Karlsruhe Reports in Informatics 2014,2
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Using a Context Knowledge Base for the Verification of Vehicle Test Processes

Richard Mrasek, Jutta Mülle, and Klemens Böhm

2014

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Using a Context Knowledge Base for the Verification of Vehicle Test Processes

Richard Mrasek[1], Jutta Mülle[1], and Klemens Böhm[1]

Karlsruhe Institute of Technology
Institute for Program Structures and Data Organization
76131 Karlsruhe, Germany
{richard.mrasek|jutta.muelle|klemens.boehm}@kit.edu

Abstract. Diagnostic frameworks in the vehicle production domain are important to guarantee the quality of the outcomes of the manufacturing process. Recently, the test processes in these frameworks are becoming more complex due to the use of a large amount of components and electronic devices in modern vehicles. To achieve high quality test processes we take a model checking approach to check that the test processes comply with relevant requirements. We have analyzed the requirements for vehicle test processes and the relevant context information for verification. We use this information to create a context knowledge base, containing information about the components and their relationships and constraints of our application domain, i.e. vehicle test processes. Our goal is to automatically verify test processes, specified in different notations, with a model checking approach. The requirement rules are dynamically generated at verification time from the context knowledge base. The approach is evaluated with actual test processes of our industrial partner.

1 Introduction

1.1 Motivation

Diagnostic frameworks are workflow management systems (WfMS) that plan and coordinate the testing and end of line manufacturing of, e.g., vehicles. The dagnostic frameworks execute test processes and invoke manual or automatic activities that either test a component or put the component into service, e.g., configuration of the software [28]. The process designers need to model for each new vehicle project a series of process schemata. Until now only minor focus has been set on these industrial test processes as an application field for workflow technique and theory. With this work we want to show that business process verification techniques can be used to increase the quality of test processes in an automotive domain. Each process schema needs to fulfill a set of requirement rules, similar to compliance rules ([15], [17]). For example, a requirement rule states that before task `X` another task `Y` has to occur. If one of these rules is violated an unwanted behaviour occurs and can lead to a deadlock or a component not tested properly. To this end, we will verify test processes against a set

of requirement rules. To apply an automatic verification technique,in particular model checking, we need to specify the requirements in a formal language like a temporal logic [22]. The specification in a temporal logic is error-prone and not feasible for domain experts untrained to formal specification. To this end, graphical modeling languages (Compliance Rule Graphs [20], BPMN-Q [3]), and specification-pattern based approaches, e.g., [10] and [24] has been proposed. In our application domain a large amount of short and relatively similar rules exist. Only few different types of requirements are relevant, but the configuration of the requirement rules depends on the context of the testing process. For example, some requirements only hold for a specific vehicle project or only when the test process is executed at a particular test station in the factory. We want to present an approach that is capable of verifying a testing process against domain-specific requirement rules.

1.2 Challenges

Most of the requirement rules are context-sensitive, i.e., only apply under specific context situations of the test process. The knowledge about the requirements is distributed in different departments and by different employees. A central documentation is missing and most requirements merely exist in the mind of individual process modelers. The relevant context information is missing a central documentation. The information is described either as XML files, in textual form, or exists in several databases.

Additionally, we are dealing with a highly heterogeneous environment. The factories are located in several countries on four continents. The process models use different notations to describe the process schema depending on the factory and the development framework used.

1.3 Contributions

With the help of our industrial partner we have analyzed the requirements for vehicle test processes and the relevant context information for verification. We use this information to create a context knowledge base, containing information about the components and their relationships and constraints of our application domain, i.e. automotive test processes. Our goal is to automatically verify test processes, specified in different notations, with a model checking approach. The requirement rules are dynamically generated at verification time from the context knowledge base. The approach is evaluated with actual test processes of our industrial partner.

2 Scenario

After manufacture the test process has to check for each vehicle produced if all of its Electronic Control Units (ECU) function correctly, and has to put the ECUs into service. To check an ECU, a number of test routines, in the following also

called tasks, need to be executed. The tasks can either be automatic or require a factory worker (manual task). Hundreds of tasks need to be executed for each vehicle. For example, for the vehicle series A6/A7 more than 2700 tasks are specified and need to be tested.
Test processes plan and schedule the test routines. Each routine communicates with exactly one component of the vehicle. The components are arranged in a Master-Slave relationship, see Figure 1. The test processes are executed at a specific station in the factory called process place. For each vehicle project and each process place at least one test process exists.

Example 1. A vehicle of the A6 series is tested at process place VP2. To this end, the test process (A6_VP2) is executed by the WfMS. The WfMS invokes tasks that the ECU automatically executes or the task is presented to a worker. One of these tasks, namely task x, checks if the injection system works properly. For this purpose test routine x communicates with the ECU of the engine MOT.

3 Notations

In this section we want to introduce the notations used in this paper, i.e. Petri Nets as formal model of a workflow used for verification, and CTL as language to specify requirements to be verified. Due to lack of space we limit the definitions to a minimum. For a more detailed introduction see one of the standard literature, e.g., [11] and [8].

A Petri net is a directed bipartite graph with two types of nodes called places and transitions. It is not allowed to connect two nodes of the same type.

Definition 1 (Petri net). *A Petri net is a triple* (P, T, F)

- P *is a set of places*
- T *is a set of transitions* $(P \cup T = \emptyset)$
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs*

We say that $p \in P$ is an input place of $t \in T$ if $(p, t) \in F$ and an output place if $(t, p) \in F$. $\bullet t$ denotes the set of input places of t and $t\bullet$ the set of output places. A mapping $M : P \to \mathbb{N}_0$ maps each $p \in P$ to a positive number of tokens. We call this distribution of tokens over places (M) a state of the Petri net. A transition $t \in T$ is activated in a state M if $\forall p \in \bullet t : M(p) \geq 1$. A transition $t \in T$ in M can fire, leading to a new state M' with:

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in \bullet t \\ M(p) + 1 & \text{if } p \in t\bullet \\ M(p) & \text{else} \end{cases}$$

Fig. 1. Relationship between Routine, ECU and Master

We call the set of states reachable from a start state M_0 of a Petri net its state space.

CTL (**C**omputation **T**ree **L**ogic) is a temporal logic to specify requirements. Automatic model checking algorithms exist to efficiently verify CTL requirements [7]. The formal syntax of CTL is given as follows:

Definition 2 (Computation Tree Logic:). *Every atomic proposition $p \in AP$ is a CTL formula. If ϕ_1 and ϕ_2 are CTL formulas $\Rightarrow \neg\phi_1$, $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\mathsf{AX}\phi_1$, $\mathsf{EX}\phi_1$, $\mathsf{AG}\phi_1$, $\mathsf{EG}\phi_1$, $\mathsf{AF}\phi_1$, $\mathsf{EF}\phi_1$, $\mathsf{A}[\phi_1\ \mathsf{U}\ \phi_2]$, $\mathsf{E}[\phi_1\ \mathsf{U}\ \phi_2]$ are CTL formulas.*

AP is in our application domain an equation of tokens in one or more places. The operators always occur in pairs: A path operator (A or E) and a temporal operator (X,G,F or U). A means that the formula holds in *all* succeeding execution paths, E means that at least one execution path *exists*,
X means that the formula holds in the next state, G means that formula holds in all succeeding states, F means that the formula holds in at least one succeeding state, and $[\phi_1\ \mathsf{U}\ \phi_2]$ means that ϕ_1 holds until ϕ_2 is reached.

4 Requirements

Our overall goal is to establish a verification tool for vehicle test sequences which is easy to use, easily adaptable to new vehicle variants and adequate for flexible test process execution. To this end, we have hold a series of interviews with the engineers of our industrial partner developing the diagnostic programs and the test schedules.

Using this information we have extracted different types of requirements of a test process, see Section 4.1 and the knowledge about the context influencing the test process and its requirements.

First, we have analyze which kinds of requirements a testing process needs to fulfill. Next we build a knowledge base that contains the information about our application field which is required for verification. In Section 4.2 we describe the requirements on the knowledge base.

4.1 Requirements of a Testing Process

R1 Syntactical Correctness
The test process should be syntactically correct and comply to the naming conventions for tasks of the company.

R2 Installed ECUs
The ECUs are only installed at specific process place. When a test task tries to communicate with a ECU that is not installed at a specific process place, either a direct error occurs or even worse, an error occurs indirectly in subsequent tasks.

R3 Connections of the ECUs

Each ECU opens a connection to one of two transport protocols (UDS or KWP-2000) supported. Each transport protocol can handle 10 open connections in maximum at the same time. In total 14 connections in maximum can be open at the same time.

R4 Task Conditions
Some tasks depend on the occurrence of other tasks in the process, e.g., they can not be run in parallel or need to occur in a certain sequential order.
Table 1 contains the different types of task conditions that can occur.

Table 1. Task and ECU Conditions

Req. Name	Description
R4.1 Sequential before	If a task A is in the testing process a task B has to occur before A.
R4.2 Optional Sequential before	If both, A and B, occur in the testing process, B has to occur before A. B can completely be missing
R4.3 Sequential after	The occurrence of task A leads to the occurrence of task B.
R4.4 Non-Parallel	Tasks A and B are not allowed to occur in parallel.
R5.1 Access	Each ECU c can only be accessed/tested by one task at the same time.
R5.2 Master	Each ECU c should never be accessed in parallel with its master ECU c_m or any other component c_2 with the same master ECU.
R5.3 Non-Parallel	Some ECU c_1 should never be tested in parallel with an ECU c_2.
R5.4 Close Connection	The connection to an ECU c needs to be closed by a certain task A_c.

R5 Component Conditions
Additionally to the conditions on the tasks, other conditions regarding the ECUs exist, see Table 1. These conditions hold for each task that communicates with the respective ECU.

4.2 Requirements of the Knowledge Base

R6 Context-Dependent
The knowledge base should contain contextual information of the test processes. First, the requirements of the test processes depend on the vehicle, i.e. on the components built in the vehicle, which have to be tested. The components are determined by the vehicle type and the concrete configuration of the produced vehicle. Second, the requirements of the test processes can be different dependent on the specific process place the component is tested. Third, there exist

dependencies between the test tasks, see Subsection 4.1.

R7 User-Friendly Specification of the Requirements
Test engineers should be supported to specify the requirements in a comfortable way. To this end, the structure of the knowledge base should support the view of these experts and not require sophisticated experience with formal modelling.

R8 Use of Existing Documents and Information
Defining the requirements should use as much information available from previous steps of the production life cycle. Information about the vehicle and its components, which have to be tested, arises during the production design and production planning. This should be used for filling the knowledge base.

5 Knowledge Base

This section describes the generation and construction of the knowledge base. The knowledge base has the goal to allow to generate automatically requirements for checking testing sequences. In order to manage knowlege about testing sequences in automotive industry we have developed a knowledge base in cooperation with the testing experts of our partner company, see [23]. Figure 2 shows the entity-relationship model of the structure of our knowledge base.

The knowledge is organized in three submodels of context information about text sequences. One submodel contains the test sequence objects and dependencies between the test sequences. Another submodel comprises the vehicle components with variants described as options of the component configurations. A further submodel describes the factories and locations where the testing sequences are executed, so-called process places. Dependencies between the submodels relate these contexts and complete the overall structure of the knowledge base.

In the industrial setting envisioned here, it is necessary to verify hundreds of requirements per process, in short time. In our setting, the requirements are dynamically generated from this knowledge base at the time of verification, see Section 6. In particular, the requirements describe sequential and parallel ordering constraints that certain tasks need to fulfill. In cooperation with our industrial partner, we use real industrial processes that specify the sequential and parallel ordering of testing tasks of an automobile right after its assembly. These processes are the ones that a German automobile manufacturer does carry out in its factories worldwide. The processes are described in OTX notation (Open Test sequence eXchange) [13], a standard to specify testing workflows.

The industrial partner has provided us with 50 processes. They contain between 6 and 813 elementary tasks, arranged in up to 14 parallel lanes. The processes are executed at different test stations, so-called process places, in the factories. For each place, each vehicle series and each factory, process designers need to specify a process by hand. A new vehicle variation again requires a mod-

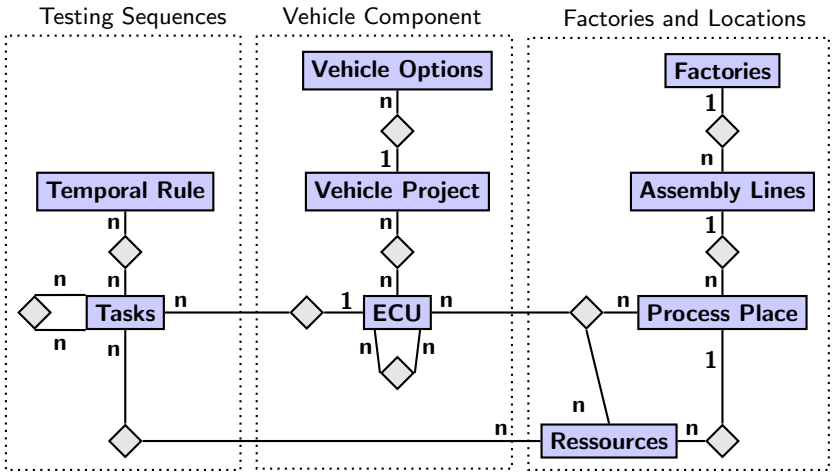

Fig. 2. Simplified Database Schema of the Knowledge Base [23]

ification of the process. The processes for the same place are quite similar in size and complexity. We have collected the requirements which the testing processes have to fulfill by interviews and have used the results to construct the respective knowledge base.

Example 1 illustrates the outcome of such an interview.

Example 2. For an automobile, different testing tasks need to be executed. Each task uses physical components of the testing environment. A task has an identifier that describes the purpose of the task. Two tasks on the same component must not be executed in parallel. Before certain tasks can be executed, another task has to validate if the component is ready and usable. When a process stops using a component, it must close the connection to it before the process can terminate.

6 Verification

The only input parameter of our verification program is the path to the test process file. We accept files in the notation of SIDIS Pro, Prodis.Automation and OTX. If a file is not in OTX notation we transform it to OTX in a preprocessing step. Next, the context information regarding the process place and the vehicle project are extracted from the test process.

Our program consists of two modules: First, the *Data-Reconciliation* and, second, the *Model-Checking*. Figure 3 shows the architecture of the verification system.

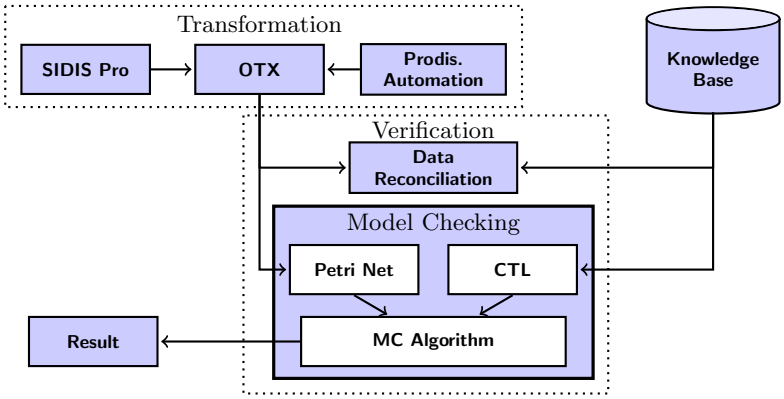

Fig. 3. Architecture of the verification process

6.1 Data Reconciliation

First, the syntactical correctness of the OTX process is tested. To this end, our program validates the test process against the XML schema of OTX. Additionally, we check each task if it complies to the naming conventions of the company (R1).

Then, we check if the task, components and their resources are available in the context of the test process. To this end, we query the Knowledge Base and evaluate if the elements of this context match the elements used in the test process.

6.2 Model Checking

To check the complex requirements we need to consider the actual execution traces of the test process, i.e., we follow a model checking approach. To enable the verification of a OTX process by model checking techniques we first need to transform the process into a formal structure, in our case Petri Nets. To transform the OTX process we transform each element of the process into a subnet and combine the nets. Our approach is similar to the transformation of a BPEL process to a Petri Net of [12] and [25].

Figure 4 shows the pattern for a task, i.e. a test routine. The state *In* marks that task A is activated and ready for execution. If a task execution starts the transition *start* fires and a token is created in the places *run-A*, *C-A* and *P-A*. The actual execution of the task is represented by *run-A*. *C-A* is the place of the component with which A communicates, and *P-A* is the bus protocol that is used by A (either **UDS** or **KWP2000** [28]). At the runtime of the verification the program queries the knowledge base to get the protocol which is used by task A.

A model checking algorithm requires the representation of the requirements in a formal language, e.g., in a temporal logic like CTL. Defining requirements with

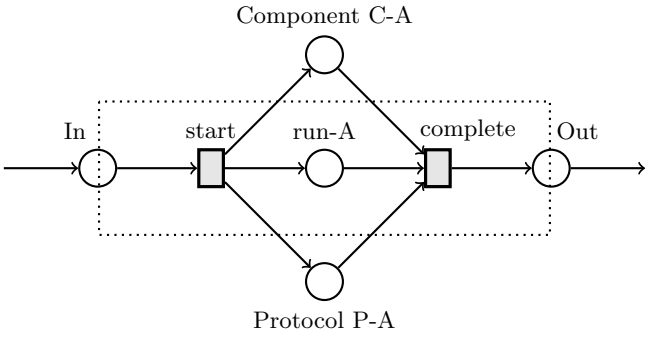

Fig. 4. The Pattern for a Task A

CTL is error-prone and complicated for end-users. Therefore, [10] has introduced a set of property specification patterns to allow the specification on a higher, more user-friendly level. We map the entries of the knowledge base to these property specification patterns and use the transformation from [10] to transform our requirements into CTL formulas.

Requirement R3 corresponds to the *absence* pattern, with *global* scope, see Table 2. The event arises if the protocol states contain more tokens than their capacity allows. The capacity is dynamically read from the knowledge base. **UDS** and **KWP2000** determine the number of tokens in the related places of the Petri Net.

The requirement R4 concerns the occurrences of a task, i.e. the *run*-States in the Petri Net. First, we query the knowledge base about the requirements of the tasks and align this list with the tasks in the test process. Then, we map each entry to a pattern according to its type, see Table 2.

For requirement R5 we collect a list of all related requirements in the test process. We use the knowledge base to get the master component $C - M$ of component C and the non-parallel relationships. For each component C a specific task close-C closes the connection to the component C. Table 2 shows the mapping of the component conditions to CTL.

7 Evaluation

For our verification we first show that we fulfill the functional requirements, see Subsection 7.1. Thus our tool is able to verify industrial test processes in a feasible time. However, this is not sufficient because the tool could only find false positives and do not fit the needs of the process developers in practice. To this

Table 2. Pattern and CTL Formula for each Requirement Type

Req.	Pattern	CTL
R3.1	Absence, Globally	$\mathsf{AG}(\neg(\mathbf{UDS} > 10))$
R3.2	Absence, Globally	$\mathsf{AG}(\neg(\mathbf{KWP2000} > 10))$
R3.3	Absence, Globally	$\mathsf{AG}(\neg(\mathbf{UDS} + \mathbf{KWP2000}) > 14))$
R4.1	B precedence A, Globally	$\mathsf{A}\ [\neg\mathbf{run\text{-}A} > 0\ \mathsf{W}\ \mathbf{run\text{-}B} > 0]$
R4.2	Absence A, Before B	$\mathsf{A}\ [(\neg\mathbf{run\text{-}A} > 0 \vee \mathsf{AG}\ (\neg\mathbf{run\text{-}B} > 0))\ \mathsf{W}\ \mathbf{run\text{-}B} > 0]$
R4.3	A response to B, Globally	$\mathsf{AG}\ (\mathbf{run\text{-}A} > 0 \rightarrow \mathsf{AF}\ (\mathbf{run\text{-}B} > 0))$
R4.4	Absence, Globally	$\mathsf{AG}\ (\neg(\mathbf{run\text{-}A} > 0 \wedge \mathbf{run\text{-}B} > 0))$
R5.1	Absence, Globally	$\mathsf{AG}\ (\neg(\mathbf{C} > 1))$
R5.2	Absence, Globally	$\mathsf{AG}\ (\neg(\mathbf{C} > 0 \wedge \mathbf{C\text{-}M} > 0))$
R5.2	Absence, Globally	$\mathsf{AG}\ (\neg(\mathbf{C} > 0 \wedge \mathbf{C\text{-}2} > 0))$
R5.3	C response to close-C, Globally	$\mathsf{AG}\ (\mathbf{C} > 0 \rightarrow \mathsf{AF}\ (\mathbf{close\text{-}C} > 0))$

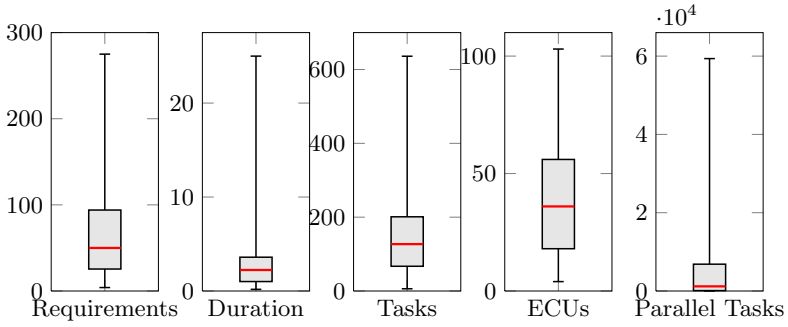

Fig. 5. Characteristics of the Evaluated Processes

end, we perform an empirical evaluation to analyze how our tool is used by the current engineers developing the diagnostic test schedules. This report gives an overview about the empirical evaluation planned, see Subsection 7.2. The results of the evaluation will be published in a coming publication.

7.1 Functional Evaluation

We have evaluated our system with 40 testing processes used by our industrial partner in practice. The processes describe the testing of three different vehicle series (AU57x, AU48x, AU64x) corresponding to the model series A4, A5, A6, A7 and A8. The processes are executed at 14 different testing stations in the factory. Figure 5 denotes the number of analyzed requirements for the processes, the combined duration of the verification, the number of tasks/ECUs and the number of parallel tasks of the 40 processes. The line shows the minimum and

maximum, the box spans from the first quartile (Q_{25}) to the third quartile (Q_{75}), the red line marks the median. Figure 5 shows that the box for the parallel tasks and the duration is getting smaller leading to a position near to bottom. This is due to the fact that the number of parallel tasks is growing exponentially with the number of tasks. The number or parallel tasks yields an increase in the overall duration. The duration for 75% of the cases is lower than 3.6s and even for the worst case is shorter than 30s on a standard PC.

The results show that the majority of the processes do not comply with the naming conventions (**R1**). Violations concerning the ECUs installed (**R1**) and connections to the ECUs (**R3**) do not occur in our processes analyzed. In total, 55 violations of task dependencies (**R4**) and component dependencies (**R5**) have been detected by verifying the 40 processes.

7.2 Empirical Evaluation

To evaluate our presented approach we developed a questionnaire for our industrial partner from the auto mobile industry. With the questionnaire we want to test three major attributes of our developed system: process quality, generality and Usability.

Process Quality: Have the system be successful in increasing the quality of the testing processes. To this end we ask for the time reduction of the development task, the number of false positive thus the number of reported violation in the process that are actual not problematic, the number of false negative thus the number of undetected rule violation in the process and we ask for the overall quality improvement of our tool.

Generality: Can the System be used in a different context within the company. For example, is the system general enough to be used in another factory possibly in a different country. We also ask if the specification form is general enough to cover all requirements.

Usability: Can the system be used in an intuitive way. Is the help of a technical person necessary for using the system?

For the usability we will use the Standard System Usability Test (SUS) [5]. SUS is a 10 item test thatis scored on a 5-point scale of strength of agreement or disagreement. The SUS has the advantage that it is technology-agnostic thus it can be used in different application domains. Due to the wide usage of the SUS-Test, a meta-test and guidelines exist to interpret the results [5].

Participants As participants in our study we will involve domain experts, i.e. employees of the test process department of our partner company.

8 Related Work

8.1 Compliance Checking

Recently, compliance checking of business processes has become an active research field [16]. Companies need to comply with standards and contractual obligations. Besides the semantic differences (usually compliance rules focus on regulation while ours capture technical aspects) our requirements of test processes can be seen similar to compliance rules. Some important research topics arise with the question of compliance and requirement checking. This research topics include the user-friendly specification of the requirement rules, the allocation of a requirement rule to a process schema, the management of the requirements and the verification of a process with such rules.

8.2 Specification

The direct specification in a temporal logic formalism like CTL is error-prone and not usable in practice for a user without experience in formal specification. To this end, different approaches have been developed to support the users in the specification task. Most business processes are modeled in a graph-based modeling language like BPMN, YAWL [2] or Petri Nets [1]. Thus many approaches base on a graphical graph-based approach for specification. [6] extends the BPMN notation with new elements that directly represent LTL operators. BPMN-Q [3] extends BPMN with new edge types that represent sequential ordering between tasks. Compliance Rule Graphs [20] allow a visual specification in a graph-based formal language. In our application field only a relatively small amount of requirement types exist. The current instances depend on the relationship between the requirement types and contextual information of the test processes, which are checked. Thus a graphic modeling is not applicable. Another approach to support the users with the specification is the use of specification patterns. [10] introduces the property patterns for specifying concurrent systems. [24] extends the pattern system to PROPEL (PROPerty ELucidation) to cover different variations of the property patterns. [9] uses a question tree to allow specifying PROPEL patterns. In our application field most of the requirements are dependent on the relationships between the test tasks and the environmental objects of test processes as well as on the typical requirement types of our application domain (which are contained in the knowledge base). The modeling of each individual requirement (even in a more user-friendly graphic notation or in a pattern based approach) would be time-consuming and impractical for most of the requirements for a test process.

8.3 Allocation and Management

[26] builds an ontology for the domain of compliance management. In contrast to our approach, our knowledge base is dedicated to vehicles, test environments for vehicles and specific compliance rules for this appliaction area. The actual

requirements used for verification are generated from the relationships of the application domain and do not focus on domain knowledge in general. Managing compliance rules includes the task of allocating the rules to the business processes. [14] clusters the compliance rules to the business processes using potential relevant activities. We dynamically generate only those requirements relevant for the test process, using the knowledge base, directly before verification.

8.4 Verification

We want to check if a business process *bp* complies with the rules given. [21] uses an approach that checks if the event log L (a set of execution traces each related to a particular *case*) complies with the temporal rules. In our case, there exist violations of rules, that are not related to an event during the process execution. Therefore, it is not part of the log and, thus, analyzing the logs is not able of detecting it. For example, it is not possible to recognize a violation of a non-parallel rule in the log of a process. Therefore, we use a model checking approach for verifying the processes. Most high-level process languages lack the direct construction of the state space required for a model-checking approach. To this end, a transformation to a formal language like Petri Nets is required. [19] gives an overview of transformations from BPMN, YAWL and WS-BPEL to Petri Nets. Our approach is similar, mainly to the approach of [12]. Beside the detection of compliance violation in existing processes (compliance by detection) it is possible to already use compliance rules during modeling of the process (compliance by design) [18]. This can be achieved by semi-automatic synthesizing approaches. [27] uses requirements in the notation of PROPOLS (an extended version of the property specification patterns [10]) to semi-automatically generate a WS-BPEL process schema. [4] uses compliance rules in the notation of LTL to synthesize process templates. Our knowledge base can be used to allow a compliance by design approach and is not limited to compliance by detection.

9 Conclusions

To verify a test process we need to specify the requirements the process must hold. Our approach consists of a knowledge base with information about the requirements and contextual information relevant for the requirements as well as of a set of types of requirements which are typical for vehicle test processes. This allows to generate concrete requirement rules tailored to a certain test process. Following these steps we have achieved, that users can specify the context of test processes in a user-friendly way filling the knowledge base. The structure of the knowledge base represents objects of the environment of test processes, i.e. components of the vehicle, test routines of the components, required and disposable resources for testing, the organisation of the physical test lines in the factory, and the relationships between these objects.

We successfully have developed a contextual knowledge base that allows to store information about the diagnostic sequences and their environment. Also,

we have developed an automatic transformation to generate on the fly formal specifications of the requirements tailored to a certain test process model from the knowledge base. Our tool developed is able of verifying the test processes of these requirements.

Moreover, we provide evidence that workflow verification methods like Petri Net model checking can be used to increase the quality of industrial testing processes. Our work clearly shows that beyond classic application domains new ones can benefit from the business process community and their research activity. Further work has been done on improving the verification step to allow applying it to real-world test processes. In future work, we will use the knowledge base as source for requirements to adapt and generate new test processes and variants.

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems and Computers 08(01), 21–66 (1998)
2. van der Aalst, W., ter Hofstede, A.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
3. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic 5240, 326–341 (2008)
4. Awad, A., Goré, R., Hou, Z., Thomson, J., Weidlich, M.: An iterative approach to synthesize business process templates from compliance rules. Information Systems 37(8), 714–736 (2012)
5. Bangor, A., Kortum, P.T., Miller, J.T.: An Empirical Evaluation of the System Usability Scale. International Journal of Human-Computer Interaction 24(6), 574–594 (2008)
6. Brambilla, M., Deutsch, A., Sui, L., Vianu, V.: The Role of Visual Tools in a Web Application Design and Verification Framework: A Visual Notation for LTL Formulae. In: Lowe, D., Gaedke, M. (eds.) Web Engineering, Lecture Notes in Computer Science, vol. 3579, pp. 557–568. Springer Berlin Heidelberg (2005)
7. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (Apr 1986)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT press (1999)
9. Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User guidance for creating precise and accessible property specifications. In: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 208–218. SIGSOFT '06/FSE-14, ACM, New York, NY, USA (2006)
10. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Second Workshop on Formal Methods in Software Practice. pp. 7–15. ACM (1998)
11. van Hee, K.M., et al.: Workflow management: models, methods, and systems. The MIT press (2004)
12. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In: Aalst, W., Benatallah, B., Casati, F., Curbera, F. (eds.) Business Process Management, Lecture Notes in Computer Science, vol. 3649, pp. 220–235. Springer Berlin Heidelberg (2005)
13. ISO, Geneva, Switzerland: Road vehicles – Open Test sequence eXchange format (OTX). ISO 13209 (2012)

14. Kabicher, S., Rinderle-Ma, S., Ly, L.T.: Activity-Oriented Clustering Techniques in Large Process and Compliance Rule Repositories (2011)
15. Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P.: On enabling data-aware compliance checking of business process models. In: Conceptual Modeling–ER 2010, pp. 332–346. Springer (2010)
16. Knuplesch, D., Reichert, M., Fdhila, W., Rinderle-Ma, S.: On Enabling Compliance of Cross-organizational Business Processes. In: Int'l Conference on Business Process Management 2013. Lecture Notes in Computer Science (LNCS), Springer, Germany (August 2013)
17. Liu, Y., Muller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal 46(2), 335–361 (2007)
18. Lohmann, N.: Compliance by design for artifact-centric business processes. Information Systems 38(4), 606–618 (2013)
19. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processes–a survey. Transactions on Petri Nets and Other Models of Concurrency II pp. 46–63 (2009)
20. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Goeser, K., Pfeifer, H., Reichert, M., Dadam, P.: SeaFlows Toolset - Compliance Verification Made Easy for Process-aware Information Systems. In: Proc. CAiSE'10 Forum - Information Systems Evolution. pp. 76–91. No. 72 in LNBIP, Springer (2010)
21. Ramezani Taghiabadi, E., Fahland, D., Dongen, B.F., Aalst, W.M.: Diagnostic Information for Compliance Checking of Temporal Compliance Requirements. In: Salinesi, C., Norrie, M., Pastor, O. (eds.) Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 7908, pp. 304–320. Springer Berlin Heidelberg (2013)
22. Schlingloff, H., Martens, A., Schmidt, K.: Modeling and model checking web services. Electronic Notes in Theoretical Computer Science 126, 3–26 (2005)
23. Schneider, T.: Specification of testing workflows for vehicles and validation of manually created testing processes (in German). Master's thesis, University of Karlsruhe (May 2012)
24. Smith, R.L., Avrunin, G.S., Clarke, L.A., Osterweil, L.J.: PROPEL: an approach supporting property elucidation. In: Proceedings of the 24th International Conference on Software Engineering. pp. 11–21. ICSE '02, ACM, New York, NY, USA (2002)
25. Stahl, C.: A Petri net semantics for BPEL, Technical Report 188. Humbold-Universität zu Berlin (2005)
26. Syed Abdullah, N., Sadiq, S., Indulska, M.: A Compliance Management Ontology: Developing Shared Understanding through Models. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 7328, pp. 429–444. Springer Berlin Heidelberg (2012)
27. Yu, J., Han, Y.B., Han, J., Jin, Y., Falcarin, P., Morisio, M.: Synthesizing service composition models on the basis of temporal business rules. Journal of computer science and technology 23(6), 885–894 (2008)
28. Zimmermann, W., Schmidgall, R.: Bussysteme in der Fahrzeugtechnik–Protokolle, Standards und Softwarearchitektur. Vieweg+ Teubner 4 (2010)