ONTOLOGY-BASED TEST DATA GENERATION USING METAHEURISTICS

Zoltán Szatmári, János Oláh and István Majzik
Department of Measurement and Information Systems, Budapest University of Technology and Economics
H-1117, Magyar Tudósok krt. 2, Budapest, Hungary

Keywords: Ontologies, Autonomous agents, Optimization with metaheuristics, Test data generation.

Abstract: Software testing is an expensive, yet essential stage in all software development models, thus there is a great effort from the research community to facilitate or even automate this step. Although much of the testing process is automated by modern software development environments (e.g., test execution, monitoring), the selection of test data remains generally a manual process.
In this paper we present a novel approach for test data generation in case of testing data dependent behaviour of autonomous software agents. The proposed method uses the metamodel of the agent's environment derived from the context ontology, and utilizes the input specifications to formulate the goal of testing. Our approach suggests the use of metaheuristic search techniques for the generation of optimal test data, usually referred to as search-based software test data generation.

1 INTRODUCTION

Software testing is the process of evaluating the quality of the *software under test* (SUT) by controlled execution, usually with the primary aim to reveal inadequate behavior or performance problems. During testing a set of *test cases* is executed to verify the expected behaviour. A test case consists of input data, precondition, expected output and postcondition.

Testing is an essential step of all software development models. However, writing test cases is expensive, labor-intensive and time consuming, thus facilitation or automation of the testing process is desired. The main challenge in test generation is to avoid ad-hoc testing and support test case generation using measurable coverage metrics and well-defined method.

One of the most important tasks in automated test generation is *test data generation*, which is the process of identifying input data that satisfy certain criteria (*test goals*). A typical test goal is the verification of the behaviour in selected (often all) states of the SUT. Considering this goal, the automated generation of realistic and feasible input data is usually difficult, because of the large state space of the SUT. However, in certain cases, goals of testing can be expressed solely by referring to the input domain of the SUT, without considering its internal states.

Such cases include testing of *autonomous software agents*. A formal definition of autonomous agents is given in (Franklin and Graesser, 1996), which states the following: *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.* Thus the goal of testing autonomous agents can be expressed as testing the behaviour in case of various configurations of the environment (context).

Construction of efficient test data (that cover all valid configurations using a minimal set of test cases) is still a difficult problem. Application of deterministic test generation algorithms is often impractical, due to the high number of potential configurations and the related semantic constraints that determine the feasible and valid configurations and influence the efficiency of testing in a nontrivial way.

In this paper we propose a novel automatic test data generation approach, which utilizes the context model of the agent and applies metaheuristics in order to generate efficient test data.

First, we propose an ontology based construction of the context model (Section 3). This way the hierarchy and relations of the elements (objects and

Szatmári Z., Oláh J. and Majzik I..
ONTOLOGY-BASED TEST DATA GENERATION USING METAHEURISTICS.
DOI: 10.5220/0003533902170222
In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2011), pages 217-222
ISBN: 978-989-8425-75-1

changes) in the environment can be precisely formulated, which can be directly utilized when defining and computing context coverage (as an important coverage metric during testing).

Second, on the basis of the context model, we express the semantic constraints that are included in the functional specification or generally characterize the domain (determining the valid context configurations, e.g., the arrangement of objects or timing of changes) in the form of model patterns (Section 4). Usually, these patterns are overlapping and constructed at different levels of the hierarchy of the context model, which makes the search for test data difficult. This is why we propose search-based software test data generation (Section 5): instead of deterministic (in worst case exhaustive) search in the space of valid context models, we rely on an iterative improvement of an initial test set by modifying configurations (adding new elements from the context model) and measuring the quality of the resulting test data. Measuring requires the construction of a so-called fitness function that incorporates a refined coverage metric with respect to the model patterns.

Finally, we propose implementation technology in the form of a model manipulation framework that allows efficient representation of model patterns and manipulation of test data (Section 6).

2 REFERENCE ARCHITECTURE

An agent can be described with an *agent function* in abstract mathematical form. The implementation of the function is called *agent program*. The environment in which the agent operates is usually referred to as its *context*.

The reference architecture is shown in Figure 1. This set up is very similar to the arrangement that authors use in (Russell and Norvig, 2003), when defining the connection between an agent and its context.

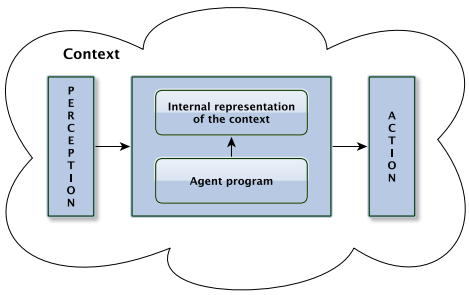

Figure 1: Architecture of an autonomous agent.

The agent program utilizes an *internal representation of the context*, that stores the knowledge of the agent about its environment. This representation should describe all the things and events that are relevant for the behaviour (control algorithms) of the agent.

The input of the agent program is provided by the *perception module*, that identifies the current situation and the changes of the context. Based on the perception information changes are applied on the internal context representation. The control of the agent may include reasoning, learning and adaptation to the evolving context. Based on the internal context representation, the internal rules and goals, the agent program makes a decision and generates the input for the *actuators*. In this paper we focus on the testing of the agent program, and we will not deal with the testing of perception and action execution.

Considering our test goal, testing is implemented through the generation and manipulation of the agent's context (this way the input data for the agent program). Specific configurations and changes in the context are considered as test data. To be able to generate these test data in an automated way, a flexible but expressive representation of the context is necessary. For this purpose we propose an ontology based modeling approach: A *context ontology* is defined that supports the description of the context elements.

3 CONTEXT ONTOLOGY

Ontologies expressed in description logic formalism (Bechhofer, 2004) are commonly used to represent knowledge base in a well-structured and expressive way. Domain experts can easily use this modelling approach since it is close to the human thinking and supports rapid development of domain specific languages.

Ontologies consist of terminologies (TBox) and model instances (ABox). A TBox describes the concepts, its relationships and properties. An ABox collects the model elements that are TBox-compliant instances. In other words terminology is a metamodel-like "dictionary" to define a model while model instances store the knowledge about the modelled things.

The *context ontology* is a domain-specific description of the objects and events in the agent's context that are relevant for its behaviour:

- The *static objects* that can be found in the context are modelled using an ontology concept hierarchy (in other words a dictionary based taxonomy).
- The *relations between concepts* are also modelled.

- Every object could have some *properties* (e.g., location). Properties can be modelled using Data properties or Object properties, that are base elements in an ontology TBox model.
- *Dynamic changes* in the environment should also be modelled using this ontology. We included this dynamic aspect in the context ontology by defining the concept of changes with regard to objects (i.e., an object appears, disappears), their properties (e.g., a property changes) and relations. Using these concepts a dynamic context can be described.

We also defined *context patterns*. Each context pattern represents a fragment of a context model as a specific arrangement of elements within the context. These patterns are originated from the specification (use cases) of the SUT.

We propose the usage of the context ontology during the requirement specification and test definition phases due to its expressiveness and the tool-supported consistency checking facility. In the later development and test generation phases the context ontology can be mapped to a domain specific metamodel and model, while the context patterns can be mapped to model patterns. Axioms in the ontology can be mapped to well-formedness constraints (with regard to the metamodel) and model patterns (that require a desired configuration of model elements).

4 TESTING CONCEPT

Through the last decades, several approaches have appeared for automatic software test data generation, however most approaches concentrate on structural testing. In (Ferguson and Korel, 1996) the authors divided these methods into three classes.

Random methods obviously select input data by random selection. *Path-oriented methods* reduce the test data generation to a path problem, where a path in the control flow is selected (usually to trigger a selected program statement), and then the task is to generate input data to execute that path. In the *goal-oriented approach*, the path selection is eliminated, thus the goal is to find particular input data which trigger a selected statement in the program code. Methods using this approach monitor the program execution with the current input data, and classify branches according to their influence on execution of the desired branch.

The weakness of these approaches in case of agent testing is the program analysis stage. Both path- and goal-oriented approaches require the analysis of program code, which can be complicated in case of large programs, partly implemented programs with component stubs, or legacy code. Furthermore, all introduced approaches handle complex data structures with difficulty, though most modern software applications deal with large and complex input data.

In this paper we focus on functional testing. Our approach is based on the *high-level behaviour specifications* of the agent program, that are relations between the program input and output. For example a specification states that if a particular configuration of objects is present in the context, the agent executes an associated action.

We utilize these specifications to formulate *requirements* regarding the context of the SUT. These requirements include the presence of particular configuration of objects stated in the input specification. These configurations can be represented by context patterns.

The goal of test data generation is to cover all context patterns. Hence a generated suite of test data is perfect for our goal, if it covers all required context patterns, this way it is appropriate to determine whether the SUT behaves as specified. We will refer to this as *sound* suite of test data.

The generated test data is an *instance model* that conforms to the metamodel constructed from the context ontology, thus it shall fulfill the *well-formedness constraints* defined by the metamodel. Additionally, the test data shall conform with the context patterns that also originate from the context ontology (for example, they require the presence of certain objects when another object is already present in the instance model). We will refer to these restrictions as *semantic constraints*. We will refer to test data that is well-formed and satisfies the semantic constraints as *valid* test data.

Since the generated test data is an instance model, manipulation of this model during the execution of the test data generation algorithm can be implemented in the form of a *model transformation* (MT). These transformations take an input model and produce an output model by the application of a transformation rule. In our case the metamodel of the input and the output instance models are the same, thus we apply endogenous MTs.

Previously we have stated that a sound and valid suite of test data covers all required context patterns and satisfies the semantic constraints. Finding the optimal set of test data is a non-trivial problem due to the large number of patterns, the hierarchy of objects and relations included in these patterns and the overlapping nature of patterns and semantic constraints (e.g., patterns may contain configurations of elements from other patterns, constraints may complementary etc.).

According to these problems, the issue of test data generation can be formulated as an *optimization problem*. A fitness function assigns a real value to each suite of generated test data. This value indicates how well a particular candidate fulfills the criteria aggregated in the fitness function. One such criterion is the coverage of the patterns, i.e., the number of patterns that is included directly or indirectly (taking into account the hierarchy and overlapping of patterns) in the set of generated test data. Another criterion that can be taken into account is the size of the set of test data (that shall be kept low to reduce the cost of testing). Our task is to locate the global maxima of the fitness function, this way to find the optimal set of test data.

5 SEARCH-BASED TEST DATA GENERATION

Search-based software engineering (SBSE) is the use of search-based optimization algorithms (usually metaheuristic search techniques) to software engineering problems. SBSE is an approach with increasing relevance, since search techniques were successfully applied to a number of software engineering problems throughout the whole software development life-cycle (Harman, 2007). Software testing is probably the most important application domain of SBSE. Furthermore the amount of research in search-based software test data generation alone is so significant that it led to a survey by McMinn (McMinn, 2004).

Metaheuristics are the primary subfield of stochastic optimization applied for a very wide range of problems. Metaheuristics can be divided into *single-state methods* (i.e., hill-climbing, simulated annealing or tabu search) and *population methods* (i.e., genetic algorithms and evolution strategy from the field of evolutionary computation, or particle swarm optimization from the class of swarm intelligence methods). An exhaustive and up to date description of metaheuristics is presented by Luke (Luke, 2009).

Metaheuristics are advantageous in problems described as *"I know when I see it"*. In our case, for example, the formulation of a deterministic algorithm would be impractical taking into account the dependency and hierarchy between semantic constraints and context patterns to cover, though we are able to score the quality of a candidate solution (test suite) and decide whether it is optimal.

The key ingredients for the application of search-based optimization to test data generation is the choice of representation of the solutions and the definition of the fitness function. In order to successfully apply metaheuristic search techniques, a good representation should fulfill the *heuristic belief* about the space of candidate solutions. This means that similar solutions behave similarly, thus small changes in parameters will result in small changes in the quality of the current solution.

As we already mentioned, the task of test data generation can be interpreted as generation of instance models, thus in this case the candidates are represented as model instances. We call the generated test data sound according to the fitness function, when all required objects are covered according to the established goals (i.e., there are matches of the context patterns within the model). The fitness function formulated to guide the test data generation should reward model instances that contain the context patterns with higher scores. The computation of the coverage of model patterns, that is the core of the fitness function, should well handle the introduced hierarchy and dependency problems.

Additionally, the formulation of operators is a fundamental question when metaheuristic algorithms are applied for optimization. These operators define how the candidate solution(s) can be updated in each iteration. In traditional problems, the candidate solutions are represented as vectors, thus updating is executed by the manipulation of values in the vectors.

Since our candidate solutions are represented by instance models, updating of a candidate can be executed by the introduced model transformations. Possible transformations of candidate solutions are defined by a set of model transformation rules prior to the execution of the test data generation algorithm. In every iteration of the applied metaheuristic algorithm, an arbitrary number of rules are selected and executed. Obviously, these rules do not violate the well-formedness constraints provided by the metamodel.

Figure 2 presents the entire workflow of the proposed test data generation algorithm.

6 IMPLEMENTATION

To implement the proposed test data generation approach a *model manipulation framework* is needed, that supports metamodel based model manipulation tasks to generate instance models conforming to the domain specific metamodel.

The set of *initial instance models*, which forms the input for the test data generation algorithm, should be constructed based on the context metamodel.

The test data generation algorithm utilizes the following functions of the model manipulation framework:

- Since the conformance to the metamodel is the

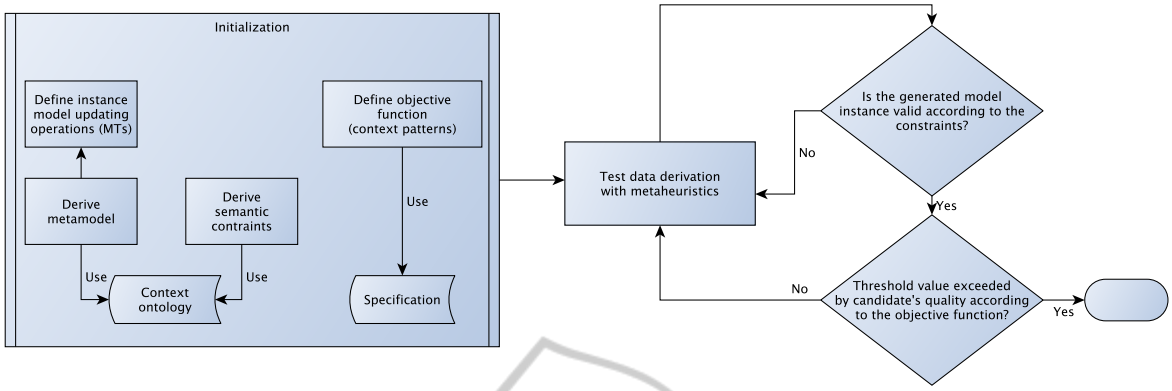

Figure 2: Workflow of the proposed approach.

primary requirement in the case of the generated instance models, the framework should support efficient *metamodel conformance checking*.

- Application specific requirements are represented using content patterns, so *checking the coverage of model patterns* should be supported.
- Since the operators in the test data generation algorithm are defined as *model transformation rules*, the efficient execution of such rules is required.

Several model transformation frameworks exist that support these functions. Graph transformation based frameworks (e.g., VIATRA2[1], AGG[2]) apply graph pattern matching and graph transformation rules for checking and manipulating model instances. Another solution could be the application of a *rule engine* (e.g., Drools[3]), which supports the construction of domain specific rules that can be used for pattern checking and model transformation.

In certain cases (e.g., in case of autonomous robots) the context model describes a real world that consists of 3D objects and dynamic behaviour of these objects. For demonstration and visualization purposes the context model can be transformed into a visualization language (e.g., X3D is an open-standard format, that is able to describe 3D scenes and objects).

7 AN EXAMPLE

We demonstrate our proposed approach on the example of a simplified version of the *Wumpus World*. This world is a popular demonstrating environment for intelligent agents, thoroughly discussed in (Russell and Norvig, 2003).

[1]See http://www.eclipse.org/gmt/VIATRA2/ for details.
[2]See http://user.cs.tu-berlin.de/˜gragra/agg/ for details.
[3]See http://www.jboss.org/drools for details.

Wumpus World is a cave with a number of rooms. Each room is represented with a square. The neighborhood of a room consists of four rooms (north, south, east and west).

In our simplified example only one Wumpus and one treasure is present on arbitrary squares. If the Wumpus is at a square, then there is stench on that square and all on its neighboring squares. If the treasure is at a square, then there is glitter on that square.

The agent in this world perceives the current square where it is located. According to the two perceivable elements (i.e., stench and glittering), these perceptions are represented by two element vectors.

The agent may turn 90° left or right, and go forward. In our case if it goes to a wall nothing happens, and if the agent advances on a square where a Wumpus is waiting, the agent is destroyed. The agent may decide to leave the cave, if it can not safely determine the treasure.

Let us consider the following input specification. The agent operating in an arbitrary Wumpus World that conforms to the rules introduced above, it is able to avoid the Wumpus, and find the treasure or quit, when no further safe move is possible. We may apply our proposed approach in order to generate test data (i.e., Wumpus world) to test whether the agent fulfills this specification.

The context ontology created for the Wumpus World contains all the elements mentioned above, with proper relations. The derived metamodel from this ontology is presented on Figure 3.

According to the ontology axioms defined for the Wumpus World, we may derive one semantic constraint. If the Wumpus is at a square, then there is stench at all neighboring squares. The model pattern expressing this constraint is shown on Figure 4. This is a *negative* pattern: if it has a match in the candidate model, then a square exists near a Wumpus without stench, i.e., the model is not valid for testing.

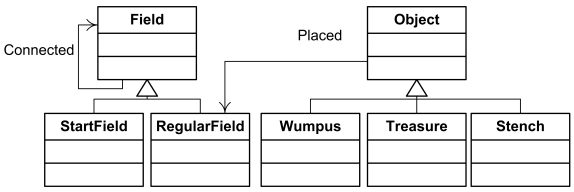

Figure 3: Metamodel of the simplified Wumpus World.

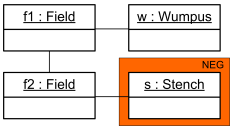

Figure 4: Semantic constraint for the test generation.

A Wumpus World generated to test the agent is sound if it contains all possible elements (Wumpus, treasure and start square). Context patterns expressing these requirements are presented on Figure 5. The fitness function which measures the quality of a candidate Wumpus World counts the patterns that are covered. For example, if the three model fragments has exactly one occurrences in the model (i.e., there is one Wumpus, one treasure and one start square), the fitness function is maximal. The proper appearance of squares with stench is guaranteed by the semantic constraint.

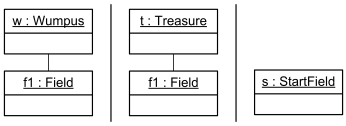

Figure 5: Context patterns that shall be covered.

Since an initial model can be an empty $n \times n$ cave, the applied model transformations may add a Wumpus, a treasure or stench to a square, or transform it to a start square. During the iteration of the selected metaheuristic, a candidate solution is checked whether it fulfills the well-formedness rules and the semantic constraints, and then the fitness function estimates its quality by checking whether there are occurrences of each context pattern in the candidate.

In this example, let us denote the number of occurrences of a context pattern i in a given candidate model with k_i. Let $x_i = 0$, if $k_i = 0 \,||\, k_i > 1$, and $x_i = 1$ if $k_i = 1$, where $i \in S$ and S is the set of all context patterns. Then the fitness function may assign $\sum_{i \in S} x_i$ to the candidate. If the selected threshold is three, the generated test data covers all context patterns once.

8 CONCLUSIONS AND FUTURE WORK

Verification of autonomous software agents is a difficult task, which requires the generation of valid and sound test data according to the system specifications.

In this paper we introduced an approach, which uses the context ontology to determine validity of the generated test data through the derived metamodel and semantic constraints, and measures the soundness of test data with context patterns derived from the system specification(s). Furthermore we proposed the use of search-based test data generation to determine optimal test data. The implementation of the proposed approach is currently under development.

The generated test data have to be sound according to various, often conflicting context patterns simultaneously. An optimal solution is *as sound as possible*, while it remains valid, thus it is usually a trade-off between the individual test goals. This type of problem is usually referred to as *multiobjective optimization problem*. In the future we plan to investigate this problem and apply hierarchical decomposition on the basis of the hierarchy of the input metamodel.

REFERENCES

Bechhofer, S. (2004). OWL web ontology language reference. W3C recommendation. http://www.w3.org/TR/owl-ref/.

Ferguson, R. and Korel, B. (1996). The chaining approach for software test data generation. *ACM Trans. Softw. Eng. Methodol.*, 5:63–86.

Franklin, S. and Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages*.

Harman, M. (2007). The current state and future of search based software engineering. In *2007 Future of Software Engineering*, FOSE '07, pages 342–357, Washington, DC, USA. IEEE Computer Society.

Luke, S. (2009). *Essentials of Metaheuristics*. Available online. (http://cs.gmu.edu/∼sean/book/metaheuristics).

McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14:105–156.

Russell, S. and Norvig, P. (2003). *Artifical Intelligence. A Modern Approach.* Pearson Education Inc., second edition.