A Framework and a Tool for Robustness Testing of Communicating Software

Fares Saad-Khorchef
LABRI, CNRS (UMR 5800)
F-33405 Talence, France
saad-kho@labri.fr

Antoine Rollet
LABRI, CNRS (UMR 5800)
F-33405 Talence, France
rollet@labri.fr

Richard Castanet
LABRI, CNRS (UMR 5800)
F-33405 Talence, France
castanet@labri.fr

ABSTRACT

Robustness testing aims at verifying the acceptable behavior of a system under unexpected conditions. In this paper we propose a framework and a tool for robustness test cases generation. Our framework consists of two phases : (1) Construction of an *increased specification* by integrating hazards in the nominal specification model written in SDL. The rule of the increased specification is to specify the acceptable behavior in presence of hazards. (2) A specific method to generate robustness test cases (in TTCN-3) from the increased specification and a robustness test purpose. We also give some experimental results on the TCP protocol.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Testing and Debugging

General Terms

Formal Testing

Keywords

Robustness testing, IOLTS, TCP protocol

1. INTRODUCTION

Nowadays, software systems tend to be complex, embedded, and often used in critical conditions. A failure of such systems may lead to catastrophic consequences (financial or human). Then, a proper validation is highly needed in order to increase the quality and the confidence of the system. Testing is a part of the validation process consisting in a direct execution of the system implementation IUT (*Implementation Under Test*) under specified conditions. The results are observed and compared to the expected behavior of the system. Testing may focus on different topics such as conformance, reliability, interoperability and robustness.

In this paper, we deal with robustness testing of communicating systems (e.g. communicating protocols). Although a precise definition of robustness is somewhat elusive, functionally the meaning is clear : "the ability of a system to function correctly in presence of faults or stressful environmental conditions" (*IEEE* [9]). The term "hazards" will be used to gather faults and stressful conditions.

Usually, system specifications do not take care about unexpected conditions. Note that it is never possible to have a complete specification of the system directly, but it is possible to specify a behavior facing a specific hazard when this latter is identified.

One contribution of our work is to provide a framework permitting to take into account these aspects. This aim is achieved by integrating representable hazards in the nominal specification of the system. The obtained model is called the *increased specification*. Because of its important size, we propose a specific method to generate robustness test cases using a test purpose.

We present the RTCG (*Robustness Test Cases Generator*) tool which implements this approach. It permits to extract robustness test cases (in the TTCN-3 [15] format) based on a given robustness test purpose and on the increased specification (written in the SDL [7] format).

The paper is organized as follows. Section 2 recalls the models used in our study. Section 3 presents our framework for robustness testing. Section 4 describes the RTCG tool and provides a case study on the TCP protocol. Section 5 gives a state of the art and we finally conclude in section 6.

2. BASIC CONCEPTS

2.1 Models of specification

Usually, communicating softwares are specified in a dedicated language (SDL, LOTOS, UML, etc...). Such formalisms are based on labelled transition system (LTS) semantics. LTS distinguishes internal and visible actions. But in black-box testing, a distinction is often made between inputs and outputs. In this paper we use the IOLTS model (Input Output Labelled Transition System).

Definition 1 (IOLTS) *An IOLTS [13] is a quadruplet* $S = (Q, \Sigma, \rightarrow, q_0)$ *such that:* Q *is a nonempty finite set of states,* q_0 *is the initial state,* Σ *is the alphabet of actions and,* $\rightarrow \subseteq Q \times \Sigma \times Q$ *is the transition relation.*

The alphabet Σ is partitioned into three sets $\Sigma = \Sigma_O \cup \Sigma_I \cup I$, where Σ_O is the output alphabet (an output is denoted by $!a$), Σ_I is the input alphabet (an input is denoted by $?a$) and I is the alphabet of internal actions (an internal action is denoted by τ). Usual notations are :

SAC'07 March 11-15, 2007, Seoul, Korea

Notation	Meaning
$q \xrightarrow{a}$	$\exists\, q' \mid q \xrightarrow{a} q'$
$q \xrightarrow{\mu_1 \cdots \mu_n} q'$	$\exists\, q_0 \cdots q_n \mid q = q_0 \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} q_n = q'$
$q \stackrel{\varepsilon}{\Rightarrow} q'$	$q = q'$ or $q \xrightarrow{\tau_1 \cdots \tau_n} q'$
$q \stackrel{a}{\Rightarrow} q'$	$\exists\, q_1, q_2 \mid q \stackrel{\varepsilon}{\Rightarrow} q_1 \xrightarrow{a} q_2 \stackrel{\varepsilon}{\Rightarrow} q'$
$q \stackrel{a_1 \cdots a_n}{\Rightarrow} q'$	$\exists\, q_0 \cdots q_n \mid q = q_0 \stackrel{a_1}{\Rightarrow} q_1 \stackrel{a_2}{\Rightarrow} \cdots \stackrel{a_n}{\Rightarrow} q_n = q'$
$q\ after\ \sigma$	$q' \in Q \mid q \stackrel{\sigma}{\Rightarrow} q'\}$; by extension, $S\ after\ \sigma = q_0\ after\ \sigma$
$Trace(q)$	$\{\sigma \in \Sigma^* \mid q \stackrel{\sigma}{\Rightarrow}\}$; by extension, $Trace(S) = Trace(q_0)$
$Out(q)$	$\{a \in \Sigma_O \mid q \stackrel{a}{\Rightarrow}\}$
$Out(S, \sigma)$	$Out(S\ after\ \sigma)$
$ref(q)$	$\{a \in \Sigma_I \mid a \not\rightarrow)\}$

An *IOLTS* S is called *deterministic* if no state accepts more than one successor with an observable action. It is called *observable* if no transition is labelled by τ. S is called *input-complete* if each state accepts all inputs of the alphabet.

2.2 Hazards

In robustness testing, a *hazard* denotes any event not expected in the nominal specification of the system. They may be internal, external or beyond the system boundaries [3] or classified according to tester controllability or/and formal representability [12].

In this paper, we deal with controllable and representable hazards related to communicating software domain. Controllability means the ability of the tester to control the presence of hazards (e.g. erroneous or unexpected inputs), and representability means that it is possible to represent the hazard in the IOLTS model (e.g. inputs or outputs). More precisely, we identify three kinds of controllable and representable hazards :

- **Invalid Inputs** In a hostile environment, exchanged messages may be infected by accidental or intentional faults. Formally, we consider as an "*invalid input*" any unspecified input. i.e, $?a' \notin \Sigma_I$.
- **Inopportune Inputs** In a hostile environment, the communicating software entity may receive delayed or untidy messages. Formally, "*inopportune inputs*" correspond to actions existing in the alphabet of the specification, but not expected in the given state. $ref(q)$ denotes the inopportune inputs in a state $q \in Q$.
- **Unexpected outputs** Taking into account the hazards can lead the system, in some cases, to send some unexpected outputs. Sometimes, such outputs may be considered as acceptable. For example, restarting a session, resetting or closing a connection may be acceptable behaviors. As a consequence, all acceptable outputs must be added to the specification (e.g. restarting or closing connection messages). Formally, $!x'$ is an acceptable output if $!x' \notin \Sigma_O$ or $!x' \in \Sigma_O \wedge !x' \notin Out(q)$.

3. PROPOSED APPROACH

In this section, we outline our formal approach to generate robustness test cases. Two phases are given : firstly we construct an increased specification, and secondly we generate robustness test cases. Note that the nominal specification describes the expected behavior in nominal conditions. In the following, it is modelled by an IOLTS denoted S.

3.1 Phase 1 : Increase of specification

This phase consists in integrating the representable hazards (invalid inputs, inopportune inputs and acceptable outputs) in the model of the nominal specification. The obtained model is called *increased specification*. The aim of the *increased specification* is to formally describe the acceptable behaviors in presence of controllable and representable hazards. Robustness of an implementation is evaluated with respect to the increased specification. In order to obtain the increased specification, we proceed to the following steps (see Fig.1) :

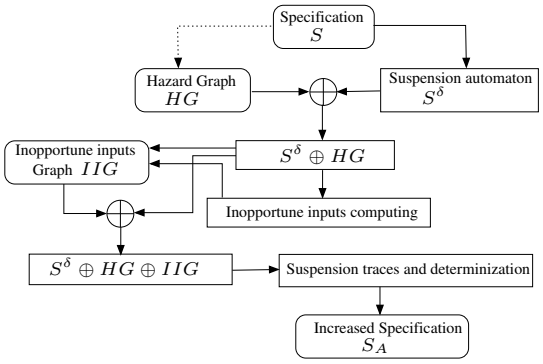

Figure 1: Obtaining the increased specification

3.1.1 Quiescence

In practice, the tester observes outputs of a system, but also the absence of events (quiescence). Several kinds of quiescence may happen in a state $q \in Q$: ***outputlock*** quiescence if the system is blocked on standby input of the environment ($Out(q) = \emptyset$), ***deadlock*** quiescence if there is no more evolution of the system ($\forall a \in \Sigma | q \stackrel{a}{\not\rightarrow}$) or ***livelock*** quiescence if $q \stackrel{\varepsilon}{\Rightarrow} q$. To model valid quiescence in $IOLTS$ model, we use the suspension automaton defined below :

Definition 2 (Suspension automaton) *The suspension graph [13] of* $S = (Q, \Sigma, \rightarrow, q_0)$ *is an IOLTS* $S^\delta = (Q, \Sigma^\delta, \rightarrow_\delta, q_0)$ *such that:* $\Sigma^\delta = \Sigma \cup \{\delta\}$ *with* $\delta \in \Sigma_O^\delta$. $\rightarrow_\delta$ *is obtained from* $\rightarrow$ *by adding loops* $q \xrightarrow{\delta} q$ *for all quiescence states.*

Thus, quiescence is seen as an observable output action. In practice, the tester identifies such event with a timeout. The first step of our approach consists in obtaining the suspension automaton S^δ associated to S.

3.1.2 Acceptable behavior

In order to check the robustness of the system, the acceptable behavior in the presence of hazards has to be given by the system designers. The acceptable behavior is supposed modelled by a specific graph called *meta-graph*. Let $S = (Q, q_0, \Sigma, \rightarrow_S)$ an nominal specification. A meta-graph G, associated to S, is a graph such that each state of G corresponds to a set of states of S having the same behaviors in the presence of the same hazards.

Definition 3 *A meta-graph associated to* S *is a triplet* $G = (V, E, L)$ *such as : (1)* $V = V_d \cup V_m$ *is a set of states.* $V_m \subseteq 2^Q$ *is called the set of* meta-states *and* V_d *is called the set of* degraded states *such that* $V_d \cap Q = \emptyset$. *(2)* L *is an alphabet of actions, (3)* $E \subseteq V \times L \times V$ *is a set of edges.*

In the following, we suppose that invalid inputs and acceptable outputs are modelled by one or more meta-graph(s) HG (*Hazards Graph*), and inopportune inputs are represented by meta-graph(s) IIG (*Inopportune Input Graph*).

3.1.3 Integrating hazards

This step consists in the composition of the nominal specification S and a hazard graph HG. The composition between an IOLTS and a meta-graph is defined by :

Definition 4 (Composition $IOLTS \oplus G$) *Let $S = (Q, q_0, \Sigma, \rightarrow_S)$ be an IOLTS and $G = (V, E, L)$ a meta-graph associated to S. The composition of S and G, noted $S \oplus G$, is the IOLTS $(Q^{S\oplus G}, q_0^{S\oplus G}, \Sigma^{S\oplus G}, \rightarrow_{S\oplus G})$ defined by: $Q^{S\oplus G} = Q \cup V_d$, $q_0^{S\oplus G} = q_0$, $\Sigma^{S\oplus G} = \Sigma \cup L$ and the following rules :*

1. $q \xrightarrow{a} q' \iff q \xrightarrow{a}_{S\oplus G} q'$
2. $(v, a, v') \in E$ *and* $v, v' \in V_d \iff v \xrightarrow{a}_{S\oplus G} v'$.
3. $(v, a, v') \in E$, $v \in V_m$ *and* $v' \in V_d \iff q \xrightarrow{a}_{S\oplus G} v'$ *for all* $q \in v$.
4. $(v, a, v') \in E$, $v \in V_d$ *and* $v' \in V_m \iff v \xrightarrow{a}_{S\oplus G} q$ *for all* $q \in v'$.
5. $(v, a, v') \in E$ *and* $v, v' \in V_m \iff q \xrightarrow{a}_{S\oplus G} q'$ *for all* $q \in v$ *and* $q' \in v'$
6. $(v, a, v) \in E$ *and* $v \in V_m \iff q \xrightarrow{a}_{S\oplus G} q$ *for all* $q \in v$.

This composition consists in adding in S the set of transitions and states of meta-graph HG. Actually, for a state q of S member of a meta-state (i.e. a set of states) v of HG, we add in S the set of transitions and/or states starting from v.

Example 1 *In Fig 2.(c), the composition $S^\delta \oplus HG$ is obtained as follows :*
Rule 1 *adds to $S^\delta \oplus HG$ the whole of transitions of S^δ ($q_0 \xrightarrow{?a} q_1$, $q_1 \xrightarrow{!x} q_2$, $q_1 \xrightarrow{?b} q_3$, $q_3 \xrightarrow{!y} q_4$, $q_0 \xrightarrow{!\delta} q_0$, $q_2 \xrightarrow{!\delta} q_2$);*
Rule 2 *adds to $S^\delta \oplus HG$ the transition $d_2 \xrightarrow{?b'} d_1$;*
Rule 3 *adds to $S^\delta \oplus HG$ the following transitions ($q_0 \xrightarrow{?a'} d_2$, $q_1 \xrightarrow{?a'} d_2$, $q_2 \xrightarrow{?a'} d_2$, $q_3 \xrightarrow{?a'} d_2$, $q_0 \xrightarrow{!x'} d_1$, $q_1 \xrightarrow{!x'} d_1$, $q_2 \xrightarrow{!x'} d_1$, $q_3 \xrightarrow{!x'} d_1$);*
Rule 4 *adds to $S^\delta \oplus HG$ the following transitions ($d_1 \xrightarrow{?a} q_0$, $d_2 \xrightarrow{?a} q_0$);*
Rule 6 *adds to $S^\delta \oplus HG$ the following transitions ($q_0 \xrightarrow{?b'} q_0$, $q_1 \xrightarrow{?b'} q_1$, $q_2 \xrightarrow{?b'} q_2$, $q_3 \xrightarrow{?b'} q_3$).*
Rule 5 *is not used because there are no transitions between the meta-states.*

After the integration of invalid inputs and acceptable outputs in S^δ, we compute the inopportune inputs (using the ref set of each set) of $HG \oplus S^\delta$. Then, system designers give the required acceptable behavior in this case. The given description is modelled by IIG (Fig 2 (d)).
We reuse the definition 4 in order to integrate inopportune inputs in $HG \oplus S^\delta$. The obtained model is $HG \oplus S^\delta \oplus IIG$ (Fig 2 (e)).

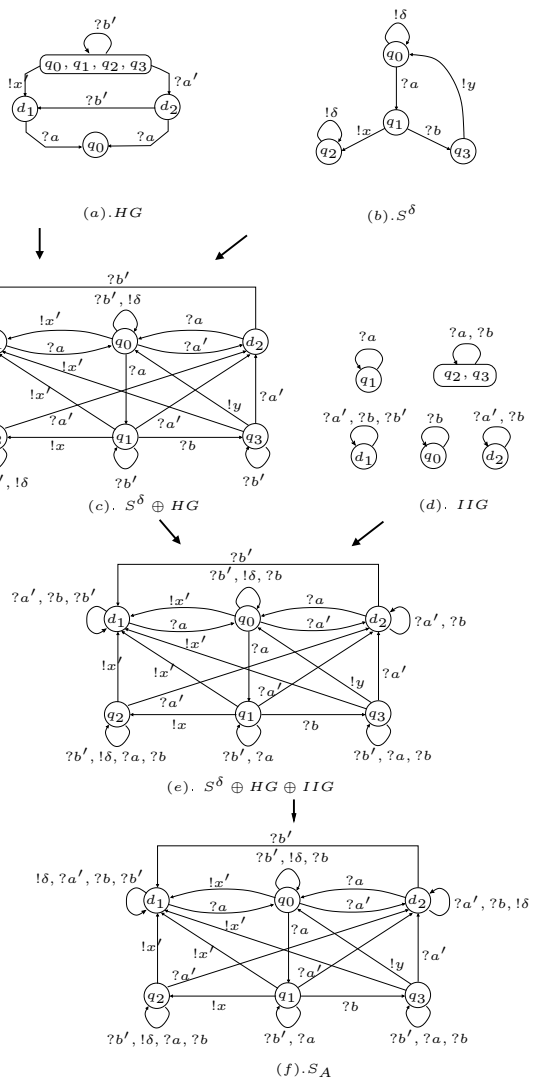

Figure 2: Construction of the increased specification

3.1.4 Determinization

As robustness testing is based on the observation of visible behaviors, test synthesis requires a determinization of the specification. The deterministic model obtained from the suspension automaton associated to $HG \oplus S^\delta \oplus IIG$ is called the increased specification (Fig 2.(f)), and denoted S_A.

3.1.5 Robustness relation

The IUT is a black box interacting with a tester. We apply the *test hypothesis* generally used in testing research, assuming that : (1) IUT is modelled by an IOLTS $IUT = (Q^{IUT}, \Sigma^{IUT}, \rightarrow_{IUT}, {q_0}^{IUT})$ such that : ${\Sigma_I}^{S_A} \subseteq {\Sigma_I}^{IUT}$ and ${\Sigma_O}^{S_A} \subseteq {\Sigma_O}^{IUT}$; (2) IUT is *input-complete* on the alphabet Σ^{S_A}. We also assume that IUT conforms to S with respect to the conformance relation ***ioco*** [13].

Let IUT be an implementation of a specification S and S_A its increased specification. The robustness relation **Robust**

is defined by :

$$IUT \textbf{ Robust } S_A \equiv_{def} \forall \sigma \in Trace(S_A) \backslash Trace(S^{\delta}) \iff Out(IUT^{\delta}, \sigma) \subseteq Out(S_A, \sigma).$$

Only the increased behaviors (added) are useful for robustness testing because the nominal behaviors (including valid quiescence) already passed the conformance testing.

3.2 Phase 2 : Robustness test generation

In this section we present a robustness test case generation technique. Using test purpose permits to reduce the test selection domain and to concentrate the efforts in order to check some critical functionalities.

We give the different steps : using the increased specification S_A and a Robustness Test Purpose RTP, we compute the synchronous product $S_A \otimes RTP$. Then, we use this result to compute a Robustness Test Graph RTG, and then a Reduced Robustness Test Graph RRTG. Finally, Robustness Test Cases (RTC) are generated using RRTG. An example is given in Fig 3, and details are given just below.

3.2.1 Robustness test purpose

A robustness test purpose (RTP) permits to select a part of the total specification in order to focus on a precise functionality (e.g, robustness property). Formally,

Definition 5 (RTP) *A robustness test purpose is a deterministic and observable IOLTS* $RTP = (Q^{RTP}, \Sigma^{RTP}, \rightarrow_{RTP}, q_0^{RTP})$ *with two sets of trap states "**Accept**" and "**Reject**", with the same alphabet as the increased specification (i.e.* $\Sigma^{RTP} \subseteq \Sigma^{S_A}$*).*

Example 2 *The RTP given in Fig 3.(b) aims at seeking any trace of the increased specification containing a reception of the invalid input* $?a'$ *followed by the acceptable output* $!x'$ *without considering the transitions labelled by* $!x$ *or* $?b$*.*

The label "*other*" is used to describe all actions of the alphabet $\Sigma^{S_A \otimes RTR}$ which are not specified in the current state.

3.2.2 The synchronous product $S_A \otimes RTP$

In order to obtain a robustness test sequence, we have to cover simultaneously the RTP and S_A until we find an adequate sequence satisfying RTP. The synchronous product is defined as follows :

Definition 6 (Synchronous product) *Let* $S_A = (Q^{S_A}, q_0^{S_A}, \Sigma^{S_A}, \rightarrow_{S_A})$ *be an IOLTS of the increased specification, and* $RTP = (Q^{RTP}, q_0^{RTP}, \Sigma^{RTP}, \rightarrow_{RTP})$ *a robustness test purpose with* $\Sigma^{RTP} = \Sigma^{S_A}$ *and with state sets "**Accept**" and "**Reject**". The synchronous product of* S_A *and* RTP*, denoted by* $S_A \otimes RTP$*, is a deterministic IOLTS* $S_A \otimes RTP = (Q^{S_A \otimes RTP}, q_0^{S_A \otimes RTP}, \Sigma^{S_A \otimes RTP}, \rightarrow_{S_A \otimes RTP})$ *defined by :*

1. $q_0^{S_A \otimes RTP} = (q_0^{S_A}, q_0^{RTP})$,
2. $Q^{S_A \otimes RTP} = \{(q_1, q_2) \mid q_1 \in Q^{S_A}, q_2 \in Q^{RTP}\}$,
3. $\Sigma^{S_A \otimes RTP} \subset \Sigma^{S_A} \cup \Sigma^{RTP} = \Sigma^{S_A}$,
4. $\rightarrow_{S_A \otimes RTP}$ *is defined by :* $(q, q') \in Q^{S_A \otimes RTP}$, $q \xrightarrow{a}_{S_A} q_1 \wedge q' \xrightarrow{a}_{RTP} q_1' \iff (q, q') \xrightarrow{a}_{S_A \otimes RTP} (q_1, q_1')$.

3.2.3 Robustness test graphs

A robustness test graph (RTG) describes all tests corresponding to a given RTP. Formally, a RTG is a deterministic IOLTS $RTG = (Q^{CRTG}, \Sigma^{RTG}, \rightarrow_{RTG}, q_0^{RTG})$, composed by three subsets of states **ACCEPT**, **REJECT** and **INCONC** such that :

- $\Sigma^{RTG} = {\Sigma_O}^{RTG} \cup {\Sigma_I}^{RTG}$ with ${\Sigma_I}^{RTG} = {\Sigma_O}^{S_A \otimes RTP}$ and ${\Sigma_O}^{RTG} = {\Sigma_I}^{S_A \otimes RTP}$ (mirror image);
- $Q^{RTG} = \textbf{ACCEPT} \cup \textbf{REJECT} \cup \textbf{INCONC}$ with
  1. $\textbf{ACCEPT} = \{q \in Q^{S_A \otimes RTP} \mid \exists \sigma \in \Sigma^{S_A \otimes RTP*}, q \xrightarrow{\sigma} \textbf{Accept}\}$ **ACCEPT** consists of states from which the state **Accept** is reachable,
  2. $\textbf{INCONC} = \{q' \in Q^{S_A \otimes RTP} \mid \exists q \in \textbf{ACCEPT}, q' \notin \textbf{ACCEPT}, a \in {\Sigma_O}^{S_A \otimes RTP}, q \xrightarrow{a} q'\}$. i.e. **INCONC** is composed of states not in **ACCEPT**, but which are direct successors of states in **ACCEPT** by an output in $S_A \otimes RTP$,
  3. $\textbf{REJECT} = \{q \in Q^{S_A \otimes RTP} \mid q \notin \textbf{ACCEPT} \wedge q \notin \textbf{INCONC}\}$.
- if ${q_0}^{S_A \otimes RTP} \in \textbf{ACCEPT}$ then ${q_0}^{RTG} = {q_0}^{S_A \otimes RTP}$, otherwise Q^{RTG} is empty.

Since RTG is often voluminous, it is necessary to reduce it by concentrating only on the behaviors accepted by RTP. Then we keep in RTG only the paths leading to an ACCEPT or INCONC states. The obtained model is called reduced robustness test graph, and denoted by RRTG.

Example 3 *Robustness Test Graph RTG (Fig. 3 d) describes the mirror image of the synchronous product (Fig. 3 c). RTG consists of three states :* ***INCONC*** $= \{(q_2,$ ***Reject***$)\}$, ***REJECT*** $= \{(d_1,$ ***Reject***$), (q_1,$ ***Reject***$), (q_0,$ ***Reject***$)\}$ *and* ***ACCEPT*** $= \{(q_0, q_0'), (d_1, q_1'), (q_1, q_0'), (q_0,$ ***Accept***$)\}$*. Reduced robustness test graph RRTG (Fig. 3 e) consists of the states and transitions of* ***ACCEPT*** *and* ***INCONC****.*

3.2.4 Robustness test case

A robustness test case (RTC) is an elementary test corresponding to a particular robustness test purpose. It describes the interactions between a tester and an implementation. It only contains observable actions.

Definition 7 *A robustness test case RTC is an IOLTS* $RTC = (Q^{RTC}, \Sigma^{RTC}, \rightarrow_{RTC}, q_0^{RTC})$ *with three sets of trap states* ***Pass***, ***Fail*** *and* ***Inconc*** *characterizing verdicts. Its alphabet is* $\Sigma^{RTC} = \Sigma_I^{RTC} \cup \Sigma_O^{RTC}$ *with* $\Sigma_O^{RTC} \subseteq \Sigma_I^{S_A}$ *(RTC emits only inputs of* S_A*) and* $\Sigma_I^{RTC} \subseteq \Sigma_O^{IUT}$ *(RTC foresees any output or quiescence of IUT).*

We make several structural assumptions on test cases : (1) states **Fail** and **Inconc** are directly reachable by inputs. Formally, $\forall (q, a, q') \in \rightarrow_{RTC} \quad (q \in \textbf{Inconc} \cup \textbf{Fail} \implies a \in {\Sigma_I}^{RTC})$; (2) from each state a verdict must be reachable. Formally, $\forall q \in Q^{RTC}, \exists \sigma \in \Sigma^{RTC*}, \exists q' \in \textbf{Pass} \cup \textbf{Fail} \cup \textbf{Inconc}, q \xrightarrow{\sigma} q'$; (3) RTC is controllable : no choice is allowed between two outputs or an input and output. Formally, $\forall q \in Q^{RTC} \forall a \in {\Sigma_O}^{RTC}, q \xrightarrow{a}_{RTC} \implies \forall b \neq a, q \not\xrightarrow{b}_{RTC}$; (4) a test case is input complete in all states where an input is possible. Formally, $\forall q \in Q^{RTC} (\exists a \in {\Sigma_I}^{RTC}, q \xrightarrow{a}_{RTC} \implies \forall b \in {\Sigma_I}^{RTC}, q \xrightarrow{b}_{RTC})$.

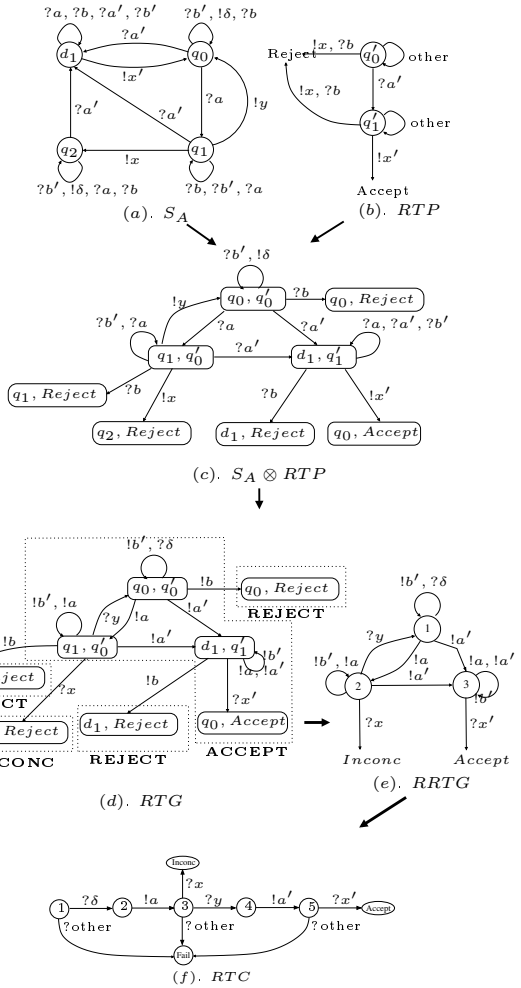

Figure 3: Robustness test cases generation

3.2.5 *Selection of robustness test cases*

In order to choose the traces which are considered in the robustness relation **Robust**, we use an algorithm based on coloration principle. Two colors distinguish the transitions of the nominal specification (first color) and those added during the construction of the increased specification (second color). Then we choose test cases favoring the second color (focusing on hazards), and we avoid any nominal trace (colored with the first color).

Example 4 *Robustness test case (RTC) given in Fig.3.(f) is derived from RRTG (Fig.3.(e)). RTC consists of two output states (states 2 and 4), and three reception states (states 1, 3 and 5).*

4. IMPLEMENTATION AND CASE STUDY

4.1 RTCG tool

RTCG is a tool automating the previous approach. It provides two functionalities :
The first one permits to build the increased specification of systems written in SDL or modelled as *IOLTS*. In order to achieve this aim, *RTCG* implements the composition method given in paragraph 3.1.3. It computes the composition of the nominal specification S with a meta-graph HG. Then, it computes inopportune inputs from the previous product and, proposes a default or a customized increase. It also computes the suspension traces and the determinization.

The second one allows to generate robustness test cases based on a robustness test purpose. The user defines both S_A and RTP files. *RTCG* checks the RTP (observability, determinism and accept states). Then, *RTCG* computes the synchronous product, the robustness test graph (RTG) and the reduced robustness test graph (RRTG). Finally it selects a robustness test case (RTC).

In the current *RTCG* version, robustness test purposes and specification files are written in the SDL or DOT format and robustness test cases are written using the TTCN-3 [6], XML or DOT formats.

4.2 Case study : TCP protocol

TCP [10] (Transmission Control Protocol) is a reliable protocol using a connection mechanism. Nowadays, it is largely used on the Internet. In case of TCP communication, a *client* requests a connection (active connection) by sending a synchronization message. If the server accepts, it replies another synchronization (passive connection); then the connection is established and the entities can exchange data. In our study, we focus on the robustness of connection phases. The nominal specification of such phases is provided in [10] written in the Finite State Machine format. The equivalent SDL translation is given below :

```
PROCESS TCP-Specification (1,1);
START NEXTSTATE Closed;
STATE CLOSED;
INPUT Passive-OPEN;
OUTPUT create-TCB;
NEXTSTATE LISTEN;
INPUT Active-OPEN;
OUTPUT create-TCB;
NEXTSTATE SYN-SENT;
STATE LISTEN;
INPUT SYN;
OUTPUT SYN,ACK;
NEXTSTATE SYN-RCVD;
INPUT SEND;
OUTPUT SYN;
NEXTSTATE SYN-SENT;
INPUT Close;
OUTPUT delete-TCB;
NEXTSTATE CLOSED;
STATE SYN-SENT;
INPUT SYN,ACK;
OUTPUT ACK;
NEXTSTATE ESTABLISHED;
INPUT SYN;
OUTPUT ACK;
NEXTSTATE SYN-RCVD;
STATE SYN-RCVD;
INPUT ACK-of-SYN;
NEXTSTATE ESTABLISHED;
INPUT Close;
OUTPUT Fin;
NEXTSTATE FIN-WAIT-1;
STATE ESTABLISHED;
INPUT Fin;
OUTPUT ACK;
NEXTSTATE CLOSE-WAIT;
INPUT Close;
OUTPUT Fin;
NEXTSTATE FIN-WAIT-1;
INPUT ACK-of-Fin;
STATE FIN-WAIT-1.
INPUT ACK-of-Fin;
NEXTSTATE FIN-WAIT-2;
INPUT Fin;
OUTPUT ACK;
NEXTSTATE CLOSING;
STATE CLOSE-WAIT;
INPUT Close;
OUTPUT Fin;
NEXTSTATE LAST-ACK;
STATE FIN-WAIT-2;
INPUT Fin;
OUTPUT ACK;
NEXTSTATE TIME-WAIT;
STATE CLOSING;
INPUT ACK-of-Fin;
NEXTSTATE TIME-WAIT;
STATE LAST-ACK;
INPUT ACK-of-Fin;
NEXTSTATE CLOSED;
STATE TIME-WAIT;
INPUT Timeout=2MSL;
OUTPUT delete TCB;
NEXTSTATE CLOSED;
END PROCESS TCP-Specification;
```

In order to apply our method, we consider the following hazards, used to increase the nominal specification : (1) Invalid inputs : messages with detected faults (e.g. bad sequence number). (2) Inopportune inputs : specified but not expected message (e.g. an acknowledgement (ACK) in an ESTABLISHED state) (3) Acceptable outputs : emitting a reset (RST) message after reception of an inopportune message. Such hazards are considered in most implementations of TCP protocol.

As an example, we consider the following test purpose :
RTP1: Opening a passive connection in presence of hazards.
The obtained robustness test case in the TTCN-3 format is :

```
tescase TCP-Tester() runs on TCP-IUT { timer ReponseTimer := 100E-3;
  Tester.send(close);
  Tester.send(passive-open);
  Tester.send(ACK-of-FIN-failed);
  Tester.send(SEND);
  Tester.send(SYN,ACK);
  ReponseTimer.start
  alt
    { [] ReponseTimer.timeout
         {  setverdict(fail);
            stop
         }
      [] Tester.receive(SYN);
         {  setverdict(pass);
            ReponseTimer.stop
          Tester.send(SYN);
          Tester.send(SYN,ACK);
          Tester.send(SYN,ACK);
          ReponseTimer.start
          alt
          {  [] ReponseTimer.timeout
                {  setverdict(fail);
                  stop
                }
             [] Tester.receive(ACK-of-SYN);
                {  setverdict(pass);
                  ReponseTimer.stop
                }
             [else] { setverdict(fail);
                      stop
                    }
           }
          }
      [else] { setverdict(fail);
               stop
             }
    }
control
  {
     execute (TCP-Tester());
  }
}
```

5. RELATED TOOLS

We have mentioned that the majority of the existing tools based on formal methods are focusing on conformance testing (e.g. TorX [14], TGV [8]). In robustness testing, some tools have been developed. The PROTOS project [11] simulates abnormal inputs with a high level of abstraction. *Crashme* [2] is also a testing tool of robustness filling an array with random data and trying to execute it as if it were code. STRESS [5] operates on GFSM model and synthesizes a set of test scenarii, protocol events and relation between topology delays and timer values that stress the protocol according to the evaluation criteria. The FIAT tool ([1]) modifies a processus binary image in memory, whereas the BALLISTA [4] tool works on data unexpected modifications.

6. CONCLUDING REMARKS

This paper presented a framework and a tool permitting to generate robustness test cases for communicating software. The proposed approach consists of two phases : the first one deals with the construction of an increased specification. The second phase deals with robustness test cases generation. The tool permits to implement the approach described above using SDL specification, and to generate TTCN-3 test cases. We also proposed a case study on the TCP protocol focusing on the robustness of connection phases.

As a future work, we intend to focus on unrepresentable hazards, and on timing constraints.

7. REFERENCES

[1] J.-H. Barton, E.-W. Czeck, Z.-Z. Segall, and D.-P. Siewiorek. Fault injection experiments using FIAT. *IEEE Trans. Comput.*, 39(4):575–582, 1990.

[2] G.-J. Carrette. CRASHME: Random input testing. http://people.delphiforums.com/gjc/crashme.html, 1996.

[3] R. CASTANET and H. WAESELYNK. Techniques avancées de test de systèmes complexes: Test de robustesse. Technical report, Action spécifique 23 du CNRS, 11 2003.

[4] J. DeVale, P. Koopman, and D. Guttendorf. The ballista software robustness testing service. In *Testing Computer Software Conference (TCSC99)*, June 1999.

[5] D. Estrin, S. Gupta, and A. Helmy. STRESS : Systematic testing of robustness by evaluation of synthesized scenarios. http://netweb.usc.edu/stress/, 1998.

[6] IEEE. *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN)*, number 9646 in 2, 2003.

[7] ITU-T. Specification and description language (sdl). ITU-T Recommendation no Z.105, International Telecommunication Union. Genève, 1999.

[8] T. Jéron, C. Jard, C. Viho, B. Caillaud, H. Kahlouche, P. Morel, J.-C. Fernandez, A. Kerbrat, and M. Bozga. Génération automatique de tests pour les protocoles: l'exemple de l'approche formelle de TGV. *Revue de l'Electricité et de l'Electronique (REE)*, 3, March 1999.

[9] I. S. G. of Software Engineering Terminology 610.12-1990. Customer and terminology standards. *In IEEE Standards Software Engineering, IEEE Press*, 1, 1999.

[10] J. Postel. Transmission control protocol. IETF, RFC793, September 1981.

[11] J. Röning, M. Laakso, and A. Takanen. PROTOS - systematic approach to eliminate software vulnerabilities. http://www.ee.oulu.fi/research/ouspg, May 2002. 2002.

[12] F. Saad-khorchef, I. Berrada, A. Rollet, and R. Castanet. Automated robustness testing for reactive systems : Application to communicating protocols. In *6th International Workshop on Innovative Internet Community Systems (I2CS 2006), Neuchâtel, Switzerland*, June 26-28 2006.

[13] J. Tretmans. Test generation with inputs, outputs, and quiescence. In T. Margaria and B. Steffen, editors, *Second Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, 1996.

[14] J. Tretmans and E. Brinksma. TorX: Automated model based testing. In *First European Conference on Model-Driven Software Engineering*, 2003.

[15] I.-T. R. Z.140-142. The testing and test control notation", version 3 (ttcn-3), rec. z.140: Ttcn-3 core language, rec. z.141: Tabular presentation format for ttcn-3 (tft), rec. z.142: Graphical presentation format for ttcn-3 (gft). ITU-T, Geneva (Switzerland), 2002.