SUPPORTING THE TESTING OF AUTONOMOUS SYSTEMS USING CONTEXT ONTOLOGIES

Zoltán SZATMÁRI
Advisor: István MAJZIK

I. Introduction

This paper is based on my previous research on the conceptual comparison of the classical modeling technologies (e.g. UML, EMF) and ontology based modeling. Both of these modeling approaches are intended for describing domain-specific knowledge using high-level models, called metamodels and constructing domain models based on these metamodels [1].

Ontologies expressed in description logic formalism [2] provide a way to represent knowledge bases in a well-structured and expressive way. Ontologies consist of terminologies (TBox) and model instances (ABox). TBox represents the metamodel, it describes the concepts, its relationships and properties. ABox collects the model elements that are TBox-compliant instances. In other words terminology is a metamodel-like "dictionary" to describe a model while model instances store the knowledge of an object.

Last year I presented a framework that supports ontology based assessment of development toolchains. In this paper I propose a method that is based on ontology models and ontology metrics, that can be used for model based test generation for testing autonomous systems.

II. Testing of agent based systems

The concept of agents is a very popular one, which has its origin in the mathematical model of concurrent computation, called *actor model* [3]. After the invention of software agents, they spread across software industry as they have proven useful in many fields from email clients to autonomous robots. In textbook [4], agent is defined as *anything, that can be viewed as* ***perceiving its context*** *through sensors and* ***acting upon that context*** *through effectors.* The agent can be described with an **agent function** in abstract mathematical form, and the implementation of the function called **agent program**. The goal of this work is supporting ontology based testing of the agent programs.

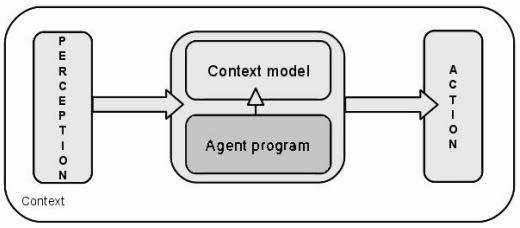


Figure 1: Architecture

In the rest of this paper I will concentrate on those agents, that can be installed in some kind of *spatial situation*, where they can move and execute different actions. Furthermore, I won't deal with the testing of perception and action execution, but only with the testing of the agent program which implements the agent function.

The reference architecture is shown in figure 1. The agent program is the **system under test**. This set up is very similar to the arrangement that authors use in [4], when defining the connection between an agent and its context.

A. Motivation

The main challenge in test generation is to avoid ad-hoc testing and support test generation based on measurable coverage metrics and well-defined method.

During the behaviour testing the focus is on testing the implementation of the the agent program.

The input is provided by the **perception module**, that provides the current situation and the changes of the context. Based on the internal rules and goals the **agent program** makes a decision and generates the input for the **actuator**.

The control of such an agent based system includes reasoning, learning and adaptation to evolving context. The testing of this function is supported by the **context model**, that stores the knowledge about the context for the agent itself. This model should describe all the things, events in the agent's context that are relevant for the control algorithms.

The context model can include **context patterns**. These patterns represent different aspects of test goals, focusing on complex combinations of things and events. Test goals can include complementary or in some cases conflicting patterns. To support test generation based on these patterns definition of **coverage metrics** and based on these metrics an **intelligent algorithm** is needed.

In order to support these approach I propose an ontology based description of the domain. A **context ontology** will be presented that supports the description of the context elements, and based on this ontology the context patterns and coverage metrics can be defined.

B. Context ontology

The **context ontology** is a domain-specific description of the objects and events in the agent's context. A **context model** is constructed based on this ontology and describes the following aspects of the context:

- The **static objects** that can be found in the context will be modeled using an ontology concept hierarchy, or in other words as a dictionary based taxonomy. The hierarchy can be used for defining coverage metrics of a context model in case of testing.
- Every object could have some **properties** (e.g., location), that are modeled using Data properties or Object properties, that are base elements in an ontology TBox model.
- The **relations between concepts** will be also utilized when test coverage is analyzed.
- **Dynamic changes**, and events should also be modeled using this ontology. I defined an extension for the ontology, that allows modeling dynamic behaviour. It describes changes (e.g. appears, disappears, moves) of objects, properties or relations. Using this aspect, based on the static model a dynamic context can be described.

OWL2 ontology language is used to represent the context models, which language provides $\mathcal{SROIQ}(\mathcal{D})$ expressive power.

In most 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 represented in a visualization language (e.g. X3D is an open-standard format, that is able to describe 3D scenes and objects).

C. Test model

The testing is based on a **behaviour specification** of the control, that are relations between input (context) and output (actions). The specification consists of a set of scenarios that represent behaviour. Such scenarios can be created manually, even on the basis of informal requirements, using the elements of the context model and action model. For this purpose a scenario language is used. [5]

The **test data** is a **context model** that is conforming to the context ontology. The **test case** contains every necessary information to execute a test on the **system under test**. It contains the agent's **initialization**, the **workload** that should be executed by the system under test, the description of the **expected behaviour** and the **test data**.

The efficiency of the testing (a set of test cases) can be measured using **coverage metrics** defined in the following way:

- *Coverage of the ontology* measures the coverage of the concepts defined in the context ontology, considering the hierarchical taxonomy. It can take advantage of the hierarchy of the ontology.

- *Coverage of context patterns* measures the coverage of various patterns defined using the concepts of the ontology.

Besides testing the correctness of behaviour, the testing of its robustness can also be addressed. Robustness is an attribute of dependability and it is used to measure the degree to which a system operates correctly in the presence of **exceptional inputs** or **stressful environmental conditions**. In this case the expected input attributes of the agent program are taken from the context model based on the range definitions of the concept attributes or based on the extreme values defined in the ontology constraints.

D. Test generation

In the proposed algorithm, **evolutionary strategy** will be used to generate test data for black-box software testing. Using evolutionary strategy means using **genetic operators** - selection and mutation - in order to generate better solution in every iteration of the algorithm. [6]

The genetic algorithm is based on an initial set of test data (**initial population**), that will be manipulated using the **genetic operators**. These operators can select some elements from the population, can modify an element (mutation) and can add new generated element to the population in order to achieve the goal of the algorithm. The set of the allowed operators and the size of the initial population depend on the used genetic strategy. The goal of the algorithm can be defined using **metrics** that can be measured on the the instances or on the whole population.

The steps of the presented algorithm are the following:

- **Initialize the test generation** with an initial context model based on *context patterns* derived from the scenarios.
- **Complete measures on context models** in order to determine the next step of the algorithm.
- **Assign execution probability** to the genetic operators based on the set of measures in order to balance the different test generation aspects.
- **Make some modification on context models** using genetic operators to get new context models.

During the implementation of the algorithm the main challenge is the connection between the context ontology, metrics and genetic operators.

The definition of the **metrics** used by the algorithm is based on the context ontology. These metrics measure how good is the generated context model in the view of the scenario being tested. In my approach the metrics will be defined on the basis of context patterns, they describe the required objects and properties. These patterns will be implemented using **graph patterns**.

The **genetic operators** are model manipulation steps and will be implemented as **model transformation rules**. The execution semantics of transformation rules provides the nondeterminism that is required by the genetic algorithms.

The final step is making **connection between metrics and genetic operators**. On the basis of the set of metrics the algorithm can get a set of measures on the generated test data. Depending on the measures different **execution probability** will be assigned to the different transformation rules, and the transformation engine can choose one transformation rule based on the probabilities. The connection between patterns (related metrics) and transformation rules (operators) will be described using **test generation rules**.

III. Forklift example

All the presented test generation steps (metrics, model transformation rules) can be implemented using **model manipulation** technologies. In the following this implementation will be presented using a Forklift example based on industrial use cases proposed in the R3-COP project [7].

Cooperating forklifts are Laser Guided Vehicles (LGVs) that physically move the goods and act as the link between the different machines within a warehouse environment. LGVs are controlled by a central unit which plans their path constructed on the basis of the warehouse map. Each forklift knows

its own position calculated using a laser positioning system and the velocity using the wheel encoder information. The context recognition is based on stereo cameras. The actions are defined as *put up things, move to a position and put down things*.

The proposed **scenario** is the following: The forklift should move an object from one position to an other one and during the operation it should not collide other objects.

First, the **context ontology** of the domain should be constructed. A simplified ontology of the forklift domain is shown on figure 2. There can be seen the **general concepts** (*position*, *dimension*, *static and dynamic objects*) and the specific elements **inherited from the use case definition** (*forklift, pallet, shelf, good*).

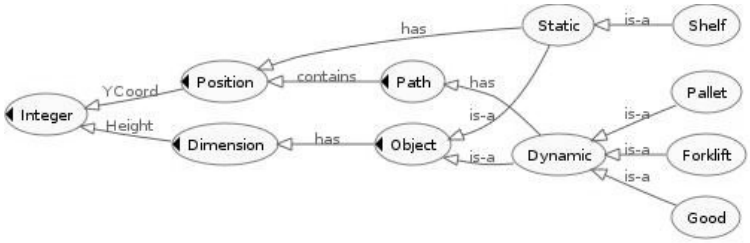
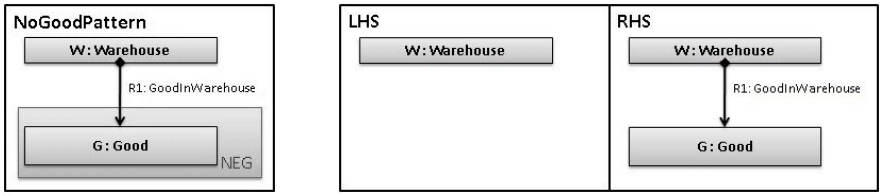


Figure 2: Simplified context ontology

The existence of simple pattern shown in figure 3a is used as a metric in test generation. (If the scenario prescribes to move some object to a different position then there should be a movable object in the context model). This pattern describes, that there is no *good* object in the model.

In figure 3b an example rule is shown, that places a new good object in the context model. This sample rule can be executed more than once, in order to generate goods in the test context.



(a) Sample pattern (b) Sample transformation rule

Figure 3: Forklift test generation examples

IV. Conclusion

I presented a method that extends the test generation algorithm described in [6] with context ontologies. Namely I proposed an implementation of the genetic algorithm based test generation that is supported by **ontology based context description and coverage metrics**.

The construction of the patterns, genetic operators and test generation rules requires domain-specific knowledge. Accordingly, most of these elements can be constructed by domain experts on the basis of the scenario model and context ontology.

References

[1] B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *Journal of Systems and Software*, 84(2):301 – 313, 2011.

[2] S. Bechhofer, "Owl web ontology language reference. w3c recommendation," Feb. 2004.

[3] William D. Clinger, "Foundations of Actor Semantics," Tech. Rep., MIT Artificial Intelligence Laboratory, 1981.

[4] S. Russell and P. Norvig, *Artifical Intelligence. A Modern Approach.*, Pearson Education Inc., second edition, 2003.

[5] Z. Micskei and H. Waeselynck, "The many meanings of uml 2 sequence diagrams: a survey," *Software and Systems Modeling*, pp. 1–26, 2010, 10.1007/s10270-010-0157-9.

[6] János Oláh, "Test data generation using metaheuristics," in *Proc. of the 18th PhD Mini-Symposium*, 2011.

[7] "R3-COP: robust and safe mobile cooperative autonomous systems," URL: https://www.artemis-ju.eu/projects.