Ontology-Based Test Case Generation For Simulating Complex Production Automation Systems

Thomas Moser, Gregor Dürr and Stefan Biffl
Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems
Vienna University of Technology, Austria
{thomas.moser, gregor.duerr, stefan.biffl}@tuwien.ac.at

Abstract—The behavior of complex production automation systems is hard to predict, therefore simulation is used to study the likely system behavior. However, in a real-world system many parameter variants need to be tested with limited resources. Therefore, test cases need to be generated in a systematic way to find suitable scenarios efficiently. This paper investigates the effort of two approaches for providing test cases based on available testing knowledge. The traditional approach uses a static generator script based on implicit testing knowledge, which takes significant effort to add new parameters. The innovative approach uses a dynamic generic generator script based on an ontology data model of the testing knowledge. We empirically evaluate these approaches with a use case from the production automation domain. Major result is that the high-level test description of the ontology-based approach takes more initial effort for setup, but increases the usability and reduces the risk of errors during the test case generation process.

Keywords - test case generation, ontology, production automation simulation, explicit testing knowledge.

I. INTRODUCTION

Production automation systems are often complex as the behavior of the overall system cannot easily be predicted from the behavior of the subsystems. Therefore, simulation is used to study the behavior of complex production automation systems. In addition to simulation accuracy the simulation system performance is an important issue, particularly if many parameters and value ranges for system behavior need to be evaluated systematically [8]. Example test parameters for assembly lines in production automation are scheduling strategy, failure handling strategy, and the number of products.

Software testing investigates the quality of the product or service under test. In order to achieve sufficient test coverage for the requirements of an application, there must be at least one test case for each requirement and relevant parameter setting, which can take considerable effort. A major goal is to generate test cases in a systematic way to efficiently find suitable scenarios for most of the requirements with limited testing resources. In this paper, the test cases define input data to simulate complex distributed assembly line systems with a simulation tool [9, 10, 17].

This paper presents an approach for extracting expert testing knowledge into an ontology and for using this explicit knowledge to efficiently generate test cases for a production automation simulator [9, 10, 17]. For this purpose, the test coverage combined with the cost to achieve this test coverage is used as performance metric. In our context, test coverage is the ratio between the number of generated test case scenarios and the number of all possible test case scenarios regarding a given set of parameters respectively their possible parameter values.

This paper investigates two approaches for providing test cases. The traditional approach implements a test as a static generator script, which is fairly simple and quick to start up, but takes significant effort to add new parameters, since all existing scripts need to be updated to accommodate the new parameter settings. In addition, the tester needs programming skills both for setting and modifying parameters.

The innovative approach models system and test know-how in an ontology-based data model and dynamically adapts a generic generator script according to the settings in the test data model. Test cases are generated and run with respect to the parameters chosen by the user. Important advantages of this approach are (a) available efficient tool support for modifying ontologies in case of changes to the parameters and/or structure of the system under test and (b) the fact that the generator script is not affected by the modification. Therefore, even testers without programming skills can add new parameters by modifying the underlying data model. Test cases are generated from the ontology and exported as XML file. The implemented generator script and the ontology are loosely coupled. Therefore, changes to the ontology do not necessarily lead to changes of the generation script. This design approach enables a flexible and high-level test description.

The evaluation part explores how the ontology-based approach can reduce the cost for test description, enables a more efficient and effective generation of test cases while aiming at an exact definition of test coverage to be achieved, and improves the changeability of the test case generation approach (e.g., the introduction of new test parameters) due to lower efforts to implement new test case parameters.
The remainder of this paper is structured as follows: Section 2 summarizes related work on software testing, production automation simulation, ontologies, and on ontology-based test case generation. Section 3 identifies the research issues and summarizes the use case. Section 4 introduces the ontology-based test case generation approach, while section 5 presents the evaluation results. Section 6 discusses the evaluation results with regard to the research issues, and finally section 7 concludes the paper and identifies further work.

II. RELATED WORK

This section presents related work on software testing and introduces a generic life cycle of a test case. Furthermore, the production automation simulation systems MAST and SAW are shortly introduced. Finally, related work on ontologies as flexible data models for system and test definition, and on ontology-based test case generation is summarized.

A. Software Testing

Testing of software is an important part of every software life cycle to demonstrate the capability of the software and identify defects before operational use. The National Institute of Standards and Technology (NIST) reported in a 2002 study that software bugs cost the U.S. economy around 60 billion dollar per year [12]. The same study showed that these costs could be reduced by more than a third by improving testing. In the IEEE Standard on the Software Life Cycle Process, a process is defined as a set of activities; process groups represent a higher level of abstraction [2].

A test process can be seen as the systematic execution of a software program. Test management and running the test object with specific data are the important parts of the test process. The goal of the test management is to plan, execute, and analyze the test run. A test run consists of one or more test cases [14].

![Figure 1. Life cycle of a test case (according to [14]).](image)

Figure 1 structures the life cycle of a test case, divided into the creative process and the automated process. For the creative process, there exists tool support (e.g., test and quality management cockpits for quality or project managers) to increase the quality of testing [14]. In the context of is work, the tasks for planning the test cases are located in the test case generation process described in section IV. The costs for searching, acquiring, and maintaining a suitable tool build up the cost triangle for testing. However, there is very little work on efficiently determining test cases for production automation systems [13].

B. Production Automation Simulation

The Manufacturing Agent Simulation Tool (MAST) is an agent-based solution for some typical manufacturing tasks by using multi-agent technologies for manufacturing control [17]. Flexible distributed manufacturing systems can be modeled as intelligent, autonomous and cooperative agents [16], where each agent manages the local behavior to meet goals locally without centralized control [17]. In addition, the agents interact to achieve the system goal cooperatively.

The Simulation of Assembly Workshop (SAW) (9, 10), which is an extension of the original MAST system, investigates processes, methods, and tool support for planning, coordination, simulation, and lab tests for work shifts in an assembly workshop. Generally, a workshop aims to effectively and efficiently carry out the work orders in a work shift. SAW helps to understand the impact of tactical decisions in the production automation environment. For instance, the user can optimize his assembly line by analyzing the effect of the different strategies. The SAW project provides a simulator that can execute the test cases for the production automation domain. In addition, the logged events during the simulation allow analyzing the simulation object under test. Thus, the results of the simulation enable checking the post condition of the test case. The explained life cycle of a test case is adopted for the simulation process of the SAW project during the elaboration of this work.

C. Ontologies – for flexible engineering data modeling

In general, ontologies are a main part of the semantic web technology and facilitate the knowledge representation of real-world concepts. Ontologies are formal models of a specific application domain, and primarily used to facilitate the exchange and partitioning of knowledge. More precisely, an ontology is a data model that represents a set of concepts within a domain and their relationships. The word ontology has its origin from the Greek words *ontos* (=being) and *logos* (=word). From a philosophical point of view an ontology refers to the subject of existence, that is the study of being as such [4]. Gruber [5] defines an ontology as an explicit specification of a conceptualization. Where a conceptualization illustrates an abstract, simplified picture of the world used for representation and designation. Each knowledge representation follows a certain degree of conceptualization, either explicitly or implicitly. Moreover, ontologies can effectively support software development processes by providing a continuous data model [3].

The emerging field of semantic web technologies promises new stimulus for Software Engineering research. However, since the underlying concepts of the semantic web have a long tradition in the knowledge engineering field, it is sometimes hard for software engineers to overlook the variety of ontology-enabled approaches to Software Engineering. Happel and Seedorf [6] propose a simple classification schema that allows a better differentiation among the various ideas of using ontologies in Software Engineering. The Ontology Driven Architecture (ODA) note at W3C serves as a
starting point to elaborate a systematic categorization of the approaches and to derive more clearly defined acronyms [15]. Happel and Seedorf [6] propose two dimensions of comparison to achieve a more precise classification. First, they distinguish the role of ontologies in the context of Software Engineering between usage at run time and at development time. Second, they look at the kind of knowledge the ontology actually compromises. Here, they distinguish between the problem domain that the software system tries to tackle, and infrastructure aspects to improve the software or its development. In the context of this paper, ontologies can be used as flexible data models for system and test definition.

D. Ontology-Based Test Case Generation

Software tests are an important part of quality assurance [1]. However, writing test cases is an expensive endeavor that does not directly yield business value. It is also not a trivial task, since the derivation of suitable test cases demands a certain amount of domain knowledge. Ontologies could help generate basic test cases since they encode domain knowledge in a machine processable format. A simple example for this would be regarding cardinality constraints. Since those constraints define restrictions on the association of certain classes, they can be used to derive equivalency classes for testing [7]. Ontologies may not be the first candidate for such a scenario, since there are formalisms like OCL that are specialized for such tasks. However, once domain knowledge is available in an ontology format, it might be feasible to reuse that knowledge.

Nguyen et al. [11] describe a framework for automated test case generation in the context of multi-agent systems. They use agent interaction ontologies that define content semantics of agent interactions to generate test inputs, guide the exploration of the input space during test case generation, and verify messages exchanged between agents with respect to the agent interaction ontology. Their results for interaction ontologies of non-trivial size show that the ontology-based method achieves higher coverage of the ontology classes than manual test case derivation. The ontology-based approach also outperformed manual derivation in terms of defects detected and coverage of input space.

III. RESEARCH ISSUES AND USE CASE

The generation of test cases based on available testing knowledge is an import factor for the simulation of complex systems, such as production automation systems. This paper investigates two approaches for providing test cases. A traditional approach based on manually derived static generator scripts, which takes significant effort to add new parameters. In addition, the users need programming skills for both setting and modifying parameters. The ontology-based approach uses a dynamic generic generator script based on an ontology as data model. Test cases are generated with respect to the chosen parameters by the user.

This work proposes an ontology as suitable data model to provide the necessary data to generate a suite of test cases with an ontology-based approach for a given set of parameters. This claim is also met by the static approach. In other words, the fact that the ontology-based approach is newer and works as well as the old static one is not enough motivation for change. However, measurable benefits are essential to accept the new ontology-based approach and the change costs. As with every new engineering method an important question is what aspects of the new approach are comparable to or better than a best-practice traditional approach. Therefore, we derive the following research issues.

RI-1. Feasibility of the ontology-based test case generation approach. The ontology-based approach aims at overcoming limitations of the traditional static approach. Firstly, the new approach should provide a high-level test description to allow the target audience to generate test cases with less effort. Secondly, a validation check and consistency check of the parameter setting is essential to reduce the risk of making mistakes during the configuration phase of the test case generation process.

RI-2. Cost-benefit potential of ontology-based test case generation approach. The following two sub research issues define a metric how the cost-saving potential can be measured with respect to the effort for test description and the effort for setting up the solution. RI-2a assumes that both the number of parameters to choose and the number of supported data types are constant, while RI-2b addresses the cost for adding new parameters with different data types.

RI-2a. Cost-benefit for a constant number of parameters. Assuming a constant amount of parameters the three cost aspects and the metrics for measuring each cost unit are listed below. The test generation description is a step-by-step instruction for the user to execute the test case generation process. The metric to categorize the test description of the generator scripts is the level of abstraction. A high-level test description increases the acceptability of the generation process since test cases can be described without the need to know details about the application domain and further for the overall simulation process since the simulation is based on the generated test cases as input data. The measurement of the effort for setting up the generator approaches by implementing the corresponding generator scripts are based on determining the effort of process steps. For this purpose the person months for implementing the static script are estimated based on domain experts’ experience. In addition, the effort to realize the ontology-based approach was part of this work and is therefore traceable. In our context the test coverage is the ratio between generated test cases and all possible test cases regarding a given set of parameters respectively their possible parameter values. For that purpose the test coverage combined with the costs to achieve this test coverage is used as performance metric. For the evaluation it is tested whether the generator approach meets the requirement of the definable test coverage by the user during the parameter setting.

RI-2b. Cost-benefit for expanding the number of parameters. This research issue focuses on the expandability of the generator scripts. It partly overlaps with the section above describing the constant number of parameters. The three different cost units and the metrics for measuring each
cost unit are listed below. The changes which are necessary for the step-by-step test description instruction to execute the generation process after a new test case parameter has been added are estimated to categorize the generator scripts. The effort for adding a new test case parameter to the existing implementation of the generator approaches is identified. For this purpose the skills required for the modification as well as the risk of making a mistake during the modification are taken into account to make the two comparable. The metric for calculating the test coverage is the same as for RI-2a.

IV. ONTOLOGY-BASED TEST CASE GENERATION APPROACH

This section describes the ontology-based test case generation approach. In the first subsection, a short overview of the simulation systems underlying ontology data model is given, while the second subsection describes the test case generation suite.

A. Simulation System Data Model

The SAW system [9, 10] uses a 5-layer ontology as underlying data model. The five layers of the simulation system and their relationships among each other are illustrated in Figure 2. In the following, each layer is described shortly. The business layer prioritizes all incoming orders with respect to their due date. In the shift layer a capacity check of the resources needed for producing the ordered product takes place. In addition, the business orders get transformed to the resources needed for producing the ordered product with respect to their due date. In the job shop layer the work orders get broken down to single tasks by the production strategies. Afterwards, the responsible production strategy orders the tasks down to single tasks by the production strategies. Afterward, the generator script generates the test cases with respect to the chosen parameter setting. The EER diagram shown in Figure 3 displays all test case parameters modeled in the ontology. Additionally, some of the test case parameters are explained as examples.

A production strategy, also called scheduling strategy, serializes the tasks to process all orders received by using dispatching rules. Every strategy creates the ordered task list with respect to their specific criteria such as, for instance, earliest due date of the order, shortest processing time of the imminent operation and total processing time of the order. Production strategies can be classified into static and dynamic strategies. The static strategies order the tasks before the simulation of the shift starts. Therefore, no shift time is taken into account. Static strategies are suitable if no system disturbances can be assumed. Mostly, the system states are not predictable in a dynamic and uncertain environment. Dynamic scheduling strategies have the ability to react to system state changes. For this purpose the order of the tasks can be changed during shift runtime.

In SAW the conveyor belt components and the machine components can be simulated as fallible. The time when a failure of such a component occurs is based on a statistical distribution. The “number of failures” test case parameter is the upper limit of the number of failures which can occur during the shift. The simulation system reroutes the pallets automatically in case of a conveyor belt failure event. In case of a machine failure different failure handling strategies can apply. These strategies should help to balance both the jobs in the system and the arriving jobs while the machine is out of service [9]. There are three rerouting policies, as well as the possibility to do nothing and just queue the jobs of the
failed machine. The aim of the different failure handling strategies is to reroute the queue of the failed machine, to reroute the new arrivals to working machines or to do both.

B. Test Suite Generation

Initially, the ontology-based approach requires a GUI that can be dynamically adapted with regard to the available test case parameters. The ontology-based approach extracts the offered test case parameters, the allowed values for the parameters, and structural information from the underlying data model. Afterwards the identified information of the ontology is passed to the GUI. This data flow makes it possible to build a dynamic GUI which corresponds to the ontology at runtime. As a consequence, modifications to the ontology do not necessitate lead to manual changes neither to the GUI nor to the dynamic generic script. This feature allows adding new parameters to the ontology with tool support. Afterwards the modification is represented by the GUI without the need to change a line of code. Therefore the user does not need programming skills, neither for adding new parameters nor for configuration of the generation process. Most of the complexity is moved from the user to the implementation of the dynamic generic approach. This is why the overall test case generation process could be simplified from the perspective of the user.

On the one hand the ontology-based approach uses the underlying ontology’s structure and restrictions to ensure the consistency of the test cases. On the other hand the data ranges of the offered parameters are used to ensure the validation of the test cases. Nevertheless, the user has to manage the underlying ontology. Therefore, the user needs skills for modifying the ontology with common graphical editor tools like Protégé. Of course, for modifying an ontology users need some experience but it involves less effort than modifying a hard-coded script. Furthermore, if something goes wrong during the modification of the ontology the user will recognize it immediately in the parameter setting configuration process.

On the contrary, the user might realize that something went wrong during the modification of the static script after the simulation run took place by analyzing the simulation results. This is a frustrating experience as users usually do not know what went wrong. In that case the user is captured in a trial and error loop. Surely, these circumstances negatively affect the acceptability of the static approach.

By using the ontology-based approach, the user can always choose from all test case parameters supported by the generator script to define the parameter setting. In addition, the ontology-based approach ensures that only valid and consistent parameter settings can be defined by the user. This circumstance can be achieved with the information of the ontology. Afterwards, the ontology-based script generates test cases based on the parameter setting. At the end of the path the dynamic generic script generates the XML file.

A useful characteristic of the “3 Phases Process Model” shown in Figure 4 is the fact that the first phase and the second phase are totally independent of the domain. This is ensured as the domain specification is hidden in the ontology. The first phase communicates with the ontology and passes the element names, valid values for element instances, and information about the structure of the ontology to phase two. After this step the second phase uses the received information about the ontology to build a dynamic GUI at runtime. The user can configure the third phase based on the data model easily. The implementation of the application’s aim is located in the third phase only. In this work the third phase uses the chosen parameter setting to generate test cases. Afterwards the generated test cases are written into an XML file. This structured file can be used as input data for the simulation tool.

By using the ontology-based approach, the user can always choose from all test case parameters supported by the generator script to define the parameter setting. In addition, the ontology-based approach ensures that only valid and consistent parameter settings can be defined by the user. This circumstance can be achieved with the information of the ontology. Afterwards, the ontology-based script generates test cases based on the parameter setting. At the end of the path the dynamic generic script generates the XML file.

A useful characteristic of the “3 Phases Process Model” shown in Figure 4 is the fact that the first phase and the second phase are totally independent of the domain. This is ensured as the domain specification is hidden in the ontology. The first phase communicates with the ontology and passes the element names, valid values for element instances, and information about the structure of the ontology to phase two. After this step the second phase uses the received information about the ontology to build a dynamic GUI at runtime. The user can configure the third phase based on the data model easily. The implementation of the application’s aim is located in the third phase only. In this work the third phase uses the chosen parameter setting to generate test cases. Afterwards the generated test cases are written into an XML file. This structured file can be used as input data for the simulation tool.

V. EVALUATION

This section focuses on the evaluation concept and explains in detail the criteria for the different objectives. The objectives are the costs for the test description, the effort for implementing test case parameters, and whether the test coverage is definable by the user.

The evaluation compares the traditional static approach and the novel ontology-based approach. Both approaches are test case generator scripts, which aim at providing test cases as input data for a simulation in an automated and structured way. Firstly, the evaluation concept determines the costs for the test description as configuration duration of the parameter setting. Secondly, the effort for implementing test case parameters is determined with respect to the experience level. Lastly, the Return on Investment (ROI) is calculated assuming experience in the use of the different technologies.

1 http://protege.stanford.edu/
All results are based on data and estimates validated with domain experts.

Figure 5 shows the costs for the test description for both generator scripts with respect to the implemented number of parameters. Therefore, the time to configure the test case generation process is measured. For the empirical evaluation, we measured the time needed for test description using 1, 4 and 10 parameters for each approach and extrapolated the results as shown in Figure 5.

In order to empirically evaluate the effort needed for additional parameters, we again measured the time needed to add 1, 3 or 5 parameters and then extrapolated the results to be able give evidences for larger use case scenarios. Figure 6 outlines the effort for implementing up to 195 test case parameters. The calculated ROI has a value of 37.59 and is visualized in the figure. As a result, the higher effort for the first time implementation of the dynamic generic script pays off after the implementation of the 38th test case parameter. The calculation is done by determining the number of implemented parameters for the static script to reach the effort for implementing the dynamic generic script for the first time. Therefore, the linear equation \( y = k \times x \) of the static script has to be solved for \( k = 1.6 \) and \( y = 1\text{PM} \).

Nevertheless, the secondary criteria such as a lower risk for mistakes during the configuration phase of the test case generation process and ensuring valid and consistent test cases as output of the generation process make the use of the dynamic generic script preferable even for a small number of test case parameters.

VI. DISCUSSION

The results of the evaluation with regard to the identified research issues are listed in Table 1. As can be seen, the ontology-based approach performs better on most objectives. The ontology-based approach demands a higher effort for the first time implementation. The evaluation shows that the Return on Investment (ROI) is achieved after 38 implemented parameters with experience of using the generator scripts assumed. In addition, the user needs to obtain the necessary skills for the used generator script. For using the static script the user needs programming skills for both generating test cases and adding new parameters. The ontology-based script assumes skills in using an ontology editor for adding new test case parameters to the ontology. The dynamic generic ontology-based approach requires no specific skills for generating test cases. The percentage values for the implementation effort are based on the effort interpolation presented in the evaluation section. For the constant parameters scenario, the effort needed for implementation of the static approach is taken as base effort (100%) since it is lower than the effort needed for implementing the ontology-based approach; and vice versa for the expandable number of parameters scenario.

Table 1 shows the information in a more structured way with the specific numerical values as result of the evaluation. However, important requirements such as usability and the risk of making mistakes during the configuration phase of the test case generation process are not included in the performance metric since it is difficult to measure these requirements. Nevertheless, the ontology-based approach ensures that the parameter setting is valid and consistent by using a dynamic GUI at runtime. In addition, a mistake during the modification of the ontology would be noticed immediately after the parameter setting form is displayed in the GUI. Therefore a failure caused by wrong modification of the data model will be detected in an early stage before the generation of the test cases takes place. In contrast, for the static approach mistakes during the addition of new parameters as well as the configuration of the test case generation process will usually be detected after the simulation is finished. Furthermore, the user has to work on code level without a validation check and a consistency check of the parameter setting. In order to add a new test case parameter the user also has to modify the XML structure of the XML file to be generated on code level. This circumstance increases the testing efforts significantly. As a result, the risk of making mistakes in the static script is higher than the risk of making mistakes in the ontology-based script.

Table 1. Results of the Evaluation.

<table>
<thead>
<tr>
<th></th>
<th>Ontology-based approach</th>
<th>Static approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Description</strong></td>
<td>high-level</td>
<td>low-level</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>260%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Test Coverage</strong></td>
<td>is defineable</td>
<td>is not defineable</td>
</tr>
</tbody>
</table>
VII. CONCLUSION AND FURTHER WORK

High test coverage is essential for testing the performance of simulation systems not only in the production automation domain. Test case generators provide such test cases as input data for the simulation in an automatic and structured way. Many solutions for test case generators use a static approach which is difficult to expand. Mostly, these solutions offer an unsatisfying usability since only a low-level test description is supported. In this work, we discussed the test case generation process and developed an ontology-based approach based on an available static one to meet the criteria of a high-level test description, lower effort for implementing additional test case parameters, and a definable test coverage.

This paper identified weaknesses of the traditional static approach and proposed and evaluated a solution to address these weaknesses. After the implementation of the new approach the effectiveness of both approaches was compared. The two generator scripts were evaluated with respect to three objectives – the test description, implementation, and the test coverage. In addition, two different application domains, i.e. a “constant number of parameters” and the “expandability of parameters” were taken into account. The ontology-based approach, however, performs very well on most objectives as the result of the evaluation shows. It could be shown that the new approach only lacks the implementation objective in the “constant number of parameters” domain. The reason for this limitation is based on the higher effort for the first establishment of the “3 Phases Process Model”. Once the dynamic generic script is running it does not have to be maintained anymore even if the number of test case parameters varies over time. A further advantage of the realization of the “3 Phases Process Model” is that the first phase and the second phase are totally independent of the domain as the domain specification is hidden in the ontology.

Future Work. Future research will include the feedback of the simulation results into the ontology. The basic idea of the feedback into the ontology is that the cyclic test case generation process has not to start from scratch each time. An important fact is that the feedback is not limited to the simulation results since the results depend on the test cases and the assembly line. Another future research topic will be the more excessive usage of ontology-based reasoning to support the deduction of test cases. At the moment the ontology is used for building a dynamic GUI at runtime to ensure that the parameter setting is valid and consistent. In addition, the structure of the XML file corresponds to the structure of the ontology. However, the deduction of test cases is probably more efficient than generating test cases as combinatorial possibilities of the parameter setting especially since the ontology is a knowledge-based system and therefore enables to infer from the stored fact base.

ACKNOWLEDGMENT

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria. This work has been partially funded by the Vienna University of Technology, in the Complex Systems Design and Engineering Lab.

REFERENCES