Exploring the use of Metaheuristic Search to Infer Models of Dynamic System Behaviour

James R. Williams, Simon Poulding, Richard F. Paige, Fiona A. C. Polack

Department of Computer Science, University of York,
Deramore Lane, York, YO10 5GH, UK.
{jw,smp,paige,fiona}@cs.york.ac.uk

Abstract. As software systems become more pervasive and increase in both size and complexity, the requirement for systems to be able to *self-adapt* to changing environments becomes more important. This paper describes a model-based approach for adapting to dynamic runtime environments using metaheuristic optimisation techniques. The metaheuristics exploit metamodels that capture the important components in the adaptation process. Firstly, a model of the environment's behaviour is extracted using a combination of inference and search. The model of the environment is then used in the discovery of a model of optimal system behaviour – i.e. how the system should best behave in response to the environment. The system is then updated based on this model. This paper focuses on investigating the extraction of a behaviour model of the environment and describes how our previous work can be utilised for the adaptation stage. We contextualise the approach using an example and analyse different ways of applying the metaheuristic algorithms for discovering an optimal model of the case study's environment.

1 Introduction

Software that can adapt to changes in its environment without, or with minimal, human intervention is a key challenge in current software development [1]. One method of deciding upon the most appropriate adaptation to perform is to utilise *metaheuristic optimisation techniques*. These techniques may be used to efficiently locate (near) optimal solutions by assessing how close candidate solutions are to solving a problem and using this information to guide the search over the the space of all possible solutions [5, 4]. Therefore, in reformulating system adaptation as an optimisation problem, these techniques can be applied.

Previous work by Ramirez and Cheng [8] uses metaheuristics to discover optimal system configurations at runtime. These configurations are represented as a graph of interconnected system components. A genetic algorithm is used to activate and deactivate links between components in the hunt for an optimal configuration, using sensory information from the environment to evaluate candidate configurations. Earlier work by Goldsby and Cheng [3] uses metaheuristics to discover optimal component behaviour models (state machines). Performance

issues, related to an expensive fitness function and the time taken to generate models, were cited for this technique as only being effective at design time [8].

The main contribution of this paper is a model- and search-based approach to self-adaptation. We focus in this paper on adaptation at the component level, providing a fine-grained level of adaptation whilst aiming to overcome the performance issues found in [3]. Rather than encoding static information from the environment in fitness functions (as in [8]), our approach uses metaheuristics to extract a model of the environment's dynamic *behaviour*. From this environment model, we apply a second metaheuristic to discover a model of the optimal system/component behaviour. In order to do this, we use a generic search-amenable representation of models – making our approach applicable to any problem domain where adaptable parts of the system, and the environment's behaviour, can be represented as a model. It is worth noting that modelling the behaviour of the environment may be very challenging and expensive. Our previous research demonstrated how we can discover optimal system models [12]. In this paper we focus on exploring the use of metaheuristics to extract a model of the environment's behaviour. We investigate ways in which we might improve the metaheuristic's efficacy in order to make this approach applicable at runtime.

To illustrate our approach, we aim to adapt our "Super Awesome Fighter" (SAF) computer game [12] to maintain the challenge of the game as a player learns new strategies. The environment to which the SAF system must adapt is thus a human player, for which a behaviour model is extracted. The objective is to evolve the game – in terms of providing suitable opponents – and tailor it for that player. We define maximal gameplay enjoyment and challenge as a condition where, irrespective of skill level, a player never plays an opponent that they have no chance of beating, nor fights an opponent that they find easy to defeat. The system component being adapted, therefore, is the opponent.

The paper is as follows. Section 2 describes our adaptation approach and introduces the SAF game, whilst highlighting how our previous work can be used to discover optimal system models. Section 3 then presents our approach to extracting models from an environmental behaviour trace at runtime, and describes how this can be applied to SAF. We evaluate the efficacy of our approach applied to SAF in section 4, examining ways to improve the performance of the metaheuristic. The paper concludes in section 5 by discussing limitations and highlighting future work.

2 Adaptation through Optimisation

Figure 1 shows our runtime adaptation approach that combines the use of models and metaheuristic search. The first stage to adapting runtime system behaviour in response to a dynamic environment is to extract a model from the behaviour trace of the environment. Utilising a meaningful representation of the sensory information can enable better runtime decision making [10]. This data model guides a metaheuristic search algorithm to extracting a model of the behaviour of the environment (e.g. a set of rules, or a finite state machine). By determin-

ing a model of the environment's behaviour, we can use a second metaheuristic algorithm to discover a model of the optimal system/component behaviour in response to the environment. In [12] we demonstrate how to automatically discover models with desirable characteristics using metaheuristic optimisation. To do this, we defined a way to search over the space of models conforming to a given metamodel of a *textual* language. We have extended this technique to all models (textual or graphical) by defining an integer-based, metaheuristic search-amenable representation of models that can encode any model conforming to any given metamodel [11]. After mapping this encoding to a model, it can be evaluated by a problem-specific fitness function. This representation allows us to easily apply metaheuristic optimisation techniques to any problem where the solution is a model, therefore by defining metamodels for the adaptable components in a system we can discover optimal configurations in response to environment changes.

We now describe the example through which we explore our adaptation approach.

2.1 Example: SAF

Super Awesome Fighter[1] has been developed to illustrate MDE concepts to high-school students. The game is played between two human-specified fighters, or one human-specified fighter and a pre-defined opponent. Fighters are specified using a bespoke language, the *Fighter Description Language* – a language developed using MDE technologies (Xtext[2] [2]) – meaning that each fighter in SAF is a model. SAF logs the state at each time step in a fight for playback purposes.

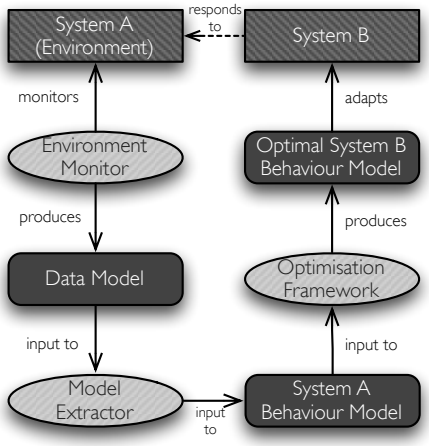

Fig. 1. The process of extracting a model of the environment and using it to adapt system behaviour.

In [12] we exploited the FDL to derive opponents with desirable behaviour using metaheuristic optimisation algorithms. In the context of SAF, the solution space is all instances of the FDL grammar. The optimisation algorithm is guided through the search space by assigning *fitnesses* to each candidate solution it examines – i.e. how close that candidate solution is to solving the designated problem. The *grammatical evolution* algorithm [7, 9] is used to instantiate candidate fighters which are then evaluated against some desirable criteria. The goal was to discover opponents that would be 'interesting and challenging' for the player, which was defined as

[1] `super-awesome-fighter.appspot.com`
[2] `www.eclipse.org/Xtext`

meaning an opponent who wins 80% of its fights against other fighters (the human's goal is to iteratively improve their player so that it beats the opponents). We demonstrated that it was possible to discover these desirable fighters, thus concluding that the optimisation algorithm was effective. Although performed in a purely illustrative context (a simple computer game), the principles extend to any domain where searching for optimal models is of interest.

The process of selecting opponents for a player can be seen as analogous to adaptation – the opponent is the adaptable component, and the game aims to adapt the opponent in order to keep the player interested. Assuming the game is directly interactive (unlike SAF where the player is already represented by a model), it is possible to extract a model of the player's behaviour. This model can then be used to select appropriate opponents to pit against the human. In other words, we use a model of the human behaviour to discover an optimal model of the adaptable system behaviour. It is purely due to the nature of our case study that both of these models conform to the same metamodel (FDL): the important point is that both the environment and the adaptable part of the system can be represented by models. Therefore, the technique presented in this paper is applicable to adapting system behaviour for any system whose adaptable components can be represented as a model and providing that this model can be evaluated to guide the optimisation algorithm.

As our approach, when applied to SAF, will be to extract and discover FDL models, we now briefly overview the FDL.

Fighter Description Language FDL conforms to the metamodel shown in figure 2. The language allows players to specify their character's Personality – the punch and kick power and reach – and define their character's behaviour using a set of Rules. Each rule is composed of a Condition, a MoveAction, and a FightAction. Listing 1.1 shows an example of a fighter described using FDL. It also illustrates some of the more complex parts of the language (which are not shown in figure 2). Firstly, there is the choose construct which allows players to specify a set of actions (movement or fighting) and the game selects one at random. Secondly, FDL supports composite conditions using the and and or constructs. Finally there is a special Condition called always. This is a rule that is always applicable; if none of the other rules' condition's are valid, then the always rule is applied.

```
1  JackieChan{
2    kickPower = 7
3    punchPower = 5
4    kickReach = 3
5    punchReach = 9
6    far[run_towards punch_high]
7    near and stronger[choose(stand crouch) kick_high]
8    weaker[run_away choose(block_high block_low)]
9    always[walk_towards punch_high]
10 }
```

Listing 1.1. An example character defined using FDL.

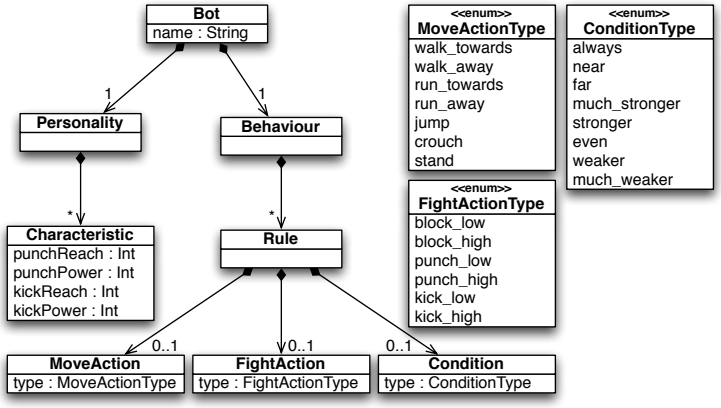

Fig. 2. Simplified metamodel for the FDL used to describe players in SAF.

At runtime, each player's rules are examined sequentially in the order they were defined and the first rule whose condition(s) are applicable in the current game state is selected for execution.

We now describe our metaheuristic-based approach to extracting a model of then environment and describe how it can be applied to SAF.

3 Model Extraction using Search

Our approach proposes that information of interest expressed by the environment be captured in a *data model*. This model becomes the starting point of a metaheuristic search algorithm which attempts to extract a model of the environment's *behaviour*. This model is then used to discover optimal system behaviour (see section 2 and [12]). This section overviews our environment behaviour model extraction process and describes its application in the context of SAF.

3.1 Model Extraction Process

Extracting a model from a corpus of data can be achieved in numerous ways, such as game theory or inference from domain knowledge [1]. In this paper, we examine an extraction process based on metaheuristic optimisation. The process assumes the existence of two metamodels: one that meaningfully captures data coming from the environment; and another that represents the environment's behaviour – this could, for example, be a state machine or a domain-specific model. The information contained in the environment data model is used to guide the metaheuristic algorithm to infer the model of environment's behaviour. In order to infer the behaviour model, we utilise our generic, search-amenable representation of models [11]. For model extraction purposes, we search over the space of environment behaviour models (defined by the metamodel) using the

environment data model to inform the search. We describe this approach now using SAF as a contextualising example.

3.2 SAF: Extracting Player Behaviour

Our goal to make SAF self-adapting needs to be met by trying to infer a FDL model of the way the human is playing the game, and using this model to select appropriate opponents. In SAF's current form, player's specify their behaviour using a FDL model, which may not accurately reflect human behaviour in a more interactive, reactive game. To address this, our experiments (section 4) create mutations of the human model as a method of introducing noise into the data to better simulate the non-uniform behaviour of a human player. Having the original FDL model of the actual human, however, does allow us to validate the extracted model by comparing it against the human model (see section 4).

Our extraction workflow is a series of model management operations, and is illustrated (with respect to the experimentation) in figure 3. To increase the chances of finding optimal solutions, we defined a simplified (but equivalent) version of the FDL metamodel which is more amenable[3] to our model-search technique. We have defined a two-way model-to-model transformation from our simplified metamodel to the FDL metamodel. During the evaluation of a candidate solution, it is transformed into FDL for use in SAF. The model management operations, and the model-search tool, are all written using the Epsilon[4] languages, and are chained together using Epsilon's orchestration framework [6].

The Environment SAF produces a log file for each fight that takes place, where each game state is formatted as follows:

$$P1_X, P1_H, P1_M, P1_F; P2_X, P2_H, P2_M, P2_F$$

where PN refers to either player one or player two; X is the player's position; H is the player's health; and M and F are the player's movement action and fight action that they are performing. Some actions take more than one time step. In these cases, SAF logs the player as being *blocked* and stops the player from performing another action. The contents of the log files are the behaviour trace being produced by the environment.

We have defined a data metamodel (available at `www.jamesrobertwilliams.co.uk/adaptation`) which captures the information contained in the logs in a more useful way - frequency counts of conditions and actions. In every game state, there are always two conditions that are applicable. This is due to the fact that conditions come in two distinct categories: those related to health (e.g. *much_weaker*, *stronger*) and those related to distance from the opponent

[3] The FDL metamodel is reference-heavy; a feature which the model-search tool struggles with. We believe this refactoring process could be automated but leave this for future work.

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

(*far*, *near*). Exactly one condition from each category will be applicable in each game state. Therefore, our data model captures the log file information in two ways. Firstly, it captures the frequencies of actions against single conditions, and secondly, it captures the frequencies of actions against pairs of conditions. The two conditions applicable in each game state are calculated using the same rules that SAF uses. For example, *near* is applicable when $|P1_X - P2_X| \leq 5$.

Logging two conditions in every state makes manually inferring the player's behaviour rules very tricky. When defining behaviour rules in FDL, the player is not required to specify two conditions and so a rule whose only condition is *near* may be executed in any of the health-related conditions. Therefore, although manual inference of the behaviour rules initially seemed trivial, further inspection suggested that it is not and therefore might be a good candidate for metaheuristic optimisation.

Partial Model Extraction The logs produced by SAF only give information about the behaviour of the fighter, and not its *personality*. This information cannot be extracted from the data model[5] and so we propose a two stage extraction process. These stages are SAF-specific but may proof useful for other case studies. Firstly, we extract the behaviour rules from the data model using a *genetic algorithm* (GA), and then we use a *hill climbing* algorithm to discover the personality parameters and thus complete the environment behaviour model. We use a GA to discover the rules because the behaviour rules are complex and interrelated, meaning that there are likely to be many local optima which makes the problem unsuitable for hill climbing. To discover the personality parameters, however, the search space is much smaller and contains fewer local optima and so hill climbing is used.

To evaluate the fitness of a candidate solution in the GA, we compare its behaviour with respect to the information contained in the data model. Each condition pair entry in the data model is passed to the candidate solution as if it were the game state. The first rule that is applicable with respect to the pair of conditions is selected and 'executed' the same number of times that the condition pairs appear in the data model. The rule is 'executed' multiple times because it may contain action choices and therefore result in different actions. The frequencies of the resulting move and fight actions are compared against the frequencies found in the data model. The fitness is calculated as the sum of squares of the distance between the target frequency and the actual frequency. We also reward partial matches (e.g. where the move action matches, but the fight action does not).

Population Seeding For many metaheuristic optimisation algorithms, the effectiveness of the algorithm can depend on where in the solution space the algorithm begins. For population-based metaheuristic algorithms, such as GAs, it is

[5] Thorough analysis of the log file can actually shed *some* light on the personality. For instance, by tracking the distance moved when running, or analysing the effects of punches that make contact. This, however, is out of scope for this paper.

common to create an initial population of diverse candidate solutions in order to promote exploration of the search space.

We can, however, make use of the information contained in the data model and attempt to create an initial population which starts the search in an area of the solution space that is closer to the optimal solution and therefore improve the performance of the search. This process is called *seeding*, and is not SAF-specific. We have implemented two different types of seeding which we compare in section 4. Although seeding can be useful, it may bias the search towards sub-optimal solutions. As such, when seeding we also include some other solutions created randomly. We assess the effectiveness of these seeding strategies in section 4.

Random Seeding Our random seeding algorithm creates a set of simple candidate solutions (fighters) by random selecting entries from the data model and creating behaviour rules. Up to 4 composite condition rules are created from randomly selected condition-pair entries in the data model, and up to 3 single condition rules are created by randomly selecting single condition entries from the data model. Once the set of rules has been created, the fighter is mapped down to its integer string form and added to the initial population.

'Intelligent' Seeding Instead of random selecting entries from the data model, we can do a small amount of inference using domain knowledge to create the rules. For example if we have two condition-pair entries which have different action sets, we can create a composite condition rule which uses FDL's choice construct to select between the actions.

Model Completion As previously mentioned, it may not always be possible to infer the entire environment behaviour model, as is the case in SAF. In this instance, we propose using a different metaheuristic optimisation algorithm called hill climbing. This algorithm attempts to find the correct allocation of personality parameters to the partial model that resulted from the GA. As SAF is aware of the previous opponents that the player has fought, we can use them to evaluate candidate personality parameters. The partial model is, therefore, updated with the candidate personality and then pit against the same opponents that produced the data in the original data model. The fitness of the personality is calculated as the distance between the resulting data model and the original data model.

We now illustrate this approach by investigating whether or not it is able to successfully extract human models of increasing complexity.

4 Exploring the Effectiveness of Search

We now evaluate our approach by attempting to successfully extract the models of three input fighters. The fighters (available at `www.jamesrobertwilliams.co.uk/adaptation`) are of increasing complexity: the *simple* fighter uses only

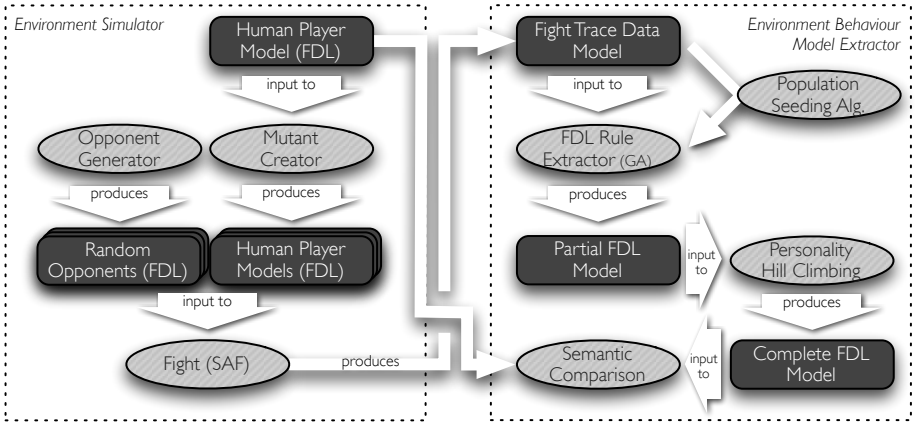

Fig. 3. The process of extracting a model of the player used for the experimentation.

atomic conditions and no action choice; the *medium* fighter has some rules with composite conditions but still no action choice; and the *complex* fighter has rules with both composite conditions and action choices.

To make the problem more challenging and simulate the non-uniform behaviour of real humans, we also automatically create new instances of each fighter with slight stochastic variations, called *mutants*. For instance, we might mutate the condition of a particular rule to produce variable behaviour. When creating the data model, one of the mutants is selected randomly to participate in the fight. This produces a noisier data model, which is intended to simulate more realistic behaviour.

Following the process outlined in figure 3, we investigate six scenarios for each of the three human models, analysing both the effects of mutants and the effects of population seeding:

1. #mutants = 0; population seeding = none
2. #mutants = 0; population seeding = random
3. #mutants = 0; population seeding = intelligent
4. #mutants = 5; population seeding = none
5. #mutants = 5; population seeding = random
6. #mutants = 5; population seeding = intelligent

For fairness, each experiment is executed ten times, each with a different seed to the pseudo-random number generator. The environment is simulated by fighting the human (and mutants) against 50 random opponents.

Algorithm Settings The GA is configured with a population of size 10, and executed for 20 generations, and the personality hill climbing algorithm was executed for 50 generations. The complete set of parameters used in for metaheuristic algorithm is available at `www.jamesrobertwilliams.co.uk/adaptation`. The

parameter values selected for a metaheuristic technique plays a crucial role in its efficacy. At this stage, we are not concerned with efficiency (although this is obviously extremely important for adaptation at runtime) and so no substantial effort was made to tune the parameters to this problem. Our goal is determine the feasibility of using this approach to adapt components at runtime, and analyse whether population seeding is a potential method of improving the performance.

Response In order to validate whether or not we successfully extracted the human model, we devised a *semantic fighter comparator*. A structural comparison of the fighters would not be enough, as different combinations of rules and personalities can result in equivalent behaviour. Our semantic comparator, therefore, aims to see if the the extracted model is semantically equivalent to the input model. In other words, when fighting against the same opponents, do they perform as well. In order to calculate a semantic similarity score, the extracted model and the input model both fight against 100 randomly generated opponents. Each opponent is fought ten times, and the number of times that the human model (extracted or input) wins is noted. If the human has mutants, one of the mutants is selected randomly for each individual fight. The semantic similarity score is then calculated as:

$$\frac{\sum_{i=1}^{100} |\#WIN^i_{input} - \#WIN^i_{extracted}|}{100}$$

where $\#WIN^i_{model}$ is the number of times out of ten that the *model* beat opponent i. Scores range between 0 (semantically equivalent) and 10 (semantically dissimilar).

It is worth emphasising that this semantic comparison would not occur at runtime (indeed, it would not be possible); it is used here purely as a way of validating the approach. In addition to a semantic similarity score, we log the time taken for the experiment to execute from start to finish (i.e. follow the entire path through figure 3).

4.1 Results

Figure 4 shows the results of each experiment. The increase in difficulty caused by adding mutants is immediately apparent when examining the results of the simple and medium humans. Without mutants the metaheuristics find near optimal solutions (when seeded), but struggle when mutants are introduced. Unexpectedly, the addition of mutants actually improved the performance for the complex human; the variation in the results suggests that the fitness landscape is noisy, which could attribute towards this phenomenon.

The simple and medium human results highlight the effect of seeding. The GA performs poorly without being seeded – likely due to the randomly initialised population containing many more complex solutions than simple ones. This randomness appears to be advantageous for the complex human where it is better to omit than perform seeding – highlighting that our seeding techniques may

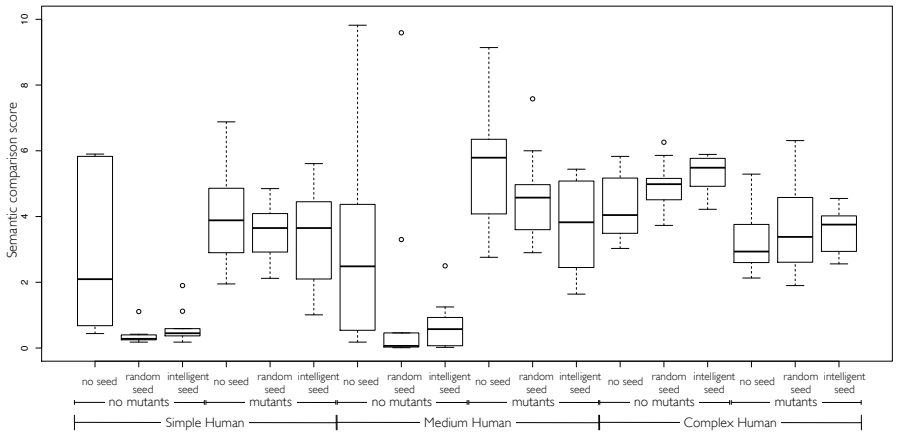

Fig. 4. The results of the experimentation. The lines in the centre of the boxes represents the median score.

need re-examining. Furthermore, the random seed outperforms the intelligent seed for a similar reason – the intelligent seeder creates much more complex solutions using choice and composite rule conditions which are not needed by the two simpler humans. We performed a three-way analysis of variance (ANOVA) and found that all three factors (human complexity, mutants, and seeding) have a significant effect ($p < 0.05$).

Execution times for the entire extraction process ranged in the region of one to ten minutes (median five). This, however, includes the semantic validation phase which would not occur at runtime. Furthermore, no effort has been made to tune the performance of the implementation of the metaheuristics or integer-to-model transformation that they use. While this approach may not be fast enough for systems requiring near-instantaneous adaptation, a tuned implementation may adapt sufficiently quickly for systems where the environment changes more slowly (such as the change in the ability of a game player considered here).

5 Discussion

The aim of this paper was to explore the potential of utilising metaheuristics to enable efficient component-based adaptation at runtime. We presented a model-driven approach to adapting systems at runtime which utilises metaheuristic optimisation in order to, firstly, extract a model of the environment, and secondly, discover the optimal system response. Our focus was on using metaheuristic optimisation techniques to extract a model of the environment's behaviour, and we illustrated using a case study the positive effects of seeding the metaheuristic.

One issue is performance. Search can be expensive to perform and may not be appropriate for systems whose adaptation time is very small. On the contrary,

search can be useful in some time-dependent scenarios (such as adaptation), as it can always return a 'good enough' solution at any point in its execution (as opposed to a constructive approach to model inference which is unable to return a solution until it completes).

There is much work that we would like to investigate in the future: further analysis of seeding strategies and search parameters to improve performance; optimisation of the search algorithms; extracting an environment model based on time (e.g. player behaviour may change at different times, such as the start or end); analysis of the realism that mutants introduce; application of the techniques to other systems; comparison against other model extraction techniques.

Acknowledgements

This research was supported by the EPSRC, through the Large-Scale Complex IT Systems project, EP/F001096/1, and the Dynamic Adaptive Automated Software Engineering project, EP/J017515/1. The authors would like to thank Louis Rose and Dimitrios Kolovos for their advice and feedback.

References

1. B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee. Software engineering for self-adaptive systems: A research roadmap. In *LNCS 5525*, pages 1–26, 2009.
2. S. Efftinge and M. Voelter. oAW xText: A framework for textual DSLs. In *Proc. Workshop on Modelling, Eclipse Con*, 2006.
3. H. J. Goldsby and B. H.C. Cheng. Avida-MDE: a digital evolution approach to generating models of adaptive software behavior. *Proc. GECCO'08*, pages 1751–1758, 2008.
4. M. Harman. The current state and future of search based software engineering. In *Proc. FOSE '07*, pages 342–357, 2007.
5. M. Harman and B. F. Jones. Search based software engineering. *Information and Software Technology*, 43(14):833–839, December 2001.
6. D. S. Kolovos, L. M. Rose, R. F. Paige, and A. Garcia-Dominguez. *The Epsilon book*. Unpublished, 2012.
7. N. O'Neill and C. Ryan. Grammatical evolution. *IEEE Trans. Evol. Comput.*, 5(4):349–358, 2001.
8. A. J. Ramirez and B. H. C. Cheng. Evolving models at run time to address functional and non-functional adaptation requirements. In *MART'2009*, 2009.
9. C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution : Evolving programs for an arbitrary language. In *LNCS 1391*, pages 83–96. Springer, 1998.
10. M. Sánchez, I. Barrero, J. Villalobos, and D. Deridder. An Execution Platform for Extensible Runtime Models. In *MART'2008*, 2008.
11. J. R. Williams, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. Search-based model driven engineering. Technical Report YCS-2012-475, Department of Computer Science, University of York, 2012.
12. J. R. Williams, S. Poulding, L. M. Rose, R. F. Paige, and F. A. C. Polack. Identifying desirable game character behaviours through the application of evolutionary algorithms to model-driven engineering metamodels. *LNCS*, 6956:112–126, 2011.