Specification Based Formal Testing: The EasyLink Case Study

Axel Belinfante[1], Jan Feenstra[1], Lex Heerink[2], René G. de Vries[1]

[1]University of Twente, P.O. Box 217, 7500 AE, Enschede
[2]Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven

lex.heerink@philips.com, {belinfan,feenstra,rdevries}@cs.utwente.nl

Abstract—**Testing is, in most cases, a manual activity that is time consuming and error prone. Automation, however, can severely reduce the associated costs. In the project** *Côte de Resyste* **(COnformance TEsting of REactive SYSTEms) theory is being developed and a protoype tool is being built to support the automatic test generation and execution from formal specifications for reactive systems such as communication protocols and embedded system software [14]. Industrial case studies are carried out by industrial partners to evaluate the theory and tools in practice, to identify potential bottlenecks and to suggest improvements. In this paper we describe our experiences with one of such case studies, the EasyLink case study. In this case study Audio/Video (AV) devices are automatically tested for compliance to the EasyLink standard. The purpose of this case study is to check the viability of the theory and the tools developed in project by applying them in an industrial setting. This paper discusses how the case study led to theoretical developments and tool improvements, and in that way provided valuable feedback that was used to guide the future direction of the project.**

Keywords—**conformance testing, test automation, formal methods, case study**

I. Introduction

Testing is a practical approach to check whether an implementation conforms, i.e., behaves as specified, to its specification. With the increasing complexity of embedded systems over the years, the testing of such systems has also become increasingly complex; more tests have to be executed in less time in order to be able to asses the quality of implementations up to a satisfactory level within reasonable time. To meet this trend, automation of the test process can be an answer. The idea behind automation is that tests can be automatically generated from a specification and executed against an implementation in a much faster and more consistent manner than humans can.

In the project *Côte de Resyste* theory is being developed and tools are being built to support the automatic generation and execution of By means of a formal specification the required behaviour of a system can be described in a precise and unambiguous manner. Furthermore, formal specifications are, unlike natural languages, amenable to interpreation by machines, so tools can be used to process such specifications. In the *Côte de Resyste* project a tool called TorX is being developed that can automatically interpret such specifications, derive tests from it, and execute these tests against an implementation. Implementations are considered to be black boxes, i.e., no internal details of the implementation are assumed to be known and the only way to access them is by providing stimuli and observing the responses.

To check the viability of the theory and the tools that were developed in the project, to identify and carry out improvements and to provide direction for future research industrial case studies are carried out. One of such case studies is the EasyLink case study. This paper describes and consolidates the experiences gained in automatically testing the preset download feature of the EasyLink protocol. The experiences of the EasyLink case study are described more or less in a chronological order, and it is discussed how the experiment itself identified problems that were solved in theory or led to tool adaptations.

The outline of this paper is as follows. Section II describes the context of the EasyLink case study, and provides some background information. Section III provides a quick overview of the relevant theoretical aspects of automated testing, section IV presents the models that were used for test generation, section V describes the tools that were used and the experiments that were carried out, and section VI describes to which extend this case study has triggered theoretical developments and tool improvements. Conclusions are presented in section VII.

II. Context: AV.Link and EasyLink

Customer behaviour of Audio Video (AV) devices is such that these devices are purchased from many different manufacturers, connected in different combinations, at dif-

ferent moments in time. This requires an interconnection mechanism that is extendable and suited for the interconnection of existing devices as well as new devices. [1] defines a point-to-point interconnection system between AV devices by means of a so-called peritelevision connector. To connect more than two devices a star topology is required, e.g., by means of a switch box with as many connectors as the number of devices to be interconnected. However, to limit the costs in such a star topology the number of connectors on the switch-box must be limited, which conflicts with the requirement of extendability. To provide a better solution a chain configuration interconnection mechanism is described in the AV.Link standard [2]. This standard is complementary to [1].

A. AV.Link

The AV.Link standard [2] specifies a protocol for control-oriented data communication between a chain of AV devices (such as a TV and a VCR), where the TV must always be situated at the top of the chain. Communication is defined over a single wire between one of the unused pins of a scart connection (which is present on most AV devices). Message exchange defined by the AV.Link standard is serial and analogous.

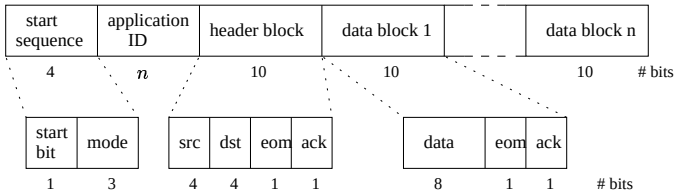

Fig. 1. Message format of AV.Link messages

The AV.Link standard defines a generic frame format for messages that can be send over the wire (see Figure 1). This generic frame format consists, a.o., of an 'application ID' bit structure. Different instantiations for the 'application ID' allow for the definition of different protocols on top of AV.Link. Furthermore, each message consist of a header block (which indicates, a.o., the source and destination address of the message), and a variable number of data blocks (each block contains an indicator whether or not more data blocks will follow).

B. EasyLink

One of the applications that have been defined on top of AV.Link is *EasyLink*. This proprietary protocol has first been defined in 1996 by Philips [3], [**?**] to facilitate communication between a TV and one or more AV devices (e.g., VCRs), and has been licensed under applicable patents as a defacto standard to many other Consumer Electronics compagnies since then. Commercial implementations of the EasyLink protocol are available under different names from different vendors.

Fig. 2. An example EasyLink configuration

EasyLink is a protocol that runs between a TV and a chain of AV devices. Figure 2 shows an example configuration of EasyLink devices chained together. EasyLink has been developed to provide the user of such devices with additional services that make them easy to program and use in combination. EasyLink (v1.3) provides the following features:

signal quality and aspect ratio matching – to automatically select the best signal quality in a signal path from one device to another, and to automatically adjust the aspect ratio (vertical vs. horizontal ratio) of the active image on the screen to the most appropiate setting (e.g., 4:3 or 16:9)

preset download – to automatically download the predefined settings (e.g., channel number, frequency, etc.) from the TV to the AV devices.

WYSIWYR (What You See Is What You Record) – to automatically record the image that is displayed on the TV screen by one of the connected VCRs.

EPG download – to download information for timed recordings from one device (the TV) to another (the VCR).

C. The 'preset download' feature

In this paper we will focus on testing of the preset download feature. It was chosen because it is based on a standard, exhibits a reasonable degree of reactive behaviour, is an industrially relevant application, and many implementations (TVs, VCRs) are available. The preset download feature is responsible for synchronizing predefined channel settings (presets) between the TV and connected AV devices. Each AV device that is equipped with a tuner keeps track of a so-called *preset list* which administrates the currently available channels (e.g., 1 to 100) and their associated settings. There are two types of presets: terrestrial presets and virgin presets. A channel that is associated to a terrestrial preset administrates the frequency to which the channel is tuned, the channel name (e.g., BBC-1 or Discovery), and some additional status information for

that channel. A channel that is associated to a virgin preset only administrates that the channel is a virgin setting, i.e., no frequency, channel name, etc. are associated to it. Table I lists the relevant messages associated to the EasyLink protocol.

start_preset_transfer() request the TV to send its presets
stop_preset_transfer() request the TV to stop sending its presets
end_preset_transfer() inform connected AV devices that all presets have been send
set_terrestrial_preset(ch,freq,status) inform peer device of frequency *freq* and *status* information of channel *ch*
set_preset_name(ch,name) inform peer device of name *name* of channel *ch*
set_virgin_preset(ch) inform peer device that channel *ch* is a virgin preset
abort(code) confirmation message informing the orginator that it could not handle the request associated with *code* correctly
set_language(code) inform all devices about the current TV language setting indicated by *code*
set_country(code) inform all devices about the current TV country setting indicated by *code*

TABLE I
EASYLINK MESSAGES

The behaviour of the preset download feature can be informally described as follows. Presets are always send from the TV to all other connected AV devices. Terrestrial presets are always transferred as a sequence of two messages: *set_terrestrial_preset* followed by *set_preset_name*. A virgin preset is transferred as a single message by means of *set_virgin_preset*. There are two different ways to initiate a preset transfer: either on request of the TV, or on request of AV devices that are connected to the TV. In case the initiative to send presets is taken by the TV the TV broadcasts its presets to the connected devices autonomously (e.g., because its preset list has changed). The connected AV devices can either accept or reject the presets (depending on their current device mode). Each connected AV device that accepts the presets is expected to update its preset list accordingly. In case the initiative to send presets is taken by another device than the TV, this device sends a special request *start_preset_transfer* to the TV. The TV can either accept or reject such request (depending on the mode of operation of the TV). In case of acceptance the TV will respond by sending its presets to all connected devices, otherwise it will inform the initiating device that it cannot download its presets by sending an *abort* message to this device. The AV device that initiated the preset download may decide to stop downloading presets during preset downloading by sending the message *stop_preset_transfer* to the TV, in which case the TV stops downloading the presets. When the TV is done with sending its presets or when it is forced to stop it broadcasts a special message (*end_preset_transfer*) to inform all peer devices that no more presets are to be expected. This may be followed by the messages *set_language* and *set_country*, which are send by the TV to all connected devices (these messages are implemented from EasyLink 2.0 and further). No ordering between these last two messages has been specified. In case an AV device is not able to respond to a request from another device it sends an *abort* message to the initiator of the request. *abort* messages are always parameterised with a code that indicates the type of request that could not be handled.

As EasyLink is defined on top of AV.Link, the EasyLink messages are mapped on AV.Link messages. This mapping is not relevant for this paper.

III. FROM THEORY ...

This section describes the relevant theoretical concepts and ingredients behind formal testing as carried out in this paper, and instantiates these for the EasyLink case study.

A. Formal testing

The purpose of testing is to check whether an implementation conforms to its specified behaviour. A description of the required behaviour is called a specification, and the object to be tested is referred to as the implementation under test (IUT). In formal testing the specification is assumed to be a described in a formal, mathematically precise, language with a clear semantics. The advantage of formal specifications over informal (natural language) specifications is that the first ones are precise, unambiguous, and that machines can process such specifications. This makes formal specifications amenable to automatic test generation. We will restrict to black box testing, i.e., the only way to access the implementation is by providing stimuli to it and observing responses from it.

In the EasyLink case study black box formal testing is done following the theory described in [13]. It is outside the scope of this paper to explain all the theoretical aspects associated with this theory. The only relevant thing to remember is that specifications and implementations can be modelled as labelled transitions systems, i.e., directed multi-graphs where the transitions are labelled with ac-

tions. A transition from one node to another represents a state change, and the label of the transition represents the cause of the state change in terms of a stimulus or response. Each transition can either indicate an input action (an action initiated by the environment to IUT) or an output action (a response from the IUT to the environment).

B. Test architecture

In order to test whether the IUT conforms to its specified behaviour, tests have to be generated and executed against the IUT. The 'test architecture' gives an abstract view on how the tester communicates with the IUT. Ideally, the tester communicates directly with the IUT at the implementation access points (IAPs). Unfortunately, in practice the IUT cannot always be directly accessed, but only indirectly via a so-called test context. In general, the tester then accesses the test context at the points of control and observation (PCOs), which in turn access the implementation at its IAPs. The system comprising of the IUT embedded in its context is called the 'system under test' (SUT).

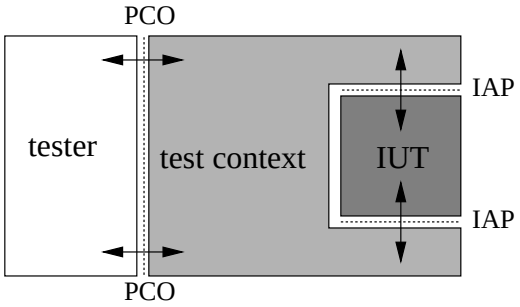

Fig. 3. Generic test architecture

To conduct a concrete testing experiment a test environment has to be set up. The test environment describes how the tester communicates with the IUT, and can be seen as a concrete instantiation of the abstract concepts such as test context, IAPs, PCOs and IUT. The test environment for EasyLink consists of a TV that is indirectly connected to a single VCR via an intermediate device called MBB by means of scart cables. This intermediate device is also connected to, and controlled by, a networked computer running our tester (TorX) via a bidirectional serial link. The TV can be operated by a uni-directional remote control. This remote control is also controlled by the computer via a human interface. Figure 4 depicts the test enviroment that was used in the EasyLink testing experiment.

The objective of the experiment is to check whether the TV implements the preset download feature correctly, i.e., the IUT is instantiated with the TV. The tester resides on the computer which is attached to the network.

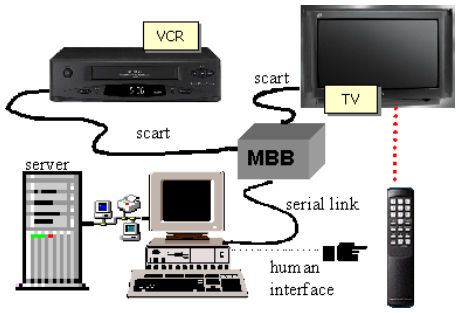

Fig. 4. EasyLink test architecture (1)

It can access the IUT indirectly by accessing the MBB via the bidirectional serial link (to provide stimuli and observe responses) and by accessing the unidirectional remote control via the human interface (to provide stimuli only). Hence, two PCOs can be identified: one represents the interface to the MBB, the other represents interface to the remote control. The IAPs are the interfaces to the software calls of the relevant software within the TV that implements the EasyLink functionality. The test context consists of everything that is needed to translate actions at the PCOs to (a series of) actions at the IAPs, and vice versa.

Interface between host and MBB – The MBB is a proprietary device that has been developed within Philips for testing purposes. No internal details of this device where known to any of us; only the functional specification describing the API commands and their effects were available. The MBB can monitor messages that are exchanged from the TV to the VCR, and downloads them to the host. It also allows the host to upload messages to the MBB, which are subsequently dispatched by the MBB to the TV or VCR (depending on the message).

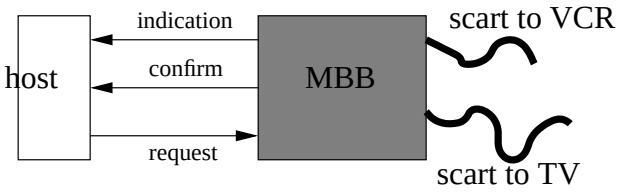

Fig. 5. Interface between the host and the MBB

Three types of messages are distinguished: a request, an indication, and a confirm (see Figure 5). A *request* message is a message that is initiated by the host and received by the MBB. Such messages may be transported by the MBB to the VCR or the TV. Request messages can be unconfirmed or confirmed. The correct handling of confirmed requests by the MBB can be notified to the host using *confirmation* messages. Messages that are received

by the MBB from the TV or the VCR can be notified to the host by means an *indication* message.

Interface between host and remote control – This interface is implemented by means of a human, i.e., a person has to push the buttons on the remote control to establish the effect on the TV required by the host. Although such interface can, in principle, be automated, this has not been done due to lack of availability of the proper equipment. For the EasyLink case study the behaviour required by the host on human is limited to shuffling channel settings on the TV. It is assumed that the human correctly implements the shuffling behaviour prescribed by the host, that is, the human will not initiate other behaviour by pressing the wrong buttons. Although the shuffling of channel settings on the TV usually requires the pressing of a sequence of buttons on the remote control, this can be modelled abstractly by a single shuffle action with two parameters: the channels to be shuffled.

Table II lists the messages at interface between the host and MBB, and the host and the human. For some messages the shortnames are indicated between brackets. These shorthands will be used in the remainder of this paper.

IV. … AND MODELS …

This section describes the behaviour of the preset download feature in combination with the (manual) shuffling of presets as a formal model. We do that in two steps: first construct a model for the preset download feature, and secondly a model for the suffling behaviour. These models are then combined into a model that specifies the combined behaviour. All models were written in the specification language PROMELA [15]. The resulting specification is used as the basis for test generation.

A. A formal model of the preset download feature

The formal models can be represented as labelled transition systems. Since communication between the tester and the IUT with respect to preset download messages runs via the host – MBB interface, the transitions in the model for the behaviour of the preset download feature are expressed in terms of messages that run over this interface (Table II). Figure 6 depicts this model as a transition system. We use the convention that inputs for the MBB (and outputs for the host) are prefixed with a '?', outputs of the MBB (and inputs for the host) are prefixed with a '!'. Sequences of actions are represented as a single transition labelled with the actions separated by a ';' (e.g., '$a; b$' represents action a followed by action b). A choice between actions is indicated by writing these actions below each other with the

messages from host to MBB
request messages
`start_preset_transfer()` (`spt`) tells the MBB to send message *start_preset_transfer* to the TV so that the TV can start downloading presets to the VCR
`stop_preset_transfer()` (`stop`) tells the MBB to send message *stop_preset_transfer* to the TV indicating that the TV can stop with sending its presets to the VCR
messages from MBB to host
indication messages
`end_preset_transfer()` (`ept`) tells the host that the MBB has received message *end_preset_transfer* from the TV indicating that the TV successfully stopped downloading its presets
`set_terrestrial_preset(ch,freq,status)` (`stp`) tells the host that the MBB has received a *set_terrestrial_preset* message from the TV
`set_preset_name(ch,name)` (`spn`) tells the host that the MBB has received message *set_preset_name*
`set_virgin_preset(ch)` (`svp`) tells the host that the MBB has received a *set_virgin_preset*
`set_language(code)` tells the host that the MBB received message *set_language* from the TV
`set_country(code)` tells the host that the MBB received message *set_country* from the TV
confirmation messages
`abort(code)` (`abort`) tells the host that the MBB was not able to process the former request message; the reason why the MBB could not handle this request message is indicated by `code`
`ack` (`ack`) tells the host that the MBB has processed the former request message successfully
`nack` (`nack`) tells the host that the MBB could not handle the former request message successfully
messages from host to human
request messages
`shuffle(ch1,ch2)` tells the user to switch the settings of channel `ch1` with those of channels `ch2`

TABLE II
MESSAGES AT THE PCOS

same indentation.

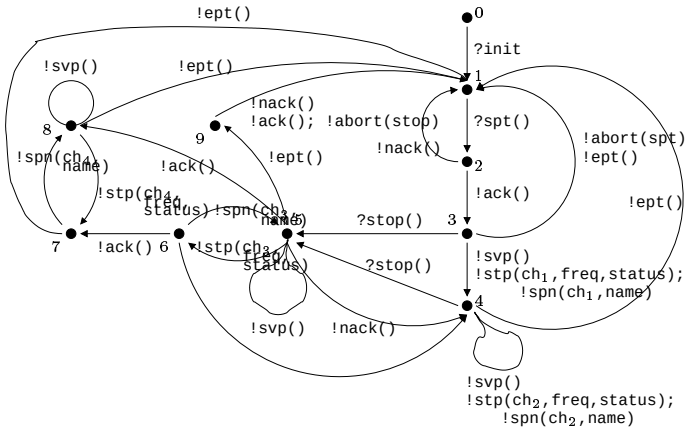

Fig. 6. Formal model of the preset download feature

The formal model presented in figure 6 is explained informally as follows. Initially, the MBB is able to receive an initialisation message `?init` (state 0) which is used to initialise the MBB (this initialisation message is unconfirmed). Next, the MBB is able to receive message `start_preset_transfer` from the host (state 1). The MBB can acknowlegde this messages back to host by sending an `!ack` (to indictae success) or a `!nack` (to indicate failure). In case of success a request to download presets has been sent to the TV. Under the assumption that the TV is in a state in which it is able to download its presets, the TV will respond with sending all its presets. These presets are received by the MBB and subsequently transferred to the host (state 3 and state 4). Each preset is either a `set_virgin_preset`, or a `set_terrestrial_preset` followed by `set_preset_name`. If the TV is done sending all of its presets, it sends an `end_preset_transfer` to the MBB. During the preset transfer, the host can send a `stop_preset_transfer` message to the MBB (which are acknowlegded by a `!ack` or `!nack`). After the sending of the `stop_preset_download` and before the reception of the `!ack` the TV can still be sending presets. Only when the TV has actually received the message `stop_preset_transfer` it will stop sending presets. This is signalled by the TV by sending the message `end_preset_transfer` message to the MBB.

Note that the model of Figure 6 explodes in size when the values of the parameters are explicitly taken into account. This artefact indeed does occur and poses problems, as will be discussed lateron (see section VI).

B. A formal model for shuffling presets

The behaviour of preset shuffling is described by means of a labelled transition system that models the causal relationships of the interactions between the host and remote control, and the effect that these actions cause on the behaviour on the host – MBB interface.

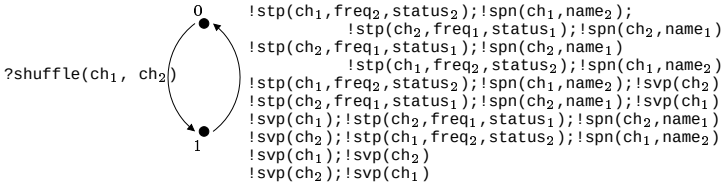

Fig. 7. Formal model of the shuffle feature

A shuffle is initiated by the host by issuing a `?shuffle(`ch_1`,`ch_2`)` request. The purpose of the shuffle operation is to switch the channel settings of channel ch_1 with those of channel ch_2. The shuffle operation itself is implemented by having a human pressing buttons on the remote control. The effect of having shuffled two channels on the TV is that the presets of the channels that are changed are broadcasted by the TV to all AV devices that are connected in the AV.Link chain so that they can update their preset list. The updated information is transfered by means of the messages `set_terrestrial_preset` and `set_preset_name` or `set_virgin_preset`. The MBB notices these messages, and informs the host. Since no restrictions on the ordering of shuffling presets are specified in [4] 8 different interleavings are possible. Figure 7 depicts all these orderings

C. Combining preset download and shuffle models

To test the preset download feature and the shuffling of presets in combination, the models of sections IV-A and IV-B must be combined. Unfortunately, these models cannot be composed in parallel because this would lead to massive state-space explosion which makes the resulting model infeasible as the basis for test generation. Therefore, it has been chosen to activate the shuffle feature only during selected phases of the behaviour of the preset download feature. For this purpose, two phases have been identified: one in which the preset download feature is not active, the other in which it is. By activating the shuffle feature in each of these phases, two different combined models have been obtained.

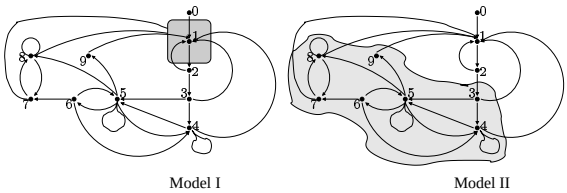

Fig. 8. Combining models features

Model I – In the first model (adopted Model I) the shuffle feature may only be activated in the phases where the preset download feature has not yet been activated. This is depicted in Figure 8 (Model I), where the shaded area denotes the area in the preset download model where the shuffle feature can be activated.

Model II – In the second model (adopted Model II) the shuffle feature may only be activated in phases where the preset download is in progress (see the shaded area in Figure 8 (Model II)). To limited the size of the resulting model, the activiation of the shuffle feature has been restricted to a limited range of preset transfers e.g., only during the first 5 presets. This range limitation is valid under the assumption that if there are no errors in the combination of the preset download feature and the shuffle features during the first 5 presets, then probably there will be also no errors for larger ranges. Such assumptions (also called uniformity hypotheses [9]) are very common in practice to restrict the number of experiments that have to be carried out.

V. ... VIA TOOLS AND EXPERIMENTS ...

This section describes the test tool TORX that was used to carry out the case study, and discusses the experimental results that were obtained.

A. TORX *tool architecture*

TORX is the prototype test generation and test execution tool that is being developed in the *Côte de Resyste* project. TORX can operate in two modes: on-the-fly and batch. For EasyLink, tests are generated and executed on-the-fly. That is, once a single test action has been selected from the specification for execution, it is immediately executed. The TORX tool architecture that is used in the EasyLink case study is depicted in Figure 9. This architecture is basically similar to the TORX architecture that was used in a former case study, the Conference Protocol [12]. It consists of the following components: EXPLORER, PRIMER, DRIVER and ADAPTOR (the System Under Test (SUT) itself is not part of the tool architecture).

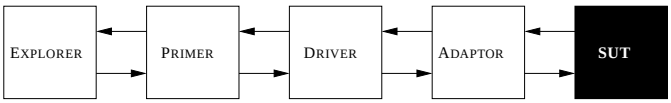

Fig. 9. TORX tool architecture

We describe the functionality of the components of the tool architecture at a global level.

- The EXPLORER is a specification language-specific component that offers functions (to the PRIMER) to explore the transition-graph of a specification and to provide, for a given state, the set of outgoing (or: enabled) transitions (actions).
- The PRIMER uses the functions provided by the EXPLORER to implement the test derivation algorithm. It offers test primitives to the DRIVER to generate inputs (stimuli) for the implementation, and to check outputs (observations) from the implementation.
- The DRIVER is the central component of the tool architecture. It controls the progress of the testing process by deciding whether to do an input or to observe and check an output. The DRIVER uses the PRIMER to obtain the input and to check whether the output of the implementation is correct. It uses the ADAPTOR to execute inputs by sending these inputs to the IUT, and to observe outputs that are generated by the IUT.
- The ADAPTOR provides the connection with the SUT. It is responsible for sending inputs to and receiving outputs from the SUT on request of the DRIVER. The ADAPTOR is also responsible for encoding and decoding of abstract actions from the DRIVER to concrete bits and bytes for the SUT, and vice versa, including the mapping of time-outs onto *quiescent* actions, see [13].

Interfaces between the components are standardised as much as possible. See [11] for more information on the components and their interfaces.

B. *Test experiments*

Infrastructure and equipment – The concrete implementation that was used in the experiment was a Philips MG98 TV, model 29PT8304/12, operating at 50 Hz AC. The TV was supposed to be compliant to EasyLink Release 2 as far as the Preset Download feature was concerned. The preset list of the TV consisted of 100 entries, that is, at most 100 channels could be programmed. The VCR that was used was a Philips VR 800/02 Natural Color video recorder, which is supposed to be compliant to EasyLink Release 2. The MBB is a proprietary device of which design details were unknown to us. This device was connected to a PC running Linux v2.0.30. This PC has an internal memory of 6828 Kb and runs on an Intel 486 microprocessor. Since this PC was slow and has a limited amount of memory, this PC was connected via an Ethernet connection to a server situated at the Philips wide LAN. Communication between the PC and the server was done via standard IP communication mechanisms such as telnet.

Tests – In total 317 invocations of the preset download feature were executed (this means that 317 times the message `start_preset_transfer` was invoked). All these invocations where automatically generated from the PROMELA specification. Of the 317 invocations, 218 tests were executed when the preset download feature was not

activated (Model I, see section IV-C), and 99 of them were carried out when the shuffling of presets was enabled during the activiation of the preset download (Model II). It took approximately one and a half day to execute the tests. The time limiting factor in the execution of the tests was the manual PCO; in case of shuffling the remote control had to be operated manually.

Detected errors – While carrying out the experiment, two (believed) errors were deteted. The first error had to do with the fact that the TV did sometimes send an 'empty' message to the VCR. Under the interpretation of MBB messages, this meant that the TV has send an 'empty' message to the VCR. The EasyLink specification did not specify such messages, and so this is considered erroneous. This error, however, was not considered a severe one; the VCR will by default reject all messages that it cannot understand. This default behaviour does not endanger the overall functionality of the preset download feature. The error seemed to occur nondeterministically; we were not able to produce this error in a controlled way.

The second error does reveal unwanted behaviour. It appears that, when the TV is in standby and the host sends the message `start_preset_transfer` to the TV (via the MBB) to start downloading presets, the TV ends up in a state with a blue screen. The only way to escape from this state is to turn the TV off and on. This second error can formally not be attributed to EasyLink, because the specification is not decisive about the fact whether the 'standby' state of the TV is a state in which the TV is able to receive presets or not (this is required for the preset download feature to operate correctly). However, even though the EasyLink specification is not clear, the observed situation causes unwanted user behaviour and is therefore considered an error from a user perspective.

VI. … BACK TO THEORY AND TOOLS AGAIN

This section describes the impact of the case study on theoretical research and tool development within *Côte de Resyste* by addressing the problems that were encountered during the case study, and by describing the solutions that were provided. We believe that some of the solutions are sufficiently generic, so that they can be applied to similar associated problems that may pop up in different case studies.

A. Problems and solutions

In this section we address some of the problems that we encountered, and we present the solutions that were provided to overcome them.

Initial state – In the EasyLink case study the initial state of the TV is unknown. That is, it is not known beforehand how many channels are available, to which frequency each channel is tuned, and what the name of the broadcasting station is. Consequently, the tester should initially be prepared to receive a huge amount of possible reactions from the IUT (i.c., every combination of channel number, frequency and name). This leads to an explosion in the number of states and number of transitions (potentially, there can be 256 different channel numbers, 65535 different frequency numbers, so the number of different combinations grows rapidly). However, once the tester has collected all initial information of the TV and knows the settings of the TV, the set of states and transitions is manageable. This means that the branching structure of the specification is such that initially the specification seems to diverge, but after the initial settings have been resolved a manageable specification (in terms of number of states and transitions) is obtained.

To avoid the initial state space explosion, two approaches have been tried out. The first approach (called the bootstrap-approach) is a two-step approach. first, the initial settings of the TV are obtained by the tester by sending a `start_preset_transfer` message to the TV. Next, a dedicated specification was generated in which the initial settings of the TV were encoded explicitly. This specification was then used as the basis for test generation. The advantage of this approach was that the initial state space explosion was avoided because the generated specification was already configured with the settings of the TV. The disadvantage, however, was that in order to obtain the initial settings the preset download feature had to be invoked. But this was also the feature under test! Therefore, this approach is only valid under the assumption that the TV initially transfers all its settings correctly to the TV.

In the second approach the outputs produced by the TV were treated symbolically. This means that the parameter values for the names and frequencies are not a priori unfolded in the transition system, but that a single transition exists with references to variables, and that these variables are bound at run-time to their concrete values. As symbolic test generation and test execution is not well-studied in theory, some theory had to be developed and the language PROMELA had to be adapted for this in order to specify such symbolic transitions. These activities were carried out in the *Côte de Resyste* project as a direct result of the EasyLink case study, and a prototype test environment was built that demonstrated the usage of symbolic output transitions.

Both approaches mentioned before have been implemented and tested. As these approaches only differ in the way transitions and states are generated but not in the func-

tionality that they represent, both approaches were equally powerful.

Discarding messages – The preset download feature is only one of the features that is provided by EasyLink. During the experiment it was observed that the TV did also send some messages that were outside the scope of the preset download feature. Without any adaption of the current testing strategy the specifiction had to take all these 'irrelevant' messages into account. This would severly complicate the specification. To avoid having to specify such 'irrelevant' messages a generic solution has been provided. The idea behind this solution is to simply ignore the irrelevant messages produced by the IUT. In that way these messages do not have to be compared with the actions in the specifictions. In our implementation we have implemented this in the ADAPTOR (see Figure 9) by mapping such irrelevant messages to a special message called 'Ignore', that is subsequently transfered to the DRIVER. The DRIVER uses this message for logging purposes only, but does not use this message for comparison with the specification.

By ignoring irrelevant messages in a systematic manner as described above, the specification itself is not polluted with such actions. We expected that such approach can, in general, severly simplify the modelling process of the specification itself.

Distribution of the tester – In this case study the tester has been distributed over a PC and a server that are connected by means of a network (see Figure 4). The reason for this distribution is that the PC itself was too slow and provided too little computational resources. Therefore, the tester was split: the less resource consuming ADAPTOR was stored on the PC, and the computationally intensive PRIMER and DRIVER were allocated to the server. It turned out that splitting the tester can be done very easily due to the message-oriented interfaces between the building blocks of the tester (see Figure 9). In general, we believe that that splitting the tester is important in case limited resources are available on the target system itself (e.g., in the case of embedded systems), or in case the IUT needs to be tested remotely. As far as we now this is the first industrial case study where TORX itself has been distributed in order to optimally use the available resources.

VII. CONCLUSIONS

In this paper the results of the EasyLink case study have been described. This case study has been carried out as part of the *Côte de Resyste* project. It is the first industrial case study that was carried out in this project. The purpose of the case study was to check the viability of the theory and the tools that were developed in the project by applying them in an industrial setting, to identify and carry out improvements and, by doing so, to provide direction to the theory and tool development of specification based formal testing in the context of *Côte de Resyste*. To carry out the case study a formal specification of the preset download feature has been written in the language PROMELA [15]. This specification was used as the basis for test generation and test execution using the test tool TORX that was developed within *Côte de Resyste*. The most time consuming phase of the case study was to get the test environment operational, and not the construction of the specification. This can be explained by the fact that this case was the first industrial case study carried out in the project and that people involved were relatively unexperienced with the test environment.

It turned out that the theory and the tool TORX could be applied in the case study, but that several improvements were needed to make the tool practical. Some of these improvements were

- the introduction and application of symbolic testing to decrease the state space and transition space explosion problem;
- the application of a generic method to systematically discard irrelevant message that are produced by the IUT, so that these messages do not have be modelled in the specification;
- the distribution of the tester over different machines in order to optimally use the available resources.

All these improvements were triggered by the case study, and have been carried out during the case study. However, also some problems were identified that have not resolved yet. It turned out that the ADAPTOR that had to be provided still required a lot of coding, and that a more systematic approach to construct an ADAPTOR is necessary. Furthermore, it turned out that the user guidance in the form of test purposes (i.e., specifications that specify the property that is to be tested) would greatly improve the usability of the tool in a practical setting. We believe that future case studies will benefit from the results of this case study, and that this case study has provided direction and insight in the issues that need to be investigated and their associated priorities in *Côte de Resyste*.

REFERENCES

[1] *Domestic and similar electronic equipment interconnection requirements: Peritelevision connector*. CENELEC standard EN 50049-1 [confidential].

[2] *Domestic and similar electronic equipment interconnection requirements: AV.Link*. CENELEC standard EN 50157, 1998 [confidential].

[3] *EasyLink – Commercial Requirements Specification* Ref.no. AR29-R-34. Philips Consumer Electronics. [Confidential]

[4] *EasyLink – version 1.3* Functional description. Ref.no. AR29-n-08. Philips Consumer Electronics. [Confidential]

[5] *EasyLink Functional Description Philips Release 3* by Project50 Team, 28 July 1998. Ref.no. APG/LT 98/35, version 1.0 [Confidential]

[6] *Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework*. International Standard IS-9646, ISO, Geneva, 1996. Also: CCITT X.290-X.294.

[7] *Project50 – Bus interface functional specification* by D.J. Woolgar. Project Note APG/LT98/30, version 3. Advanced Project Group, Philips consumer Electronics, Redhill, UK [Confidential]

[8] *Project50 – Functional Description for non-SCART systems* by Project50 Team, 18 August 1998. Ref.no. APG/LT 98-12, version 1.0 [Confidential]

[9] Olivier charles and Roland Groz. Formalisation d' hypothèses pour l'évaluation de la couverture de test. In: *Colloque Francophone sur 'Ingenierie des Protocols (CFIP)*, pp. 484-497, 1996. Bennani, Dssouli, Benkiran and Rafiq, editors.

[10] Côte de Resyste – Project Description. Available at `http://fmt.cs.utwente.nl/projects/CdR-html/`.

[11] Côte de Resyste – Working tool document. Available at `http://fmt.cs.utwente.nl/projects/CdR-html/` (restricted access).

[12] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, Int. Workshop on Testing of Communicating Systems XII, pages 179-196. Kluwer Academic Publishers, 1999.

[13] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, vol. 17, no. 3pp. 103-120, 1996. Also: Technical Report no. 96-26, Centre for Telematics and Information Technology, University of Twente, The Netherlands.

[14] René de Vries, Jan Tretmans, Axel Belinfante, Jan Feenstra, Loe Feijs, Sjouke Mauw, Nicolae Goga, Lex Heerink, Arjen de Heer. Côte de Resyste in Progress. *PROGRESS 2000: First Workshop on Embedded Software and Systems*, p.157-164.

[15] SPIN. On-the-Fly LTL Model Checking with SPIN. URL: `http://netlib.bell-labs.com/netlib/spin/whatisspin.html`