VALID 2016

The Eighth International Conference on Advances in System Testing and Validation Lifecycle

ISBN: 978-1-61208-500-5

August 21 - 25, 2016

Rome, Italy

VALID 2016 Editors

Xinli Gu, Huawei Technologies Co., Ltd., USA

Claus-Peter Rückemann, Leibniz Universität Hannover /

Westfälische Wilhelms-Universität Münster /

North-German Supercomputing Alliance (HLRN), Germany

VALID 2016

Forward

The Eighth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2016), held on August 21 - 25, 2016 in Rome, Italy, continued a series of events focusing on designing robust components and systems with testability for various features of behavior and interconnection.

Complex distributed systems with heterogeneous interconnections operating at different speeds and based on various nano- and micro-technologies raise serious problems of testing, diagnosing, and debugging. Despite current solutions, virtualization and abstraction for large scale systems provide less visibility for vulnerability discovery and resolution, and make testing tedious, sometimes unsuccessful, if not properly thought from the design phase.

The conference on advances in system testing and validation considered the concepts, methodologies, and solutions dealing with designing robust and available systems. Its target covered aspects related to debugging and defects, vulnerability discovery, diagnosis, and testing.

The conference provided a forum where researchers were able to present recent research results and new research problems and directions related to them. The conference sought contributions presenting novel result and future research in all aspects of robust design methodologies, vulnerability discovery and resolution, diagnosis, debugging, and testing.

We welcomed technical papers presenting research and practical results, position papers addressing the pros and cons of specific proposals, such as those being discussed in the standard forums or in industry consortiums, survey papers addressing the key problems and solutions on any of the above topics, short papers on work in progress, and panel proposals.

We take here the opportunity to warmly thank all the members of the VALID 2016 technical program committee as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and efforts to contribute to VALID 2016. We truly believe that thanks to all these efforts, the final conference program consists of top quality contributions.

This event could also not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the VALID 2016 organizing committee for their help in handling the logistics and for their work that is making this professional meeting a success. We gratefully appreciate to the technical program committee co-chairs that contributed to identify the appropriate groups to submit contributions.

We hope the VALID 2016 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in system testing and validation. We also hope Rome provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful historic city.

VALID 2016 Advisory Committee

Andrea Baruzzo, University of Udine / Interaction Design Solution (IDS), Italy
Cristina Seceleanu, Mälardalen University, Sweden

Mehdi Tahoori, Karlsruhe Institute of Technology (KIT), Germany
Mehmet Aksit, University of Twente - Enschede, The Netherlands
Amir Alimohammad, San Diego State University, USA
Hironori Washizaki, Waseda University, Japan
Radu Grosu, Vienna University of Technology, Austria
Shlomo Mark, SCE - Shamoon College of Engineering, Israel
Stefan Wagner, University of Stuttgart, Germany

VALID 2016 Research Institute Liaison Chairs

Alexander Klaus, Fraunhofer Institute for Experimental Software Engineering (IESE), Germany
Kazumi Hatayama, Gunma University, Japan
Vladimir Rubanov, Institute for System Programming / Russian Academy of Sciences (ISPRAS), Russia
Tanja Vos, Universidad Politécnica de Valencia, Spain

VALID 2016 Industry Chairs

Abel Marrero, Bombardier Transportation Germany GmbH - Mannheim, Germany
Juho Perälä, Bitfactor Oy, Finland
Xinli Gu, Huawei Technologies Co., Ltd., USA
Philipp Helle, Airbus Group Innovations, Germany

VALID 2016

Committee

VALID Advisory Committee

Andrea Baruzzo, University of Udine / Interaction Design Solution (IDS), Italy
Cristina Seceleanu, Mälardalen University, Sweden
Mehdi Tahoori, Karlsruhe Institute of Technology (KIT), Germany
Mehmet Aksit, University of Twente - Enschede, The Netherlands
Amir Alimohammad, San Diego State University, USA
Hironori Washizaki, Waseda University, Japan
Radu Grosu, Vienna University of Technology, Austria
Shlomo Mark, SCE - Shamoon College of Engineering, Israel
Stefan Wagner, University of Stuttgart, Germany

VALID 2016 Research Institute Liaison Chairs

Alexander Klaus, Fraunhofer Institute for Experimental Software Engineering (IESE), Germany
Kazumi Hatayama, Gunma University, Japan
Vladimir Rubanov, Institute for System Programming / Russian Academy of Sciences (ISPRAS), Russia
Tanja Vos, Universidad Politécnica de Valencia, Spain

VALID 2016 Industry Chairs

Abel Marrero, Bombardier Transportation Germany GmbH - Mannheim, Germany
Juho Perälä, Bitfactor Oy, Finland
Xinli Gu, Huawei Technologies Co., Ltd., USA
Philipp Helle, Airbus Group Innovations, Germany

VALID 2016 Technical Progam Committee

Fredrik Abbors, Åbo Akademi University, Finland
Jaume Abella, Barcelona Supercomputing Center (BSC-CNS), Spain
Mehmet Aksit, University of Twente - Enschede, The Netherlands
Amir Alimohammad, San Diego State University, USA
Giner Alor Hernandez, Instituto Tecnologico de Orizaba - Veracruz, México
María Alpuente, Technical University of Valencia (UPV), Spain
César Andrés Sanchez, Universidad Complutense de Madrid, Spain
Aitor Arrieta, Mondragon Unibertsitatea, Spain
Selma Azaiz, CEA List Institute - Gif-Sur-Yvette, France
Cesare Bartolini, ISTI - CNR, Pisa, Italy
Andrea Baruzzo, University of Udine / Interaction Design Solution (IDS), Italy
Armin Beer, Beer Test-Consulting, Austria
Serge Bernard, LIRMM, Franmce

Massimo Marchiori, University of Padua / European Institute for Science, Media and Democracy, Italy
Shlomo Mark, SCE - Shamoon College of Engineering, Israel
Oded Margalit, IBM CCoE, Israel
Abel Marrero, Bombardier Transportation Germany GmbH - Mannheim, Germany
Maria K. Michael, University of Cyprus, Cyprus
Brian Nielsen, Aalborg University, Denmark
Roy Oberhauser, Aalen University, Germany
Johannes Oetsch, Vienna University of Technology, Austria
Nguena-Timo Omer-Landry, LaBRI/University Bordeaux 1, France
Yassine Ouhammou, ENSMA / LIAS-lab, France
Sachin Patel, Tata Consultancy Services, India
Bernhard Peischl, Technische Universität Graz, Austria
Juho Perälä, Bitfactor Oy, Finland
Mauro Pezzè, Università della Svizzera Italiana, Switzerland
Miodrag Potkonjak, Univeristy of California, Los Angeles (UCLA), USA
Paolo Prinetto, Politecnico di Torino, Italy
Henrique Rebêlo, Federal University of Pernambuco, Brazil
Eike Reetz, University of Applied Sciences Osnabrück, Germany
Patrick Rempel, Technische Universität Ilmenau, Germany
Oliviero Riganelli, University of Lugano, Switzerland
Giedre Sabaliauskaite, Singapore University of Technology and Design, Singapore
Goiuria Sagardui Mendieta, Mondragon University, Spain
Hiroyuki Sato, University of Tokyo, Japan
Christian Schanes, Vienna University of Technology, Austria
Cristina Seceleanu, Mälardalen University, Sweden
Nassim Seghir, University of Oxford, UK
Sergio Segura, University of Seville, Spain
Vipul Shah, Tata Consultancy Services, India
Lwin Khin Shar, University of Luxembourg, Luxembourg
Julien Signoles, CEA LIST, France
Dimitris Simos, SBA Research, Austria
Mehdi B. Tahoori, Karlsruhe Institute of Technology (KIT), Germany
Nur A. Touba, University of Texas - Austin, USA
Spyros Tragoudas, Southern Illinois University Carbondale, USA
Andreas Ulrich, Siemens AG, Germany
Jos van Rooyen, Bartosz ICT, Netherlands
Miroslav N. Velev, Aries Design Automation, USA
R. Venkatesh, Tata Consultancy Services, India
Bart Vermeulen, NXP Semiconductors - Eindhoven, The Netherlands
Arnaud Virazel, Université Montpellier 2 / LIRMM, France
Vincent von Hof, UniversityofMünster, Germany
Tanja E. J. Vos, Universidad Politécnica de Valencia, Spain
Stefan Wagner, University of Stuttgart, Germany
Hironori Washizaki, Waseda University, Japan
Kristian Wiklund, Ericsson AB / Mälardalen University, Sweden
Robert Wille, Johannes Kepler University Linz, Austria / DFKI GmbH, Germany
Lina Ye, CentraleSupélec, Gif sur Yvette, France / LRI, Univ. Paris-Sud 11, France

Cemal Yilmaz, Sabanci University - Istanbul, Turkey
Zeljko Zilic, McGill University, Canada

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Automatic Job Generation for Compiler Testing Testing of Generated Compiler

Ludek Dolihal

Department of Information Systems,
Faculty of Information Technology,
Brno University of Technology
Brno, Czech Republic
Email: idolihal@fit.vutbr.cz

Tomas Hruska

Department of Information Systems,
Faculty of Information Technology,
Centre of Excellence IT4Innovations,
Brno University of Technology
Brno, Czech Republic
Email: hruska@fit.vutbr.cz

***Abstract*—Hand in hand with the design of the new core goes the need for thorough testing, which is highly automated. Tools for hardware/software codesign allow very fast design of the new core and generation of the complete tool-chain. The tool-chain that is used for the programming of the newly developed core and also descriptions of the core in various languages are generated automatically and it is the role of automatic testing to ensure that there is no regression. As the pace of the development is high also the techniques for the testing must be able to cover the testing in very short period of time. In this article, we will introduce the generator of jobs for the continuous integration server Jenkins. Through the job generation we reach the higher level of automation of the whole process of the core development and also speed up the process of testing.**

Keywords–Compiler testing; Continuous integration; Hardware/Software codesign; Test generation.

I. INTRODUCTION

Each software product must be tested. In the article, we will address the testing of tools for hardware/software codesign [1]. Hardware/software codesign deals with the design of the new Application Specific Instruction-set Processors (ASIPs). Such kind of systems can be found in wide variety of devices such as network routers or printers.

The production of ASIPs is growing as the need for the small and low power cores that can be used for specific purposes is still bigger. For example Texas Instruments released 4 new cores in the last 6 months [2]. Hence, this area is extremely important. The development of today's ASIPs must be done in a very short period of time. To do so, it is common to use the tools for hardware/software codesign. Some Architecture Description Language (ADL) is usually in the core of such systems [3]. The development is done in a modern Integrated Development Environment (IDE) that allows the designer to generate all the necessary tools, such as compiler, assembler and simulator. In the same environment the user is able to perform any step needed for the development of the core, such as simulation or profiling.

Such kind of development environment shortens the development time significantly [4]. However, each piece of software contains errors, and environments for hardware/software codesign are not an exception. Some of the tools are more error prone than others. From our point of view, the Software Development Kit (SDK), and especially the compiler, are the most critical parts. Because in case we have error in the compiler, the compiled program does not have to work properly. If the compiler does not work correctly, it is crucial to discover the error in the shortest possible time. For this purpose we use a continuous integration server to run testing jobs.

The continuous integration server will be used for execution of jobs that will be automatically generated. We will introduce the generator of the jobs that will bring the higher level of automation and also speed up the process of testing.

The paper is structured as follows: Section II, gives the short overview of the Lissom project. In Section III, we explain the continuous integration process. Section IV discuss the related work. In Sections V and VI, we explain the generator of the testing jobs and achieved results. Finally, in Section VII, we present the conclusions.

II. LISSOM PROJECT

In this section, we will describe the Lissom research project [5], which creates background for the testing methods that are described in this article. The Lissom project started in 2004 and is located at the Brno University of Technology, Faculty of Information Technology, Czech Republic.

The Lissom project has two main areas of interest. The first one is the development of the Architecture Description Language (ADL) called CodAL, which serves for the ASIP description. The description of the language can be found in detail here [6].

The second scope of the project is the generation of the full tool-chain from the description in the ADL CodAL. The generated tool-chain contains a C compiler, assembler, linker, disassembler, two types of simulators (instruction and cycle accurate), the debugger and few more tools. As the language is designed for description of the ASIPs, the scale of processors that can be described, without making any modifications to the language, is large.

However, there are also other ways how to utilise such language. One of them is to use the language for description of architectures that already exist. Hence, it is possible to model in the CodAL language architectures such as MIPS [7], ARM [8], RISC-V [9] and many others. The generated tool-chain or just separate tools can be used as a replacement of existing tools in case they are not in good shape. This gives large

 ISBN: 978-1-61208-500-5

possibilities in case the core is upgraded and new tool-chain is needed. For certain cores also, some of the tools might be missing and by designing the given architecture in ADL the missing tool can be easily generated.

All the tools are generated from the description in the CodAL language. In the beginning, the model in the CodAL language is validated and compiled. The result of the compilation is the XML representation of the model. The XML format was chosen intentionally as there are other tools that use this form and there is also large number of generators and parsers working over XML.

Once the XML is created there are two tools working over it. The first tool is the tool-chain generator, also called toolsgen. The second one is the semantics extractor or semextr.

The tool-chain generator produces tools, such as simulator, assembler, debugger and many others. The tools that are generated by the tool-chain generator consist of two types of files. Both types of files are compiled and linked together.

1) Files that are platform independent are the same for all architectures. Into this category falls user interfaces with parsers of the command line arguments, or in case of profiler the generation of the graphical output.
2) Automatically generated files that contain the platform dependent information. Into this category belong the instruction decoders in the simulators or assembler printer in the C compiler.

The second tool is the semantics extractor. The execution of the extractor is the prerequisite for the compiler generation. Moreover, there are other tools that use the outputs of the semantics extractor, such as Quick EMUlator (QEMU) or documentation generator and also decompiler that is described in the thesis [10].

The main role of the semantics extractor is to extract the assembler syntax, binary encoding and semantics of each instruction described in the model.

The development of the new core in the Electronic Design Automation (EDA) tool [11] can be done very swiftly. The experienced designer can create an instruction accurate model of a core in a few hours. Modification of a core can be created even faster. It is very simple to add some instructions and/or create larger register field for example. This process can give birth to the versions of the processors that can be optimized for speed, size of the code or power consumption.

All such variants of the core should be tested, so there is a need for simple generation of the testing infrastructure. Hence we need a generator of the jobs, that will perform the testing. We need to speed up the whole process and reduce the amount of manual work.

III. Continuous Integration

In this section we will describe the Continuous Integration (CI) and introduce the job format, which we will use in the further sections.

The main idea of continuous integration [12] is to avoid the integration problems in the later stages of the development. The developers are encouraged to merge with the main development line several times a day and execute the tests over the merged line. By this approach they are encouraged to keep an eye on the integration continually.

The technique was mentioned for the first time by Grady Booch [13], and was called Booch method. Later it was adopted by extreme programmers and resulted in performing an integration in once or more times a day.

Today, the continuous integration servers are used in every larger company. The most widespread CI server is called Jenkins [14]. Jenkins is an open source automation server that can provide not only continuous integration but also continuous deployment. It uses the system of plugins to enhance the basic functionality. Nowadays, there are plugins available all the Version Control Systems (VCS) as well as plugins for visualisation of pipelines etc.

The basic block of the Jenkins server is called a job. The main action for every job is the execution. The job has the data that are typically taken from the VCS and action that is usually execution of some script.

The jobs are stored at the Master server. Master server is the computer that keeps the installation of the Jenkins and all the jobs are kept here. In case of the single master installation. The job is represented by a file in the XML format that is stored in the given folder on the Master server. The format in the markup language is in our case a great advantage as there is a lot of generators of the XML and also there are other tools that can work with the description.

A. Jenkins job format

Jenkins supports several types of jobs. The basic ones are the freestyle project and the multiconfiguration project. The main difference between the two is the fact that multiconfiguration project can be executed on multiple machines. There are also special types of jobs that are tied to the various plugins. There is a maven job, external job or various views.

Below we listed the basic description of the multiconfiguration job, as it is the job, which we are the most interested in. Though we need to work with the other job types as well, the configuration of the basic kind of job will be suffictient for demonstration purposes now.

```
<?xml version='1.0' encoding='UTF-8'?>
<matrix-project plugin="matrix-project@1.4
    ">
  <actions/>
  <description></description>
  <keepDependencies>false</
      keepDependencies>
  <properties>
    <com.sonyericsson.rebuild.
        RebuildSettings
     plugin="rebuild@1.22">
       <autoRebuild>false</autoRebuild>
     </com.sonyericsson.rebuild.
        RebuildSettings>
    <hudson.model.
        ParametersDefinitionProperty/>
  </properties>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false
```

```
  </blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false
  </blockBuildWhenUpstreamBuilding>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <axes>
    <hudson.matrix.LabelAxis>
      <name>label</name>
      <values>
        <string>CentOS-6.5-32</string>
      </values>
    </hudson.matrix.LabelAxis>
  </axes>
  <builders>
    <hudson.tasks.Shell>
      <command>echo \$(pwd)</command>
    </hudson.tasks.Shell>
  </builders>
  <publishers/>
  <buildWrappers/>
  <executionStrategy class="hudson.matrix.
DefaultMatrixExecutionStrategyImpl">
    <runSequentially>false</
        runSequentially>
  </executionStrategy>
</matrix-project>
```

On the second line, we can see that it is the matrix project, which means that it can deploy multiple axis, and one of them is the configuration of the nodes. For simplicity the job does not download any data from the VCS. Another important tag is the one called *axes*. This tells us that this job is built only on one node called CentOS-6.5-32. It is important to note that this job does not have parameters. If it had, the parameters would be visible in the top of the configuration.

There are also sections *builders* and *publishers*. Section *builders* says that there is the shell script executed, and only command it runs it the *echo $(pwd)*. The job has no results, hence, the part *publishers* is empty. The execution strategy is default. It is important to know, how the configuration of the job looks like as we will work with the representation in the later sections.

IV. Related Work

Let us have a look at the current development at the field of the job generation. We can distinguish two types of solutions. The are tools in Jenkins that were designed for this purpose and then there are several works that try to deal with the problem of job generation outside of the Jenkins environment.

First we will have a look at the solutions inside the Jenkins. One of them is the template plugin [15]. Via the template project plugin the user can set up an template project containing the settings the user want to share. Is is possible to set for example the VCS repositories that are common for the jobs or the script that should be executed and so on. Then it is possible to create inside the Jenkins another project from the created template. So the generation has to be performed manually by using the template several times. Hence, the possibilities of the automation are limited.

Other possibility provided by Jenkins server itself is the job generator plugin [16]. This plugin is based on template, which is the job itself and the parameters, which can be global or local. This plugin is very powerful in combination with other plugins such as plugin for conditional resolution. However, it has limitations in form of what types of jobs can be generated and it can not use time triggers. Moreover, it is very difficult to generate more complex jobs. The hierarchy and conditions can become very complex and the whole process is error prone. We also did not find a way how to set the desired nodes in the multiconfiguration project.

Now we will mention several approaches that try to deal with job generation outside the Jenkins environment. The interesting ideas are proposed in the article at Jenkins User Conference [17]. The article deals with the automation of testing in the area of robotics. The author uses combination of various Jenkins plugins for packaging and static analysis. Nevertheless, the process of build and testing is very complicated and hardly maintainable. The author proposes use of Domain Specific Language (DSL) for specification of the informations and then generation of the Jenkins jobs. It seems that the author just uses Jenkins for the build. However, the system seems to be slow and has problems with synchronisation of the jobs. Also there are problems with the graphical side of the solution.

Some interesting ideas connected with the job generation are in the Shaw article [18]. The article also introduces the possibility of job generation from the templates and use of the Jenkins command line interface. Nevertheless, the article does not provide any examples of the templates or scheme how the system works.

Above we have mentioned several possibilities in the area of job generation. None of the approaches that were mentioned suits our needs. In our project we need to generate all kinds of jobs, as it is crucial to test the various aspects of the newly developed core. This includes the tests of various features that can be tied to very specific kinds of jobs. The approach mentioned in [17] seems to be interesting. For our use it appears to be too cumbersome. The lightweight solution with the command line interface would suit our needs better.

V. Job Generation

The main task that we need to address is the generation of the various jobs, which will ensure the complex testing of the core. As we plan to use the whole system also from the command line, we wanted to avoid the graphical interface, at least in the first version of the project. We may add the graphical interface in the later versions, but we need to keep the command line interface, as we would like to use the solution from the command line. This is also one of the reasons, why we can not use the plugins provided by Jenkins. They have very poor documentation and are primary focused for usage via the web interface.

The basic scheme of our system is illustrated in Figure 1. We can see that the whole system consists of just a few steps. The first part of the system is the sniffer. It works over the git repository in our case. Once the generation is triggered the job generator uses the templates to generate corresponding jobs. We will now give more detailed description of the aforementioned parts.

A. Sniffer

We called this part of the generation process a sniffer as it sniffs in the git repository for a new branches. The main

 ISBN: 978-1-61208-500-5

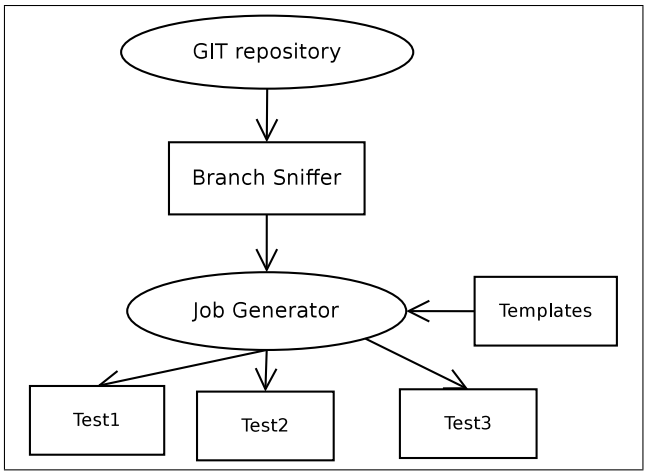

Figure 1. Scheme of the system.

role of the sniffer is to detect the creation of the new branch in given git repository and trigger the generation. The whole system is designed in a way that the sniffer can be replaced by a different component. In the future, we would like to add the support for other VCS. It also does not have to be present at all and can be completely removed. The generator can be started by a different tool, if it is compatible with the defined interface.

Though currently the role of the sniffer is to notify that the new branch has been created and deliver this information to the job generator. The sniffer has no further intelligence and the whole system is designed in a way that all the decisions should be made in the generator itself. In the latest version the sniffer has the shape of the unix script that is executed repeatedly by the operation system.

B. Templates

The second input into the job generator are the templates. We have various kinds of templates as we need to test various parts of the newly developed core. The main areas that has to be covered by test job generation are:

- compiler testing,
- functional verification,
- assembler testing,
- tools generation.

Please note that these are just the areas that needs to be covered, not the jobs. Under each domain there is a variety of jobs that are generated and later on executed. There is usually just one template per domain, just in case of the functional verification we need to have several templates, as this area is very vast and we were not able to stick to just one template.

As far as the templates itself are concerned, they are very simple and do not keep any intelligence. The intelligence, for example the name of the node, where the job will run is kept in the generator. The templates are in the XML format and are similar to the example in Section III. Consider for example that we want to generate the name of the node, where our job will be executed. The corresponding part in the template will have the following form:

```
<string>@NODE_NAME@</string>
```

C. Job generator

Now, when we described the inputs of the generator we will move to the generator itself. The job generator consists of several parts that are pictured in Figure 2.

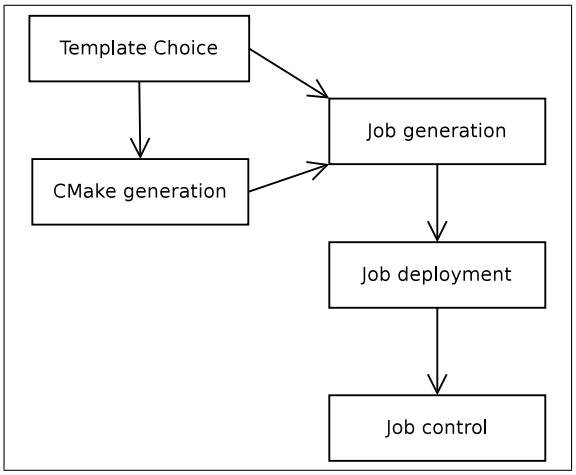

Figure 2. Scheme of the generator

We decided to implement the generator in Python language because it allows very fast development and the code is very easy to read and modifications are simple.

One of the first steps of the generation is the template selection. This part of the generator works over the configuration file that is present at the specific directory in the model branch that should be tested. We have proposed a simple format of the configuration file, which specifies the tested features. The other possibility we have is to automatically detect what features should be tested but we have chosen the configuration file, because some of the features can not be automatically detected. From the specification file we are able to determine what templates should be used. The specification file has two major tasks:

- define features that should be tested,
- specify parameters for the generators.

However, the automatic detection of the features that should be tested was not completely abandoned. The detection is present, but plays only the supplementary part.

Once the phase of the templates selection is finished we need to generate the CMake files that will fill into the templates the desired information. The generated CMake files are template specific as each template has different fields. Currently we generate one CMake file per template and we do so in the separate directories.

From the two above mentioned inputs, we can generate the job. The job generation is in fact just insertion of the data into the templates. We decided to do this via CMake, because it is one of the cleanest ways to do so. The most frequent facts that are generated are the following:

- branch used for testing,
- node, where the job is executed,

 ISBN: 978-1-61208-500-5

- bash script and the parameters,
- job name and view, where the job is placed.

The above mentioned information can be determined in the following way. The branch is one of the input parameters. It is delivered by the sniffer, but can be also delivered different way, it can be for example specified by the user.

The script that is executed could be the part of the template, however, this would increase the number of the templates significantly. Hence, we try to determine the name of the script. This could be done based on the information from the configuration. Some of the scripts may have variable number of parameters, but this we are able to determine from the directory structure of the model. Here we can see the supplementary part of the automatic detection.

The job name and view, where the job should be places, are also determined from the configuration file and repository name. We also plan in the future to use directory plugin in our installation, however, this should not be a problematic step.

The most complicated task is the selection of the correct node, where the job should be executed. There are certain jobs that can be executed only on the specific set of nodes. Typically this is true for the jobs that perform tests of the functional verification or tests of the synthesis. We have a special groups of nodes and special templates with the predefined sets of nodes. Nevertheless, for the majority of jobs we do not have to solve such issues. We keep a simple table of nodes, which is divided into the sections, which define what nodes are used for specific jobs. We choose the jobs with the smallest number of assigned jobs and optionally we modify the assigned value by hand.

There are also other information that can be filled into the template. But the four above mentioned are the most common ones. We have the predefined default values for all the parameters that would suit the most cases.

Very often we generate the parameters into the templates.

They are stored in the *parameters* section and later this parameters are used in the *builders* section. However, there are also parameters that are node dependent, or are defined globally in the Jenkins.

Very often the generated job needs to use the artifacts from the other jobs. Nevertheless, we try to keep the generator as lightweight as possible and do not want modify other jobs. The compatibility in this case is assured by the wildcards, and the name of the new job must fit into the wildcard. For example if the job is named Test-compiler-xxx the wildcard can be Test-compiler-*.

Once we have generated the jobs that are needed for the testing of the newly developed branch, we have to upload these jobs to the CI server. For this purpose we use the Jenkins command line interface that performs the job upload and also registers the job.

VI. RESULTS

With the current implementation of the simple job generator we have performed several tests. We have tried to generate the set of tests that are typical for our project. The tests are divided into two sets. The basic set consists of tests that test compiler and assembler and full set adds also tests for functional verification. The templates that are needed for generation of such tests were added into the template set. The basic set consists of three jobs and full set consists of 12 jobs. We have set the polling time to 6 minutes, so every 6 minutes is the VCS server polled for the new branches.

The times needed for the generation are summarised in the following table. We have performed 10 different runs: five for basic set of tests and five for the full set of tests. The last run was triggered manually.

TABLE I. COMPARISON OF GENERATION TIMES.

Run	*Basic set*	*Full set*
1	124s	516s
2	248s	524s
3	194s	212s
4	150s	317s
5	91s	412s
Manual run	42s	178s

We can see in the Table I that the generation of the three jobs takes 42 seconds, which gives exactly 14 seconds per job. When we try to generate the full set of 12 jobs, it takes 178 seconds. That is approximately 15 seconds per job. All of the jobs we generate are multiconfiguration jobs. The generation times vary for the basic set from 91 to 248 seconds. That is perfectly accurate, as the delay caused by the front end is up to 360 seconds. The generation of the full set is also affected by the front end and should be from 178 seconds up to 538 seconds. Our measurements confirm that.

We have also tried to create the jobs manually. The group that created the jobs consisted of two persons. We tried to create the basic set of testing jobs, and then the full set of jobs. The basic set of tests include the generation of three jobs and covers the compiler and assembler. The full set of jobs contains also jobs for verification. Together this set contains 12 jobs. Hence, the sets are the same as in the previous measurement.

TABLE II. COMPARISON OF CREATION TIMES.

Method	*Basic set*	*Full set*
Lissom Generator	182s	499s
Manual creation	486s	2197s

In the Table II we can see that the manual creation of the jobs was very slow in comparison with the generator. Especially in case we have to create the set of 12 jobs the task was very time consuming.

The last comparison we made was with the Jenkins job generator plugin. We used the Jenkins server in version 1.656 and the plugin was in the version 1.22. The Jenkins server was running on the server with the 4 cores Intel i5 and has 8 GB of the memory. The same set of jobs as above was generated.

TABLE III. COMPARISON OF CREATION TIMES.

Method	*Basic set*	*Full set*
Lissom Generator	103s	361s
Jenkins generator plugin	148s	839s

The results are gathered in the Table III. It is clear, that Lissom generator was fastest in both tested cases. However, in case of generation of just three jobs, the times were comparable. And in case of maximal 360 seconds delay, the Jenkins job generator can be even faster. Nevertheless, in case

 ISBN: 978-1-61208-500-5

of generation of the big set the generator had clear advantage even in case of maximal delay caused by the front end. Moreover when compared to the times without delay, the speed of Lissom job generator can not be matched.

Other advantage of the job generator is the fact that it is very lightweight and can be used for any kind of jobs. This largely depends on the templates that will be created.

VII. Conclusion

In this paper, we sketched the simple generator of the Jenkins jobs that would suite our needs in the Lissom project. We need the generator that can be started by various ways is lightweight and can generate all kinds of jobs. This was one of the basic requirements that was not met by any plugin that is currently available for Jenkins. We also wanted the tool to be at least partly independent of Jenkins as it is not rare that the plugins do not cooperate well.

The current implementation of our generator is dependent just on the internal representation of the job. This is not a problem, as it is very simple to deploy new templates. At the same time, the internal job representation is not likely to change as it would imply the changes in all plugins currently used by Jenkins.

We also put the generator under the tests and the gathered results are very positive. As far as the speed of the generator is concerned it can not be matched by any tool that is currently available. In the future we would like to add to the generator also other functionality such as work with the directory plugin and also ability to register the jobs for artifact download.

We created a tool that helps us to generate new sets of test every time the new core is developed. It gives us the higher level of test automation.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602.

References

[1] G. De Micheli and W. Rolf, E.and Wolf, Readings in Hardware/Software Co-design. Morgan Kaufmann, 2001, ISBN: 9781558607026.

[2] “Texas Instruments,” http://www.ti.com/general/docs/newproducts.tsp (July 2016), 2016.

[3] F. Oquendo, “π-adl: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures,” ACM SIGSOFT Software Engineering Notes, vol. 29, no. 3, 2004, pp. 1–14.

[4] J. Teich, “Hardware/software codesign: The past, the present, and predicting the future,” Proceedings of the IEEE, 2012.

[5] Lissom, “Project Lissom Webpages,” http://www.fit.vutbr.cz/research/groups/lissom/ (August 2014), 2014.

[6] K. Masarik, “System for hardware-software codesign,” Master’s thesis, Faculty of Information Technology, Brno university of Technology, 2008.

[7] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, “A 300 mips/w risc core processor with variable supply-voltage scheme in variable threshold-voltage cmos,” in Custom Integrated Circuits Conference, 1997., Proceedings of the IEEE 1997. IEEE, 1997, pp. 587–590.

[8] B. Smith, “Arm and intel battle over the mobile chip’s future,” Computer, vol. 41, no. 5, 2008, pp. 15–18.

[9] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, “The risc-v instruction set manual, volume i: Base user-level isa,” EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62, 2011.

[10] J. Kroustek, “Retargetable analysis of machine code,” Master’s thesis, Faculty of Information Technology, Brno university of Technology, 2014.

[11] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Electronic design automation: synthesis, verification, and test. Morgan Kaufmann, 2009.

[12] M. Fowler and M. Foemmel, “Continuous integration,” Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf, 2006, p. 122.

[13] G. Booch, Object-oriented Analysis and Design with Applications (2Nd Ed.). Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994.

[14] Jenkins, “Jenkins website,” https://jenkins.io/ (July 2016), 2016.

[15] “Template Project Plugin,” https://wiki.jenkins-ci.org/display/JENKINS/Template+Project+Plugin (July 2016), 2016.

[16] “Job Generator Plugin,” https://wiki.jenkins-ci.org/display/JENKINS/Job+Generator+Plugin (July 2016), 2016.

[17] F. Lier, J. Wienke, and S. Wrede, “Jenkins for flobi–a use case: Jenkins & robotics,” in Jenkins User Conference, 2013.

[18] K. Shaw, “Generating New Jenkins Jobs From Templates and Parameterised Builds,” http://www.blackpepper.co.uk/generating-new-jenkins-jobs-from-templates-and-parameterised-builds/ (July 2016), 2012.

 ISBN: 978-1-61208-500-5

SAT-Based Testing of Diagnosability and Predictability of Centralized and Distributed Discrete Event Systems

Hassan Ibrahim and Philippe Dague

LRI, Univ. Paris-Sud, CNRS, Univ. Paris-Saclay
Orsay, France
Email: firstname.lastname@lri.fr

Laurent Simon

LaBRI, Univ. Bordeaux, CNRS
Bordeaux, France
Email: lsimon@labri.fr

***Abstract*—In the general framework of safety analysis, diagnosability of a system, i.e., the guarantee to surely identify any fault in a finite delay after its occurrence, based on the available observations, is a key property to be verified at design stage. Diagnosability analysis of discrete event systems received a lot of attentions in the past twenty years, firstly in the centralized, then in the distributed case. In particular, a satisfiability-based approach was proposed in 2007 in the centralized case. We extend in this work this approach to cover also distributed discrete event systems, by handling both observable and unobservable synchronous communication events at the same time. Then, we adapt the method to analyze, in both centralized and distributed cases, fault predictability, a stronger property than diagnosability, which guarantees that any fault can be correctly predicted before its occurrence, based on observations. We provide experimental results for both diagnosability and predictability.**

Keywords–Discrete Event Systems; Distributed Systems; Diagnosability; Predictability; Satisfiability.

I. INTRODUCTION

Nowadays, there is an increasing interest to ensure from the design stage of a system that partial observations given by the sensors will allow a precise diagnosis of potential faults that could occur in that system, once built. This will actually save high costs of adding new sensors for this task during the operating mode of the system. This raises the problem of diagnosability and of predictability which are essential properties to verify while designing the system model. Once this verification has been done (possibly thanks to a modification of the system model), both the system and its diagnoser or predictor (which can be automatically derived from diagnosability or predictability analysis) can be built with a guarantee of correctness and precision of the diagnosis, at least for those faults anticipated at design stage. Diagnosability is a property that determines the possibility to distinguish any possible behavior in the system with a given fault from any other behavior without this fault. A fault is diagnosable if it can be surely identified from the partial observation available in a finite delay after its occurrence. A system is diagnosable if every possible fault in it is diagnosable. Predictability is similarly an important system property, stronger than diagnosability, that determines at design stage whether a considered fault can be correctly predicted before its occurrence, based on available observations. If a fault is predictable, the fault management system can be designed to warn the operator, to halt the system or to take preventive measures.

The main difficulty in diagnosability and predictability checking is related to the states number explosion. Methods to cope with this problem and scale the studied system size in the case of discrete event systems (DES) resort to adopting a succinct representation of the system or a distributed modeling and to use powerful checking tool. For these reasons, we extend in this work to distributed discrete event systems (DDES) the succinct representation and the use of a satisfiability (SAT) solver introduced in [1] for centralized DES. Then, we adapt the SAT-based method to predictability analysis, in both centralized and distributed cases.

The paper is structured as follows. We first present related works in section II. In section III, we introduce the system transition models for DES and recall the traditional definition of diagnosability in those models and the state of the art of encoding it as a satisfiability problem in propositional logic. Then, in section IV, we present our first contribution, an extension of this SAT-based diagnosability analysis to DDES with observable and unobservable synchronous communication events in the same model, and give experimental results of this extension. Then, in section V, after having recalled the usual definition of predictability in DES, follows our second contribution, the encoding of this property as a satisfiability problem for both DES and DDES, and presentation of experimental results. Finally, in section VI, we conclude and outline our perspectives for future work.

II. SELECTION OF RELATED WORKS

The first introduction to the notion of diagnosability was by [2], who gave its formal definition (see Def. 2 in section III) and studied it for labeled transition systems (LTS) by constructing a deterministic diagnoser to test it. However, this approach is exponential in the number of states of the system, which makes it impractical. In order to overcome this limitation, [3] introduced the *twin plant* approach, a structure built by synchronizing on their observable events two identical instances of a nondeterministic fault diagnoser. Then a so-called *critical path* is searched in this structure, i.e., a path with an observed cycle made up of ambiguous states, i.e., states that are pairs of original states, one reached by going through a fault and the other not. Fault diagnosability is thus equivalent to the absence of such a critical path. This approach turns the diagnosability problem in a search for a path with a cycle in a finite automaton, and this reduces its complexity to be polynomial of degree 4 in the number of states. The works by

 ISBN: 978-1-61208-500-5

[4] and [5] generalize simple faults modeled as distinguished events to supervision patterns, given as arbitrary suffix-closed rational languages of events.

The first work that addressed diagnosability analysis in DDES was [6], who introduced an incremental diagnosability test that avoids to build the twin plant for the whole system if not needed. Thus, one starts by building a local twin plant for the faulty component to test the existence of a local critical path. If such a path exists one builds the local twin checkers of the neighboring components (structure similar to local twin plant, except that there is no fault information in it) and one tries to solve the ambiguity resulting from the local critical path by exploiting the observable events in the neighboring components. This is done by synchronizing on their *communication events* the local twin plant with the local twin checker of one neighboring component. The process is repeated until the diagnosability is answered, so only in the worst case has the whole system to be visited. The work by [7] has optimized this construction by exploiting the different identifiers given to the communication events at the observation synchronization level (depending on which instance, left or right, they belong to) to assign them directly to the two behaviors studied. This helped in deleting the redundant information, then in abstracting the amount of information to be transferred later to next steps if the diagnosability was not answered. The generalization to supervision patterns in DDES was introduced by [8].

After the reduction of the diagnosability problem to a path finding problem by [3], it became transferable to a satisfiability problem as for planning problems [9]. This was done by [1] which formulated the diagnosability problem (in its twin plant version) into a SAT problem, assuming a centralized DES with simple fault events, modeled as a succinct labeled transition system (SLTS). We provide in subsection III-B a summary of this approach, on which our work is based. Our prior work [10] focused on using incremental SAT for diagnosability analysis in DDES.

Works on the predictability property for DES are fewer and more recent. A deterministic diagnoser approach was proposed by [11], with exponential complexity in the number of system states, and later a polynomial method by [12], that checks predictability directly on a twin plant. But the whole twin plant is built, which we avoid here by forcing the search after the fault occurrence in the correct sequence only (see subsection V-B). The generalization to supervision patterns was introduced by [13] and to DDES by [14] and [15].

III. SAT-Based Diagnosability Analysis of Centralized Systems

We recall the definitions of DES models we use, of the diagnosability property and of its SAT-based analysis.

A. Preliminaries

Traditionally, since the seminal work [2], LTS are used as a modeling formalism, where faults are simply modeled as particular unobservable events. Following [1] we will use an equivalent but more compact representation than LTS called SLTS, that are expressed in terms of propositional variables, allowing an easier translation to a SAT problem of the twin plant method proposed by [3] for checking diagnosability. The system states are represented by the valuations of a finite set A of Boolean state variables where valuation changes reflect the transitions between states according to the events. The set of all literals issued from A is $L = A \cup \{\neg a | a \in A\}$ and $\mathcal{L}$ is the full propositional language over A that consists of all formulas that can be formed from A and the connectives $\vee$, $\wedge$, $\neg$, $\rightarrow$ and $\leftrightarrow$. Each event is described by a set of pairs $\langle \phi, c \rangle$ which represent its possible ways of occurrence by indicating that the event can be associated with changes $c \in 2^L$ in states that satisfy the condition $\phi \in \mathcal{L}$.

Definition 1. A **succinct labeled transition system** (SLTS) is described by a tuple $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \delta, s_0 \rangle$ where:

- A is a finite set of state variables,
- Σ_o is a finite set of observable correct events,
- Σ_u is a finite set of unobservable correct events,
- Σ_f is a finite set of unobservable faulty events,
- $\delta : \Sigma = \Sigma_o \cup \Sigma_u \cup \Sigma_f \rightarrow 2^{\mathcal{L} \times 2^L}$ assigns to each event a set of pairs $\langle \phi, c \rangle$,
- s_0 is the initial state (a valuation of A).

It is straightforward to show that any LTS with a set of states X can be represented as an SLTS with $\lceil log(|X|) \rceil$ Boolean variables and reciprocally that any SLTS can be mapped to an LTS (see Definition 2.4 in [1]).

The formal definition of diagnosability of a fault f in a centralized system modeled by an LTS or SLTS T was proposed by [2] as follows.

Definition 2. Diagnosability. A fault f is diagnosable in a system T if and only if (iff)

$$\exists k \in \mathbb{N}, \forall s^f \in L(T), \forall t \in L(T)/s^f, |t| \geq k \Rightarrow$$
$$\forall p \in L(T), (P(p) = P(s^f.t) \Rightarrow f \in p).$$

In this formula, $L(T)$ denotes the prefix-closed language of T whose words are called trajectories, s^f any trajectory ending by (a first occurrence of) the fault f, $L(T)/s$ the post-language of $L(T)$ after the trajectory s, i.e., $\{t \in \Sigma^* | s.t \in L(T)\}$ and P the projection of a trajectory on its observable events. The above definition states that for each trajectory s^f ending with fault f in T, for each t that is an extension of s^f in T with enough (depending only on f, not on its occurrences) events, every trajectory p in T that is equivalent to $s^f.t$ in terms of observation should contain in it f. As usual, it will be assumed that $L(T)$ is live (i.e., for any state, there is at least one transition issued from this state) and convergent (i.e., there is no cycle made up only of unobservable events).

A system T is said to be diagnosable iff any fault $f \in \Sigma_f$ is diagnosable in T, which is equivalent to each fault being separately diagnosable (i.e., the other faults being considered as unobservable correct events). Thus, to avoid exponential complexity in the number of faults during diagnosability analysis, only one fault's diagnosability is checked at a time, without loss of generality. It will thus be assumed in the following that there exists only one fault event f ($\Sigma_f = \{f\}$), without restriction on the number of its occurrences. Diagnosability checking has been proved in [3] to be polynomial in the number $|X|$ of states for LTS, so exponential in the number $|A|$ of state variables for SLTS (actually the problem is NLOGSPACE-complete for LTS and PSPACE-complete for SLTS [16]).

B. SLTS Diagnosability as Satisfiability

An immediate rephrasing of definition 2 shows that T is nondiagnosable iff it exists a pair of trajectories corresponding to cycles (and thus to infinite paths), a faulty one and a correct one, sharing the same observable events. This is equivalent to the existence of an ambiguous cycle in the product of T by itself, synchronized on observable events, which is called *twin plant* structure introduced in [3]. A cycle is ambiguous iff it is made up of pairs of states respectively reachable by a faulty path and a correct path. This nondiagnosability test was formulated in [1] as a satisfiability problem in propositional logic and we recall below this encoding, where superscripts $t \in \mathbb{N}$ refer to time points and (e_o^t) and $(\hat{e}_o^t)$ refer respectively to the faulty and correct events occurrences sequences of a pair of trajectories witnessing nondiagnosability. These two sequences share the same observable events represented by (e^t) and forming a cycle. The states are described by valuations of (a^t) and $(\hat{a}^t)$.

In order to represent the occurrence of the fault f and differently from the original encoding in [1], which does not exploit any relation between the fault occurrences at the different time steps, we added the variables f^t, whose truth value is $True$ iff f has occurred before t. This helped us to propagate the fault information automatically and guide the solver to search this specific information about the fault occurrence which is essential to decide the diagnosability test (it will be required also for our predictability encoding in SAT). Each time step increase corresponds to triggering at least one transition and so the extension by an event of at least one of the two trajectories. $T = \langle A, \Sigma_u, \Sigma_o, \Sigma_f, \delta, s_0 \rangle$ being an SLTS, the propositional variables required for the encoding are:

- a^t and $\hat{a}^t$ for all $a \in A$ and $0 \leq t \leq n$,
- e_o^t for all $e \in \Sigma_o \cup \Sigma_u \cup \Sigma_f$, $o \in \delta(e)$ and $0 \leq t \leq n-1$,
- $\hat{e}_o^t$ for all $e \in \Sigma_o \cup \Sigma_u$, $o \in \delta(e)$ and $0 \leq t \leq n-1$,
- e^t for all $e \in \Sigma_o$ and $0 \leq t \leq n-1$,
- f^t for all $0 \leq t \leq n$.

The following formulas express the constraints that must be applied at each t or between t and $t+1$.

1) The event occurrence e_o^t must be possible in the current state:
$$e_o^t \rightarrow \phi^t \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{1}$$
and its effects must hold at the next time step:
$$e_o^t \rightarrow \bigwedge_{l \in c} l^{t+1} \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{2}$$
We have the same formulas with $\hat{e}_o^t$.

2) The present value ($True$ or $False$) of a state variable changes to a new value ($False$ or $True$, respectively) only if there is a reason for this change, i.e., because of an event that has the new value in its effects (so, change without reason is prohibited). Here is the change from $True$ to $False$ (the change from $False$ to $True$ is defined similarly by interchanging a and $\neg a$):
$$(a^t \wedge \neg a^{t+1}) \rightarrow (e^t_{i_{1_{o_{j_1}}}} \vee \cdots \vee e^t_{i_{k_{o_{j_k}}}}) \tag{3}$$
where the $o_{j_l} = \langle \phi_{j_l}, c_{j_l} \rangle \in \delta(e_{i_l})$ are all the occurrences of events e_{i_l} with $\neg a \in c_{j_i}$.
We have the same formulas with $\hat{a}^t$ and $\hat{e}^t_{i_{l_{o_{j_l}}}}$.

3) At most one occurrence of a given event can occur at a time and the occurrences of two different events cannot be simultaneous if they interfere (i.e., if they have two contradicting effects or if the precondition of one contradicts the effect of the other):
$$\neg(e_o^t \wedge e_{o'}^t) \quad \forall e \in \Sigma, \forall \{o, o'\} \subseteq \delta(e), o \neq o' \tag{4}$$
$$\neg(e_o^t \wedge e_{o'}^{\prime t}) \quad \forall \{e, e'\} \subseteq \Sigma, e \neq e', \forall o \in \delta(e), \forall o' \in \delta(e') \text{ such that } o \text{ and } o' \text{ interfere} \tag{5}$$
We have the same formulas with $\hat{e}_o^t$.

4) The information about f occurrence is propagated by expressing that f has occurred before $t+1$ ($t \leq n-1$) iff it has occurred either before t or between t and $t+1$.
$$f^{t+1} \leftrightarrow f^t \vee \bigvee_{e \in \Sigma_f, o \in \delta(e)} e_o^t \tag{6}$$

5) The formulas that connect the two events sequences require that observable events take place in both sequences whenever they take place (use of e^t):
$$\bigvee_{o \in \delta(e)} e_o^t \leftrightarrow e^t \text{ and } \bigvee_{o \in \delta(e)} \hat{e}_o^t \leftrightarrow e^t \quad \forall e \in \Sigma_o \tag{7}$$

6) To avoid trivial cycles (silent loops with no state change at some step) we require that at every time point at least one event takes place:
$$\bigvee_{e \in \Sigma_o} e^t \quad \vee \bigvee_{e \in \Sigma_u \cup \Sigma_f, o \in \delta(e)} e_o^t \quad \vee \bigvee_{e \in \Sigma_u, o \in \delta(e)} \hat{e}_o^t \tag{8}$$

The conjunction of all the above formulas for a given t is denoted by $\mathcal{T}(t, t+1)$.

A formula for the initial state s_0 is:
$$\mathcal{I}_0 = \neg f^0 \wedge \bigwedge_{a \in A, s_0(a)=1} (a^0 \wedge \hat{a}^0) \wedge \bigwedge_{a \in A, s_0(a)=0} (\neg a^0 \wedge \neg \hat{a}^0) \tag{9}$$

At last, the following formula can be defined to encode the fact that a pair of executions is found with the same observable events and no fault in one execution but one fault in the other (first line), which are infinite (in the form of a cycle, necessarily non trivial by (8)) at step n (second line), witnessing non diagnosability:
$$\Phi_n^T = \mathcal{I}_0 \wedge \mathcal{T}(0,1) \wedge \cdots \wedge \mathcal{T}(n-1, n) \wedge f^n \wedge \bigvee_{m=0}^{n-1} \left(\bigwedge_{a \in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \right)$$

From this encoding in propositional logic, follows the result (theorem 3.2 of [1]) that an SLTS T is not diagnosable iff $\exists n \geq 1, \Phi_n^T$ is satisfiable. It is also equivalent to $\Phi_{2^{2|A|}}^T$ being satisfiable, as the twin plant states number is an obvious upper bound for n, but often impractically high (see in the same reference some ways to deal with this problem).

IV. SAT-Based Diagnosability Analysis of Distributed Systems

We extend from centralized to distributed systems the satisfiability framework above for testing diagnosability and we provide some experimental results.

 ISBN: 978-1-61208-500-5

A. DDES Modeling

In order to model DDES with SLTS, we need to extend these ones by adding communication events *to each component*. So we introduce the following definition for a distributed SLTS with k different components (sites):

Definition 3. A **distributed succinct labeled transition system** (DSLTS) with k components is described by a tuple $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \Sigma_c, \delta, s_0 \rangle$ where (subscript i refers to component i):

- A is a union of disjoint finite sets $(A_i)_{1 \leq i \leq k}$ of component own state variables, $A = \cup_{i=1}^{k} A_i$,
- Σ_o is a union of disjoint finite sets of component own observable correct events, $\Sigma_o = \cup_{i=1}^{k} \Sigma_{oi}$,
- Σ_u is a union of disjoint finite sets of component own unobservable correct events, $\Sigma_u = \cup_{i=1}^{k} \Sigma_{ui}$,
- Σ_f is a union of disjoint finite sets of component own unobservable faulty events, $\Sigma_f = \cup_{i=1}^{k} \Sigma_{f_i}$,
- Σ_c is a union of finite sets of (observable or unobservable) correct communication events, $\Sigma_c = \cup_{i=1}^{k} \Sigma_{ci}$, which are the only events shared by at least two different components (i.e., $\forall i, \forall c \in \Sigma_{ci}, \exists j \neq i, c \in \Sigma_{cj}$),
- $\delta = (\delta_i)$, where $\delta_i : \Sigma_i = \Sigma_{oi} \cup \Sigma_{ui} \cup \Sigma_{f_i} \cup \Sigma_{ci} \rightarrow 2^{\mathcal{L}_i \times 2^{L_i}}$, assigns to each event a set of pairs $\langle \phi, c \rangle$ in the propositional language of the component where it occurs (so, for communication events, *in each component separately* where they occur),
- $s_0 = (s_{0i})$ is the initial state (a valuation of each A_i).

Synchronous communication is assumed. More precisely, a transition by a communication event c may occur in a component iff a simultaneous transition by c occurs in all the other components where c appears (has at least one occurrence). The global model of the system is thus the product of the models of the components, synchronized on communication events. Notice that we allow in whole generality communication events to be, partially or totally, unobservable (which is not allowed up to now in any model, to the best of our knowledge), so one has in general to wait further observations to know that some communication event occurred between two or more components. On the other side, assuming these communications to be faultless is not actually a limitation. If a communication process or protocol may be faulty, it has to be modeled as a proper component with its own correct and faulty behaviors. In this sense, communications between components are just a modeling concept, not subject to diagnosis. It will be also assumed that the observable information is global, i.e. centralized, allowing to keep definition 2 (as, when observable information is only local to each component, distributed diagnosability checking becomes undecidable [17]).

B. DSLTS Diagnosability as Satisfiability

Let T be a DSLTS made up of k components denoted by indexes i, $1 \leq i \leq k$. In order to express the diagnosability analysis of T as a satisfiability problem, we have to extend the formulas of the centralized case to deal with communication events between components. Let $\Sigma_c = \Sigma_{co} \cup \Sigma_{cu}$ be the communication events, with $\Sigma_{co} = \cup_{i=1}^{k} \Sigma_{coi}$ the observable ones and $\Sigma_{cu} = \cup_{i=1}^{k} \Sigma_{cui}$ the unobservable ones. The idea is to treat each communication event as any other event in each of its owners and, as it has been done with events e^t for $e \in \Sigma_o$ for synchronizing observable events occurrences in the two executions, to introduce in the same way a global reference variable for each communication event at each time step, in charge of synchronizing any communication event occurrence in any of its owners with occurrences of it in all its other owners. We use one such reference variable for each trajectory, e^t and $\hat{e}^t$, for unobservable events $e \in \Sigma_{cu}$, and only one for both trajectories, e^t, for observable events $e \in \Sigma_{co}$ as it will also in addition play the role of synchronizing observable events between trajectories exactly as the e^t for $e \in \Sigma_o$. So, we add to the previous propositional variables the new following ones:

- e_o^t, $\hat{e}_o^t$ for all $e \in \Sigma_c$, $o \in \delta(e) = \cup_i \delta_i(e)$ and $0 \leq t \leq n-1$,
- e^t for all $e \in \Sigma_c$, $\hat{e}^t$ for all $e \in \Sigma_{cu}$ and $0 \leq t \leq n-1$.

Formulas in $\mathcal{T}(t, t+1)$ are extended as follows.

1) Formulas (1), (2), (3) and (5) extend unchanged to e_o^t and $\hat{e}_o^t$ $\forall e \in \Sigma_c$.
2) Formulas (4) extend to prevent two simultaneous occurrences of a given communication event in the same owner component, i.e. apply $\forall e \in \Sigma_c, \forall i, \forall \{o_i, o_i'\} \subseteq \delta_i(e), o_i \neq o_i'$ (the same with $\hat{e}$)
3) The new following formulas express the communication process itself, i.e. the synchronization of the occurrences of any communication event e in all its owners components ($S(e)$ being the set of indexes of the owners components of e) and extend also formulas (7) to observable communication events:
$$\bigvee_{o_i \in \delta_i(e)} e_{o_i}^t \leftrightarrow e^t \text{ and } \bigvee_{o_i \in \delta_i(e)} \hat{e}_{o_i}^t \leftrightarrow \hat{e}^t \ \forall e \in \Sigma_{cu} \ \forall i \in S(e)$$
$$\bigvee_{o_i \in \delta_i(e)} e_{o_i}^t \leftrightarrow e^t \text{ and } \bigvee_{o_i \in \delta_i(e)} \hat{e}_{o_i}^t \leftrightarrow e^t \ \forall e \in \Sigma_{co} \ \forall i \in S(e)$$
4) Finally, the clause (8) is adapted to take into account both observable and unobservable communication events:
$$\bigvee_{e \in \Sigma_o \cup \Sigma_c} e^t \quad \vee \quad \bigvee_{e \in \Sigma_{cu}} \hat{e}^t \quad \vee \quad \bigvee_{e \in \Sigma_u \cup \Sigma_f, o \in \delta(e)} e_o^t \quad \vee \quad \bigvee_{e \in \Sigma_u, o \in \delta(e)} \hat{e}_o^t$$

We have thus the result that a DSLTS T is not diagnosable iff $\exists n \geq 1, \Phi_n^T$ is satisfiable (proof analog to that in the centralized case). It is also equivalent to $\Phi^T_{2^{2\sum_{i=1}^{k}|A_i|}}$ being satisfiable.

C. Implementation and Experimental Testing

We have implemented the above extension in Java, our experiments were run on 64-bit Windows 7 machine with an Intel(R) Xeon(R) CPU @2.80GHz processor and 8 GB of RAM. We used the well designed API of the SAT solver Sat4j [18] as it fitted well our clause generator written in Java. We have tested our tool on small examples with several communication events with multiple occurrences, with global communication (all components share the same event) or partial communication (only some components share the same event), as in Fig. 1, adapted from the running example in [6], which is made up of three communicating components. The results are in Table I, where the columns show the system and the fault considered (4 cases separated by horizontal

lines), the steps number n, the answer of the SAT solver, the numbers of variables and of clauses and the runtime in ms.

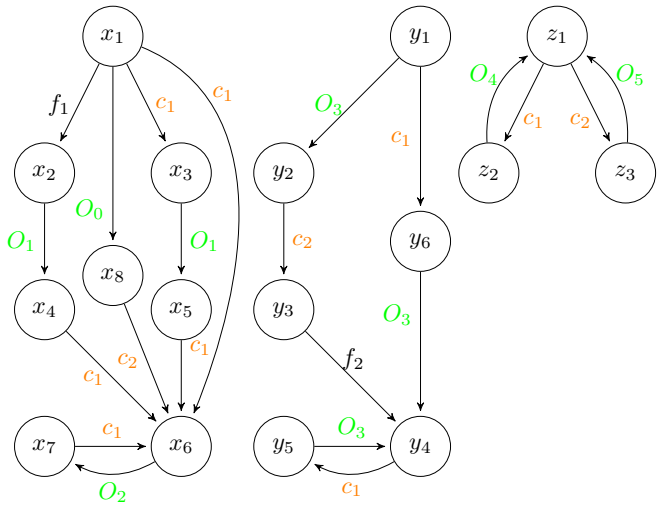

Figure 1. A DDES made up of 3 components C1, C2 and C3 from left to right. $ci_{,1\leq i\leq 2}$ are unobservable communication events, $Oi_{,0\leq i\leq 5}$ are observable events and $fi_{,1\leq i\leq 2}$ are faulty events.

TABLE I. DIAGNOSABILITY RESULTS ON THE EXAMPLE OF FIG 1.

System	Fault	|Steps|	SAT?	|Variables|	|Clauses|	Time(ms)
$C2$	$f2$	4	No	112	561	6
$C2$	$f2$	5	No	138	699	11
$C2$	$f2$	6	Yes	164	837	15
$C1, C2$	$f2$	6	No	356	356	25
$C1, C2$	$f2$	32	No	1838	12751	94
$C1, C2$	$f2$	64	No	3662	25487	225
$C1, C2$	$f2$	128	No	7310	50959	112
$C1, C2$	$f2$	256	No	14606	101903	180
$C1, C2$	$f2$	512	No	29198	203791	1855
$C1, C2$	$f2$	1024	No	58382	407567	784
$C1, C2$	$f2$	4096	No	233486	1630223	23453
$C2, C3$	$f2$	6	No	252	1237	15
$C2, C3$	$f2$	32	No	1292	6541	46
$C2, C3$	$f2$	64	No	2572	13069	71
$C2, C3$	$f2$	128	No	5132	26125	61
$C2, C3$	$f2$	256	No	10252	52237	216
$C2, C3$	$f2$	512	No	20492	104461	143
$C2, C3$	$f2$	1024	No	40972	208909	381
$C1, C2, C3$	$f1$	8	No	586	3723	40
$C1, C2, C3$	$f1$	9	Yes	657	4186	45
$C1, 10 \times C2, 10 \times C3$	$f1$	9	Yes	3862	22907	342
$C1, 20 \times C2, 20 \times C3$	$f1$	9	Yes	7112	42087	592
$C1, 50 \times C2, 50 \times C3$	$f1$	9	Yes	16862	99627	3141
$C1, 100 \times C2, 100 \times C3$	$f1$	9	Yes	33372	196723	26930

Which means that $f2$ is not diagnosable in $C2$ alone while it becomes diagnosable when synchronizing $C2$ with either $C1$ or $C3$. For proving these two last results, we have increased the steps number, verifying that the answer remained UNSAT, until reaching the theoretical upper bound $2^{2\sum_i |A_i|}$ (equal to $2^{2(3+2)} = 1024$ for $\{C2, C3\}$ and to $2^{2(3+3)} = 4096$ for $\{C1, C2\}$). While $f1$ is not diagnosable even after synchronizing all three components together. These 4 tests are mentioned as a proof of concept. But actually, numbers of variables and clauses are small in comparison to what SAT solvers can handle (up to hundred thousands propositional variables and millions of clauses). This is why we extended the last case (non-diagnosability of $f1$) to bigger systems obtained by duplicating (10, 20, 50 and 100 times) components $C2$ and $C3$, keeping unchanged their communication events and renaming their proper local events. This shows the efficiency of the method (less than 27s for 201 components). Notice that here the steps number remains unchanged as occurrences of non-interfering events are processed simultaneously in the same step, thanks to the succinct encoding in this representation. The number of states in the last tested system is very large, which proves the efficiency of this approach in detecting the non-diagnosability of a system if the length of a potential critical path stays short. The case where diagnosability analysis requires checking very long potential critical paths is still impractical and needs a more abstract induction-proof approach.

V. PREDICTABILITY AS SATISFIABILITY

We recall the definition of the predictability property, adapt the framework above to define SAT-based predictability analysis for both centralized and distributed systems and provide experimental results.

A. Definition

The formal definition of predictability of a fault f in a centralized system modeled by an LTS or SLTS T was proposed by [11] as follows.

Definition 4. Predictability. A fault f is predictable in a system T iff

$$\exists k \in \mathbb{N}, \forall s^f \in L(T), \exists \eta \in \overline{s^f}, \forall p \in L(T), \forall p' \in L(T)/p$$
$$(P(p) = P(\eta) \wedge f \notin p \wedge |p'| \geq k \Rightarrow f \in p')$$

The above definition, where $\bar{t}$ denotes the set of strict prefixes of t, states that a fault f is predictable iff for any trajectory s^f ending with a first occurrence of f, there exists at least one strict prefix of s^f, denoted by η (thus η does not contain f), such that for every correct trajectory p with the same observations as η, all the long enough (depending only on f) continuations of p should contain f. In other words, the non-predictability of f is equivalent to the existence of a finite faulty sequence that ends with a first occurrence of f and of an infinite (i.e. corresponding to a cycle) correct sequence that is synchronized with the faulty sequence on observable events before the occurrence of f. Predictability is thus stronger than diagnosability (if f is predictable, then f is diagnosable).

B. SLTS Predictability as Satisfiability

Unlike diagnosability, predictability checking process has two different phases, before and after the fault occurrence in the faulty sequence: the synchronization on observable events between the two sequences is required only up to this fault occurrence and, after it, only the correct sequence has to be extended and searched for the presence of a cycle in it. The new or modified formulas expressing the constraints to be applied at each time step t are as follows.

1) The synchronization of observable events between the two sequences holds only up to the fault occurrence, i.e. (7) is replaced by:

$$\begin{aligned} f^t \vee \Big(\bigvee_{o \in \delta(e)} e_o^t \leftrightarrow e^t \Big) \quad \forall e \in \Sigma_o \\ f^t \vee \Big(\bigvee_{o \in \delta(e)} \hat{e}_o^t \leftrightarrow e^t \Big) \quad \forall e \in \Sigma_o \end{aligned} \tag{10}$$

2) The formula (8), requiring that at every time point at least one event takes place in either one or the other sequence, remains valid up to the fault occurrence; after it, we require that at least one event takes place in the correct sequence:

$$f^t \vee \bigvee_{e\in\Sigma_o} e^t \vee \bigvee_{e\in\Sigma_u\cup\Sigma_f, o\in\delta(e)} e^t_o \vee \bigvee_{e\in\Sigma_u, o\in\delta(e)} \hat{e}^t_o$$
$$\neg f^t \vee \bigvee_{e\in\Sigma_o\cup\Sigma_u, o\in\delta(e)} \hat{e}^t_o \quad (11)$$

The conjunction of the formulas (1-6), (10) and (11) for a given t is denoted by $\mathcal{S}(t, t+1)$.

The formula (9) for the initial state s_0 is unchanged.

Finally, the formula to encode the non predictability property is obtained as Φ^T_n, where the presence of a cycle at step n is required only in the correct sequence:

$$\Psi^T_n = \mathcal{I}_0 \wedge \mathcal{S}(0,1) \wedge \cdots \wedge \mathcal{S}(n-1,n) \wedge f^n \wedge \bigvee_{m=0}^{n-1} \left(\bigwedge_{a\in A} (\hat{a}^n \leftrightarrow \hat{a}^m) \right)$$

It follows that an SLTS T is not predictable iff $\exists n \geq 1, \Psi^T_n$ is satisfiable, which is also equivalent to $\Psi^T_{2^{2|A|}}$ being satisfiable (proof analog to that for diagnosability).

C. DSLTS Predictability as Satisfiability

Let T be now a DSLTS. The extension of predictability analysis to distributed systems adapts what we presented for diagnosability analysis. As the synchronization of observable events holds only before the fault occurrence, we will decouple it from the synchronization of communication events. So, the only change concerning the variables is that we use now one reference variable for each sequence for observable communication events, as for unobservable ones, i.e. we have:

- e^t, $\hat{e}^t$ for all $e \in \Sigma_c$ and $0 \leq t \leq n-1$.

Formulas in $\mathcal{S}(t, t+1)$ are extended as those in $\mathcal{T}(t, t+1)$ were extended, except the following.

1) The synchronization of the occurrences of any communication event e in all its owner components in $S(e)$ is expressed in each sequence and in the same way for both observable and unobservable events:

$$\bigvee_{o_i\in\delta_i(e)} e^t_{o_i} \leftrightarrow e^t \text{ and } \bigvee_{o_i\in\delta_i(e)} \hat{e}^t_{o_i} \leftrightarrow \hat{e}^t \ \forall e \in \Sigma_c \ \forall i \in S(e)$$

while the synchronization of the occurrences of any observable event in the two sequences before the fault occurrence, expressed in the centralized case by formulas (10), is extended to any observable communication event:

$$f^t \vee (e^t \leftrightarrow \hat{e}^t) \quad \forall e \in \Sigma_{co}$$

2) The clauses (11) are extended to take into account communication events:

$$f^t \vee \bigvee_{e\in\Sigma_o\cup\Sigma_c} e^t \vee \bigvee_{e\in\Sigma_{cu}} \hat{e}^t \vee \bigvee_{e\in\Sigma_u\cup\Sigma_f, o\in\delta(e)} e^t_o \vee \bigvee_{e\in\Sigma_u, o\in\delta(e)} \hat{e}^t_o$$
$$\neg f^t \vee \bigvee_{e\in\Sigma_c} \hat{e}^t \vee \bigvee_{e\in\Sigma_o\cup\Sigma_u, o\in\delta(e)} \hat{e}^t_o$$

We have thus the result that a DSLTS T is not predictable iff $\exists n \geq 1, \Psi^T_n$ is satisfiable, which is also equivalent to $\Psi^T_{2^{2\sum_{i=1}^k |A_i|}}$ being satisfiable (proof analog to that for diagnosability).

D. Experimental Results

We used the same example (Fig. 1) as for diagnosability and studied the predictability of the faulty events $f1$ and $f2$. Table II shows the results obtained. It is found that $f2$ is not predictable in $C2$ alone, which was expected as it is not diagnosable in $C2$. We saw that it became diagnosable in the system composed of $C1$ and $C2$ and we find that it is actually even predictable in this system, by obtaining the UNSAT answer up to the theoretical upper bound 4096. On the contrary, although we saw it became also diagnosable in the system composed of $C2$ and $C3$, we find that it remains not predictable in this system. And here again, we extend this test to bigger systems by duplicating component $C3$, with the same steps number and very good efficiency. Concerning the fault $f1$, it is found not predictable in the whole system made up of the three components, which was expected as it has been shown not diagnosable in this system.

TABLE II. PREDICTABILITY RESULTS ON THE EXAMPLE OF FIG 1.

System	Fault	|Steps|	SAT?	|Variables|	|Clauses|	Time (ms)
$C2$	$f2$	3	No	92	414	7
$C2$	$f2$	4	Yes	120	549	12
$C1, C2$	$f2$	1024	No	66574	404495	10109
$C1, C2$	$f2$	4096	No	266254	1617935	91299
$C2, C3$	$f2$	4	No	196	817	14
$C2, C3$	$f2$	5	No	242	1018	21
$C2, C3$	$f2$	6	Yes	288	1219	27
$C1, C2, C3$	$f1$	3	No	267	1399	29
$C1, C2, C3$	$f1$	4	Yes	350	1859	40
$C2, 10 \times C3$	$f2$	6	Yes	1408	5219	24
$C2, 20 \times C3$	$f2$	6	Yes	2528	9219	50
$C2, 50 \times C3$	$f2$	6	Yes	5888	21219	125
$C2, 100 \times C3$	$f2$	6	Yes	11488	41219	277

VI. Conclusion and Future Works

By extending the state of the art work for centralized DES [1], we have expressed diagnosability analysis of DDES as a satisfiability problem by building a propositional formula whose satisfiability, witnessing non-diagnosability, can be checked by SAT solvers. We allow both observable and unobservable synchronous communication events in our model. We have then applied the same method to express predictability analysis as a SAT problem, both for centralized DES and for DDES. In each case, we have provided experimental results.

In order to conduct more experiments to check precisely the scalability of the method and to compare it with other approaches referenced above (for which no software is available and in general no experimental results are given), we have implemented classical twin plant based algorithms and achieve implementing an automatic benchmarks generator, tuned by several parameters and whose diagnosability and predictability will be known by construction. We have also designed and are implementing the extension of this work from simple faulty events to supervision patterns. All our programs will be made available as open source. We also aim at investigating relations between our work and the problem of opacity of discrete event systems [19], in order to treat this problem with SAT-based methods. Finally, relationships between satisfiability and bounded or unbounded model checking methods to encode and analyze fault diagnosability and predictability will be studied. In particular, SAT-based model checking [20] allows incremental solving, which significantly improves both the capacity and the speed of solving. Research of invariants by full-proof methods like temporal induction should avoid unrolling to a theoretical bound, as it is the case here when the system is not diagnosable.

 ISBN: 978-1-61208-500-5

REFERENCES

[1] J. Rintanen and A. Grastien, "Diagnosability testing with satisfiability algorithms." *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 532–537, 2007.

[2] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems." *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555–1575, 1995.

[3] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems." *IEEE Transactions on Automatic Control*, vol. 46, no. 8, pp. 1318–1321, 2001.

[4] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier, "Supervision Patterns in Discrete Event Systems Diagnosis." *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06)*, pp. 262–268, 2006.

[5] S. Genc and S. Lafortune, "Diagnosis of patterns in partially-observed discrete-event systems." *Proceedings of the 45th IEEE Conference on Decision and Control (CDC'06)*, pp. 422–427, 2006.

[6] Y. Pencolé, "Diagnosability Analysis of Distributed Discrete Event Systems." *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pp. 43–47, 2004.

[7] L. Ye and P. Dague, "An optimized algorithm for diagnosability of component-based systems." *Proceedings of the 10th International Workshop on Discrete Event Systems (WODES'10)*, pp. 143–148, 2010.

[8] Y. Yan, L. Ye, and P. Dague, "Diagnosability for patterns in distributed discrete event systems." *Proceedings of the 21st International Workshop on Principles of Diagnosis (DX'10)*, pp. 345–352, 2010.

[9] H. Kautz and B. Selman, "Planning as Satisfiability." *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pp. 359–363, 1992.

[10] H. Ibrahim, P. Dague, and L. Simon, "Using Incremental SAT for Testing Diagnosability of Distributed DES." *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX'15)*, pp. 51–58, 2015.

[11] S. Genc and S. Lafortune, "Predictability in discrete-event systems under partial observation." *Proceedings of the 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'06)*, pp. 1461–1466, 2006.

[12] S. Genc and S. Lafortune, "Predictability of Event Occurrences in Partially-observed Discrete-event Systems." *Automatica*, vol. 45, no. 2, pp. 301–311, 2009.

[13] T. Jéron, H. Marchand, S. Genc, and S. Lafortune, "Predictability of Sequence Patterns in Discrete Event Systems." *Proceedings of the 17th World Congress*, pp. 537–453, 2008.

[14] L. Ye, P. Dague, and F. Nouioua, "Predictability Analysis of Distributed Discrete Event Systems." *Proceedings of the 52nd IEEE Conference on Decision and Control (CDC'13)*, pp. 5009–5015, 2013.

[15] ——, " A predictability algorithm for distributed discrete event systems." *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM'15)*, pp. 201–216, 2015.

[16] J. Rintanen, "Diagnosers and diagnosability of succinct transition systems." *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 538–544, 2007.

[17] L. Ye and P. Dague, "Undecidable Case and Decidable Case of Joint Diagnosability in Distributed Discrete Event Systems." *International Journal On Advances in Systems and Measurements*, vol. 6, no. 3 and 4, pp. 287–299, 2013.

[18] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2." *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.

[19] F. Lin, "Opacity of discrete event systems and its applications." *Automatica*, vol. 47, no. 3, pp. 496–503, 2011.

[20] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving." *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

 ISBN: 978-1-61208-500-5

Automatic Test Evaluation for Driving Scenarios Using Abstraction Level Constraints

Steffen Wittel*, Daniel Ulmer* and Oliver Bühler*
*Steinbeis Interagierende Systeme GmbH, Esslingen, Germany
Email: {steffen.wittel,daniel.ulmer,oliver.buehler}@interagierende-systeme.de

***Abstract*—Sophisticated Driver Assistance Systems (DASs) on the way to highly automated driving require extensive testing activities to verify the functionality and the safety of the developed software. With each step towards autonomous driving, the automobile manufacturers are increasingly taking on responsibility for driving maneuvers automatically performed by such systems in unknown environmental situations. Whereas recent DASs use the driver as fallback, in the future this fallback will be only available after a legally prescribed period of time since the driver might be distracted by other activities. To take account of this, robustness testing becomes more and more important to ensure a safe operation at different environments. This paper presents a constraint based approach that applies automatic testing to evaluate DASs. Thereby, the focus is set on the determination of the expected responses that are the basis for the automatic evaluation of the generated test scenarios. The introduced approach is working on different levels of abstraction in combination with an analysis of the observed behavior to classify individual situations of the scenarios after the test execution. The approach enables a full automation of the evaluation, which is the bottleneck of current state-of-the-art scenario generators.**

Keywords–Automotive Testing; Test Generation; Test Evaluation; Test Automation.

I. INTRODUCTION

The decreasing development times and product cycles in combination with technical advances already require a high testing effort to ensure that the vehicle's built-in DASs are working correctly. It is expected that with each step in the direction to highly automated driving, the testing effort increases to cover the new functionality and to ensure a safe operation of the vehicle.

While the first assistance systems, like the Electronic Stability Control (ESC) [1] or the Antilock Braking System (ABS) [1], were intervening in critical situations, only the current generation of DASs supports the driver during his entire drive, but without taking on responsibility for the maneuvers performed. Even during an intervention of a DAS, the driver is still responsible for the vehicle and the possible damage. On the way to highly automated driving, this responsibility of the driver will be only given after a legally prescribed time limit, because it is assumed that the driver is distracted by other activities and can only handle the situation after a certain time.

Additionally for autonomous driving, it is not sufficient that systems are working as expected in a defined environment, but also in unknown situations. Each drive is different from the previous one, e.g., in respect to the environmental conditions like traffic or weather. An early and safe hand over to the driver would be a technical solution. But especially in the premium market, the customers do not tolerate a system, which is rarely available. The automobile manufactures have thus to find a balance between safety and availability.

The automatic test generation is an approach, which is not intended to replace the already performed testing, but rather extends them to cover a broad functional range of a system by creating a large number of different test cases. In most cases, the commercial off-the-shelf scenario generators do not determine the test result and leave it to the tester to define the expected behavior of the System Under Test (SUT), which limits the degree of automation. To get around this, an approach is presented to determine the expected response of the SUT using constraints on different levels of abstraction. An automatic analysis of the observed behavior supplements the approach to classify individual situations of the scenarios after the test execution.

The following section shows the related work. Section III and Section IV of this paper give an overview about the SUT and the automatic testing. In Section V, the approach for an automatic evaluation of the generated test scenarios is presented. Finally, Section VI shows a case study.

II. RELATED WORK

In [2], a framework is described to construct a generic course of the road for a virtual driving scenario using a stochastic approach. It combines Markov Chain and Markov Chain Monte Carlo methods to test different input combinations. By using this automation, there would be the possibility that parameter sets, which were forgotten or erroneously ignored during the manually test creation, are tested.

A test generator is presented in [3][4], which creates, executes and evaluates test scenarios automatically. The algorithm behind the generator tries on the one hand to maximize the coverage of the reached system states by changing the input of the SUT during the test execution. On the other hand, it searches for system states that do not fulfill the given evaluation criteria. It is left to the test engineer to configure the test generator in such a way that no invalid test scenario is created and the evaluation criteria are valid for all generated test scenarios.

According to [5][6], the formal verification is currently the only known way to ensure that a system works as specified. This means that the implementation strictly follows the specification and thus it is possible to determine its behavior in every situation. To perform a formal verification, the specification must meet some requirements. Among others, the specification must be complete and correct. This is a big challenge, especially in large projects with many dependencies to external components from different suppliers.

 ISBN: 978-1-61208-500-5

A comparative study on methods to automatically determine the expected response of the SUT is given in [7]. Such methods are necessary for the automatic testing. Otherwise, the response has to be verified manually. The presented approaches are mostly limited to their application field and cannot be generally applied.

III. System Under Test

The SUT, which is also named as "test object", is a physical or logical unit as illustrated in Fig. 1 that is tested for correctness against the specification. It has an input interface X and an output interface Y. A stimulus $x \in X$ at the input causes a response $y \in Y$ at the output, as can be seen in (1), where the stimulus x can change over time.

$$x(t) \xrightarrow{\tau} y(t) \tag{1}$$

For the reproducibility of the test results, a defined start state of the SUT is required at the beginning of the test execution. Given that, it is possible to obtain an identical response when repeating a test case or after changing the execution order of test cases in a test run. For this purpose, however, the SUT must meet the following properties:

a) time-invariant
b) memoryless

According to [8], a system is called time-invariant, if a delay of the input causes the same delay at the output as shown in (2), and memoryless, if the response does only depend on the current input and not on an input from the past. If both properties are satisfied by the SUT, exactly one $y \in Y$ can be associated for each $x \in X$ regardless of the point in time and the sequence of the stimulation. Other systems that do not meet these properties can show different responses to the same stimulus.

$$x(t) \xrightarrow{\tau} y(t) \implies x(t+\delta) \xrightarrow{\tau} y(t+\delta) \tag{2}$$

Decisions in autonomous driving are dependent on the environment and usually on the history of events, which means that different stimulation sequences over time are needed for the testing. A static input pattern or a small number of scenarios are not sufficient to verify a DAS.

A. Stimulation

The input interface of the SUT consists of signals providing data from other Electronical Control Units (ECUs). It represents the lowest level of stimulation and can be stimulated at a Model-in-the-Loop (MIL) or Software-in-the-Loop (SIL) test bench. At a Hardware-in-the-Loop (HIL) test bench, a bus interface and a Residual Bus Simulation (RBS), which could have effects to the time behavior of the stimulation as shown in [9], are required.

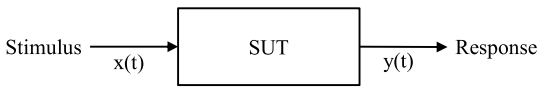

Figure 1. Schematic representation of the SUT

A direct access to the input interface on the signal-level allows a precise stimulation of the SUT. The large fan-in leads to an exponential number of test cases, which can be generated. State-of-the-art DASs have hundreds of input signals, which have to be consistently stimulated with the correct values over time. Many input signals that are not in the scope of the current test case, but are necessary for the proper operation of the SUT, must be correct and should not be manipulated by the test case generator. Deviations from the correct sequence are usually detected by the SUT and leads to a functional degradation to bring the SUT into a safe state. This outcome must be considered at the evaluation of the test.

To cope with the complexity of the input interface, it is a common practice to use models to abstract the input signals and thus to simplify the stimulation of the SUT. The models are acting as an intermediate layer between the signal-level and the used level of stimulation and ensure that the stimulation is performed in a consistent way. The advantage of the simplification is achieved by losing direct control of the input interface, which could complicate the stimulation, if specific signal characteristics are needed.

B. Evaluation

Depending on the stimulation of the SUT a response can be observed at the output interface, which has to be checked for its correctness. The evaluation of the signal characteristics can be done at selected points in time or over a certain period of time. Within these time intervals, the observed response of the SUT is compared against the expected response to ensure that the behavior corresponds with the specification. Thereby, the specification is a single point of failure. If the specification is not reliable, the tests do not recognize an abnormal behavior of the SUT in specific situations.

IV. Automatic Testing

As explained in the previous section, it is not sufficient to test only static input pattern or a small number of scenarios to verify the functionality and the safety of a DAS. Rather, it is required to test a broad range of different situations as they can be found in real-world environments. The variety of environmental conditions makes it difficult to verify the DAS within its operating range and to ensure that a safe state is always reached. For this reason, automatic testing in addition to the already performed testing is thought to play an important role.

A. Setup

Fig. 2 shows the setup for the automatic testing as used by the presented approach. The *Test Generator* comprises two parts. In the first part, the *Scenario Generator* composes a *Scenario* and the corresponding *Stimulus* for the *Test Bench* that is responsible for the test execution. In the second part, the *Response Determination* determines the *Expected Response* based on the generated *Scenario*. The *Test Bench* applies the *Stimulus* created by the *Scenario Generator* to the SUT, while observing the *Response*. The *Evaluator* compares the observed *Response* of the SUT with the *Expected Response* provided by the *Response Determination*. Differences outside a specified tolerance value cause the *Result* to fail as described in [10].

 ISBN: 978-1-61208-500-5

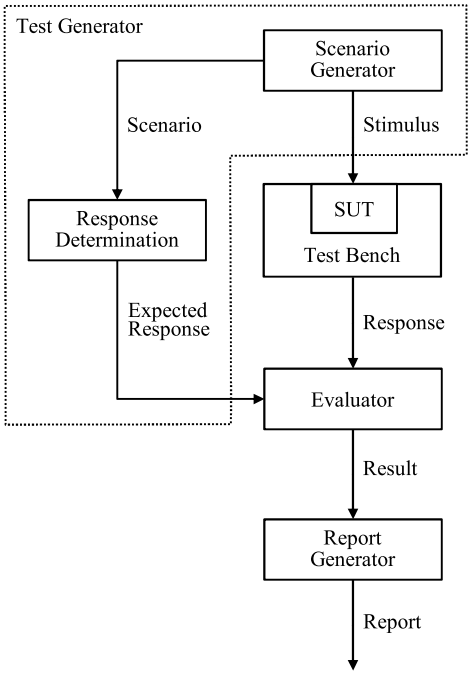

Figure 2. Setup for the automatic testing

The *Report Generator* processes the raw data provided by the *Evaluator* and creates a detailed *Report*, which allows a person with appropriate domain knowledge to analyze failed test cases and to verify successfully executed test cases. To speed-up the analysis, relevant signal characteristics are diagrammed and, where required, derived values and signals are calculated in advance.

B. Need for an Automatic Evaluation

Tools for generating scenarios usually do not provide the expected responses of the SUT, which constitutes the basis for an automatic evaluation. They leave it to the tester to define them. Without an automatic evaluation the generated scenarios can be executed at a test bench, but the actual behavior of the SUT cannot be automatically evaluated. This means that each scenario must be analyzed manually by an expert before the first execution. In this manner the evaluation can be done for individual scenarios, but this is not feasible in practice for the expected large number of generated scenarios needed for the testing.

V. Evaluation Based on Abstraction Level Constraints

In order to benefit from the advantages of automatic testing, a constraint based approach is presented to determine the response of the SUT working on different levels of abstraction. It uses an automatic analysis of the observed behavior to classify individual situations of the scenarios on the system-level after the test execution. Thereby the approach recognizes an unusual behavior of the SUT initially on the system-level from the viewpoint of an outside observer and is subsequently going down to lower levels.

For being able to implement the determination of the response on the signal-level, a profound expert knowledge is necessary to determine whether an output signal provides a correct value, or not. Dependencies between signals complicate the evaluation in addition. On the input interface, it is a common practice to use models to abstract the large number of input signals. The same is done by the approach on the output interface. Through the use of models the abstraction at the output is increased to a higher level. As a result of this, less knowledge about the functionality of the DAS is necessary to evaluate the response. However, missing parts of the overall system have to be simulated due to the increasing of the abstraction. The higher the level of the abstraction, the more parts are missing. For the evaluation of a DAS on the system level, at least a physical model of the system vehicle and simulations of all other involved ECUs are necessary.

The approach uses a parameterized model for the evaluation on the system-level, which spans a safety area around the road objects. As shown in Fig. 3, the safety area is thereby defined by the variables d_1, d_2, d_3 and d_4 that represents the safe distance in each direction. These variables can be changed over the time and thus dynamically adapted to the current situation. The safety area can be, e.g., increased in specific directions depending on the vehicle velocity.

A. Classification

For the evaluation, the observed behavior of the DAS is analyzed after the test execution to classify individual situations of the performed scenario according to the following four classes.

The class “Fallback” is a specified and thus explicitly allowed state of a DAS. The state is not necessarily critical, but rather it indicates a situation that cannot be handled by the system. As a precaution, the DAS returns the vehicle control to the driver at the expense of its availability. In relation to highly automated driving, these situations still reveal functional restrictions of the system.

Definition 1 (Fallback): The fallback is a state of the DAS, which is entered if the system cannot handle the situation by its own and returns the vehicle control to the driver.

A “Hazardous Situation” is a critical situation without a collision that is either caused by the DAS itself or by at least one object included in the scenario. On the one hand, an object vehicle can cause such a situation, e.g., during a lane change if the scenario generator has ignored the safety distance and thus the object gets too close to the system vehicle. On the other hand, the DAS can cause the critical situation, e.g.,

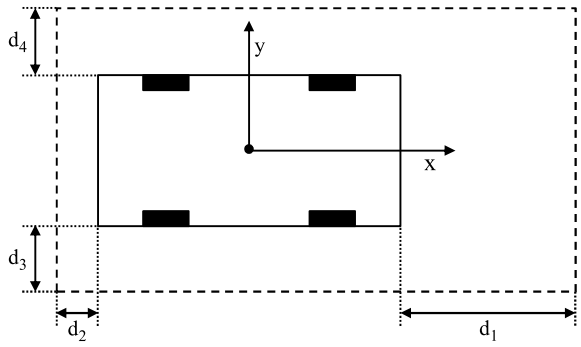

Figure 3. Safety area spanned around a vehicle

by following too closely on an object vehicle. From when a situation is considered as hazardous heavily depends on the conditions given by the scenario. The change of only one condition can lead to a new driving scenario with a different hazard potential.

Definition 2 (Hazardous Situation): A hazardous situation occurs, when:

a) the system vehicle leaves the lane without cause.
b) the minimum distance between the system vehicle and an object is less than a specified value.

The class "Event of Damage" means that the DAS was not able to avoid a collision. Further investigations are required to find out, whether there is any misconduct of the system, or not. The event of damage is usually preceded by a hazardous situation.

Definition 3 (Event of Damage): An event of damage occurs, when:

a) the system vehicle leaves the road.
b) a collision between the system vehicle and one or more objects cannot be avoided by the DAS.

All other situations are classified as "Unsuspicious", which is used as the default class.

Definition 4 (Unsuspicious): A situation is unsuspicious, if an event of damage or a hazardous situation does not exist, as well as, the DAS is not in the fallback state.

In general, the operating mode of the DAS during a situation is crucial for the evaluation. If the DAS is in an emergency situation, its behavior is different from the behavior in the driving mode. While a collision in the driving mode is not acceptable, an unavoidable collision that was mitigated in an emergency situation might be tolerable.

B. Determination of the Expected Response

The determination of the expected response works on a knowledge base individually created for each DAS on the basis of the available specifications. In the knowledge base, a hierarchical structure ensures that the behavior of the SUT is stored dependent on its abstraction level. At the beginning only the behavior described on a high level is used to fill the knowledge base. Already after a short time, a state is thus reached, which allows the determination of the expected response of the DAS. This turns out to be sufficient to execute the first test cases and to get a test result, which guides the system developer to refine the behavior on lower levels and to establish relations between different abstraction levels.

In addition, the use of constraints allows a selective description of the behavior. Based on the stimulus, a distinction can be made at each abstraction level to describe deviating responses of the SUT. In this way, different situations can be handled.

VI. Case Study

In this section, two examples, which were done as a case study, are discussed to show the idea behind the approach. All scenarios of both examples are executed at a SIL test bench. The first example demonstrates a passing maneuver that is analyzed using dynamically expandable safety areas around the objects. It describes, how individual situations of a scenario are classified corresponding to the Section V-A. Following this, a second example is presented that uses an algorithm based on [11] as a SUT to provide the functionality of a CMS. It explains the determination of the expected response on the system-level and discusses the results with respect to the performed scenarios.

A. First Example: Classification of a Scenario

The safe distance between objects in the road traffic heavily depends on a variety of factors, such as the vehicle velocity or the weather, and cannot be specified by generally valid values. Even in the law, usually no specific values are specified, but a sufficient safe distance is stipulated, e.g., in the German Road Traffic Act [12], to avoid hazardous situations or collisions with other road users. The distances considered as safe vary with the velocity, the driving direction or environmental conditions.

As the basis for the example, the model of the safety area used for classification was parameterized according to the two-second rule [13][14] (in some states also known as three-second rule), which states that a driver of a vehicle should stay at least two seconds behind the vehicle in front. During the test execution the safety area are dynamically expanded in the driving direction based on the vehicle velocity with a lower saturation at one meter. The other parameters of the model, i.e., the safe distance to the left and to the right, as well as, the safe distance to the rear, are set to a constant value of one meter.

The scenario used for the example represents a passing maneuver at high speed, as illustrated on the left side of Fig. 4, in which an object vehicle passes the system vehicle on the left lane. The timing of the passing maneuver has been chosen such that it is hazardous by violating the safety distance but not damaging.

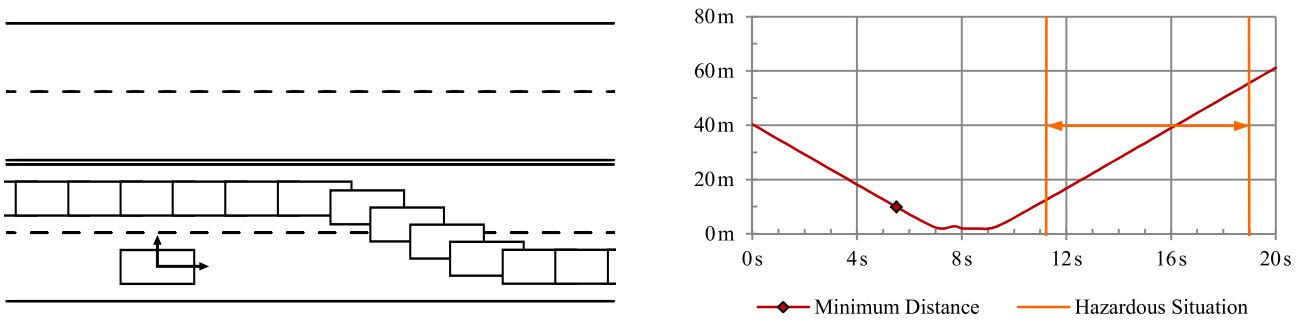

Figure 4. Passing maneuver illustrated as a difference representation (left side) and the corresponding test result (right side)

The test results diagrammed on the right side of Fig. 4 show the detection of a hazardous situation. Thereby, it is striking that the situation has been classified as hazardous after the minimum distance between the system vehicle and the object vehicle had already reached its minimum value. The minimum value is achieved, when both vehicles are at the same level. At this point in time, the distance between the left side of the system vehicle and the right side of the object vehicle are considered by the model only. The first instance of violating the safe distance can be seen during the lane change. At this, the safe distance of the system vehicle is violated by the object vehicle and the hazardous situation is recognized by the model based on the two-second rule.

The example shows that the obtained sequence of classified situations describes the scenario on the system-level, which can be used in the next step to evaluate the behavior of the SUT. It is thus not necessary to directly cope with signals.

B. Second Example: Behavior Analysis in Different Environmental Conditions

The Collision Mitigation System (CMS) [15], which can be found nowadays in almost all new vehicles, monitors the traffic around the vehicle and warns the driver of potential collisions in hazardous situations. If the driver does not react to the warning, an automatic braking is performed. The success of the system, whether a collision can be avoided or at least the effects of a collision can be reduced, is highly determined by the environmental conditions.

On the one hand, the automotive manufacturer has to ensure that no unnecessary triggering of an automatic braking is performed by the CMS, which can cause a threat on the road. On the other hand, the system should provide an added value to the driver in as many situations as possible. As elucidated in [16], there is a trade-off for the automobile manufacturers between safeness and availability.

In contrast to other DASs, the driver sees the CMS only in action in hazardous situations or at collisions. The same applies for testers, which have to put themselves in danger for the testing of the system. Although there is the opportunity to simulate virtual objects for the system vehicle [17] and thus to reduce the risk, the number of tests that can be performed is limited and in no relation to the possible scenarios. Through automatic testing, a variety of different scenarios can be executed on test benches. Thereby, the presented approach provides the expected responses of the DAS for the generated scenarios on different abstraction levels, which can be compared with the observed response obtained from the test bench.

The determination of the expected response is based on the specification of the CMS. Only the behavior described on a high level is used in the following to get a test result within a short time. In further work, the determination can be extended to support additional abstraction levels down to the signal-level.

Two characteristic scenarios for the CMS are presented in the following:

1) Reaching the Tail End of a Traffic Jam at Low Speed without the Driver Applying the Brake: Based on the scenario a hazardous situation is determined, where it comes to a brake intervention by the CMS. Through the intervention, the system vehicle should be slowed down to standstill, before there can be a front-end collision with the object vehicle ahead. As shown by the test results on the left side of Fig. 5, the minimal distance between the system vehicle and the object vehicle decreases over the time. A hazardous situation is recognized, but there is, as expected, no collision. Shortly before standstill, the situation is no longer evaluated to be hazardous due to the automatic braking.

2) Reaching the Tail End of a Traffic Jam at High Speed without the Driver Applying the Brake: In contrast to the previous scenario, the system vehicle has a much higher velocity in this situation than before. Due to the increased velocity, a brake intervention with subsequent collision is determined. The test results, as diagrammed on the right side of Fig. 5, shows that the minimal distance between both vehicles rapidly decreases. Also a hazardous situation is recognized, but this time there is an event of damage caused by the collision of the system vehicle and the object vehicle. After the collision, no further information was provided by the test bench.

The example shows that the determination of the expected response on the system-level can be used for an automatically evaluation of driving scenarios within unknown environmental situations. The abstraction leads to a simplification of the evaluation.

VII. Conclusion and Future Work

Automatic testing, which can be used within traffic simulations, would have the potential for analyzing the response of a DAS based on a large number of different scenarios with reasonable effort. However, an appropriated determination of the expected response and a convincing approach for an evaluation are mostly missing nowadays. Due to the expected large number of generated test cases for the automatic testing and the time-consuming definition of the expected responses, the

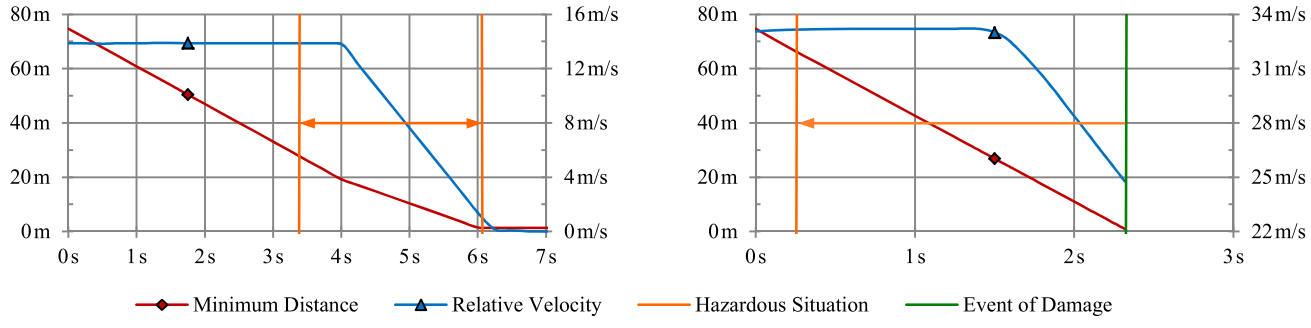

Figure 5. Test result of the slow maneuver (left side) and the test result of the fast maneuver (right side)

determination and the evaluation should be done automatically by the test generator and not manually performed by experts with appropriate domain knowledge about the DAS and its included functionality.

It has been shown that the complexity of the determination and the evaluation, which arises due to the number of signals in the output interface of the SUT, is manageable through the use of models. The higher the level of abstraction at the output interface is chosen, the less domain knowledge is necessary for the evaluation. The abstraction and the resulting simplification cause that not all information from the signal-level are available at each level of abstraction. By increasing the level of abstraction, parts of the overall system must be simulated. The closer the simulation to reality, the more reliable the results obtained. However, for software-driven testing issues a sufficient imitation of the real-world system is supposed to be precise enough. A simulation that considers all eventualities is usually not necessary.

Furthermore, it has been shown that the behavior of a SUT can be determined after the test execution by classifying the response observed at the test bench. Thereby, the obtained sequence of classified situations describes the driving scenario on the system-level. This sequence can be automatically evaluated based on the determined response und used to find errors or missing parts in the specification.

It is left for future work to apply the current approach to a DAS with several assistance functions. One aspect might be to analyze, how many abstraction levels are required to model the behavior of the SUT and to examine at which abstraction level constraints are necessary to correctly determine the expected response based on the stimulus. Another aspect might be to optimize the scenario generator to increasing the search space coverage with a minimum number of additional test cases.

REFERENCES

[1] A. Zanten and F. Kost, Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort. Cham: Springer International Publishing, 2016, ch. Brake-Based Assistance Functions, pp. 919–967. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12352-3_40

[2] S. Prialé Olivares, N. Rebernik, A. Eichberger, and E. Stadlober, Advanced Microsystems for Automotive Applications 2015: Smart Systems for Green and Automated Driving. Cham: Springer International Publishing, 2016, ch. Virtual Stochastic Testing of Advanced Driver Assistance Systems, pp. 25–35. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-20855-8_3

[3] A. Rink, E. Chrisofakis, and M. Tatar, "Automating test of control software," ATZelektronik worldwide, vol. 4, no. 6, pp. 24–27, 2009. [Online]. Available: http://dx.doi.org/10.1007/BF03242245

[4] M. Tatar, "Test and validation of advanced driver assistance systems automated search for critical scenarios," ATZelektronik worldwide, vol. 11, no. 1, pp. 54–57, 2016. [Online]. Available: http://dx.doi.org/10.1007/s38314-015-0574-1

[5] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive formal verification of an os microkernel," ACM Trans. Comput. Syst., vol. 32, no. 1, pp. 2:1–2:70, Feb. 2014. [Online]. Available: http://doi.acm.org/10.1145/2560537

[6] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "sel4: Formal verification of an os kernel," in Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 207–220. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629596

[7] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, "A comparative study on automated software test oracle methods," in Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on, Sept 2009, pp. 140–145.

[8] M. D. Adams, Continuous-Time Signals and Systems. Victoria: University of Victoria, 2013, ch. Continuous-Time Signals and Systems, pp. 7–44, ISBN: 978-1-55058-495-0.

[9] D. Ulmer, S. Wittel, K. Hünlich, and W. Rosenstiel, "Testing Platform for Hardware-in-the-Loop and In-Vehicle Testing Based on a Common Off-The-Shelf Non-Real-Time PC," International Journal on Advances in Systems and Measurements, vol. 4, no. 3 & 4, pp. 182–191, 2011, ISSN: 1942-261x. [Online]. Available: http://www.iariajournals.org/systems_and_measurements/

[10] K. Hünlich, D. Ulmer, S. Wittel, and U. Bröckl, "Optimized Testing Process in Vehicles Using an Augmented Data Logger," International Journal on Advances in Systems and Measurements, vol. 6, no. 1 & 2, pp. 72–81, 2013, ISSN: 1942-261x. [Online]. Available: http://www.iariajournals.org/systems_and_measurements/

[11] H. Winner, Fundamentals of Collision Protection Systems. Cham: Springer International Publishing, 2016, pp. 1149–1176. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12352-3_47

[12] H. Janker, Road Traffic Law - Text Edition ("Strassenverkehrsrecht - Textausgabe"), 53rd ed. Stuttgart: Dt. Taschenbuch-Verlag, 2015.

[13] Road Safety Authority Ireland, "Driving safely in traffic - the two second rule," 2016, URL: http://www.rotr.ie/rules-for-driving/speed-limits/speed-limits_2-second-rule.html [retrieved: July, 2016].

[14] New York State Department of Motor Vehicles, "Driver's Manual & Practice Tests - Chapter 8: Defensive Driving," 2016, URL: https://dmv.ny.gov/about-dmv/chapter-8-defensive-driving/ [retrieved: July, 2016].

[15] L. Walchshäusl, R. Lindl, K. Vogel, and T. Tatschke, Advanced Microsystems for Automotive Applications 2006. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ch. Detection of Road Users in Fused Sensor Data Streams for Collision Mitigation, pp. 53–65. [Online]. Available: http://dx.doi.org/10.1007/3-540-33410-6_6

[16] A. Pruckner, R. Stroph, and P. Pfeffer, Handbook of Intelligent Vehicles. London: Springer London, 2012, ch. Drive-By-Wire, pp. 235–282. [Online]. Available: http://dx.doi.org/10.1007/978-0-85729-085-4_11

[17] Steinbeis-Stiftung für Wirtschaftsförderung (StW), "Virtual Testing of Reality," 2015, URL: http://www.steinbeis.de/en/publications/transfer-magazine/edition-042015/virtual-testing-of-reality.html [retrieved: July, 2016].

 ISBN: 978-1-61208-500-5

Anomaly-Detection-Based Failure Prediction in a Core Router System

Shi Jin
and Krishnendu Chakrabarty

Duke University
Durham, NC, USA
Email: shi.jin@duke.edu
krish@ee.duke.edu

Zhaobo Zhang,
Gang Chen and Xinli Gu

Huawei Technologies Co. Ltd.
San Jose, CA, USA
Email: zhaobo.sc.zhang@huawei.com
Gang.C@huawei.com, xinli.gu@huawei.com

***Abstract*—Prognostic health management is desirable for commercial core router systems to ensure high reliability and rapid error recovery. The effectiveness of prognostic health management depends on whether failures can be accurately predicted with sufficient lead time. However, directly predicting failures from a large amount of historical logs is difficult. We describe the design of an anomaly-detection-based failure prediction approach that first detects anomalies from collected time-series data, and then utilizes these "outliers" to predict system failures. A feature-categorizing-based hybrid anomaly detection is developed to identify a wide range of anomalies. Furthermore, an anomaly analyzer is implemented to remove irrelevant and redundant anomalies. Finally, a Support-Vector-Machine (SVM)-based failure predictor is developed to predict both categories and lead time of system failures from collected anomalies. Synthetic data generated using a small amount of real data for a commercial telecom system, are used to validate the proposed anomaly detector and failure predictor. The experimental results show that both our anomaly detector and failure predictor achieve better performance than traditional methods.**

***Keywords*—Anomaly Detection; Failure Prediction; SVM.**

I. INTRODUCTION

A core router is responsible for the transfer of a large amount of traffic in a reliable and timely manner in the network backbone [1]. The complex hardware and software architectures of core router systems make them more vulnerable to hard-to-detect/hard-to-recover errors [2]. For example, a wide range of failures can occur in a complex multi-card chassis core router system:

- Hardware failures: The cards that constitute the chassis system and the components that constitute a card can encounter hardware failures. Moreover, connectors between cards and interconnects between different components inside a card are also subject to hard faults. A multi-card chassis system can have tens of different cards, each card can have hundreds of components, and each component consists of hundreds of advanced chips. Each chip in turn has hundreds of I/Os and millions of logic gates, and the operating frequency of chips and I/Os are now in the GHz range. Such high complexity and operating speed lead to an increase in incorrect or inconsistent hardware behaviors. Moreover, in such a large-scale complex system, whenever a hardware failure occurs in the chassis system, it is difficult for debug technicians to accurately identify the root cause of this failure and take effective corrective action [3][4].
- Software failures: The entire chassis system and each card have their own software platforms to control and manage different network tasks. However, since the performance requirement of network devices in the core layer is approaching Tbps levels, failures caused by subtle interactions between parallel threads or applications have become more frequent. These failures often arise because software applications tend to distribute their tasks into parallel agents in order to improve performance [4][5].

All these different types of faults can cause a core router to become incapacitated, necessitating the design and implementation of fault-tolerant mechanisms for reliable computing in the core layer.

Due to the non-stop utilization (99.999% uptime) requirement of core router systems deployed in the network backbone, a traditional fault-diagnosis system is of limited applicability here because it aims at repair after failures occur. Such solutions inevitably stall system operation. In contrast, prognostic health management is promising because it monitors the system in real time, triggers alarms when anomalies are detected, and takes preventive actions before a system failure occurs. Therefore, it ensures non-stop utilization of the entire system [6]. The effectiveness of prognostic health management depends on whether system failures can be predicted in a timely manner [7]. Therefore, in this paper, we present the design of an efficient anomaly-detection-based failure predictor that can be applied to a commercial core router system.

The remainder of this paper is organized as follows. Section II discusses the anomaly detection and failure prediction problems in more detail and highlights the contributions of this paper. In Section III, a number of time-series-based anomaly detection techniques are discussed and a feature-categorization-based hybrid method is proposed. Then, a correlation-based anomaly analyzer is described to select representative anomalies. Section IV discusses how to predict failures based on

 ISBN: 978-1-61208-500-5

historical anomaly events. In Section V, experimental results on a synthetic data set generated from a commercial core router system are used to demonstrate the effectiveness of the proposed method. Finally, Section VI concludes the paper and discusses future works.

II. Problem Statement

Prognostic health management can benefit from reasoning-based and data-driven techniques [8], as shown in Fig. 1. The system is monitored by recording different Key Performance Indicators (KPIs). The logged KPI data is then fed to an anomaly detector. When anomalies occur, an anomaly analyzer can be used to filter redundant and irrelevant anomalies. Then, a failure predictor can be triggered to forecast the occurrences of different system failures [7]. Finally, appropriate preventive actions can be executed on the monitored system to avoid failures in advance. We can see that the anomaly detector and failure predictor are two essential components in a data-driven prognostic health management scheme.

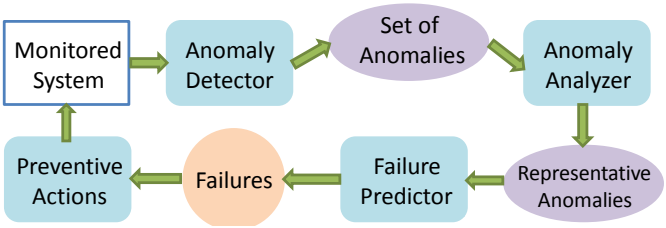

Figure 1. An illustration of data-driven prognostic health management.

Anomaly detection, which is also sometimes referred to as outlier detection, has been widely used in other domains, e.g., intrusion detection and fraud detection [9][10]. For example, density-based techniques, such as K-Nearest Neighbor (KNN) have been used in detecting outliers in high-dimensional datasets [11]. Machine-learning methods, such as Artificial Neural Networks (ANN) have also been applied to detect fraud in large multivariate databases [12]. A Multivariate State Estimation Technique (MSET) has been used to reduce or eliminate No-Trouble-Found [13]. This technique is sensitive to subtle changes in the signal trend, making it effective for detecting indirect anomalies [14].

Failure prediction has also been studied to assess the reliability, availability and serviceability of complex computing systems [15]. For example, a semi-markov reward model has been used to forecast the resource exhaustion problem in software systems [16]. Machine-learning methods, such as Naive Bayes have also been applied to predict hard disk drive failures [17]. A rule-based model has been built to predict attacks in computer networks and illegal financial transactions [18]. However, little research has focused thus far on combining failure prediction with anomaly detection in a high-performance and complex communication system.

The difficulty of developing an efficient anomaly detector and failure predictor for a complex communication system can be attributed to the reason that features extracted from communication systems are far more complex than those from a general computing system. For example, as shown in Fig. 2, a multi-card chassis core router system uses monitors to log a large amount of features from different functional units. These features include performance metrics (e.g., events, bandwidth, throughput, latency, jitter, error rate), resource usage (e.g., CPU, memory, pool, thread, queue length), low-level hardware information (e.g., voltage, temperature, interrupts), configuration status of different network devices, and so on. Each of these features can have significantly different statistical characteristics, making it difficult for a single type of anomaly-detection/failure-prediction technique to be effective.

We, therefore, address the important practical problem of designing an anomaly-detection-based failure predictor that can be effectively applied to a commercial core router system. To achieve this, a feature-categorization-based hybrid method is developed to detect a wide range of anomalies; a correlation-based anomaly analyzer is implemented to select the most important anomalies; and a machine-learning-based failure predictor is developed to predict different failures from

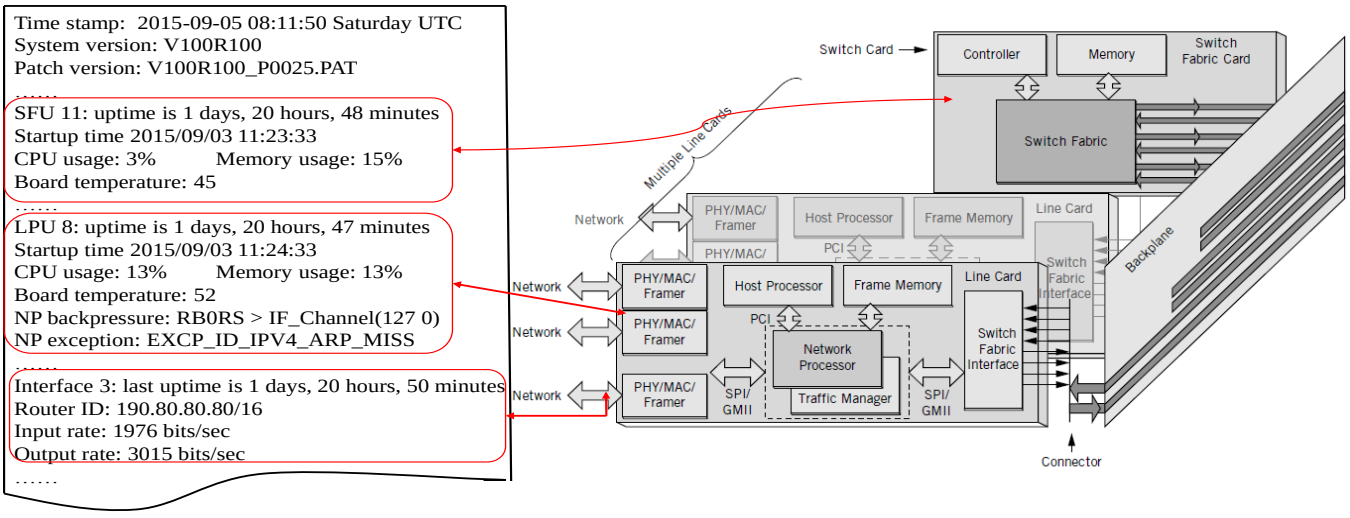

Figure 2. A multi-card chassis core router system and a snapshot of extracted (monitored) features.

historical anomalies.

III. Anomaly Detection and Analysis

In complex communication systems, such as a core router, data is collected in the form of time-series. Three kinds of techniques have been studied in the literature to detect anomalies in time-series data [9]. The first one is distance-based anomaly detection, which utilizes a distance measure between a pair of time-series instances to represent the similarity between these two time-series. The smaller the overall "distance" is, the closer this pair of time-series instances would be. Instances far away from others will be identified as being abnormal. The second one is window-based anomaly detection. This method divides time series instances into overlapping windows. Anomaly scores are first calculated per window and then aggregated to be compared with a predefined threshold. Only when the overall anomaly score of a single time-series instance significantly exceeds a predefined threshold, this instance will be identified as being abnormal. The third one is prediction-based anomaly detection. First, a machine-learning-based predictive model is learned from historical logs. Next, predicted values are obtained by feeding test data to this predictive model. These predicted values are then compared with the actual measured data points. The accumulated difference between these predicted and the actual observations is defined as the anomaly score for each test time-series instance.

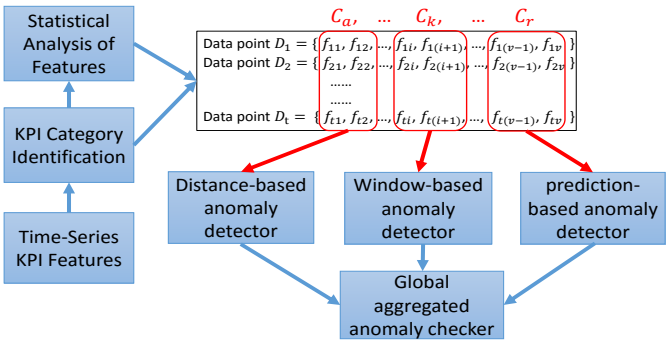

Figure 3. A depiction of feature-categorization-based hybrid anomaly detection.

However, a single class of anomaly detection methods is effective for only a limited number of time-series types. Therefore, we propose a feature-categorization-based hybrid method whereby each class of features can be classified by the most appropriate anomaly detection method. Fig. 3 illustrates the proposed feature-categorization-based hybrid anomaly detection. First, time-series data of different features extracted from the core router system is fed to a KPI-category identification component. Since features belonging to different KPI categories often exhibit significantly different statistical characteristics across the timeline, natural language processing techniques are utilized here to ensure that different KPI categories, such as configuration, traffic, resource type, and hardware can be identified. However, it is also possible that features belonging to different KPI categories have similar trend or distribution across time intervals; therefore, a statistical analysis component is needed to ensure that all features that exhibit similar statistical characteristics are placed in the same class. After these steps, a data point D_t with v features can be divided into different groups $C_a, C_b, \ldots, C_k, \ldots, C_r$, where each group has different statistical characteristics. Next, each group of features is fed to the anomaly detector that is most suitable for this type of features. Finally, the results provided by different anomaly detectors are aggregated so that we can detect an anomaly in terms of the entire feature space.

Although the proposed feature-categorization-based hybrid method can help us detect a wide range of anomalies, not all anomalies are useful and necessary for predicting system failures. First, the temporal and spatial localities of neighboring components lead to co-occurrences of similar anomalies. Second, some anomalies are caused by workload variations or temporary external noise, which makes them irrelevant for predicting system failures. Since the number of possible anomalies will increase from hundreds to tens of thousands when more new features are identified and extracted from the raw log data, anomaly analysis is needed in order to remove irrelevant and redundant anomalies before predicting failures.

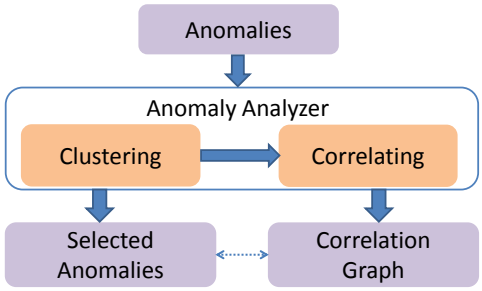

Figure 4. Overview architecture of the proposed Anomaly Analyzer.

Fig. 4 presents an outline of the proposed anomaly analyzer. A set of detected anomalies is fed to the anomaly analyzer. It then goes through two components: the clustering component and the correlating component in sequential order. The clustering component groups anomalies that occur "simultaneously" (within the same small time interval) and have similar statistical characteristics. Only one anomaly is selected from each cluster and then fed to the next component. The correlating component first identifies both linear and non-linear relationships among these anomalies and then group anomalies that have strong correlations. Finally, the anomaly analyzer outputs a number of correlated anomaly groups. An effective anomaly subset can be generated by selecting the most representative anomalies from these correlated groups. Furthermore, detailed relationships among anomalies within each group can be represented by a correlation graph $G = (V, E)$, where the set of vertices V represent anomalies and the set of edges E represent correlations between anomalies. Therefore, a correlation graph is generated for each group of anomalies.

 ISBN: 978-1-61208-500-5

IV. Failure Prediction

Fig. 5 shows the temporal relationship between faults, anomalies, and failures. Assume that a fault occurs in the system at time point t_r. After a period of time, a wide range of anomalies begin to appear at time point t_{as}. Finally, at time point t_f, the system encounters a fatal failure and crashes. Two important time intervals are defined here: δt_l, referred as the lead time, is the time interval between the occurrence of the last anomaly and the occurrence of the predicted failure. It is defined as $\delta t_l = t_f - t_{ae}$. Only if this lead time is larger than the time required to take preventive actions can our prediction become useful in reality. The parameter δt_d is defined as the time interval between the occurrence of the first and last anomaly, i.e., $\delta t_d = t_{ae} - t_{as}$. Since our failure prediction is based on the detected anomalies in the system, δt_d can be considered to the temporal length of our dataset.

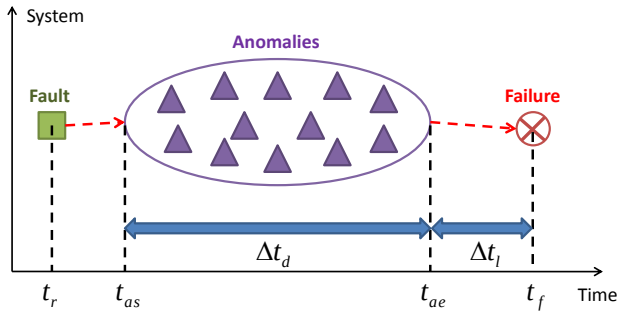

Figure 5. Temporal relationship between faults, anomalies, and failures.

Using the proposed anomaly detector and analyzer, representative anomalies can be identified and recorded. Correlating them with logs of system failures, two types of anomaly event set can be formed: failure-related anomaly event set and non-failure-related anomaly event set. An example of these two types of anomaly event sets is shown in Fig. 6. We can see that A_i represents the ID of each anomaly and F_j represents the ID of each failure. The failure-related anomaly event set consists of records that always end with a failure event F_j while the non-failure-related anomaly event set consists of records that do not have any failure events.

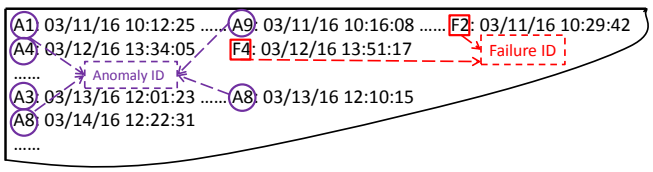

Figure 6. An example of anomaly event sets.

An efficient failure predictor should not only predict whether failures will occur, but also predict the type/category and occurrence time of those failures. Therefore, as shown in Fig. 7, the proposed failure predictor consists of two main components: the classifier and the regressor so that both the category and the lead time of failures can be predicted. First, the historical logs including both failure-related and non-failure-related anomaly events are fed as training data to both the classifier and the regressor in order to build corresponding learning models. Second, a set of newly detected anomalies of is fed to these learning models. Finally, the learnt classifier outputs which type of system failures will be triggered by the current anomalies, and the learnt regressor will output the predicted lead time for this type of system failure.

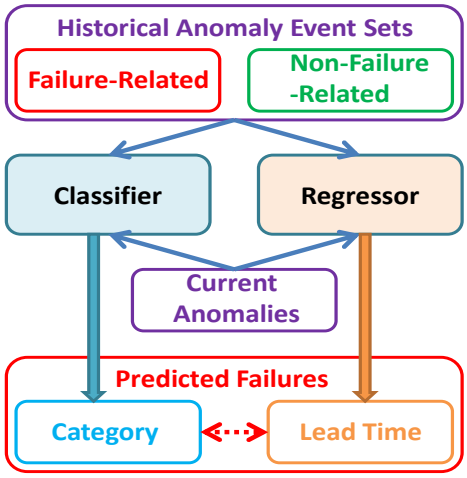

Figure 7. Overview architecture of the proposed failure predictor.

One key step implicit in Fig. 7 is to build training datasets from historical anomaly event sets for both the classification component and the regression component. Suppose we have identified a set of anomalies $\mathbf{A} = \{A_1, A_2, ..., A_N\}$ and a set of system failures $\mathbf{F} = \{F_1, F_2, ..., F_M\}$ from our historical log $\mathbf{H}$. The training dataset $\mathbf{D}$ for the classification component can then be built. For each record H_i in the historical log, it can contain one or more anomalies and either no failure or one failure. If the anomaly A_j appears in the record H_i, $D_{ij} = 1$, otherwise $D_{ij} = 0$. Note that $D_{i(N+1)}$ represents the failure category of the record H_i: If the failure F_k appears in the record H_i, $D_{i(N+1)} = k$. If no failures occur in the record, $D_{i(N+1)} = 0$. The process of building the training dataset $\mathbf{T}$ for the regression component is similar. The only difference is that the occurrence times of anomalies and failures needs to be included now. If the anomaly A_j appears at time t_j in the record H_i, $T_{ij} = t_j$, otherwise $T_{ij} = 0$. If the failure F_k appears at time t_k in the record H_i, $T_{i(N+1)} = t_k$. If no failures occur in the record, $T_{i(N+1)} = 0$.

Different machine-learning techniques can be applied for classification and regression in the proposed failure predictor. Among these techniques, the Support Vector Machine (SVM) algorithm offers several advantages, such as overfitting control through regularisation parameters and performance improvement via custom kernels [19]. Therefore, we utilize SVM-based techniques in this paper. Specifically, we apply multiclass SVM for the classification component and support vector regression for the regression component.

V. Experiments and Results

The commercial core router system used in our experiments consists of a number of different functional units, such as the

 ISBN: 978-1-61208-500-5

main processing unit, line processing unit, switch fabric unit, etc. A total of 602 features are monitored and sampled every 5 minutes for 15 days of operation of the core router system, generating a set of multivariate time-series data consisting of 4320 time points.

To evaluate the performance of the proposed anomaly detection and failure prediction methods, we use a 4-fold *cross-validation* method, which randomly partitions the extracted time series dataset into four groups. Each group is regarded as a test case while all the other cases are used for training. The Success Ratio (SR), referred to as a percentage, is the ratio of the number of correctly detected anomalies/predicted failures to the total number of anomalies/failures in the testing set. For example, a SR of 70% means that 7 out of 10 anomalies are correctly detected. In addition to SR, the Non-False-Alarm Ratio (NFAR) is also considered as an evaluation metric. It is defined as the ratio of the number of correctly detected anomalies/predicted failures to the total number of alarms flagged by the anomaly detector/failure predictor.

A. Anomaly Detection and Analysis

To evaluate the effectiveness of feature-categorization-based hybrid anomaly detection, five base algorithms are implemented: KNN is a distance-based anomaly detection method, and for each test instance, its distance to its kth nearest neighboring instance will be considered as its anomaly score. Window-based KNN and window-based SVM are two window-based methods, and the difference between them is the way they calculate their per-window anomaly score. SVR and AR are two prediction-based methods, and the difference is that the former one uses support vector regression to predict values while the latter uses auto-regression for forecasting.

The results are shown in Fig. 8-9. We can see that for the six anomaly detection methods, i.e., KNN, Window-based KNN, window-based SVM, SVR, AR, and the feature-categorization-based hybrid method, the success ratios are 82.7%, 84.5%, 86.4%, 88.2%, 78.6% and 95.1%, respectively, and the non-false-alarm ratios are 73.1%, 76.3%, 80.7%, 88.1%, 71.6% and 92.1%. The reason that the proposed feature-categorization-based hybrid method achieves much higher SR and NFAR than other methods is that it can overcome the difficulty of adopting a single class of anomaly detection to features with significantly different statistical characteristics.

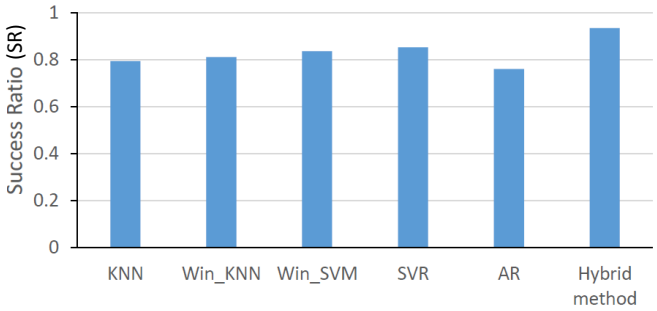

Figure 8. Success ratios of different anomaly detection methods.

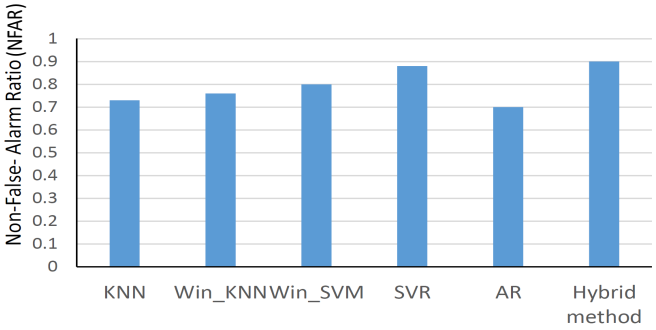

Figure 9. Non-False-Alarm ratios of different anomaly detection methods.

Initially, 467 anomalies are detected by the proposed anomaly detector. The anomaly analyzer can then partition these anomalies into disjoint clusters based on their inner-similarity and inter-correlation. The results of such clustering and correlating are summarized in Table I. We can see that only 15 out of 467 anomalies are identified as being in independent groups (clusters with a single element), which implies that most anomalies are correlated. Moreover, if we choose a single anomaly within each cluster to represent this cluster, only 105 anomalies are needed to represent the entire anomaly set, reducing the number of anomaly dimensions by 77.5%.

TABLE I. RESULTS AFTER CLUSTERING AND CORRELATING OF ANOMALIES.

Size of clusters	Number of clusters	Number of anomalies
1	15	15
2	38	76
3	14	42
4	9	36
6	10	60
10	13	130
15	4	60
21	1	21
27	1	27

B. Failure Prediction

To evaluate the effectiveness of the SVM-based classifier and the SVR-based regressor in the proposed failure predictor, two base algorithms are implemented. For the classification component, a rule-based approach is used: first, a rule model is built from the historical log; each rule takes the form "IF {anomaly A_1, anomaly A_2, ..., anomaly A_i}, THEN {failure F_j}". Second, for each new anomaly set, if a matched rule can be found, the failure label of that rule is assigned to the new anomaly set; otherwise, a random failure label is assigned. For the regressor component, a simple linear regression method is used to predict the lead time of a failure from the occurrence time of its related anomalies.

Fig. 10-11 show the SR and NFAR values for the SVM-based and the rule-based approaches. Eight failure categories are identified from the historical log, and are denoted as A, B, ..., H in the figures. The results can be summarized as follows:

1) For all eight failure categories, the SVM method achieves higher SR and NFAR than the rule-based method. One

 ISBN: 978-1-61208-500-5

possible explanation is that the effectiveness of the rule-based method highly depends on whether the rule model covers a sufficient range of "IF anomalies, THEN failure" rules. However, there are always new anomaly sets that do not match any existing rules, and therefore cannot be predicted well by the rule-based method. In contrast, the SVM method can learn "implicit rules" during its training phase, making it more suitable for predicting failure categories of new anomaly patterns.

2) Both methods perform better in predicting failure categories A and G, but they are worse in predicting failure categories C and F. After analyzing the anomaly event sets related to these failure categories, we find that the anomaly event sets for A and G are significantly different while the anomaly event sets for C and F are very similar. In some cases, the anomaly sets of C and F have exactly the same anomaly events and the only difference is the sequential order of these anomaly events. Since both SVM and rule-based methods do not take this sequential information into consideration, it is quite likely that misclassification will occur when predicting the failure category C and F.

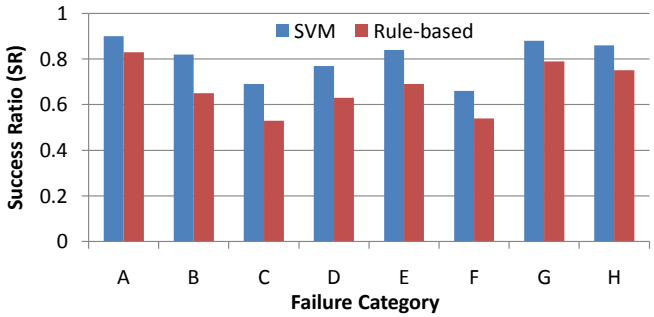

Figure 10. Success ratios of two failure category prediction methods.

Figure 11. Non-False-Alarm ratios of two failure category prediction methods.

The classical metric Root Mean Square Error (RMSE) is used to evaluate the effectiveness of the failure lead time prediction methods. For our experiments, we define RMSE as the square root of the average squared distance between the actual lead time and the predicted lead time. The results are shown in Fig. 12. We can see that the SVR method achieves much lower RMSE than the linear regression method.

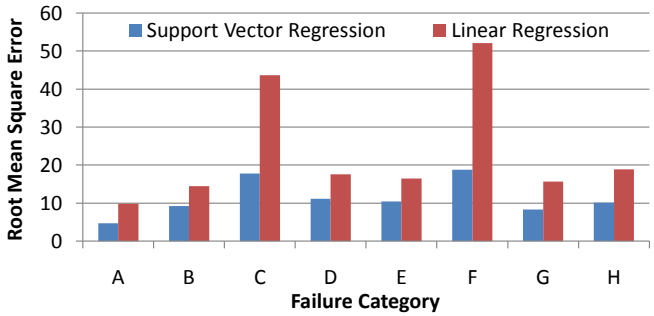

Figure 12. RMSE of two failure lead time prediction methods.

The reason is that in most cases, the temporal relationships between anomalies and failures are not linear. Also, we find that the RMSE for the SVR method for most failure categories is not greater than 10 minutes, which means the lead time predicted by the SVR method can be considered as a realistic approximation.

VI. Conclusion and Future Works

We have described the design of a anomaly-detection-based failure predictor for a complex core router system. Both a feature-categorization-based hybrid anomaly detector and a correlation-based anomaly analyzer have been implemented to detect and identify important anomalies. A SVM-based failure predictor has also been developed to predict the category and lead time of different failures from anomaly event sets. Data collected from a commercial core router system has been used to evaluate the effectiveness of the proposed methods. The experimental results show that the proposed anomaly-detection-based failure predictor achieves not only higher success ratio and non-false-alarm ratio than traditional rule-based method in predicting failure categories, but also lower root mean square error than linear regression method in predicting failure lead time.

However, several drawbacks exist in current work and need to be addressed in the future:

1) The proposed anomaly detector did not take correlations among features into account. Therefore, this method cannot capture anomalies caused by abnormal combination of multiple features. A correlation engine will be investigated in the future to detect multivariate anomalies.
2) The proposed failure predictor did not take sequential information of anomaly events into consideration. Therefore, this method cannot accurately identify failure categories if they share similar anomaly event set. A time-series-based failure predictor will be investigated in the future to better forecast different types of failures.
3) A key assumption in current work is that failures and anomalies are well-correlated. However, this assumption may not always hold true in real scenarios. Whether a sequence of anomalies will trigger a specific failure depends on a variety of factors such as software aging, hardware update, or even human intervention. Therefore, data from other sources such as business scenarios, system configurations, expertise experiences will be incorporated

 ISBN: 978-1-61208-500-5

and investigated in the future to build more fine-grained relationships among anomalies and failures.

ACKNOWLEDGMENT

The work of S. Jin and K. Chakrabarty was supported in part by a grant from Huawei Technologies Co. Ltd.

REFERENCES

[1] V. Antonenko and R. Smelyanskiy, "Global network modelling based on mininet approach." in Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 2013, pp. 145–146.

[2] M. Médard and S. S. Lumetta, "Network reliability and fault tolerance," Encyclopedia of Telecommunications, 2003.

[3] S. Tanwir, S. Prabhu, M. Hsiao, and L. Lingappan, "Information-theoretic and statistical methods of failure log selection for improved diagnosis," in Proc. IEEE International Test Conference (ITC), 2015, pp. 1–10.

[4] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," in Proceedings of the International Conference on Dependable Systems and Networks, 2006, pp. 249–258.

[5] P. K. Patra, H. Singh, and G. Singh, "Fault tolerance techniques and comparative implementation in cloud computing," International Journal of Computer Applications, vol. 64, 2013, pp. 1–6.

[6] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008, pp. 43:1–43:12.

[7] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into HPC systems," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 77:1–77:11.

[8] H. H. Chen, R. Hsu, P. Yang, and J. J. Shyr, "Predicting system-level test and in-field customer failures using data mining," in Proc. IEEE International Test Conference (ITC), 2013, pp. 1–10.

[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys (CSUR), vol. 41, 2008, pp. 15:1–15:58.

[10] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," Computer Networks, vol. 51, 2007, pp. 3448–3470.

[11] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," Computers & Security, vol. 21, 2002, pp. 439–448.

[12] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in Proc. Int. Joint Conf. Neural Networks, vol. 2, 2002, pp. 1702–1707.

[13] F. Bockhorst, K. Gross, J. Herzog, and S. Wegerich, "MSET modeling of crystal river-3 venturi flow meters," in Proc. Int. Conf. Nuclear Engineering, 1998, pp. 1–17.

[14] K. Vaidyanathan and K. Gross, "MSET performance optimization for detection of software aging," in Proc. IEEE Int. Symposium on Software Reliability Engineering (ISSRE), 2003.

[15] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," ACM Comput. Surv., vol. 42, 2010, pp. 10:1–10:42.

[16] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in Proc. International Symposium on Software Reliability Engineering, 1999, pp. 84–93.

[17] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in Proc. International Conference on Machine Learning, 2001, pp. 202–209.

[18] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss, "Predictive algorithms in the management of computer systems," IBM Syst. J., vol. 41, 2002, pp. 461–474.

[19] C. Cortes and V. Vapnik, "Support-vector networks," Journal of Machine Learning, vol. 20, 1995, pp. 273–297.

 ISBN: 978-1-61208-500-5