify, and usually require understanding of significant parts of system structure and component behaviour to interpret. We present ongoing work on a set of techniques, based on Model-Driven Engineering principles and practices, for helping to reveal explanations for system complexity. We outline the techniques abstractly, and then illustrate parts of them with three examples from the health care, system security and Through-Life Capability Management domains.

1 Introduction

Complex systems exhibit behaviour that is not directly predictable or traceable from the behaviour of their constituent components. For large-scale complex IT systems (LSCITS), complexity arises due to combinations of attributes: increasing system size; increasing rates of changing requirements that must be addressed; increasing numbers and types of stakeholders. The roots of complexity are typically *hidden*, and often these do not become apparent during the system engineering phase. When tested or deployed, emergent behaviour readily becomes apparent, as illustrated by some recent failures, e.g., the Mars Polar Lander.

The LSCITS initiative[1] has argued that existing reductionist approaches to engineering complex systems do not address essential complexity. Instead, new approaches, which allow engineers to understand and control complexity, are needed.

Model-Driven Engineering (MDE) is a systems engineering approach that promotes models —abstract descriptions of phenomena of interest— to first-class engineering artefacts. The ‘engineering’ in MDE focuses on constructing, analysing and manipulating models in rigorous ways, particularly with automated tools that support required tasks. MDE requires organisations to invest substantial effort into constructing models and supporting the modelling process.

[1] www.lscits.org

Models range from design artefacts (e.g., in UML, SysML, or bespoke domain-specific languages), to requirements, to documentation and reports, to what-if models, and beyond. The tasks that are typically applied to models in MDE include *calculations* (e.g., calculating a representative response time based on a model of a network), *transformations* (e.g., transforming a system model into a detailed design model), *comparisons* (e.g., comparing two versions of a model to identify the changes made during a configuration management process) and more. Numerous tools now exist to support MDE tasks, and MDE is now practiced in industry, and on very large complex systems engineering projects and problems.

MDE, as its name suggests, is most typically used for engineering complex systems. It is used less frequently for *understanding* systems, *explaining* systems, and *revealing inherent complexity* in systems. It is the latter that is the focus of this paper.

We contribute towards an approach for modelling complex systems, designed to help engineers better understand — and ultimately control — complexity. The approach, based on MDE principles, practices and tools, focuses on *domain modelling* and use of bespoke *simulation and analysis* of said models. The analyses and simulations are designed to help disentangle relationships – both structural and behavioural – between entities and components of complex systems. They are not intended to *hide* complexity but to reveal just enough detail for engineers to understand specific interactions between components. This, in turn, may help engineers understand (1) the impact of engineering decisions on overall system complexity; and (2) how to better manage and control complexity in a deployed system. The analyses and simulations are produced via use of MDE transformations, which, as a side-effect, produce traceability information that can be used to connect the results of analysis/simulation back to the models that capture essential characteristics of the problem domain. This "closing of the loop" can thus help to support evolution and change of models, as a deeper understanding of complexity develops.

The rest of the paper is structured as follows. We start with an abstract overview of the general approach for modelling complex systems (Section 2), which is inspired by approaches to domain-specific language design. Then, in Section 3, we present three modelling examples that focus on revealing different kinds of complexity. The examples are all related to LSCITS socio-technical problems and concepts; while not all LSCITS must exhibit socio-technical characteristics, many do, and in turn the complexity of such systems is significant (in part because of the increased number of events and nondeterminism that arises with humans and organisations involved). In Section 4 we provide pointers and concrete ideas for future work.

2 Domain-Specific Modelling Approach

The approach we take to modelling complex systems is inspired by domain-specific language approaches to MDE. The general strategy is to model domain concepts of interest (e.g., components, behaviours, requirements) in a language specifically developed for the domain. As such, the semantic gap between the domain and the language used for expressing that domain should be reduced (and therefore the complexity of the problem domain is the focus of the modelling problem and the engineers). After constructing a suitable domain-specific language (with supporting tools for manipulating models), task-specific simulations and analyses are specified and developed. These analyses, which are designed to make explicit views on the behaviour and structures of the complex system, are developed and encoded using MDE operations (e.g., the transformations or comparisons mentioned earlier). The analyses can then be used to ask questions of the complex system model, e.g., what is the source of a set of events, how do events get transformed through parts of the system, what is the cost associated with processing particular types of events. In some cases, these questions may generate a *view* on the original system model; in other cases, it may elaborate part of the system model that was previously left abstract or obscured from an engineer. It is, ultimately, the engineer's responsibility to decide how to make use of the elaborated system models that are produced as a result of applying the analyses. Importantly, by using MDE operations (particularly model-to-model transformations or model-to-text transformations) to implement the analyses and simulations, we obtain full traceability (via a trace model) to original (domain) models. As such, when an analysis is executed, its results (e.g., model checking counter-examples, simulation outputs) can be traced back to important modelling elements, thus helping to explain the output – and complexity – in a more precise and analytic way. The traceability information can also be used to elaborate or refine the source models to take into account any improvements that may be needed as a result of the analysis or simulation.

In summary, the general modelling approach is as follows.

1. Identify domain concepts and relationships of interest; this in turn identifies and clarifies the scope of the DSL to be used, and of the modelling that is to take place.
2. Encode domain concepts and relationships in a domain-specific language, including the language's abstract and concrete syntax. Ideally, provide an editor for the language; the tool support will help to reduce errors and increase trust in the models.
3. Encode analyses of interest, by transforming (via model transformations of different kinds) domain-specific models into models amenable to analysis. The transformations should effectively produce views or elaborations of the original model, e.g., by simulating algorithms or calculating values.
4. Present the results of analysis to engineers, ideally in a perspective similar to the original editor (where feasible). Exploit the traceability information generated from running the transformations to enable this (e.g., following the approach of [6]).
5. Exploit the same traceability information to refactor and evolve the domain-specific language, where needed.

We now illustrate parts of this approach with several small examples.

3 Illustrations

Our illustrations focus on constructing and analysing domain-specific languages, and thereafter models of complex systems, to help to reveal explanations for their emergent behaviour. We present three examples.

1. A process modelling example wherein complex behaviours (particularly exceptional behaviours) are revealed through the modelling approach;
2. A secure systems example wherein complex inter-relationships between actors are revealed through the modelling approach; and
3. A Through-Life Capability Management class of problem, wherein LSCITS must be obtained to satisfy disparate goals, where there are multiple (optimal) ways in which the goals can be satisfied.

3.1 Failures in Healthcare Processes

Healthcare is a complex system [1, 15, 16]. Here we use term *'system'* generically, to indicate a conceptual entity whose components interact in rich and fine-grained ways because they continually affect each other and operate towards a common sense of purpose. There are many factors which may contribute to the complexity of healthcare systems; the focus of our example is the structures of healthcare systems, and the patterns created by the interaction of their components. To describe these, a business process model can be constructed.

A *business process* [7] is a collection of tasks designed to produce a specific output (e.g., a product or service). A *business process model* defines a specific ordering of tasks across time and space. A business process may have a hierarchical structure, i.e., tasks may include or trigger further business processes. As such, they may also be recursive, i.e., a business process may invoke itself. A task is normally made up of activities (carried out by actors), resources (which support activities) and constraints.

A business process consists of a *normal* set of tasks and constraints, as well as *exceptional* tasks and constraints, designed to deal with situations outside the norm. Exceptions in business processes have been widely studied, including work on both identifying exceptions as well as exception handling. The traditional approach is to anticipate beforehand, and ideally exhaustively, all exceptional conditions that may arise, and augment business process models with the additional conditional elements that represent exception handling activities. This, however, makes business process models more complicated, and introduces complexity through interactions between normal conditions and so-called *exceptional* conditions, which model exceptional behaviour. This complexity is normally hidden from the modeller, until the process model is reviewed and validated. Even then, for large business process models, it is easy to overlook or misunderstand the complex interactions between normal tasks and exceptional tasks.

We applied the approach from Section 2 to the problem of understanding exceptional behaviour in complex business process models. The key concept that we chose to focus on in developing a DSL (according to the process of Section 2) was that of exceptional conditions. We chose to treat exceptional conditions (and hence, tasks that were executed following exceptional conditions) as *failures*, and carry out a *failure propagation analysis* on a business process model. This in turn would illustrate (a) the impact of an exception on the overall business process model; and (b) the sensitivity of the business process model to exceptional conditions (namely, by illustrating the types of exception that tended to consolidate in specific parts of the business process model). In a nutshell, our DSL would include many familiar concepts for business modelling, but would be targeted at modelling of exceptional behaviour.

In terms of the approach of Section 2, we commenced by producing a small DSL for modelling business processes; this was effectively a subset of the Business Process Description Metamodel (BPDM), tailored for the example we were planning to use for experiments. However, the subset of BPDM was extended to include concepts for modelling the possible *failure modes* of the business process. This requires elaboration, because it is non-trivial and also because it is the key modelling challenge associated with this problem.

The problem we are interested in solving is identifying the failure modes of a business process. Our claim is that most process faults have a direct association with a task in the process — i.e., we assume that in the majority of cases the failure of a business process is initiated by a failure in a task of the process, and the failures introduced by individual tasks propagate through the process until the process delivers failure behaviour. So, to start analysing the behaviour of a business process in the presence of exceptions, we must first identify the possible failure modes of each task in the business process.

Let us illustrate this with a small healthcare system example. Consider Figure 1, which shows parts of a prototypical healthcare (business) process. We focus particularly Task 15 (Investigations); this task consists of one activity. Briefly, this task is as follows.

> "Patients who have had a suspected stroke should have specialist assessment and investigation within 24 hours of onset of symptoms and be transferred to the acute stroke unit."

The activity associated with this task is to investigate the condition of the suspected stroke patient. Resources available include acute stroke/TIA specialists, required medical documents (e.g., results of early assessment, patient medical history), and the availability of an acute stroke unit. There is also a constraint to carry out this activity within 24 hours of the onset of symptoms.

Given the activities, resources and constraints, we can now identify failure modes, i.e., the ways in which the business process fails to deliver its service. In effect, failure modes identify mismatches between expected outcomes and desired qualities. Based on an analysis of the literature [8, 9, 14] and a domain analysis of health care [2, 5], we argue that the qualities of a business process have the following five dimensions:

- **completeness**, whether the outcome of the process is complete;
- **validity**, whether the outcome of the process meets its requirements;

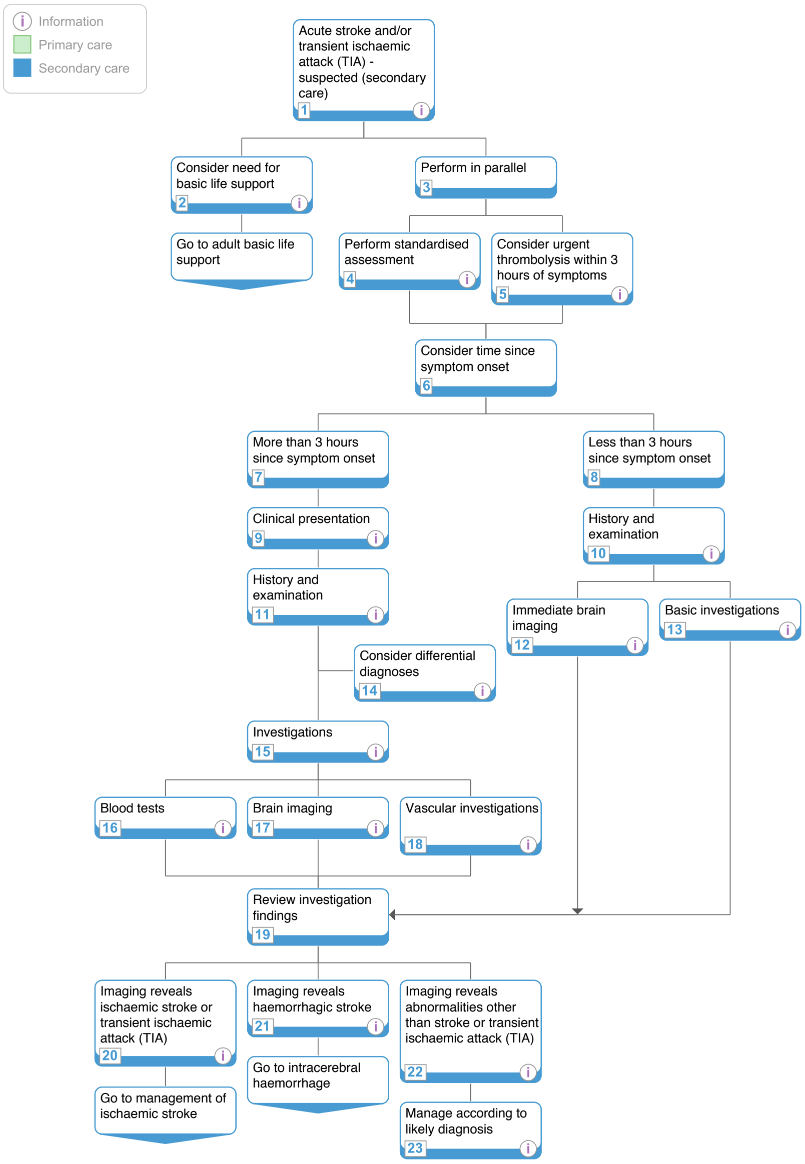

Information
Primary care
Secondary care
Acute stroke and/or transient ischaemic attack (TIA) - suspected (secondary care) 1
Consider need for basic life support 2
Perform in parallel 3
Go to adult basic life support
Perform standardised assessment 4
Consider urgent thrombolysis within 3 hours of symptoms 5
Consider time since symptom onset 6
More than 3 hours since symptom onset 7
Less than 3 hours since symptom onset 8
Clinical presentation 9
History and examination 10
History and examination 11
Immediate brain imaging 12
Basic investigations 13
Consider differential diagnoses 14
Investigations 15
Blood tests 16
Brain imaging 17
Vascular investigations 18
Review investigation findings 19
Imaging reveals ischaemic stroke or transient ischaemic attack (TIA) 20
Imaging reveals haemorrhagic stroke 21
Imaging reveals abnormalities other than stroke or transient ischaemic attack (TIA) 22
Go to management of ischaemic stroke
Go to intracerebral haemorrhage
Manage according to likely diagnosis 23

Fig. 1. Acute Stroke/TIA - suspected (secondary care) (derived from [13])

- **consistency**, whether the outcome of the process is consistent whenever the process is executed;
- **timeliness**, whether the outcome of the process is generated on time; and
- **adequacy**, whether the outcome of the process is fit-for-purpose.

In each quality dimension, we can determine the possible failure modes of each activity, resource and constraint of a task. For example, considering the human resources of Task 15, the possible failure modes might be:

- *incomplete*, the human resources assigned to the task are insufficient. For instance, the task requests a specialist and a nurse to assist the investigation, but in reality, there is no nurse available when the task is performed.
- *Late*, the team of specialists are late in carrying out the task.
- *Inadequate*, the task should be performed by a stroke specialist, but in reality, it is done by a different specialist.

Process models expressed in our DSL can be annotated with tokens indicating the type of potential failure for each kind of activity, resource or constraint. Importantly, this failure modelling is carried out on the *domain model*, in the vocabulary of process models. Additionally, these failures can then be elaborated to capture the *propagation* and *transformation* behaviour of activities, resources and constraints. For example, an activity may receive faulty outcomes from preceding activities. The new activity may compound the failure (by transforming it into a new failure), may simply propagate the failure, or may mitigate the failure in an appropriate way. Our modelling approach includes mechanisms for modelling such failure behaviours.

However, the (annotated) process model constructed as above only captures the component failure behaviour; it says nothing about the whole-system (emergent) failure behaviour. To understand and reveal this, we *transform* the annotated process model to a new model amenable to failure propagation analysis. The details of this are outside the scope of this paper, but the overall approach used is consistent with what was presented in Section 2: a model transformation maps the annotated process model into an interval timed coloured Petri Net, which can then be simulated to reveal the overall business process model failure behaviour. The overall algorithms used for simulation are inspired by those in the Failure Propagation and Transformation Calculus [17]. The transformation itself is not particularly complicated, requiring only several hundred lines of code (written in a specialised model transformation language); in part, the transformation is straightforward because of some similar structures arising in both the DSL for process modelling, and in the dialect of Petri nets.

We applied this approach to the health care example illustrated in Figure 1. This process includes sixteen key activities[2] We focused on one particular failure

[2] Excluding the final activity —i.e., determining whether the patient has experienced an acute stroke, TIA, haemorrhagic stroke, or other attack— because we are interested in analysing this example to identify key potential failures leading to incorrect judgement.

that was identified during the modelling stages: *incorrect judgement after activities related to reviewing the investigating results* (activity A19 in Figure 1). As a result of elaborating the business process model via transformation to a failure propagation model, we identified a number of reasons that could lead to this particular failure. Perhaps more importantly, we identified a number of 'vulnerable' activities that may be the ultimate cause of such a failure. These include A5 (judgement of application of urgent thrombolysis within 3 hours), A9 (clinical presentation), A10/11 (review history of health care and examinations) and A17 (brain imaging). During the analysis, the conclusion was that the quality of the services provided by these activities rely substantially on the skills of the personnel carrying them out. Thus, we can argue that to better mitigate such a failure, improvements should be concentrated on improving the skills and experience of the personnel carrying out these critical activities. These 'critical' activities were identified by expert analysis, assisted and supported by the traceability information derived from the model transformation mapping the process model into Petri nets. The results of the Petri net analysis was (manually) traced back, via the underlying trace model, to the activities of the process model. In this particular situation, the reflection of analysis results on the original model was not done automatically, but expert analysis was assisted by automated tools.

The modelling and analysis approach helped in this example to reveal inherent complexity due to failure behaviour. In turn, the analysis helped us identify critical parts of business processes, and how these parts contribute to overall failures and exceptional conditions.

Our second example uses a similar approach to modelling, but for different effect: to understand the complex inter-relationships that exist between actors.

3.2 Secure Transaction Problem

The recently cancelled UK National ID card programme was intended to introduce new large-scale complex IT systems, with particularly significant socio-technical concerns; similar (though not as controversial and pervasive) ID card systems exist in other countries world-wide. The intention for the UK ID card programme was to use a unified ID card, with secure transactions, to support many identification scenarios now currently supported by a collection of mechanisms (e.g., non-biometric cards). Many of the intended scenarios, and secure transactions, involve both network and cryptographic protocols and the interactions of different humans participants in a real-world environment. Modelling these scenarios is complex, in part due to the fuzziness of some of the properties involved. In this illustration, we use our modelling approach to unveil complexity in these scenarios, and by doing so address two other key contributors to complexity: *nondeterminism* and *unpredictability.*

Consider the following concrete example.

Alice has an identification card issued by her national government. She applies, in person, for a tax certificate for her car. The clerk takes Alice's identification card and tests it for various properties, related to Alice's *identifying features (name, photograph, biometrics, address), which are needed to ensure that a tax certificate is properly issued to the person who should hold it. Specifically, the clerk at the tax office uses the identification card as a means of verifying any claims that Alice makes about herself (e.g., related to name and age).*

Such scenarios occur widely. They appear to be simple (i.e., can Alice be issued a tax certificate for her car?) yet have significant inherent complexity. This complexity arises for at least two reasons.

1. decisions taken within the scenario are not based purely on boolean values (i.e., they are taken or not taken). Nor are they taken based on a probability (e.g., 75% of the time the certificate is issued, 25% of the time it is not[3]). Instead, decisions are taken based on probability distributions. A seemingly binary decision (*e.g.*, can Alice acquire a tax certificate?) is underpinned by a number of fuzzy decisions (*e.g.*, how accurately does the ID card presented identify Alice? To what extent does the clerk believe that the ID card represents Alice?).
2. the scenarios are inherently *generic* and must be *configured* to be fully analysable. For example, different mechanisms of different quality can be used within the scenario to support Alice demonstrating that she is at least 18. Such instantiations must take into account the scenario context (e.g., how effective will biometric scanners be in this particular building during these particular hours), and must be taken into account in any analysis process.

At different steps of the scenario (*e.g.*, when Alice enters the tax office, when the clerk first observes Alice), decisions are taken (*e.g.*, Alice is over 18, Alice's ID is valid). A simplistic view would be that these decisions are binary, and one branch of the decision structure (*e.g.*, Alice is over 18) is taken with probability p, and the other (*e.g.*, Alice is not over 18) taken with probability $1 - p$. Such a structure would naturally lend itself to state exploration and property checking using a probabilistic model checker, *e.g.*, PRISM [10] or a probabilistic process algebra, *e.g.*, pCSP [11]. A different view is that decisions are taken based on probability *distributions*. When Alice enters the office, the clerk forms a belief about Alice's age; conceptually, this belief is not that Alice is or is not 18, but that it can be represented as a distribution *around* Alice's age and represents the clerk's ability to perceive a person's age. Similarly, when Alice presents an ID card, there is a probability distribution associated with the clerk's perception of this ID card (*e.g.*, does the photo accurately depict Alice; does it appear that the card has been tampered with). This distribution is in turn dependent on the clerk's ability to perceive if a card has been tampered with. Later distributions may depend on an earlier perception, such as the ability of a third party to forge or modify a card.

Modelling scenarios like these as a set of binary decisions with real-valued probabilities hides much of the complexity of how decisions are made. By hiding

[3] Though post facto analysis of many instances of the scenario could lead to sufficient data about the probability of a decision being made.

this complexity, we make it difficult to understand the *impact* of changes in the scenario, *e.g.*, the use of ID cards that are claimed to provide fewer false positives. This in turn makes it more difficult to analyse scenarios and carry out *what-if* analyses, particularly to establish the benefit of improvements in the mechanisms used in specific configurations of scenarios.

We have applied the modelling approach of Section 2 to scenarios like the tax certificate one above. In [3] we identified the concepts for and the abstract syntax of a domain-specific language for modelling configurable scenarios. The DSL includes concepts for modelling actors (e.g., agents, subjects, cheats), mechanisms (e.g., cards, card readers), locations, and events (e.g., test a card, validate a transaction). As a result, the DSL contains slots for carrying out configurations, for example, to introduce particular cards or particular events.

Transformations have then been defined to map models written in the DSL to input to a probabilistic state exploration tool. This would allow the use of probabilistic model checkers such as PRISM, or state exploration tools like Casper, FDR2, or ProBE to do exhaustive runs of specific scenarios, or to work in an interactive mode. These would support probability distributions related to events, but it would not allow the manipulation of those distributions. In order to provide support such manipulations, we transform models to to serve as input into a bespoke state exploration tool. This tool allows us to exhaustively explore scenario instances, and as result obtain a better understanding of the inherent complexity of such scenarios. The transformation – in this case, a model-to-text transformation – is, like the previous example, not particularly complex. Much of the complexity arises with supporting the configuration of the scenario, e.g., to introduce cards with particular capabilities. In this particular case, because the number of configurations we needed to work with was relatively small – just a few different types of cards and around 10 different events – we configured the models interactively. In other words, we asked the end-user to select which configurations they wanted to execute (as the model-to-text transformation was running). For larger configurations we would likely re-implement the transformation to take two models as input – one for the scenario model, the other encoding configuration parameters. It is worth noting that the transformation tools we used support transformations involving multiple input models.

What does the elaboration of complex scenario models actually provide to us? For one, it allows us to determine the impact of mechanisms on outcome. For example, we might ask: what, truly, is the impact of biometric ID cards on identification scenarios such as this? Before biometric ID cards, successful completion of the scenario (i.e., offering Alice a tax certificate when she was legitimately entitled to one; and refusing when she shouldn't receive one) depended predominantly on the beliefs formed by the clerk. Do biometric ID cards offer any improvement? The work in [3] included a number of instances of similar scenarios and as a result made the following observations:

– use of biometric ID cards led to small, though noticeable increases in true positives and small, but noticeable reductions in false negatives (when compared with using no cards at all);

- when compared with alternative mechanisms (e.g., non-biometric ID cards), biometric ID cards made very small improvements (on the order of 1%).

While many more scenarios and experiments would need to be carried out to better bound the overall impact, it is arguable whether more expensive biometric ID cards add value when compared with less sophisticated, less accurate mechanisms of identification.

With this example, we were able to use modelling and overall approach to make explicit the relationships between actors in a scenario, and to tease out the semantics underpinning these relationships — i.e., that they were based on belief models instead of discrete probabilities. In constructing our domain-specific language and our analysis tools, we have configured the approach in a way that allows experiment with different semantics. For example, we could experiment with Bayesian models underpinning the belief systems set up in the scenarios, without having to change the overall DSL or the modelling approach — only the back-end state exploration or modelling tool would need to be changed.

3.3 Through-Life Capability Management

Large organisations typically manage LSCITS projects in terms of procuring the equipment, facilities, personnel and software that they require. Procurement exercises are invariably large and complex, requiring different entities (e.g., computers, lifeboats) to be compared in different ways.

It has been recognised by numerous government departments and large organisations that management of projects in terms of *capability* could potentially offer increased efficiencies. In classical procurement, an organisation may have defined a problem in terms of a *requirement* for a piece of equipment (e.g., a lifeboat); said equipment would then be acquired. In capability-based management, the problem would be defined in terms of a capability of *rescuing people and equipment at sea*, and a range of possible solutions could be considered. In effect, capability-based management means moving from defining problems in terms of concrete solutions, to defining problems in terms of abstract needs.

The challenge associated with capability-based management is that it opens up the solution space: a problem is now specified in terms of abstract needs, which can be met in a number of ways. Each potential solution must be identified and costed. Moreover, potential solutions are themselves capabilities, and are associated with their own individual requirements. To put it another way, solutions come with new problems, and as such understanding how solutions interact is difficult. Modelling – particularly with domain-specific languages – can help provide engineers with means to manage the solution space.

Through-Life Capability Management (TLCM) [12] is a specific instance of capability-based management, which also takes into account long-lived capabilities, e.g., logistics, personnel, training, deployment and decommissioning. It is of particular interest to military organisations (e.g., the UK Ministry of Defense). An example of a TLCM problem arises with acquisition of long-lived assets such as aircraft carriers: not only must the equipment (the carrier) be acquired, but so must airplanes, personnel, and fixtures and fittings. Moreover, personnel must be trained (at the right time) so that the aircraft carrier can be deployed on-time and on-budget. Furthermore, personnel and other equipment must be moved off the aircraft carrier in time for it to be decommissioned. There are, of course, multiple ways in which acquisition of these various assets can be implemented. In general, TLCM problems exhibit the following characteristics:

– They exhibit multiple objectives (e.g., protect a country, minimise costs).
– To solve them requires heterogeneous tradeoffs (e.g., between training and equipment). This is sometimes called the 'Apples and Wednesdays' problem, as it requires comparing and deciding between very different things.
– Different solutions may be optimal or near-optimal at different times.
– There may be multiple solutions, and understanding how each solution contributes to solving the overall problem may be difficult.

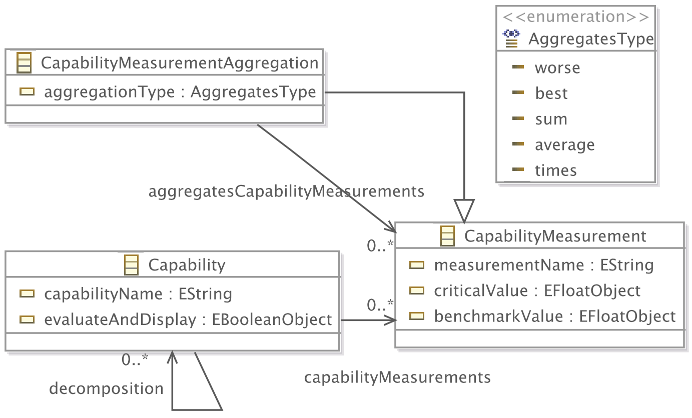
We applied the approach in Section 2 in a number of experiments that constructed DSLs for modelling TLCM problems. At the same time, we developed MDE techniques for calculating optimal and near-optimal solutions to these problems. The full details of our approach to calculating optimal solutions (based on search-based software engineering techniques) have recently been presented in [4], including a small example. We give a concise overview of our modelling approach here.

There are two basic modelling problems associated with TLCM.

1. Modelling TLCM problems in a domain-specific way, which reveals the complexity of the problem but says nothing about potential solutions. In this manner, we use abstraction and domain-specific concepts to allow us to focus on the challenges of *understanding the problem*, instead of concerning ourselves with characteristics of the solution space.
2. Modelling *components*, which are artefacts that can be used to (partially or fully) satisfy the goals inherent in a TLCM problem. In effect, the components are the basic building blocks of the solution space, and when components are – perhaps in combination or aggregation – used to satisfy goals, a solution to the overall TLCM problem is derived.

To support this, we used the approach in Section 2 and designed and implemented two domain-specific languages: one for modelling TLCM goals, and the other for modelling components. In the first DSL, goals can be decomposed further; moreover, each goal is itself associated with an arbitrary number of ways in which its satisfaction can be determined. For example, some goals may be satisfied in a probabilistic way (e.g., 100%, 75%); others may be satisfied in a measurable but discrete way (e.g., best, worst). The overall abstract syntax of the problem/goal DSL is illustrated in Fig. 2. We use UML's concrete syntax to represent the abstract syntax; in practice, we usually implement the abstract syntax using Ecore and the Eclipse Modelling Framework (EMF)[4]. For reasons of compliance with TLCM vocabulary, a goal is called a Capability in this diagram.

[4] `www.eclipse.org/emf`



<<enumeration>>
AggregatesType
worse
best
sum
average
times
CapabilityMeasurementAggregation
aggregationType : AggregatesType
aggregatesCapabilityMeasurements
0..*
CapabilityMeasurement
measurementName : EString
criticalValue : EFloatObject
benchmarkValue : EFloatObject
Capability
capabilityName : EString
evaluateAndDisplay : EBooleanObject
0..*
0..*
decomposition
capabilityMeasurements

Fig. 2. Capability and goal DSL

The second DSL is used for modelling components that can be used to satisfy goals (capabilities). Components are associated with a particular cost, as well as a description of the capabilities they provide, and the capabilities they require (it is typical of TLCM problems that acquired capabilities require other assets to also be acquired). This is illustrated in Fig. 3

The first DSL is used by the TLCM expert who wishes to model and understand a TLCM problem; the second DSL may be used by a procurement expert who understands what assets are available, what they may cost, and what is required to acquire and deploy them. Models expressed in the two DSLs are then input in to an optimisation algorithm (implemented using the Epsilon model management framework[5], which will automatically calculate optimal ways in which the components can be configured to satisfy the overall TLCM objectives. The details of this algorithm can be found in [4]. Effectively, the implementation of the algorithm uses a chain of different kinds of model transformations to calculate optimal solutions. The chain of transformations maintains traceability between source models (scenarios and components) and optimal solutions; the chain is itself very complex, involving around 5K lines of transformation code. However, the traceability information is not specifically needed to trace the results from solution back to source models, because the way in which the transformations produce results, both solutions *and* relevant source models are presented to the end-user (in other words, the traceability information is captured explicitly in the output). Explicitly representing traceability information in this specific case is necessary to allow the end-user to consider the tradeoffs between solutions that are available.

[5] `http://www.eclipse.org/epsilon`

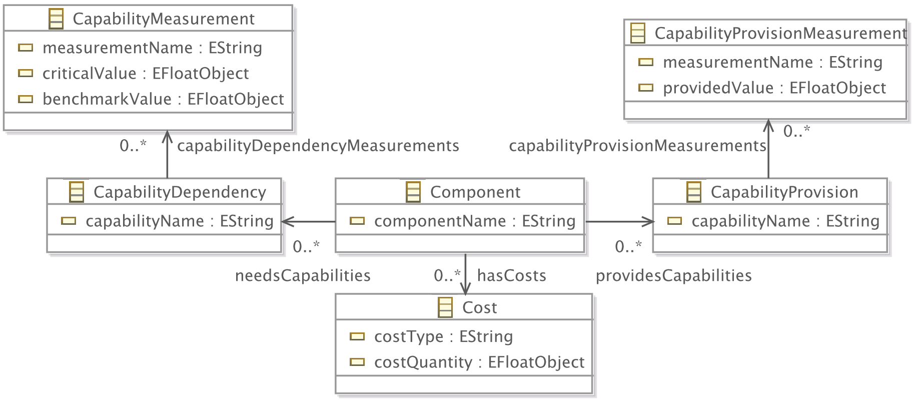

CapabilityMeasurement
measurementName : EString
criticalValue : EFloatObject
benchmarkValue : EFloatObject
0..*
capabilityDependencyMeasurements
CapabilityProvisionMeasurement
measurementName : EString
providedValue : EFloatObject
capabilityProvisionMeasurements
0..*
CapabilityDependency
capabilityName : EString
Component
componentName : EString
CapabilityProvision
capabilityName : EString
0..*
needsCapabilities
0..*
hasCosts
0..*
providesCapabilities
Cost
costType : EString
costQuantity : EFloatObject

Fig. 3. Component and measurement DSL

We have applied the modelling approach and the optimisation algorithm to a number of TLCM problems, including the next-release problem (i.e., determining the optimal features to include in a next software release), a lifeboat problem, and a crisis management problem. With this approach we were able to make explicit the relationships between solutions and problems, particularly when those relationships were extremely complex (e.g., when an aggregate of components was used in a non-obvious way to satisfy a complex decomposition of goals). Effectively, the approach can be used to disentangle the most appropriate ways to satisfy goals, to exhaustively enumerate all possible optimal solutions to TLCM problems, and as a result guide TLCM practitioners through the process of making tradeoffs between these solutions. DSLs and modelling helped us to reveal complexity, but also manage it in efficient and controlled ways.

4 Conclusions

We have outlined a modelling approach, based on the construction of domain-specific languages, for improving understanding of complex systems. Typical approaches to use of domain-specific languages (e.g., as exemplified in Model-Driven Engineering) focus on generating new applications; we instead focus on using the principles of domain-specific languages to produce views of complex systems that help us better understand interactions between components. We have used this approach, in concert with model transformation technology, to better understand complex business processes (to help identify failure behaviour, both locally and in subsystems) and to better understand complex scenarios (to help identify the relationships between actors in scenarios). We are current applying domain-specific languages to wider search-related problems, particularly for validation and verification of models and mechanisms for manipulating models. One of the critical problems in modelling is determine the validity of the model; we are attempting to evaluate this by building tools that manipulate models (e.g., via transformation, code generation, or comparison) and then using search-based techniques to automatically test the tools, thus ideally giving us more confidence in the validity of the models, the DSLs themselves, and the tools.

References

1. Baxter, G.: White paper: Complexity in health care. Technical report, Large Scale Complex IT System, LSCITS (2010)
2. Brook, R.H., McGlynn, E.A., Cleary, P.D.: Quality of health care: measuring quality of care. New England Journal of Medicine 335, 966–970 (1996)
3. Brooke, P.J., Paige, R.F., Power, C.: State exploration and property checking for fuzzy scenarios (under review, 2012)
4. Burton, F.R., Paige, R.F., Rose, L.M., Kolovos, D.S., Poulding, S., Smith, S.: Solving Acquisition Problems Using Model-Driven Engineering. In: Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H., Kolovos, D. (eds.) ECMFA 2012. LNCS, vol. 7349, pp. 428–443. Springer, Heidelberg (2012)
5. Donabedian, A.: The Definition of Quality and Approaches to Its Assessment. Health Administration Press (1980)
6. dos Santos, O.M., Woodcock, J., Paige, R.F.: Using model transformation to generate graphical counter-examples for the formal analysis of xuml models. In: ICECCS, pp. 117–126 (2011)
7. Object Management Group. Business process definition metamodel (BPDM), process definitions (2008), http://www.omg.org/cgi-bin/doc?dtc/2008-05-09
8. Haywood-Farmer, J., Alleyne, A., Duffus, B., Downing, M.: Controlling service quality. Business Quarlerly 50(4), 62–67 (1986)
9. Haywood-Farmer, J.: A conceptual model of service quality. International Journal of Operations and Production Management 8(6), 19–29 (1988)
10. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Symbolic Model Checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 113–140. Springer, Heidelberg (2002)
11. Lowe, G.: Probabilistic and prioritized models of timed CSP. Theoretical Computer Science 13(2), 315–352 (1995)
12. McKane, T.: Enabling acquisition change - an examination of the Ministry of Defence's ability to undertake Through Life Capability Management. Technical report (June 2006)
13. NHS. Acute stroke and transient ischaemic attack suspected (January 2010), http://healthguides.mapofmedicine.com/choices/map/stroke2.html
14. Parasuraman, A., Zeithaml, V.A., Berry, L.L.: A conceptual model of service quality and its implications for future research. Journal of Marketing 49, 41–50 (1985)
15. Plsek, P.E., Greenhalgh, T.: The challenge of complexity in health care. British Medical Journal 323, 624–628 (2001)
16. Sweeney, K., Griffiths, F. (eds.): Complexity and Healthcare: an introduction. Radcliffe Medical Press (2002)
17. Wallace, M.: Modular architectural representation and analysis of fault propagation and transformation. Electr. Notes Theor. Comput. Sci. 141(3), 53–71 (2005)