Fraunhofer
IESE

Iterative Refinement of Specification for Component Based Embedded Systems

Authors:
Muzammil Shahbaz
K.C. Shashidhar
Robert Eschbach

IESE-Report No. 030.11/E
Version 1.0
July, 2011

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
(Executive Director)
Prof. Dr.-Ing. Peter Liggesmeyer
(Scientific Director)
Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Iterative Refinement of Specification for Component Based Embedded Systems

Muzammil Shahbaz
Fraunhofer IESE
Kaiserslautern, Germany
shahbaz@iese.fhg.de

K.C. Shashidhar
Max Planck Institute SWS
Kaiserslautern, Germany
shashi@mpi-sws.org

Robert Eschbach
Fraunhofer IESE
Kaiserslautern, Germany
eschbach@iese.fhg.de

ABSTRACT

The current practice of component based engineering raises concerns in industry when the specification of proprietary components suffers from inaccuracy and incompleteness. The engineers face difficulties in producing quality systems since they lack knowledge on the interoperability of components. In order to address this issue, we present a novel framework for iterative refinement of specification for component based systems. The novelty is the use of a preliminary behavioral model as a source for triggering refinement iterations. Moreover, it exploits rigorous formal techniques to achieve high-level system validation as an integral part of the refinement procedure. The framework has been evaluated on an automotive system in which the embedded software control units were developed by third-party vendors. The final results produced an improved formal system specification that identified several behaviors that were previously unknown.

Keywords

Component Based Systems, Specification Refinement, Interoperability, Reverse Engineering, System Validation

1. INTRODUCTION

Component based engineering enables a systematic and cost-effective reuse of prefabricated parts – a characteristic of mature engineering disciplines. Most embedded system domains, especially the automotive industry, are relying on component based approach to meet time-to-market and economical constraints. Among the several challenges that are faced in its adoption, an important one arises from the nature of component specifications available in practice. Ideally, they should reflect their precise behaviors. However, practice shows that specifications are never complete and accurate [7]. Also, they are often available in textual format, whose informal semantics lead to misinterpretations in different engineering environments. Often, engineers have to use intuition and/or have to carry out extensive simulations to identify additional information about the components.

1.1 Motivation from the Automotive Industry

Electronic Control Unit (ECU), a fundamental electronic building block of a modern automobile, used to be a relatively simple, hardware oriented system. Today, it is a multi-purpose computer system where functionality is often delivered in software than in hardware. The complexity of functions ranges from ergonomics, like power seat and power windows, to active-safety features, like drive/brake-by-wire, blind-spot detection, stability and emission control systems, that are included as a standard functional package in the new generation of automobiles. At the same time, the advent of variety of functions and needs for future automobiles pose challenges to automotive OEMs for integrating components which are designed and developed by third-party vendors. Most of the ECUs are currently used as a black-box solution, resulting in ECUs of different complexities and capabilities integrated in a single vehicle.

The lack of precise specifications in the automotive industry poses challenges to achieving interoperability and conformity of different components in an integrated system. At the hardware level, standard communication protocols[1] e.g., CAN, TTCAN, LIN, FlexRay, have been well-adopted by the automotive industry. On the contrary, there has been a long term effort for adopting a common architecture at the software level. OSEK/VDX was the first initiative launched by the French-German consortium of automotive industries towards building an open standard. Several other initiatives, like AEE and EAST-EEA at European level, Forst-Automotive and HIS in Germany have been a continuing effort for building common software platforms. Now a novel and well-versed standard in automotive community, called AUTOSAR, is said to be the *lingua franca* for the development of ECUs between automotive OEMs and suppliers as well as their use within different vehicle types.

Despite these efforts, it is still very hard to synchronize the application development among different suppliers having different maturity levels. The common platform for developing vehicle functions, such as AUTOSAR, provides means for cooperation among various OEMs and suppliers on interface standards. But at the same time, the flexible software architecture and challenges of new vehicle functions provides room for competition in implementation among suppliers, thus giving rise to varying qualities and interoperability choices. The lack of synchronization also stems from the fact that the specification of functions that is available in textual format does not provide details of the specific design

[1]For various technologies related to automotive electronics mentioned in this section, refer to [13] for details.

decisions and implementations. For example, the timing behavior of modules in AUTOSAR is not specified at a level detailed enough to ensure that a specific module from one vendor could expect an output from another module (supplied by another vendor) in a time frame matching for both modules. Similarly, a large group of functionalities, interfaces and configurations specified as basic modules is not mandatory. The suppliers are free to implement any combination[2] of functionalities and configurations. The consequence is that many suppliers offer different sets of similarly interacting functions that further results in mismatching behaviors and challenges for integration [7].

1.2 Our Approach and Contribution

In order to address the motivated issue, we present an approach for systematic refinement of the specification of a system that is composed of various black-box components. At the same time, the approach helps in identifying potential compositional and nonconformance problems in the system through the application of rigorous formal methods. The approach has been evaluated on automotive embedded systems, but it can also be applied to other domains where component based systems are prevalent.

Our approach exploits the application of inference techniques [5] to iteratively refine the specification of an integrated system of black-box components by learning state machine models through observing system executions. In general, inference techniques require counterexamples to successively improve the conjectured models [23, 5]. Therefore, the traditional approaches using inference techniques employ different testing strategies to generate extensive tests in search of counterexamples (cf. [22]). In contrast to the traditional approach, our approach does not take into account any testing strategy to trigger refinements; instead it relies on a preliminary behavioral model that can be an effective source for finding counterexamples. Such a behavioral model can be often derived from the available system artifacts viz., textual specification, interface descriptions, domain knowledge, etc. Note that this model is only as good as the quality of the artifacts, and therefore, often inaccurate and incomplete. But at the same time, it provides valuable insights into how the components should interact in the integrated system, which may be hard to deduce solely from inference techniques. Exploiting this benefit, we can compute discrepancies between the inferred model and the behavioral model through a systematic state space exploration. Such discrepancies (or counterexamples), as we will show, can be used to iteratively obtain a *refined specification*.

A global view of our approach is shown in Figure 1. It consists of two procedures, namely, *inference procedure* and *conformance checking procedure*, that are unified under one framework. The inference procedure infers the formal models of the black-box components individually by employing an existing inference algorithm [18]. It tests each component interactively and deduces its formal model, based upon the observations from testing. The inferred models are then combined as an asynchronous product that potentially represents the system's global behavior. The product is analyzed for any compositional problems, e.g., live-locks, that may be present in the system. If the problem is identified on the system, the procedure terminates by reporting

[2]There are about 50 basic software modules and 3344 configuration parameters in AUTOSAR specification Release 4.0.

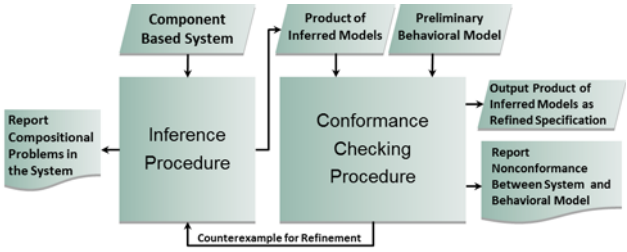

Figure 1: Our Approach to Specification Refinement

the problem in the integrated system. If there is no compositional problem, the product is fed to the conformance checking procedure. A preliminary behavioral model, built independently, is provided as another input to the procedure. The procedure performs reachability analysis of the product of the inferred models and the behavioral model. This analysis computes discrepancies between the two models, if any. If a discrepancy identifies a counterexample for the product, the product is refined by iterating the inference procedure using the counterexample. Since the behavioral model is incomplete, and may also be inaccurate, it is possible that a discrepancy also identifies nonconformance between the behavioral model and the system. In this case, the procedure terminates by reporting the nonconformance. When no discrepancy is detected, the procedure terminates by outputting the product of the inferred models as a refined specification.

The novelty of our approach is in using a preliminary behavioral model that provides means for: (1) inference techniques to compute discrepancies between the inferred models and the system, such that the discrepancies can lead to the refinement of the inferred models; and (2) high-level system validation as an integral part of the specification refinement procedure. Additionally, it is important to note that the approach does not apply inference techniques directly to the full system, but to its individual components. The inferred models for the components are composed into a single product that potentially represents the global system. In this way, it systematically explores the component interactions within the system and thereby provides additional insights into the behaviors of the integrated system.

Outline of the paper. Section 2 discusses existing approaches for specification refinement. Section 3 provides the formal settings of our approach. Section 4 describes our specification refinement framework and its illustration on an example. Section 5 gives a full account on its evaluation. Section 6 concludes the paper with some perspectives.

2. RELATED WORK

The problem of informal and partial specification for the purpose of understanding system behaviors and its validation has been addressed extensively, both in academia and industry. There is a wealth of literature on this topic but we focus on a particular approach called *specification mining* [1] that attempts to obtain formal specification directly from the system artifacts (e.g., source code, interfaces, execution traces etc.). When the system is a black-box, it largely relies on the execution traces as the available artifact. In this context, specification mining is akin to the *grammatical in-*

ference [5] approach that aims at inferring a grammar or an automaton model for an unknown language, given +ve and -ve traces of the language. The common issue, though, is that the accurate models can only be inferred with a sufficiently broad set of traces. Theoretically, the number of traces required to infer an accurate model is unknown in the context of a black-box system [23]. In practice, the techniques using this approach require many refinement iterations in which further traces are obtained to help eliminating inaccuracies in the inferred models. Most approaches in specification mining suffer from this major weakness and do not provide explicit mechanisms for triggering refinement iterations. To overcome this weakness, researchers have used different testing strategies that are designed to produce tests that, when applied, distinguish the inferred models from the black-box system. Eventually, the tests can be included in the set of existing traces for inferring the model. Shu and Lee [19] and Walkinshaw et al. [22] performed random walks to generate tests from the inferred model and applied tests to the system. In their more recent work, Walkinshaw et al. [21] used coverage based criteria for generating tests from the inferred models to distinguish it from the unknown specification of the Linux TCP/IP stack. In another work, Dallmeier et al. [4] proposed to generate tests by mutating an existing test-suite in order to find additional behaviors from the ones inferred initially. Then each mutant is applied to the system in isolation and new behavioral models are inferred from the execution of the modified test-suite. Finally, the initial model and the new models are merged. In all such testing strategies, a main problem is that they rely on heuristics that may not be effective in certain cases. Additionally, generating and running a sufficiently large test suite is often infeasible in practice, especially when tests are expensive to execute (like in the automotive domain).

In contrast to relying on given execution traces, a different inference approach, called *active inference* [5], learns the model by observing the system executions dynamically. In this approach, tests of arbitrary lengths are generated from the input alphabet of the system. Then, these tests are applied on the system and the corresponding execution traces are collected. In view of those traces, more tests are generated such that more behaviors could be covered that have not been observed by the initial tests. In this way, the hidden state space is explored systematically until tests no more observe new behaviors. The inference approach has found successful application in testing and verification (see for e.g., [14, 2, 20, 16]). The main reason for the success is having the advantage of testing the system dynamically based upon the observations that provide a good clue of the execution paths for further exploration in search of new behaviors. However, this approach eventually faces the same dilemma concerning how many tests can guarantee the exploration of the complete state space [2]. Peled et al. [14] solved this problem by assuming a prior knowledge on the upper bound on the state space size. They showed that by generating tests of increasing lengths up to the known size, the complete state space can be explored with an exponential cost in the limit. This solution, however, is not realistic in a real world scenario where it is hard to assume the upper bound on the number of hidden states. In the same spirit, Groz et al. [9] proposed a k-step lookahead algorithm to probe additional states. The advantage is the flexibility in probing states with k-length tests, where k can be selected according to the given resources. However, one must enumerate each state of the inferred model to perform lookahead. So unless the state space is small, it becomes often intractable to enumerate the full state space.

Despite the various pros and cons, most of the mentioned work apply specification refinement approaches either at a component-level, or at a system-level, where the system is taken as one unit. To the best of our knowledge, there is no related work that considers specification refinement of a system of black-box components wherein each component is inferred individually and then combined to obtain a specification of the system that captures the interactions between the components. For compositional verification, Păsăreanu et al. [16] have used active inference to synthesize assumptions about each component's environment. The inferred assumptions are refined by model checking the premises of different assume-guarantee rules. Nevertheless, their work is restricted to the interface level of the system and they do not consider components as black-boxes.

3. FORMAL SETTINGS

The formal settings of the approach are described in this section. Most of the definitions are standard and recalled from the relevant literature, e.g., [11, 15].

3.1 Finite State Machine

A Finite State Machine (FSM) $A = (S, s_0, I, O, D, \delta, \lambda)$, where S is a finite set of states with the initial state s_0, I and O are finite sets of inputs and outputs respectively, $D \subseteq S \times I$ is a specification domain, $\delta : D \rightarrow S$ is a transition function, and $\lambda : D \rightarrow O$ is an output function.

An FSM A is said to be *complete* if $D = S \times I$. We will omit the specification domain D in the case of complete machines. If D is a proper subset of $S \times I$, then A is called *partial*. Given a sequence $\alpha = x_1, \ldots, x_k$ of the set I^* of all possible finite input sequences, α is said to be a *defined* input sequence at state $s \in S$ if there exists states $s_1, \ldots, s_k, s_{k+1}$, where $s_1 = s$, such that $(s_i, x_i) \in D$ and $\delta(s_i, x_i) = s_{i+1}$ for all $i = 1, \ldots, k$. In words, α is a defined input sequence at state s if the behavior of A for the sequence α is defined. We use $\Omega(s)$ to denote the set of all defined input sequences for state s. We note that, for a complete machine, $\Omega(s) = I^*$ for any state $s \in S$, while a partial FSM can have states where the set of defined sequences is empty.

We extend the transition and output functions from input symbols to defined input sequences, including the empty sequence ϵ, as usual. For simplicity, we use the same notations δ and λ for the extended functions. Let $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$ for any $s \in S$. Suppose that β is a defined input sequence at state s and $\delta(s, \beta) = s'$. Then, for any $x \in I$ such that $(s', x) \in D$, we define $\delta(s, \beta x) = \delta(s', x)$ and $\lambda(s, \beta x) = \lambda(s, \beta)\lambda(s', x)$.

Given states $s, t \in S$, s and t are *equivalent*, if $\Omega(s) = \Omega(t)$ and $\lambda(s, \alpha) = \lambda(t, \alpha)$ for all $\alpha \in \Omega(t)$. States s and t are *distinguishable* if there exists an input sequence $\alpha \in \Omega(s) \cap \Omega(t)$ such that $\lambda(s, \alpha) \neq \lambda(t, \alpha)$. An FSM is said to be *reduced* if, for every pair of states $s, t \in S$, s and t are distinguishable. As opposed to complete FSMs, partial FSMs may have several distinguishable reduced forms (see, e.g., [8], for details). Figures 2 and 3 are examples of complete reduced FSMs.

The notions of equivalence and distinguishability could also be extended from states to machines. Let $A = (S, s_0, \ldots)$ and $B = (T, t_0, \ldots)$ be two FSMs, such that $S \cap T = \emptyset$.

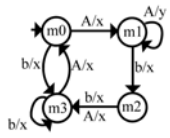

Figure 2: FSM M

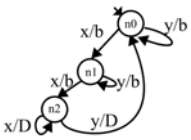

Figure 3: FSM N

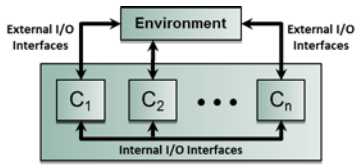

Figure 4: Architecture of the FSM system

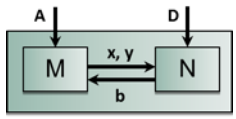

Figure 5: Example of an FSM System

Then we say that A and B are equivalent, if s_0 and t_0 are equivalent; A and B are distinguishable if s_0 and t_0 are distinguishable.

3.2 System Configuration

The architecture of the system composed of FSMs (or simply, *FSM system*) is described as follows. An FSM system is composed of n components, i.e., $\{C_1, \ldots, C_n\}$, where for each $i \neq j, 1 \leq i, j \leq n$, $C_i = (S_i, s_{0i}, I_i, O_i, \delta_i, \lambda_i)$ and $C_j = (S_j, s_{0j}, I_j, O_j, \delta_j, \lambda_j)$ are complete FSMs, such that $I_i \cap I_j = \emptyset$ and $O_i \cap O_j = \emptyset$. The components C_i and C_j communicate if $I_i \cap O_j \neq \emptyset$ or $O_i \cap I_j \neq \emptyset$. Let I be the union of all inputs, i.e., $I = \bigcup_{i=1}^{n} I_i$, and O be the union of all outputs, i.e., $O = \bigcup_{i=1}^{n} O_i$. The set $I_{ext} = I \backslash O$ contains the external inputs of the system that can be given from the environment and $O_{ext} = O \backslash I$ contains the external outputs of the system that can be given to the environment. We consider a system with at least one external input and one external output. For each component C_i it holds that if $a \in I_i$ then either $a \in I_{ext}$ or $a \in O_j$, and if $a \in O_i$ then $a \in O_{ext}$ or $a \in I_j$ of some C_j. Figure 4 shows the architecture of the FSM system.

The system executes in a run-to-completion fashion and it receives inputs only when it is in a stable mode. The system is in stable mode when the states of all its components are stable. A state is *stable* when none of its outward transitions is enabled; otherwise it is said to be *unstable*. Initially, the system is in stable mode and can be stimulated by providing an external input from I_{ext}. A component receives the input and produces either an external output to the environment or an internal input to another component. The other component receives the internal input and produces either an external output to the environment or an internal input to another component. Finally, the system produces an external output to the environment and stays in the stable mode until it receives another external input. Note that the components have disjoint sets of inputs, therefore only one component can be stimulated by any input in/to the system. Owing to space limitations, we omit a discussion on how such a configuration would suffice in the considered domain. An example of the FSM system composed of components M and N is given in Figure 5. The external inputs (outputs) are labeled in uppercase letters and the internal inputs (outputs) conversely.

3.3 Product of FSMs

Given n FSM components $C_1, \ldots, C_n$, their product, denoted by $\mathcal{P}$, is an FSM $(S, s_0, I, O, D, \delta, \lambda)$, where $I = \bigcup_{i=1}^{n} I_i$ and $O = \bigcup_{i=1}^{n} O_i$ are finite sets of all inputs and outputs respectively. The set of states is defined as $S \subseteq S_1 \times \cdots \times S_n \times (I \cup \{\epsilon\})$. Since the system executes in a run-to-completion fashion, a state in S receives only one input at a time. The symbol ϵ in the state represents no input, i.e., the system is in stable mode and the state is a stable state of the system waiting for an external input. The state $s_0 = (s_{01}, \ldots, s_{0n}, \epsilon) \in S$ is the initial state and $D \subseteq S \times I$ is the specification domain. The functions δ and λ are obtained by applying the following rules. For $1 \leq i \leq n$, let $C_i = (S_i, s_{0i}, I_i, O_i, \delta_i, \lambda_i)$, then for $s_i \in S_i$ and $a \in I$

1. If $(s_1, \ldots, s_n, \epsilon) \in S$ and $a \in I_{ext} \cap I_i$ then $\lambda((s_1, \ldots, s_n, \epsilon), a) = \lambda_i(s_i, a)$ and
   - If $\lambda_i(s_i, a) \in O_{ext}$ then $\delta((s_1, \ldots, s_n, \epsilon), a) = (s_1, \ldots, \delta_i(s_i, a), \ldots, s_n, \epsilon)$
   - If $\lambda_i(s_i, a) \notin O_{ext}$ then $\delta((s_1, \ldots, s_n, \epsilon), a) = (s_1, \ldots, \delta_i(s_i, a), \ldots, s_n, \lambda_i(s_i, a))$
2. If $(s_1, \ldots, s_n, a) \in S$ and $a \in I_i \backslash I_{ext}$ then $\lambda((s_1, \ldots, s_n, a), a) = \lambda_i(s_i, a)$ and
   - If $\lambda_i(s_i, a) \in O_{ext}$ then $\delta((s_1, \ldots, s_n, a), a) = (s_1, \ldots, \delta_i(s_i, a), \ldots s_n, \epsilon)$
   - If $\lambda_i(s_i, a) \notin O_{ext}$ then $\delta((s_1, \ldots, s_n, a), a) = (s_1, \ldots, \delta_i(s_i, a), \ldots, s_n, \lambda_i(s_i, a))$

By construction, all the stable states of the product have a transition for each external input, i.e., for all $(s_1, \ldots, s_n, \epsilon) \in S$ and $a \in I_{ext}$, $((s_1, \ldots, s_n, \epsilon), a) \in D$.

The product has a *compositional loop* (or simply, *loop*) if there exists a sequence of internal inputs $\alpha \in (I \backslash I_{ext})^*$, such that $\delta(s, \alpha) = s$, where $s \in S$ is an unstable state. In words, the product has a loop if there is a cycle of internal inputs and outputs over unstable states with no stable state in between. Let $\alpha' \in I^*$ such that $\delta(s_0, \alpha') = s$, then $\alpha'\alpha$ is a *witness* to the loop in the product. The product is called *loop-free* if it contains no loops.

3.4 Distinguishing Machine

Let $\mathcal{B}$ and $\mathcal{P}$ be two FSMs. To check if $\mathcal{B}$ and $\mathcal{P}$ are distinguishable, a designated FSM, called a *distinguishing machine* has been used by Petrenko and Yevtushenko [15], whose states are pairs of states of $\mathcal{B}$ and $\mathcal{P}$, its initial state is the pair of initial states of the two machines and the remaining states are determined by performing reachability analysis. A designated output *fail* is used to signal when the two machines do not agree on a common input.

Formally, given FSM $\mathcal{B} = (S, s_0, I, O, D_{\mathcal{B}}, \delta_{\mathcal{B}}, \lambda_{\mathcal{B}})$ and FSM $\mathcal{P} = (T, t_0, I', O', D_{\mathcal{P}}, \delta_{\mathcal{P}}, \lambda_{\mathcal{P}})$ such that $\Omega(s_0) \subseteq \Omega(t_0)$, the FSM $DM = (Q, q_0, I, O \cup \{fail\}, D, \Delta, \Lambda)$ is the *distinguishing machine* of $\mathcal{B}$ and $\mathcal{P}$, where $Q \subseteq \{S \times T\}$, $q_0 = (s_0, t_0) \in Q$, $fail \notin O \cup O'$, $D = \{((s,t), x) | (s,t) \in Q, (s,x) \in D_{\mathcal{B}}\}$ and the functions Δ and Λ are obtained as follows:

- $\Delta((s,t), x) = (\delta_{\mathcal{B}}(s,x), \delta_{\mathcal{P}}(t,x))$, if $(s,x) \in D_{\mathcal{B}}$

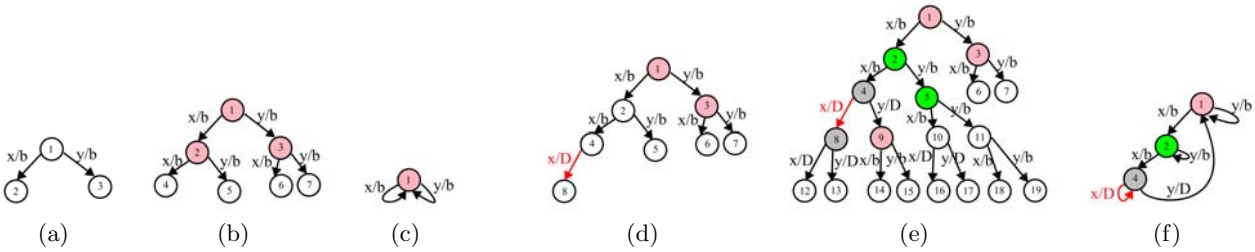

Figure 6: Learning FSM N

- $\Lambda((s,t),x) = \begin{cases} \lambda_{\mathcal{B}}(s,x) & \text{if } (s,x) \in D_{\mathcal{B}} \text{ and} \\ & \lambda_{\mathcal{B}}(s,x) = \lambda_{\mathcal{P}}(t,x) \\ fail & \text{if } (s,x) \in D_{\mathcal{B}} \text{ and} \\ & \lambda_{\mathcal{B}}(s,x) \neq \lambda_{\mathcal{P}}(t,x) \end{cases}$

The state set Q is the smallest set obtained by the application of the above rules. If $\mathcal{B}$ has n states and $\mathcal{P}$ has m states, the size of Q is bounded by nm. If $\mathcal{B}$ and $\mathcal{P}$ are distinguishable, then any $\alpha \in I^*$ that, applied at the initial state of DM, covers a transition with the output *fail*, distinguishes $\mathcal{B}$ and $\mathcal{P}$. Such a sequence is called a *fail-trace*.

3.5 Model Inference

We use the active inference algorithm [18] in our specification refinement approach that learns an FSM from a black box component via testing. To save space, we describe the algorithm with an illustration succinctly, while referring to the original paper [18] for the theoretical foundations and details on the algorithm complexity.

Assume that the input alphabet of the component is known. The algorithm calculates a test-suite from the alphabet and applies tests to the component. It observes and records the component behaviors in response to those tests. Based upon the observations, it calculates a new test-suite and applies the tests to the component. It iterates in this fashion and records observations at each iteration until some conditions on the overall observations are satisfied. Finally, it calculates a complete reduced FSM that is consistent with the collected observations. Here, a demonstration of the algorithm is given with the help of a simple example.

Let the machine shown in Figure 3 be a black box component defined over input alphabet $\{x, y\}$. The algorithm starts by applying all the inputs in the set on the initial state of the component. The component responds with b and b, on the inputs x and y, respectively. These observations are recorded in some data structure. Here, we use the tree structure and borrow the definitions for nodes and edges, as defined for states and transitions, respectively (Section 3.1), for simplicity. Figure 6(a) shows the tree after recording the observations from the initial tests. This tree is further extended by calculating tests from each leaf node of the tree. In this example, the tests xx, xy, yx and yy are applied to the component and the tree is extended with new observations as shown in Figure 6(b). After each extension, the algorithm checks whether the extended node is *equivalent* to some predecessor node in the tree. Here, nodes 2 and 3 are equivalent to node 1 in Figure 6(b). At this point, the tree is not extended any further, assuming that all future behaviors from nodes 2 and 3 would be same as of node 1. The algorithm merges the equivalent nodes in the tree, i.e., 1, 2 and 3, which shapes the FSM as shown in Figure 6(c).

As is clear, the FSM in Figure 6(c) is not a correct conjecture for the original machine shown in Figure 3. Suppose a counterexample xxx is provided that distinguishes the two machines when applied to their initial states. The algorithm then processes the counterexample to produce a refined model such that it cannot be distinguished anymore with the same counterexample. In order to process the counterexample, the algorithm takes the last observation tree (i.e., Figure 6(b)) and extends it with the counterexample and the corresponding behavior, as shown in Figure 6(d). Node 2 is now no longer equivalent to node 1 since the sequence xx produces different behaviors on the two nodes. The algorithm extends the tree from node 2 in order to check for the equivalence. The extension is shown in Figure 6(e) that depicts the fan out from nodes 4 and 5. It can be seen that node 5 is equivalent to node 2, node 8 to node 4 and node 9 to node 1. Finally, the algorithm merges the equivalent nodes in the tree, i.e., $1, 3, 9$; $2, 5$ and $4, 8$, which shapes the FSM as shown in Figure 6(f).

3.6 Behavioral Model

A salient feature of our approach lies in the use of a preliminary behavioral model, that is incomplete, and may even be inaccurate. Verification often relies on the existence of a model of a system that captures the behavior of the system to be verified. Such a model is derived by abstracting away the details of the system from its implementation that are not relevant for the property to be verified (for example, cf. [6]). However, in our context of black-box components, the available artefacts are specification and design documents, interface descriptions etc. To suit this, we derive a preliminary behavioral model by formalizing the information available in these artefacts. For example, a technique like *Sequence Based Specification (SBS)* [3] can be used here. It proposes a set of techniques for stepwise construction of traceably-correct black-box specification that is based on a sequence enumeration procedure and basic requirements analysis skills. The end result can be easily converted into a formal representation. In our case, the formal representation is a (partial) FSM model of the available specification of a given automotive system.

4. ITERATIVE REFINEMENT

4.1 Framework

The framework for iterative refinement of specification for an integrated system of black-box components is depicted in Figure 7. The framework consists of two procedures, namely *inference procedure* and *conformance checking procedure*.

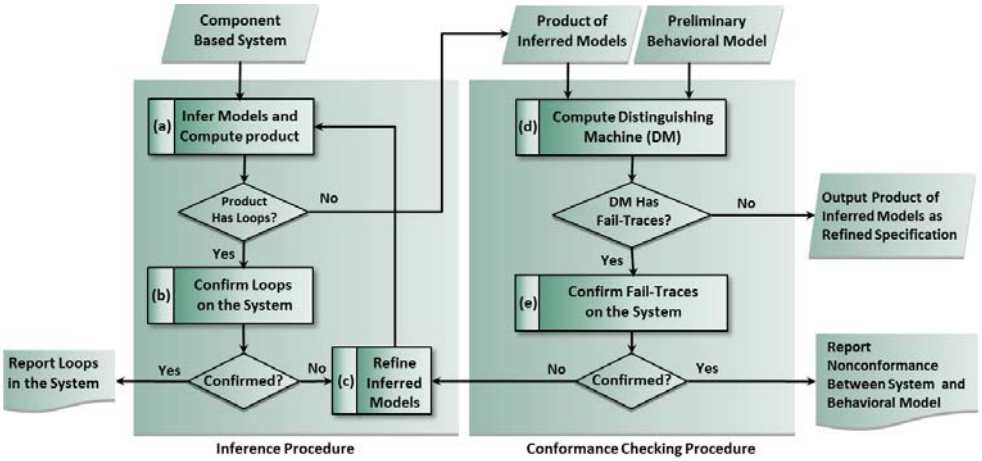

Figure 7: Framework of Iterative Refinement of Specification

4.1.1 Inference Procedure

The inference procedure infers models of the components individually using the inference algorithm illustrated in Section 3.5, and then computes the product of the inferred models as described in Section 3.3, in Step (a).

If the product has a compositional loop, the witness to the loop has to be confirmed on the real system, in Step (b). A compositional loop can be confirmed on the real system by executing the witness on the system and matching its output behavior with that of the product. Imperatively, since the components in the systems are black-boxes, it is not known how many times a cycle should be observed to declare a loop. In our framework, we declare the loop in the system, if the cycle is observed twice. Thus, the loop (witnessed by $\alpha'\alpha$) is confirmed, if the product and the system produce the same output behavior, when simulating $\alpha'\alpha\alpha$ starting at their initial states.

If the loop is confirmed, the procedure terminates by reporting the loop in the system. Otherwise, the loop is spurious, which is due to the inaccuracy of the inferred models. In this case, the witness to the loop acts as a counterexample to refine models by the inference algorithm, in Step (c). The inference procedure then iterates with the refined models and terminates when a loop-free product is obtained.

4.1.2 Conformance Checking Procedure

The conformance checking procedure starts when the loop-free product is obtained by the inference procedure. A preliminary behavioral model, that is built independently, is provided as another input to the procedure. The procedure computes the distinguishing machine of the product and the behavioral model as described in Section 3.4, in Step (d).

Any fail-trace computed by the distinguishing machine implies either a counterexample for the product or a nonconformance between the behavioral model and the system. In order to confirm this, the fail-trace has to be simulated on the system, in Step (e). A fail-trace α is confirmed if the product and the system produce the same output behavior, when simulating α starting at their initial states.

If a fail-trace is not confirmed, then it is spurious, which is due to the inaccuracy of the inferred models. In this case, the fail-trace acts as a counterexample, using which the models are refined by the inference algorithm in Step (c). The inference procedure thus iterates to provide a new product of the refined models.

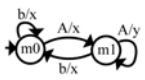

Figure 8: Model $M^{(1)}$

Figure 9: Model $N^{(1)}$

On the contrary, if a fail-trace is confirmed, it implies that the fail-trace does not distinguish the product and the system, but the behavioral model and the system. In this case, the fail-trace points to a nonconformance between the behavioral model and the system. The procedure then terminates by reporting the nonconformance.

When there are no fail-traces in the distinguishing machine, the procedure terminates by outputting the product of the inferred models as a refined specification.

4.2 Illustration

The approach is illustrated on the system given in Figure 5, that is composed of components M (Figure 2) and N (Figure 3). The illustration shows how specification refinement and system validation are simultaneously achieved in an iterative fashion using a preliminary behavioral model $\mathcal{B}$ (Figure 12).

At first, the inference procedure infers the models of components M and N individually as shown in Figure 8 and 9[3]. Then it computes the product $\mathcal{P}^{(1)}$ of the inferred models as shown in Figure 10. $\mathcal{P}^{(1)}$ has a loop that is witnessed by the sequence $Axbxb$. In order to confirm the loop on the system, the sequence is extended as $Axbxbxb$ and simulated on the system. The output behavior of the system on this sequence is $xbxbxD$, whereas the output behavior of $\mathcal{P}^{(1)}$ is $xbxbxbx$. Therefore, the loop is not confirmed and $Axbxbxb$ is a counterexample for the inferred models.

In order to find the model that is to be refined, the counterexample is projected to the input alphabet of each component. This is achieved by eliminating all symbols that do not belong to the input alphabet of the particular com-

[3]Note that the inference of N is shown in Section 3.5.

Figure 10: Product $\mathcal{P}^{(1)}$ of $M^{(1)}$ and $N^{(1)}$. The states in bold are the stable states in the product.

ponent. Each projection is then applied to its respective component in isolation. If the behavior of the component is different from its inferred model, then the model is to be refined. In this example, the projection of $Axbxbxb$ to the input alphabet of M and N gives the sequences $Abbb$ and xxx respectively. It is found that the component M responds same as its model $M^{(1)}$ on the sequence Abb. However, the component N responds as bbD and its model $N^{(1)}$ responds as bbb on the sequence xxx. This implies xxx is a counterexample for $N^{(1)}$.

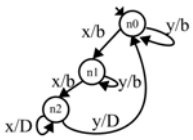

Figure 11: Model $N^{(2)}$

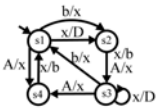

Figure 12: Preliminary Behavioral Model $\mathcal{B}$

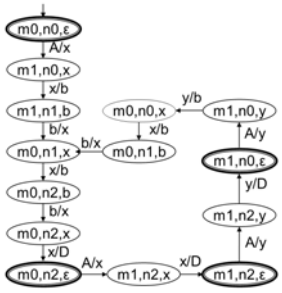

Figure 13: Product $\mathcal{P}^{(2)}$ of $M^{(1)}$ and $N^{(2)}$

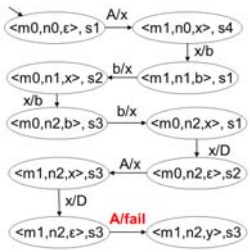

Figure 14: Distinguishing Machine $DM^{(1)}$ of $\mathcal{B}$ and $\mathcal{P}^{(2)}$

$N^{(1)}$ is refined after processing the counterexample xxx through the inference algorithm as $N^{(2)}$ (Figure 11)[4]. Now the product $\mathcal{P}^{(2)}$ of $M^{(1)}$ and $N^{(2)}$ is computed, which is shown in Figure 13. $\mathcal{P}^{(2)}$ is a loop-free product, hence the inference procedure terminates by providing the product to the conformance checking procedure.

Let the partial FSM in Figure 12 be the preliminary behavioral model $\mathcal{B}$. Then the distinguishing machine $DM^{(1)}$ of $\mathcal{B}$ and $\mathcal{P}^{(2)}$ is computed as shown in Figure 14. The machine has a fail-trace, i.e., $AxbxbxAxA$, which is to be confirmed on the system. The output behavior of the system on the fail-trace is $xbxbxDxDx$, whereas the output behavior of $\mathcal{P}^{(2)}$ is $xbxbxDxDy$. Therefore, the fail-trace is not confirmed and $AxbxbxAxA$ is a counterexample for the inferred models.

In order to find the model that is to be refined, the fail-trace is projected to the input alphabet of each component.

[4]Note that the refinement of $N^{(1)}$ is shown in Section 3.5.

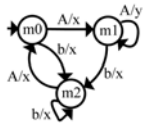

Figure 15: Model $M^{(2)}$

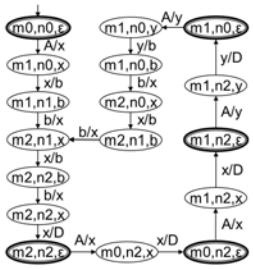

Figure 16: Product $\mathcal{P}^{(3)}$ of $M^{(2)}$ and $N^{(2)}$

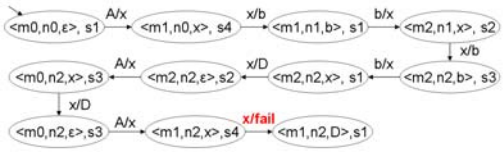

Figure 17: $DM^{(2)}$ of $\mathcal{B}$ and $\mathcal{P}^{(3)}$

Therefore, the projection of $AxbxbxAxA$ on the input alphabet of M and N gives input sequences $AbbAA$ and $xxxx$ respectively. Each projected sequence is applied to its respective component. It is found that the component N responds same as its model $N^{(2)}$ on the sequence $xxxx$. However, the component M responds as $xxxxx$ and its model $M^{(1)}$ as $xxxxy$ on the sequence $AbbAA$. This means $AbbAA$ is a counterexample for $M^{(1)}$.

$M^{(1)}$ is refined after processing the counterexample $AbbAA$ through the inference algorithm as $M^{(2)}$ (Figure 15). Now the loop-free product $\mathcal{P}^{(3)}$ of $M^{(2)}$ and $N^{(2)}$ is computed, which is shown in Figure 16.

This follows computing the distinguishing machine $DM^{(2)}$ of $\mathcal{B}$ and $\mathcal{P}^{(3)}$ as in Figure 17. The machine has a fail-trace, i.e, $AxbxbxAxAx$, to be confirmed on the system. The output behavior of the system on the fail-trace is $xbxbxDxDxD$, which is same as the output behavior of $\mathcal{P}^{(3)}$. Therefore, the fail-trace is not confirmed. The procedure terminates by reporting the nonconformance between the system and the behavioral model that is identified by $AxbxbxAxAx$. $\mathcal{P}^{(3)}$ is the refined specification obtained in this example.

5. EVALUATION

The approach of iterative refinement of specification has been evaluated on an automotive system, precisely, an embedded door control system. The system is located inside the physical car doors and it controls functions related to mirrors, windows and lockings. The textual specification [10] of a driver's side door was provided by our industrial partner with regard to the general functionalities, but without representing the real existing system. The specification consisted of 60+ pages, detailing the system characteristics and diagrams of its physical interfaces. The specification of the internal ECUs was by no means complete and the detailed description of all the interactions between ECUs was missing. Only general behaviors of the ECUs and some input/output signals were specified. The ECUs were black-boxes that were collected from external sources and integrated by our indus-

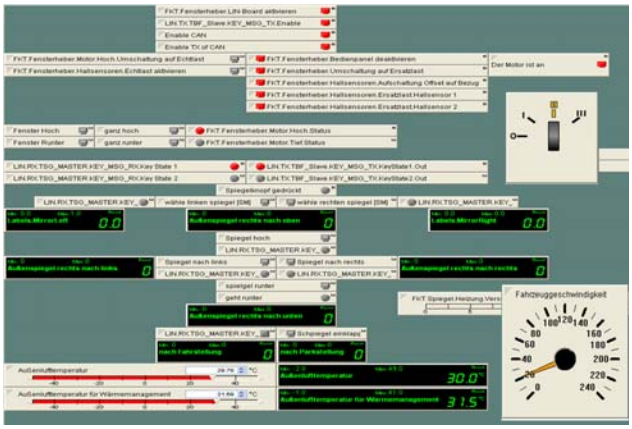

Figure 18: Snapshot of PROVEtech:TA GUI

trial partner in the final door control system.

5.1 Identification of System Boundaries

The first step in the evaluation study was to identify system boundaries in order to execute inference and conformance checking procedures. There were multiple interfaces that could be taken into account to estimate the boundaries. They were CAN/LIN bus inputs (messages from the CAN/LIN into the ECUs), CAN/LIN bus outputs (messages put by the ECUs onto the CAN/LIN), actuators (output signals sent as a result of ECU behaviors), user inputs (messages from the human operators to the ECUs) and sensors (messages from the devices such as temperature sensors to the ECUs). In our case, some information about the interfaces and the concrete signals was given but scattered all over the specification document. Moreover, only those signals were given that were necessary to explain interfacing and/or describing general behaviors of the ECUs.

This partial information was insufficient to extract a broad spectrum of behaviors during the inference procedure. Therefore, other sources were sought out such as domain expertise and knowledge of the testing tools that are commonly used in the automotive domain. For our industrial partner, PROVEtech:TA [12] is a well-known test runner that automates the test execution process for ECUs. The tool is also equipped with a GUI that provides monitors for observing sensors and actuators of ECUs graphically (see Figure 18 for a snapshot). It also provides a common vocabulary of signal names for CAN/LIN buses so that the tester can observe the signal values while ECUs are active. Using these monitors and signals, the ECUs were simulated on some basic functions relative to each ECU one by one. A script was written in the tool to capture data for signals that were stimulated when a sensor or an actuator became active. Finally, those signals became a part of our larger input set. To avoid exhaustiveness, only relevant values were considered, and furthermore, continuous stimuli (e.g., time, temperature) were abstracted into discrete values.

5.2 System Configuration

The door control system was composed of mainly three ECUs: *Mirror Control Unit (MCU)*, *power Window Control Unit (WCU)* and *Locking System (LS)*. The ECUs were embedded with additional subsystems such as telemetric and temperature sensors. The three ECUs were real systems but the subsystems were simulated by our execution environment. According to the available specification, a simplified view of the ECU behaviors can be summarized as follows:

- *MCU* controls mirror movement around its horizontal and vertical axis and also folding and expansion of the mirror frame with specialized motors. It is also equipped with a heater that responds to the outside temperature and turns on/off heating automatically.
- *WCU* controls the window's opening/closing movement with the help of specialized motors. It is equipped with position sensors to stop motor rotating when the window is opened/closed completely. An obstacle detector is also embedded that interrupts window closing when an obstacle is detected in the window frame.
- *LS* controls automatic locking/unlocking of the door.

5.3 Behavioral Model

Using the 60+ page textual specification document [10], a total of 44 functional requirements were extracted for MCU, WCU and the Locking System. Not all the requirements were relevant as some of them were only obligated to hold in certain contexts. The preliminary behavioral model was derived by slightly modifying the sequences enumeration procedure [3]. This was done by starting from the initial state and empty sequence, enumerating sequences of all inputs one by one, annotating the corresponding outputs with each input, adding the next state. Finally, the enumeration is folded into an FSM by deriving the behavior equivalence relation between different sequences. Whenever the behavior of a particular sequence was unclear or unspecified, further enumeration of the sequence was halted since the next state of the path was uncertain. This resulted in a partial FSM model where some inputs on the states were undefined. The model was produced after peer-reviews by the domain experts that helped removing further inconsistencies. The final model $\mathcal{B}^{(1)}$ consisted of 23 states and 305 transitions.

5.4 Tool Support

The framework has been implemented in our in-house tool called RALT. It is equipped with GUI, which helps user to plug the system to the tool via test adapters. The user can also interact with the tool during the framework execution and can visualize outputs (e.g., inferred models, fail-traces) at the intermediate steps. The visualization is provided by external tools (e.g., Graphviz[5]) that are connected with RALT via their relevant automata libraries.

PROVEtech:TA [12] was used as a test runner that automates the test execution process for the ECUs. It uses WinWrap Basic for writing test scripts in which signal values were set for each ECU. Thus, the test adapter for RALT was developed by writing test scripts in PROVEtech:TA that executes tests and provides the observations. The mapping from abstract inputs/output to actual signals (and vice versa) was hard-coded in the scripts, but is planned to be extended towards a more generic solution.

The real-time interfacing between PROVEtech:TA and the door control system and the electrical emulation of exterior sensors and actuators was provided by the hardware-in-the-loop (HiL) simulator from dSPACE[6]. It eventually

[5] `www.graphviz.org`
[6] `www.dspaceinc.com`

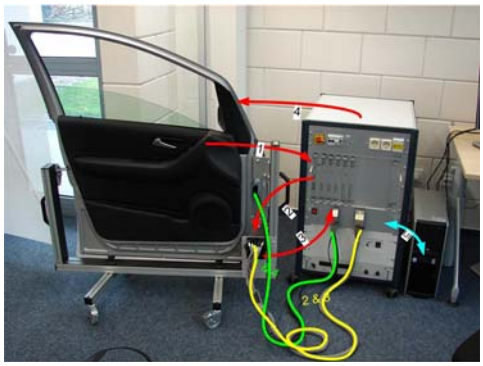

Figure 19: Physical Setup for the Evaluation Study

applied tests to the target ECUs and received output signals accordingly. Figure 19 shows the physical setup of our evaluation study in which the car door and the HiL simulator are visible. Once the setup was prepared, the framework execution was completely automatic.

5.5 Initialization

The preconditions of the experiments were set according to an initialization setup. The door was locked and all ECUs were switched off. The window was closed completely and the mirror was folded. These conditions were also viewed as system reset before applying each test sequence. In order to control timing issues, appropriate delays were added between two inputs in a test sequence. The delays were selected after a careful study of the physical latencies in the system. For example, it takes almost three seconds for the power window ECU to open/close window completely starting from the initial point. Similarly, appropriate delays were injected to capture values of the selected output signals.

5.6 Experimental Results

The framework was executed in a total of five iterations for inferring a refined specification of the door control system. In the first iteration, the inference procedure yielded a loop-free product $\mathcal{P}^{(1)}$. The conformance checking procedure computed the distinguishing machine of $\mathcal{P}^{(1)}$ and $\mathcal{B}^{(1)}$ that produced fail-traces. The fail-traces occurred due to the coarse behavioral model that had missed few non-obvious initial states of the start-up/wake-up behavior in ECUs. For example, it is required for activating the mirror heating function that the revolving motor (for mirror movement) should also be in a running state. The framework reported the nonconformance between $\mathcal{B}^{(1)}$ and the door control system.

We added the initial states in the behavioral model and created a new version $\mathcal{B}^{(2)}$. The framework was restarted for the second iteration. In this iteration, the distinguishing machine of $\mathcal{P}^{(1)}$ and $\mathcal{B}^{(2)}$ produced fail-traces, which were not confirmed on the door control system. The fail-traces occurred due to the inaccurately inferred model of WCU. The behavioral model showed that the window opening function halts if the window closing signal is given in a time span less than 5 seconds. As a consequence, the window stops opening in the middle of the frame and it does not close in response to the closing signal. The fail-trace acted as a counterexample and the inferred model of WCU was refined through the inference procedure in the third iteration, automatically.

The third iteration computed the new product $\mathcal{P}^{(2)}$ and then the distinguishing machine of $\mathcal{P}^{(2)}$ and $\mathcal{B}^{(2)}$. It produced fail-traces, which in fact identified an interaction between the mirror folding/expansion function and the telemetric sensor. The mirror cannot be expanded if the car's velocity is greater than 50 mph. This behavior was partially reflected in the product. Actually, the first few states of the inferred MCU model represented this behavior via different transitions for different velocities. As the model grew in size during its inference, the latter states could not distinguish this behavior on all velocities. This was due to the fewer observations recorded during the inference procedure that converged the behavior on those states for all velocities. A longer counterexample could distinguish those states. This longer counterexample was provided by the fault-trace of the distinguishing machine, which was not confirmed on the door control system. This triggered the fourth iteration automatically to refine the inferred model of MCU and then computed the new product $\mathcal{P}^{(3)}$. It is important to note that $\mathcal{B}^{(2)}$ also did not distinguish states for mirror expansion function on different velocities. The fault-trace occurred only due to the difference between the first few states of $\mathcal{B}^{(2)}$ and $\mathcal{P}^{(2)}$. Therefore, the distinguishing machine of $\mathcal{P}^{(3)}$ and $\mathcal{B}^{(2)}$ in the fourth iteration identified fail-traces, which were confirmed on the system. Thus, the framework reported the nonconformance between $\mathcal{B}^{(2)}$ and the door control system.

We corrected this nonconformance in the behavioral model and computed a new version $\mathcal{B}^{(3)}$. After this, the framework was started again for the fifth iteration. This time, the distinguishing machine of $\mathcal{P}^{(3)}$ and $\mathcal{B}^{(3)}$ did not yield any fail-traces and hence the procedure terminated by outputting $\mathcal{P}^{(3)}$ as the refined specification. $\mathcal{P}^{(3)}$ comprised of 45 states, compared to 40 states of $\mathcal{B}^{(3)}$. The root cause for this difference was that $\mathcal{B}^{(3)}$ was still partial (as not all the inputs were defined on all the states), hence the distinguishing machine did not compute the transitions on those states as per its definition.

5.7 Effort and Estimation Data

The evaluation study required relatively high initial effort. This included setting system boundaries, developing test adapters and preliminary behavioral model. However, the large degree of automation and automatic switching between inference and conformance checking procedures paid-off this effort. The aggregated effort data is presented in Figure 20.

As shown in the table, the major effort (34.5%) was incurred on scrutinizing the specification document for understanding the default behavior of the system. Many inconsistencies, unclear requirements and vague statements were found during this process. In most cases, the problems were resolved by consulting the domain experts with knowledge about the possible implementation (e.g, voltage, sensor readings) and conceptual issues (e.g., mechanical maneuvers).

The identification of system boundaries and finalizing the inputs/outputs sets required a considerable effort (17.2%) that included a careful review of signals (given in the document) and monitors (given in PROVEtech:TA [12]). Once the sets were prepared, developing the test adapters required lesser effort (10.3%) as it involved only mapping abstract inputs/outputs to concrete signals in the test scripts.

Constructing the preliminary behavioral model was little time consuming, but a relatively easy task after completing the activity #1. The task was shared among three engineers

#	Activity	Effort (Person Hours)	Effort (%)
1	Scrutinizing Specification (includes reading, requirement extraction, peer discussions)	80	34.5%
2	Setting System Boundaries (includes identification and finalization of input/output sets)	40	17.2%
3	Development of Test Adapters	24	10.3%
4	Constructing Behavioral Models (all iterations)	60	25.9%
5	Execution and Data Collection	8	3.5%
6	Analyzing Results	20	8.6%
	Total	**232**	**100%**

Figure 20: Aggregated Effort Data

who were also involved in the activity #1. The effort (25.9%) shown in the table is the total effort for constructing all behavioral models (i.e., $\mathcal{B}^{(1)}$, $\mathcal{B}^{(2)}$ and $\mathcal{B}^{(3)}$) during the study.

The execution phase took much less effort (3.5%) as it was supported by a high degree of automation setup. The data collection (e.g, models, fail-traces) was represented graphically thanks to the external automata tools.

The effort spent over analyzing the results (8.6%) provided the complete landscape of all iterations discussed in the previous section. Finally, the delta of the refined specification ($\mathcal{P}^{(3)}$) and the behavioral model ($\mathcal{B}^{(3)}$) highlighted the missing transitions in the behavioral model.

5.8 Traditional vs. Current Approach

We have evaluated our approach against the traditional inference approach, i.e., inference without taking into account any behavioral model. We conducted experiments for the traditional approach on the same door control system whose results have been presented in our previous work [17]. In those experiments, each component was inferred individually through the active inference algorithm [18] and then refinement iterations were triggered by applying random testing. Figure 21 compares the two approaches on the door control system by measuring the number of states and transitions for each component.

As using the behavioral model is the crux of the difference between the two approaches, the current approach appeared to be more effective. The inference of *Locking System* has produced same number of states in both approaches. But the number of transitions obtained in the current approach is greater. The additional transitions were produced due to the discrepancies from the behavioral model that specified interactions between *MCU* and *Locking System*. However, the tests generated in the traditional approach could not produce those transitions. For all the rest of the ECUs, many more states and transitions were obtained in the current approach compared to the traditional approach. We also noted that the provision of the behavioral model expedites the inference procedure because the systematic computation of counterexamples explores the hidden states more efficiently. Thus, the current approach has a better chance to quickly converge upon the hidden model.

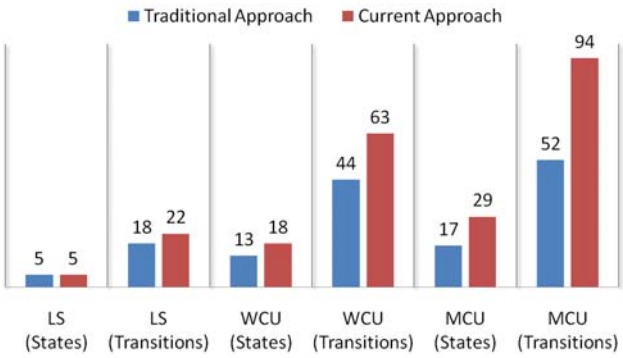

Figure 21: Traditional vs. current approach for refining specification of the door control system

5.9 Threats to Validity

We identify the following threats to the validity of our approach. Firstly, the refined specification obtained is only complete to the extent that the approach has been able to infer with the additional information from the behavioral model. Therefore, all possible behaviors are not captured in the specification. Secondly, there is a possibility of *error masking*. This can happen when there is an inaccuracy in the preliminary behavioral model that coincides with the product of the inferred models. The distinguishing machine in this case will not have a fail-trace and the refined specification obtained will carry the inaccuracy. However, both these threats are not particular to our proposed approach. While the first one is due to the theoretical limitations of inference techniques, the second is common to techniques relying on reference models. We believe that such threats can be mitigated by involving domain experts for validating the behavioral model before its use, and later, for ascertaining the quality of the refined specification. Finally, our evaluation is based on relatively smaller, but real black-box components. As discussed, the approach has demonstrated its benefit in unraveling previously unknown behaviors, compared to the traditional approach, with an acceptable additional effort. However, further evaluation with larger examples is necessary to argue for its scalability.

5.10 Lessons Learned

Many of the lessons learned from this evaluation study are not new, but might not always be explicitly stated in similar studies. The principal lessons are summarized below.

- The specification should never be treated as a *golden reference*, but as an alternate source of description for system behaviors. In reality, even a mature specification contains errors, ambiguities and inconsistencies especially when different parts are provided by different partners (OEMs, suppliers). Scrutinizing the evolving documents is costly and often leads to different interpretations in different engineering environments.
- Inference of black-box components is more efficient when supported by additional artifacts. Otherwise, the procedure would most likely terminate by outputting inaccurate models. In our study, this additional artifact was a behavioral model, which even though incomplete, proved effective in triggering the refinement iterations.

- The text-to-model formalization step for the behavioral model is mainly a manual effort. Our experience suggests that it is better to involve only a small number of engineers for this task. This reduces confusions caused by conflicting interpretations and provides faster convergence.
- Selecting an appropriate set of system inputs for specification inference is not an easy task. Even in white-box case (e.g., when source code is available), setting the range of input values requires a certain level of domain knowledge, especially for hybrid systems. Similarly, mapping abstract with concrete data is not trivial. A real system works on concrete signals, timing notions, tags, and other data fields. One has to abstract from all these details to apply formal techniques, which is a formidable task.

6. SUMMARY AND PERSPECTIVES

The paper presented a solution for an increasingly challenging problem in component based engineering practice. The problem stems from the fact that different components are developed in different environments that give rise to mismatching behaviors and interoperability problems in an integrated system. The informal specification available with these components often remains inaccurate and also lacks details of the specific design decisions and implementations.

Our contribution is a framework for iterative refinement of specification of component based systems. The framework exploits formal techniques of inference, conformance checking and reachability analysis. The novelty of our approach is the use of a preliminary behavioral model as an additional source of information that triggers specification refinement iterations in the inference procedure. Moreover, it is also useful in achieving high-level system validation as an integral part of the specification refinement procedure.

The evaluation of the framework has been conducted on an automotive system, i.e., the door control system, which is composed of a number of ECUs developed by third-party suppliers. Although evaluated for embedded systems, the framework can also be applied to other domains of component based engineering where similar problems emerge.

A perspective to improve this framework is to devise techniques for automatic construction of behavioral models. This is planned for future work on specification refinement for automotive systems: *Adaptive Cruise Control* and *Automated Gear Selection* systems, developed by our industrial partner. Furthermore, better modeling techniques are required to capture pertinent details of embedded systems, especially for hybrid systems, which have both discrete and continuous dynamics.

7. ACKNOWLEDGMENTS

This research was funded by the Fraunhofer Commercial Vehicle Innovation Cluster project.

8. REFERENCES

[1] G. Ammons, R. Bodik, and J. R. Larus. Mining specifications. In *POPL*, pp. 4–16, 2002.

[2] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen. On the correspondence between conformance testing and regular inference. In *FASE*, pp. 175–189, 2005.

[3] J. M. Carter and J. H. Poore. Sequence-based specification of feedback control systems in Simulink®. In *CASCON*, pp. 332–345. ACM, 2007.

[4] V. Dallmeier, N. Knopp, C. Mallon, S. Hack, and A. Zeller. Generating test cases for specification mining. In *ISSTA*, pp. 85–96, 2010.

[5] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. CUP, 2010.

[6] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich. Flow analysis for verifying properties of concurrent software systems. *ACM Trans. Softw. Eng. Methodol.*, 13(4):359–430, 2004.

[7] C. Farcas, E. Farcas, I. Krüger, and M. Menarini. Addressing the integration challenge for avionics and automotive systems – from components to rich services. *IEEE Proceedings*, 98(4):562–583, Apr. 2010.

[8] S. Gören and F. J. Ferguson. On state reduction of incompletely specified finite state machines. *Comput. Electr. Eng.*, 33(1):58–69, 2007.

[9] R. Groz, K. Li, A. Petrenko, and M. Shahbaz. Modular system verification by inference, testing and reachability analysis. In *TestCom*, pp. 216–233, 2008.

[10] F. Houdek and B. Paech. Das Türsteuergerät. Eine Beispielspezifikation. Technical Report 002.02/D, Fraunhofer IESE, 2002.

[11] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pp. 1090–1126, 1996.

[12] MBtech. PROVEtech:TA. `www.mbtech-group.com`.

[13] N. Navet and F. Simonot-Lion. *The Automotive Embedded Systems Handbook*. CRC Press, 2008.

[14] D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.

[15] A. Petrenko and N. Yevtushenko. Testing from Partial Deterministic FSM Specifications. *IEEE Trans. Computers*, 54(9):1154–1165, 2005.

[16] C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.

[17] M. Shahbaz and R. Eschbach. Automatic discovery of unspecified behaviors in automotive control software. In *TAIC PART*, pp. 181–188, 2010.

[18] M. Shahbaz and R. Groz. Inferring Mealy Machines. In *Formal Methods*, pp. 207–222, 2009.

[19] G. Shu and D. Lee. Testing security properties of protocol implementations - a machine learning based approach. In *ICDCS*. IEEE, 2007.

[20] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for FIFO automata. In *FSTTCS*, pp. 494–505, 2004.

[21] N. Walkinshaw, K. Bogdanov, J. Derrick, and J. Paris. Increasing functional coverage by inductive testing: A case study. In *ICTSS*, pp. 126–141, 2010.

[22] N. Walkinshaw, J. Derrick, and Q. Guo. Iterative refinement of reverse-engineered models by model based testing. In *Formal Methods*, pp. 305–320, 2009.

[23] E. J. Weyuker. Assessing test data adequacy through program inference. *ACM TOPLAS*, 5:641–655, 1983.

Document Information

Title:	Iterative Refinement of Specification for Component Based Embedded Systems
Date:	July, 2011
Report:	IESE-030.11/E
Status:	Final
Distribution:	Public Unlimited