Computer and Information Science; Vol. 12, No. 4; 2019
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Model Checking of WebRTC Peer to Peer System

Asma El HAMZAOUI[1], Hicham BENSAID [1,2], Abdeslam EN-NOUAARY[1]

[1] National Institute of Posts and Telecommunications, Smart, Embedded, Enterprise, and Distributed Systems, (SEEDS) Team, Allal Al Fassi Avenue, Rabat, Morocco

[2] labmia, faculté des sciences Mohammed V University Rabat, Morocoo

Correspondence: Asma El HAMZAOUI, National Institute of Posts and Telecommunications, Rabat, Morocoo.

Received: September 10, 2019 Accepted: October 9, 2019 Online Published: October 22, 2019

doi:10.5539/cis.v12n4p56 URL: https://doi.org/10.5539/cis.v12n4p56

Abstract

The establishment of the multimedia session is crucial in the WebRTC architecture before media and data transmission. The preliminary bi-directional flow provides the network with all the information needed in order to control and manage the communication between end-users. This control includes the setup, management, and teardown of a session and the definition, and the modification of multiple features that will be enabled in the ongoing session. This is performed by a mechanism named Signaling. In this work, we will use the formal verification to increase confidence in our SDL model by checking the consistency and reliability of the WebRTC Peer to Peer system. The verification and validation are proved the most efficient tools to avoid errors and defects in the concurrent system designs. Indeed, by using model-checking techniques we will verify if the WebRTC system adheres to standards if it performs the selected functions in the correct manner. To achieve that, we will first translate the SDL model to an intermediate format IF that will be retranslated to a Promela Model. Second, using the SPIN model checker, we will verify the general correctness of the model before checking if the desired properties are satisfied using the Linear Temporal Logic (LTL).

Keywords: WebRTC, Promela generation, formal modeling, V&V, SPIN, LTL

1. Introduction

WebRTC is the evolution of VoIP to the browser world. Indeed, WebRTC makes it feasible for developers to create a full soft client for audio and video communication on a browser, or to include VoIP into their Web-based applications. This approach revolutionizes the field of collaboration as soon as web multimedia applications using WebRTC can be created without any needs for plugins or installations.

By the standardization effort, and via standard HTML5 tags and JavaScript APIs (Application Programming Interfaces), WebRTC aims to enable RTC technology in all existing web applications and give developers the opportunity to innovate rich applications in their products using a standardized and interoperable technology.

In order to establish a WebRTC connection, clients need to check the capabilities of the two entities before proceeding to send the media. The exchange of session descriptions, which contain details in the form of data that shall be transmitted, and the way it will be transported is named signaling. In fact, signaling in WebRTC is decided to remain completely abstract by the standardizing bodies (W3C and IETF) (Alan B. Johnston and Daniel C. Burnett, 2013), which means that there is no specification that defines how it will be performed. Consequently, even if the Javascript application is allowed to control the signaling plane of the multimedia session through the interface specified in RTCPeerConnection API (Alvestrand, 2017), the technologies involved in the implementation of signaling are widely different.

Currently, there are several browser implementations of the WebRTC signaling specification, either for P2P or for multiparty conferencing (Appear.in, 2015; Meet.jit.si, 2017; Talky.io, 2013). However, those implementations suffer from a number of limitations. They are currently in alpha or beta status and as a result, have a number of bugs and may terminate unexpectedly. In fact, the API specification realized by W3C and the protocols defined by the IETF for the WebRTC architectures does not have a formal model and haven't been verified by formal analysis technics. This leaves room for ambiguities (lack of global comprehension of the system, deviations between the specification and its execution…), and misinterpretations of several concepts (Distribution of servers, and NAT (Network Address Translation) and firewall traversal issues...), which limits the reliability of WebRTC technology.

Indeed, tests and simulations are not enough in the context of critical applications that require a very high level of confidence. It must be able to provide formal evidence of specifications respect. To our knowledge, this aspect is very little studied for WebRTC.

In previous work (Asma El Hamzaoui, Bensaid, & En-nouaary, 2016), we presented formal modeling and validation for WebRTC signaling using SDL. The formal model described the structure of the system ~~of~~ in terms of blocks and processes, the data exchanged within the system and between the system and its environment, and the behavior of each process within the system, especially in terms of offered services according to the W3C specifications. The resulting SDL model was validated using the reachability analysis technique (Asma El Hamzaoui, En-nouaary, & Bensaid, 2018) with the support of the IBM Rational TAU suite (Corporation, n.d.). The validation consists of a demonstration that the system will meet its specification and that each of the blocks is correctly specified.

The next step is verification. It's a process used to determine if the system is consistent, uses reliable techniques, and performs the selected functions in the correct manner. However, the SDL language does not allow such verification. In this work, we intend to use the powerful SPIN Model Checker tool to realize the Model Checking step. The system requirements are expressed as logical temporal properties LTL to be verified. To achieve that, we must first translate the SDL model into the input language for SPIN that is Promela. This is not an easy task if we execute the translation directly.

In this paper, we use an intermediate format IF as an intermediate language to generate the Promela model from the SDL one. This choice is motivated by the simplicity of this method even if it is not automatic. After the translation of the SDL Model into Promela, the V&V (Verification and Validation) step deals with the satisfaction of the finished product, the system, with the intended behavior, represented as LTL properties.

The remainder of this paper is organized as follows. The next section describes the background of the work. It gives a summary of our specification of WebRTC P2P, then it presents the related work to Promela model translation from SDL. Section 3 details the two steps of our approach of the translation. The first step introduces the translation of the SDL model into the intermediate format IF. Some structural, behavioral and data aspects are explained. The second one highlights the if2pml tool used to generate a Promela model. Section 4 presents the model checking of our generated model. It describes some LTL properties to be checked and the result of this verification. The last section concludes the paper and presents future work.

2. Specification of WebRTC P2P Network

RTC has a characteristic that is always common in all technologies; there must be signaling or agreement between the two entities, either with the central node or with the other user. The purpose is to check the capabilities of the two entities before proceeding to send the media. For instance, the signaling channel is used to negotiate the codec agreement and NAT traversal methods that are used at the same time as all the multiple features that will be enabled in the new session.

Once signaling is done data starts to flow to the receiver, this stream may include media (audio or video) and different types of data (e.g binary, text, etc.). Conversely, NAT and firewall issues are factors that may cause session establishment to deny between peers. In fact, the Internet addressing system is still using IPv4 (Internet Protocol version 4) and because of that, most of our devices are behind one or more layers of NAT. On one hand, NAT is a mechanism that allows devices located on a private network to communicate transparently with devices located on an external network, such as the internet. It does so by modifying packets traveling between the networks, replacing the private source and destination IP addresses and ports with public ones where necessary, while maintaining an internal lookup table of these mappings, (P. Srisuresh, 1999). On the other hand, firewalls serve to filter packets received according to a predefined policy.

Actually, NAT and Firewalls reinforce the security of private networks by controlling the traffic sent or received from other communicating parties. The use of those mechanisms is very common on the internet. Thus, these issues arise as a crucial point in the WebRTC standard. In particular, peer-to-peer setting up a communication path through restrictive environments is difficult, and can even be impossible. Therefore, the WebRTC standard involves NAT traversal utilities while exchanging session descriptions to ensure the establishment of reliable connections. Those aspects are considered in our SDL model.

The SDL model of this system, cited in a previous contribution (Asma El Hamzaoui et al., 2016), consists of three interacting blocks as depicted in Figure 1.

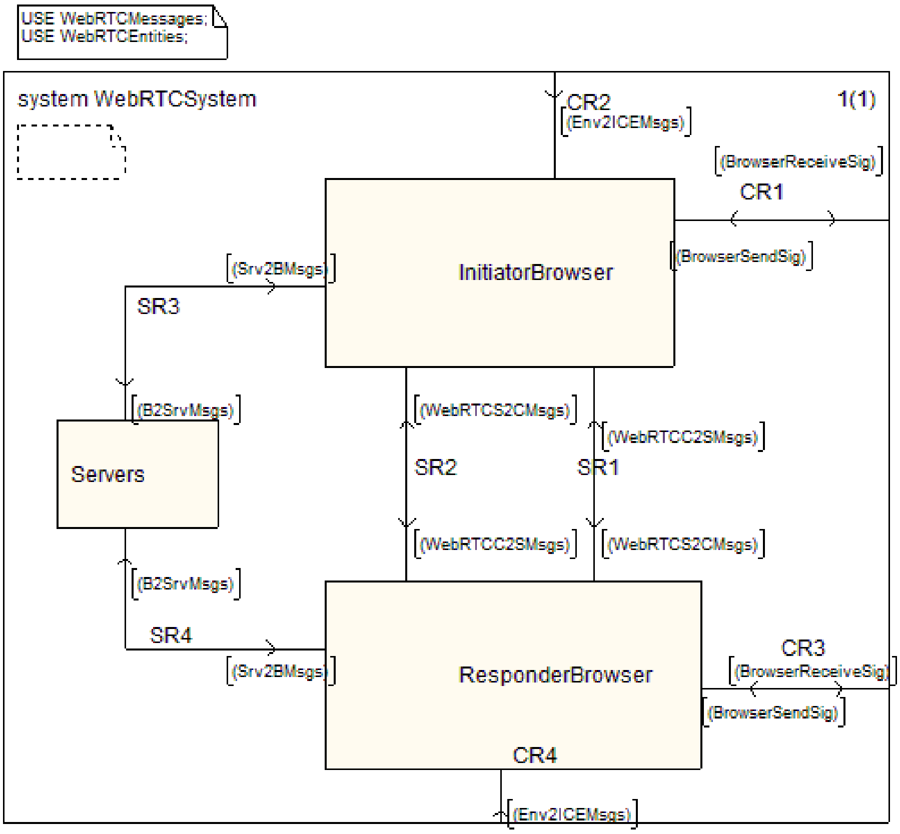

Figure 1. System view of WebRTC SDL model

The WebRTC peers interact with a block server including STUN server and TURN server and with the environment. The handling of environment signals sent to and received from the browser peer is done by those signallists: (BrowserSendSig) and (BrowserReceiveSig).

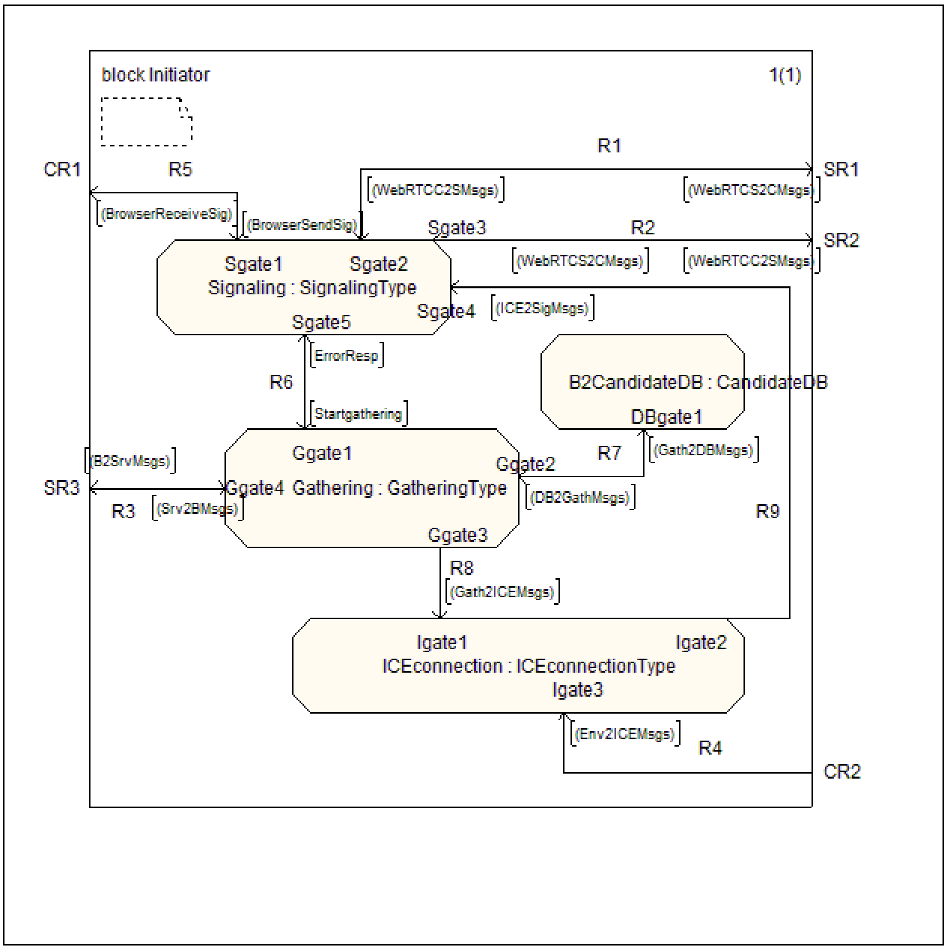

Figure 2. The block 'Initiator'

It represents the control panel signals and the audible tones by a human WebRTC user.

The behavior of a Browser Peer is captured in four collaborating process types: “Signaling”, “Gathering”, “ICE connection” and “Candidate DataBase”, see Figure 2. Each process represents a transition of states that describe the service. In addition, the process must have a set of temporary and permanent variables for its operations. For example, the permanent variable like ‘LocalStreamSet’ stores the current sent streams values of the call session and ‘RemoteStreamSet’ stores the current received streams values. The temporary variables store the consumed message values for further processing.

The validation of the resulting model was done using the reachability analysis technique (A. El Hamzaoui, En-Nouaary, & Bensaid, 2018), with the support of the IBM Rational TAU suite.

However, the verification of the model behavior has not been done yet because the SDL language does not support such verification. Thus, we plan in this paper to accomplish the system behavior verification by model checking. To that aim, we need to examine the model checker tools.

Many researchers have used model checking to deal with distributed systems, especially to verify complex communication protocols. In addition, many automatic tools for model checking have been developed for a long time. As examples of earlier model checkers, there is SPIN, Symbolic Model Verifier (SMV/NuSMV), UPPAAL, KRONOS, HYTECH and so on.

Authors in (Mazzanti & Ferrari, 2018) exposed the similarities and differences between ten formal model checkers, namely UMC, SPIN, NuSMV, mCRL2, CPN Tools, FDR4, CADP, TLA+, UPPAAL and ProB through a case study. It was the verification of an algorithm for Automatic Train Supervision. It presents the impact of these languages on the model.

In fact, making a choice which model checker to use is not an obvious task. Many considerations must be taken like the system architecture, the type of properties to be verified and so on. In this study, we have an SDL model as an input. Therefore, we need to decide on which verification language we will translate the model. In theory, there are many possibilities. Nevertheless, in practice, the SDL/Promela is a mature and commonly used pair. Another commonly utilized model checker is UPPAAL (Guasch, 2013). However, the translation from SDL to the intermediate language SDLxta and afterward to xta language used in UPPAAL tool is an intricate procedure (Frappier, Fraikin, Chossart, Chane-Yack-Fa, & Ouenzar, 2010). Besides, in the SMV language that is used in NuSMV, all the assignments, array indexes or parameters must be constant. This is not suitable for the data in our model. Moreover, taking the comparison in (Mazzanti & Ferrari, 2018) into consideration, SPIN has a reasonable range of time for verification. Consequently, since Promela is the input verification language of SPIN (Bošnački, Dams, Holenderski, & Sidorova, 2000), we selected SPIN model checker in this work.

The first step was to transform our SDL model into Promela. This transformation is not a simple task. Numerous techniques found in the literature have tackled this challenge. The most approved technique is the utilization of an intermediate format IF (Bozga, Graf, Mounier, & Sifakis, 1999). Firstly, the model is converted into IF using sdl2if tool, at that point the Promela model is created from the subsequent model utilizing if2pml tool. This process does not support some important SDL properties like the Timer and the Save operator. To address the issue, researchers proposed in (Prigent, Cassez, Dhaussy, & Roux, 2002) an extension of if2pml tool to translate the save operator from IF to Promela. The idea consists of adding local queues to which the saved signals are sent. However, this tool is not available for IBM Rational SDL Suite. It works only with ObjectGeode API, which is an obsolete product.

Furthermore, an interesting contribution was presented in (Vlaovic, Vreze, & Brezocnik, 2017; Vlaovič, Vreže, Brezočnik, & Kapus, 2007). Authors concretize the idea of directly generating a Promela model from the SDL specification by the implementation of sdl2pml (Vreže, Vlaovič, & Brezočnik, 2009) that is an automated generation tool. To the best of our knowledge, this tool is the only one that supports the translation of all SDL features. However, this tool is not as accessible as a free or commercial product.

In this work, the methodology we adopt is based on the intermediate format IF. This format was designed especially for the representation of timed asynchronous systems and provides a common model between various languages that have different description styles (e.g. SDL, LOTOS…).

We translated manually the SDL model into IF language and then the IF model is pushed through the pipe of if2pml translator to obtain a DT Promela script that serves as input to Spin or DT Spin. The result of a negative verification experiment (For example an erroneous trace) has to be checked manually against the SDL specification. The process is depicted in Figure 3.

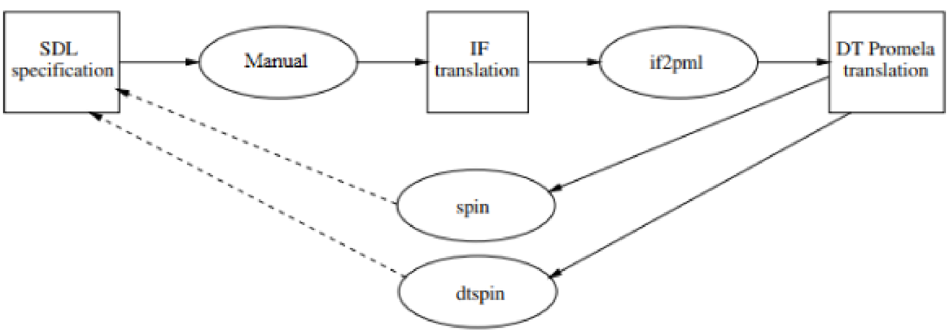

Figure 3. The Methodology used in the verification

In the next section, we will introduce our work on the model following this process.

3. Generation of Promela Model

To achieve the first step, we must do some modifications to the SDL model to be compatible with SPIN after translation. In fact, asynchronous input queues cause state explosion. The solution found for this problem is to embed the environment into the system. The next section presents the required modifications to create a closed model from the model described above and the translation process.

3.1 A Closed Model for HAN

In order to obtain a closed system, we supplemented the specification with a model of its environment. Figure 4 shows the architecture after this modification. The block environment as illustrated in figure 5 consists of two processes: 'Env2ICE' and 'CallProgress', which represent the interaction of the human and the ICE utility with the system. Figure 6 represents the process view of one of them. We can notice that in the process decisions are left nondeterministic using the reserved word "Any" and additionally, we didn't restrain the "environment" with the order in which the signals were sent. We note that this is only one possible example of the block "environment".

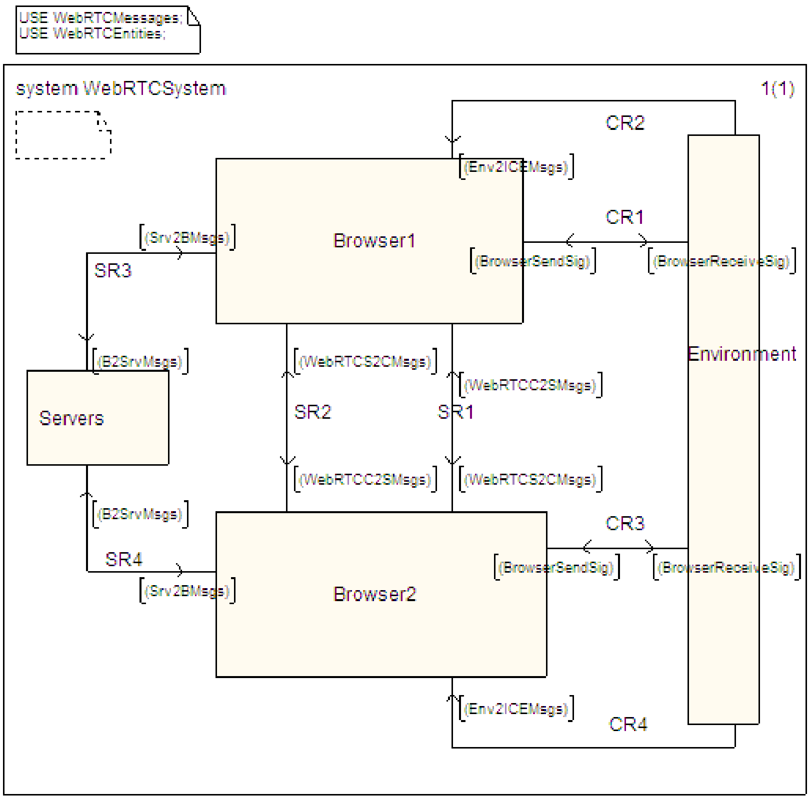

Figure 4. SDL model of WebRTC with its environment

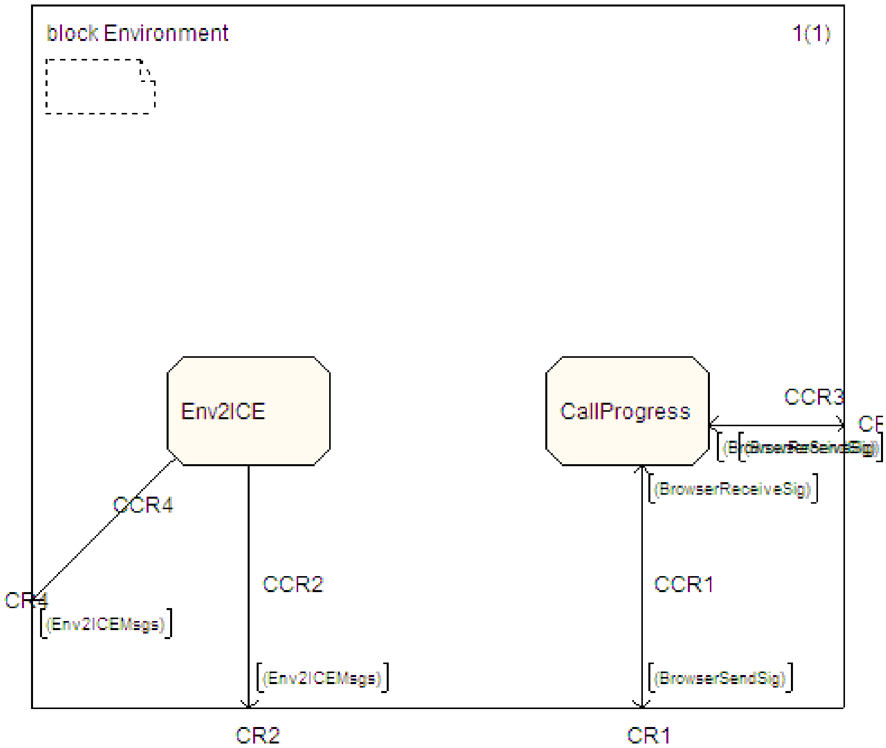

Figure 5. Block Environment

The second adjustment that we brought to the model is the elimination of the need for user's intervention during the verification step. Therefore, the user's arbitrary decision statements, specified as an informal text, were replaced with nondeterministic decision statements. This can be accomplished by using the keyword ANY. No other changes in the first WebRTC system specification were made. Furthermore, this new model is also validated using the three techniques described in (Asma El Hamzaoui et al., 2018).

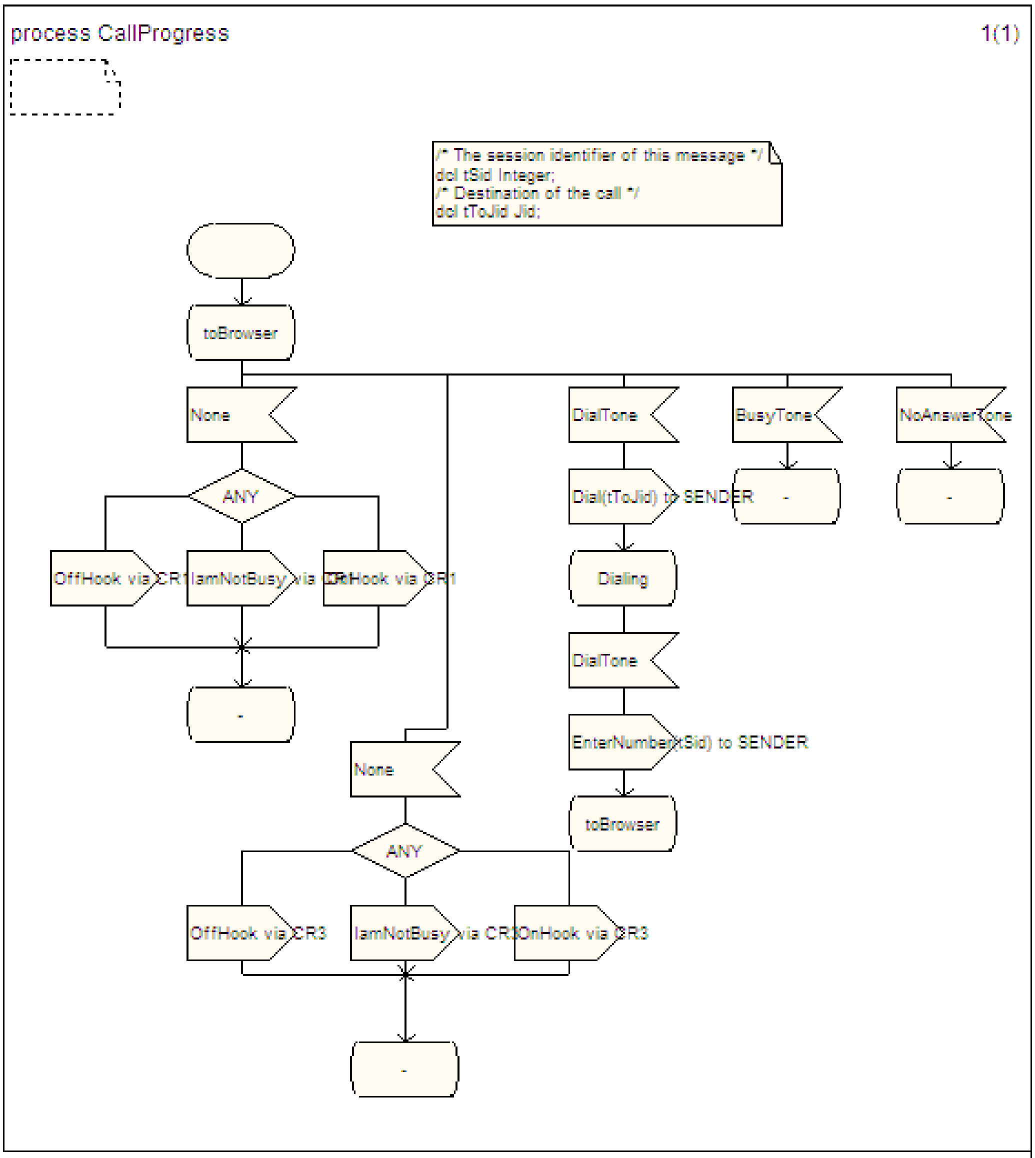

Figure 6. Process CallProgress

In the next section, we will introduce the manual translation from SDL to IF.

3.2 SDL to IF Translation

We mentioned previously that we translated first our model into an intermediate format IF. There were some limits in this conversion due to the static and simple nature of IF language. In this subsection, we present some SDL feature translation aspects concerning the structure, the behavior, and the data.

First, in the structure, SDL and IF are different. In fact, SDL provides object-oriented concepts like inheritance and the SDL model can contain blocks, processes, and procedures. Otherwise, the IF models are composed only of processes in one level which means that IF models are flat. This difference has not been a problem since, at execution time, even an SDL model is composed only of processes that react with each other. Consequently, we did not translate the structure of SDL into IF.

Furthermore, there are two types of states in IF: "stable" and "nostable". All SDL states are by default transformed into "stable" states. The "nostable" states are used to divide a long transition into small transitions. The process did not stop at those states. While blocking the other processes, it continues the execution of the transitions until it reaches a stable state.

All triggers and actions are defined in an IF transition (the minimal path between two IF states) and are executed in the same order. The transition is by default of type "eager" because it has a higher priority than time progress. In addition, the SDL input signals are directly represented with an asynchronous IF inputs. However, unlike SDL, the remote imported/exported variables declared inside processes. They are defined only once at the IF system level to be used by concerned processes. In addition, we created only one instance of some processes that could be created dynamically because IF does not support the dynamic creation of process, (Bozga et al., 1999).

On the other hand, all predefined data types used in our SDL model have their equivalent in IF except the “Charstring” type. To solve this issue, we changed the variable declared in SDL with “Charstring” into “Character”. In fact, “Character“is translated into an IF user-defined type range 1:: 256.

For example, the declaration: Sessiontype Charstring := 'SessionInitiate'; Was replaced in the model by Sessiontype Character: = 'I';

In addition to that, the abstract data types are converted into an IF ADT with the same signature. Table 1 presents an illustration of the Data type translation.

Table 1. Data type translation illustration

SDL Representation	IF Representation
/*Type for a Jingle calling browser*/ /* Example : asma@inpt.ac/office */ newtype Jid struct IdBrow Integer; Uid Charstring; Domain Charstring; Ressource Charstring; endnewtype;	/*Type for a Jingle calling browser*/ Jid = **record** IdBrow : int; Uid : Character; Domain : Character; Ressource : Character; **end**
/*Type to index the array of streamSet*/ syntype Index = Natural Constants 1:10 endsyntype; /* Type to organize candidates */ /* of a same streamSet by their Id*/ newtype streamset Array(Index, Integer) endnewtype;	/* Type to organize candidates */ /* of a same streamSet by their Id*/ streamset = **array**[0..10] of int;

Another difference between SDL and IF: the “Sender” variable is defined implicitly in SDL. In IF, we define explicitly an additional variable of type PID that contains always the PID of its sender. In table 2, we can see that for signal “AllocateSucessResp” the first parameter is "TURNServer1" which represents the PID of the process. In fact, the process name in IF is its PID.

3.3 IF to Promela Translation

In the next step, we used the if2pml tool to generate automatically a Promela model from the IF script described earlier. Although, we had to fix some bugs.

First, unlike SDL and IF, we could not send global variables through signals. To solve this problem we use local variables with the value of the global variables in the signals. For example, in the process ‘STUNServer1’, we add the variable rCand1 of type Candidate and we use it to send the value of the global variable rCandidate1 via the output ‘STUNSuccessResp’.

```
from Idle : eager
input STUNReq(Sender,tJid, tSrv) from qSTUNServer1
do rCand1 := rCandidate1
,output STUNSuccessResp(STUNServer1, tJid, tSrv,  rCand1 ) to qB1Gathering
to Idle;
```

Another issue, the if2pml tool do a mistake in the translation of type 'Array' from IF to Promela. For example, in IF we define a type "Streamset = array[0..10] of int;". However, in the Promela file, it is transformed into "int Streamset[10];" which is a variable, not a type. Consequently, in Promela we can't define variables of type StreamSet, because this type does not exist.

As a solution to this error, we decide to change the Promela file manually in two steps:

- Delete the line int Streamset[10];
- For each variable (Global or local) declared with the 'StreamSet' type, we must change it with an array of 'int'.as an illustration, 'Streamset rStreamset1;' must be changed by: 'int rStreamset1[10];'

We run the syntax check until no error is detected.

Another aspect missed in Promela is the notion of time. In fact, to be able to describe real-time proprieties, we worked with the new version DTPromela (Bošnački et al., 2000). By default, the if2pml tool translates the model into this language. The new definition of a timer is added to the system as a Promela macro, contained within a special header file "dtimesdl.h".

This file must be included at the beginning of the DTPromela model. The obtained model can be verified with an extended version of SPIN model-checker named DTSpin.

To illustrate the translation from SDL to Promela, table 2 gives an example of the process type "TURNServerType". We defined an IF process with the name "TURNServer1" as the original process. It is associated with a default input queue "qTurnServer1". As mentioned before, we defined an additional variable of type PID to represent the "Sender" variable explicitly. We have four variables declaration. When the process is in the state start, it can receive the "AllocateReq" signal. Then it checks a boolean parameter that indicates the existence or not of candidates to allocate then in the 'True' case it sends a signal 'AllocateSuccessResp', or in the 'False' case it sends a signal 'ErrorResp'. After that, it returns to the IDLE state again.

Table 2. SDL to Promela translation example

<table>
<tr><td>SDL representation</td><td><pre>process type TURNServerType;
gate TURNGate1 out with (TURNMsgs); in with AllocateReq;
gate TURNGate2 out with (TURNMsgs); in with AllocateReq;
dcl
   tJid Jid;
dcl
   tSrv Srv;
dcl
   rCand3 Candidate;
dcl
  exist Boolean;
start;
      nextstate Idle;
state Idle;
   input AllocateReq(tJid, tSrv);
      task 'Ip&PortAllocation';
      decision exist;
      (True) :
         output AllocateSuccessResp(tJid, tSrv, rCand3) to sender;
         nextstate Idle;
      (False) :
         output ErrorResp to sender;
         nextstate Idle;
      enddecision;
endstate;
endprocess type TURNServerType;</pre></td></tr>
<tr><td>IF representation</td><td>/* process TURNServer1 */</td></tr>
</table>

	process TURNServer1 : buffer qTurnServer1; var sender : pid; tJid : Jid; tSrv : Srv; rCand3 : Candidate; exist(false) : bool; state start : init; Idle; q11: nostable; transition from start : eager to Idle; from Idle : eager input AllocateReq(Sender,tJid, tSrv) from qTURNServer1 to q11; from q11 : eager if exist = true do output AllocateSuccessResp(TURNServer1, tJid, tSrv, rCand3) to qB1Gathering to Idle; from q11 : eager if exist = false do output ErrorResp(TURNServer1) to qB1Gathering to Idle; /* END process TURNServer1 */

PROMELA representation	proctype TURNServer1()
	{
	byte
	sender;
	Jid
	tJid;
	Srv
	tSrv;
	Candidate
	rCand3;
	exist
	bool;
	start:
	atomic{
	if
	:: goto Idle;
	fi;
	}
	Idle:
	atomic{
	if
	:: qTURNServer1?AllocateReq(Sender,tJid,tSrv)->
	goto q11;
	fi;
	}
	
	atomic{
	skip;
	q11:
	if
	exist :: true->
	qB1Gathering!AllocateSuccessResp(_pid, tJid, tSrv, rCand3)->
	goto Idle;
	exist :: false->
	qB1Gathering!ErrorResp(_pid)->
	goto Idle;
	fi;
	}
	};

In the next section, we intend to verify the consistency of the SDL model with the generated Promela model.

Moreover, we write some Linear Temporal Logic (LTL) properties to define the system requirement and verify them using SPIN.

4. Model Checking Verification

Confidence in protocol correctness can be increased in different ways. Testing is one method that involves building a prototype and observing it, or observing the behavior of a real system.

The main disadvantage of testing is that it can be used to show errors, but not to prove correctness. Simulation is a method based on the construction of an executable model of the system and its observation. The simulation requires the generation of test cases. Although, simulation and testing can provide interesting results in the performance analysis of a system. In the case of critical properties, simulation is not believed to provide sufficient confidence.

The other point of view to look at protocol correctness is the automated formal techniques. Model checking is a formal analysis method that verifies iteratively the system properties based on a model of the system. It performs exhaustive simulation on the system's state-space and produces verification results. Either the latter terminates successfully or it produces a counterexample to illustrate the failure case. The counterexample can be used to check the system errors and to take corrective actions. As a whole, the formal analysis increases our confidence that: (1) there are no inherent design flaws; and (2) that the acquired model satisfies enough system properties expressed informally in the system requirements document. This process is described in the next two subsections.

4.1 General Verification

After the generation of our Promela model, we realize the verification using XSpin (Holzmann, 2003), which is a graphical interface that provides displays of, results (message flows, data values and so on) while executing SPIN command in the background.

All verifications were performed on a Solaris (200 MHz) with 5 Go of memory. Spin/XSpin version 4.1.1 and Tcl/Tk version 8.5/8.5 were used in all cases. The environment of verification is depicted as figure7.

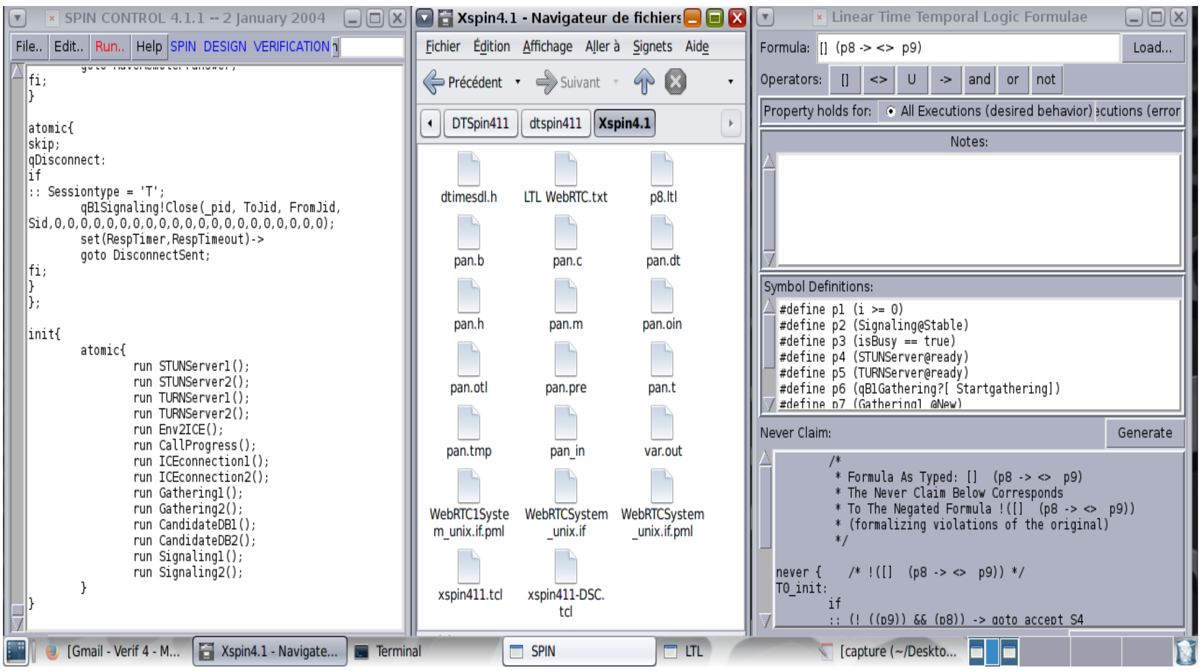

Figure 7. The environment of verification

The first step is to fix the syntax errors found. Second, we run a random simulation to correct if some evident abnormalities appear. This simulation is called 'Random' because there is no human interaction to resolve non-deterministic decisions. The interactive simulation was not possible due to the model's big number of states.

We run the simulation many times to detect obvious faults in the design. Each time, we vary the "Seed" value to get different results. Violations were detected in most cases within seconds, and generally under 5 min.

The next step was to generate an optimized verifier in order to execute exploration in some of the three main search modes: exhaustive verification, bitstate, or hash-compact approximation, (Bozga et al., 1999; Holzmann, 1997, 2003; Ober, Graf, & Ober, 2004; Prigent et al., 2002; Vlaovic et al., 2017; Vlaovič et al., 2007; Vreže et al., 2009).

Although the first search mode is the strongest one, we could not run the exhaustive exploration of our model due to its size versus the available memory that we had. In fact, we combine the bitstate search mode with the Partial Order Reduction (POR) (Baelde, Delaune, & Hirschi, 2015) to have a manageable state space to verify. Indeed, POR is a popular state-space reduction technique that mitigates the effects of state space explosion that results from concurrency. In fact, the resulting size of executed actions can reach the size of the cartesian product of the state spaces of the individual processes. However, many transitions are local to a particular process and do not influence the other processes. In this situation, it is not necessary to verify all the orderings of these transitions. To sum up, using POR can enable to construct a reduced state graph. Moreover, no relevant information is missed by verifying only the reduced graph behaviors, (Baelde et al., 2015).

In this general verification, we could detect some violations that we corrected to get a verified model without any violations. An example of a problem detected in this step is ~~about~~ the Cancel Request messages. A non-progress cycle was detected by Spin that is caused by the inclusion of Cancel Request messages in the model. WebRTC peer could repeatedly send a session Request and Cancel Request messages infinitely often to the other peer. The detected problem highlight the importance of a careful design of “Cancel” functionality. In this work, the WebRTC model has been modified to become free of non-progress cycles. Instead of enabling ~~to~~ a peer to send any session request after a “Cancel” request, the other peer returns an error message indication informing that the request did not complete. This new behavior was verified successfully and was included in all runs for the LTL properties below.

In the next section, we explore some safety and liveness properties using LTL. By default, XSpin involves satisfying the never claims, where the aim is to prove that an undesirable condition or specification is never reached: “Safety conditions”, (Igor Konnov Marijana Lazic Helmut Veith, 2017). They are generally of the form ‘both processes will not be in a critical section simultaneously.’ However, the tool can check also liveness properties:” Under certain conditions, something will ultimately occur”. Examples of such properties include ‘any request will ultimately have a response’ or ‘the program will terminate.’ will be presented in the next section.

4.2 LTL Verification

After our verification work on the model presented in the previous section, we explore in this section some system requirements that are formalized in LTL and the Spin model checker is used to determine their validity.

In fact, the task of deriving LTL formulas from the informal requirements is a point of difficulty in the verification process. The correctness and the meaningfulness of the formula in the context of the verification influence deeply the results. Consequently, this automatic verification process depends significantly on the experience of the designer. Moreover, after ensuring that there are no deadlocks or unreachable instructions, the selection of properties that are important to address is a crucial point. We worked to elicit a representative subset of WebRTC signaling requirements. Thirteen temporal formulae are defined below. We formalized them to build LTL properties:

- p1 (i > 0)
- p2 (Signaling@Stable)
- p3 (isBusy == true)
- p4 (STUNServer@ready)
- p5 (TURNServer@ready)
- p6 (qB1Gathering?[Startgathering])
- p7 (Gathering1 @New)
- p8 (qB1ICEconnection?[Completegathering] || qB1ICEconnection?[Partialgathering])
- p9 (qB1Signaling?[CompleteICEResp])
- p10 (qTurnServer1?[AllocateReq A])
- p11 (qTurnServer1?[AllocateReq B])

- p12 (qB1Gathering?[AllocateSuccessResp A])
- p13 (qB2Gathering?[AllocateSuccessResp B])

Property 1: [] p1

The variable "i" refers to the number of candidates registered by a peer. For a given peer, it must have at least one candidate registered. We must ensure that this value is never negative.

This first property represents a safety property. It is an unwanted situation that should never occur. In this case, it is in the form of an unwanted negative value for the variable.

Property 2: [] <> p4

The STUNServer process waits for the participant peers to send STUN requests. After that, it responds by their public Ip address that will be registered as a candidate. While being in the responding state, other peers may be connected to the system at this time. Consequently, the process must not hang at the state processing for a long time. This property states that if p4 (i.e. STUNServer is at the state ready) become false at any time in a run, it is always guaranteed to become true again.

Property 3: [] <> p5

This property is the same as the previous one, but for the TURN Server. It ensures that TURN Server always eventually returns to the state ready.

The second and third properties are of type liveness. They represent the required behavior that indicates the positive functionality of our system.

Property 4: [] (p3 -> <> p2)

The connected peer to the WebRTC system may be disconnected or may become busy at any time. The availability and presence of the peer are represented by a global Boolean 'isBusy'. The latter can be modified by several procedures. We must guarantee that once the variable value becomes 'true', the process 'Signaling' stops all activities and returns to the 'Stable' state.

Property 5: [] (p6 -> <> p7)

The candidates gathering phase is common to both the caller and callee and can be triggered on the two endpoints. When the peer receives a 'Startgathering' signal from the queue qB1Gathering, the process Gathering1 asks both TURN and STUN servers and the local database. After running the algorithm, it must send the signal 'Completegathering' to the queue of ICEconnection process and goes to the initial state 'New'.

Property 6: [] (p8 -> <> p9)

Interactive Connectivity Establishment (ICE) is a standard algorithm that describes how to coordinate STUN and TURN protocols to make a connection between endpoints. The algorithm used in this standard, and described in RFC (Rosenberg, 2010), is composed of many timers eventually fired and of loops. We must guarantee that the ICEconnection process can reach a stable state at certain times. The last three properties are also liveness properties of a special type called response.

SPIN generates a never claim from each property to be verified. All the properties above are the desired ones. Consequently, they are negated. For instance, the last property [] (p8 -> <> p9) is negated and then converted to the never claim as follow:

```
never {     /* !([] (p8 -> <>  p9)) */
T0_init:
    if
    :: (! ((p9)) && (p8)) -> goto accept_S4
    :: (1) -> goto T0_init
    fi;
accept_S4:
    if
    :: (! ((p9))) -> goto accept_S4
    fi;
```

}

Property 7: [] ((p10∧p11∧<> p12) -> (⌉ p13∪ p12))

This property indicates that requests from peers should be processed by STUN and TURN servers in the same order that they were issued by each of them. We write this property for a TURN server. This requirement is not part of the WebRTC specifications, nevertheless, it represents a useful feature for ICE protocol. This requirement describes an ordering relationship between when multiple requests are issued and when they are processed. For instance, if a request from A and request from B have both been sent to a server, then request B should not be processed until request A has received a response first. To formulate this property we used the absence pattern with the between scope described in (Dwyer, Avrunin, & Corbett, 1998). The absence pattern constraints some states not to be reached in a given temporal context. In this case, the property specifies the absence of the behavior in which request B is processed between the moment when request A is sent and processed.

Safety properties (1,4,5,6) lasted between 40 min to 1 h for each property. Liveness verification on properties (2,3) lasted between 3 to 4 h for each property. All requirements were validated successfully with the exception of property 7, which failed to hold. During the verification of property 7, it was realized that the order preservation requirement was not met by the WebRTC model. By examining the message trail, it was discovered that, in the general case, it is not possible to guarantee that requests will be serviced in the order they were issued because there is no synchronization in the gathering process between the peers. Although the last requirement is not verified successfully, it serves to illustrate how Model Checking can be used to help the designer to gain a better understanding of the limitations of the model.

Finally, it should be emphasized that we do not claim to accomplish a complete verification or proof of the correctness of our Promela model. Indeed, the validation runs were only possible using non-exhaustive state exploration; hence, the exhaustive model checking could reveal execution scenarios that violate some of our properties. However, the methods we employed are certainly sufficient for increasing our confidence that there are no residual design flaws in our model, and that the model achieves the requirements of WebRTC signaling.

5. Conclusion

In this paper, we integrate the formal methodologies into the development process of the complex P2P WebRTC call processing software in order to increase the quality of the design. First, we decided to start by translating manually our SDL model into an intermediate format IF before executing an automatic mapping between IF and Promela language using if2pml tool. This semi-automatic method was judged as being a possible and efficient way to get the Promela model. Once the model was ready for the verification and validation step, we checked the correctness of our model against properties like deadlocks, infinite loops or exceeded queue lengths. After, we defined desired properties in LTL, to be verified using the SPIN model checker. A secondary goal of this article was to evaluate the suitability of the formal analysis techniques that we have chosen, namely the Promela language and the Spin model checker in this empirical study of practical network models.

In future work, we intend to propose a larger model for the multiparty WebRTC system and extract a Promela model to be verified. In addition, we plan to use an SDL to C compiler to generate the C code from this SDL specification for a successful implementation.

References

Alan, B. J., & Daniel, C. B. (2013). *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web* (2nd ed.).

Alvestrand, H. (2017). *Overview: Real Time Protocols for Browser-based Applications.*

Appear.in. (2015). *video collaboration with appear.in.* Retrieved from https://appear.in/

Baelde, D., Delaune, S., & Hirschi, L. (2015). Partial Order Reduction for Security Protocols. In *26th International Conference on Concurrency Theory (CONCUR 2015)* (Vol. 42, pp. 497-510).

Bošnački, D., Dams, D., Holenderski, L., & Sidorova, N. (2000). Model Checking SDL with Spin. In *Proceedings of TACAS'2000, vol. 1785* (pp. 363-377). https://doi.org/10.1007/3-540-46419-0_25

Bozga, M., Graf, S., Mounier, L., & Sifakis, J. (1999). IF: An Intermediate Representation for SDL and its applications. In *SDL Forum* (pp. 423-440). https://doi.org/10.1016/B978-044450228-5/50028-X

Corporation, I. B. M. (n.d.). *Manual Getting Started : IBM Rational SDL and TTCN Suite 6.3*.

Dwyer, M. B., Avrunin, S. G., & Corbett, J. C. (1998). Property Specification Patterns for Finite-State Verification. In *FMSP '98 Proceedings of the second workshop on Formal methods in software practice* (pp.

1-7). https://doi.org/10.1145/298595.298598

El Hamzaoui, A., En-Nouaary, A., & Bensaid, H. (2018). On Formal Modeling and Validation of Signaling Protocols for Web Real-Time Communications using SDL. In *Procedia Computer Science* (Vol. 130). https://doi.org/10.1016/j.procs.2018.04.105

El Hamzaoui, A., Bensaid, H., & En-nouaary, A. (2016). A formal model for WebRTC signaling using SDL. In D.-G. C. (eds) Abdulla P. (Ed.), *Networked Systems. NETYS 2016.* LNCS, vol 9944, pp. 202-208. Springer, Cham. https://doi.org/10.1007/978-3-319-46140-3_16

El Hamzaoui, A., En-nouaary, A., & Bensaid, H. (2018). On Formal Modeling and Validation of Signaling Protocols for Web Real-Time Communications using SDL. *The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)*, *00*, 1-8. https://doi.org/10.1016/j.procs.2018.04.105

Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R., & Ouenzar, M. (2010). Comparison of model checking tools for information systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6447 LNCS*, 581-596. https://doi.org/10.1007/978-3-642-16901-4_38

Guasch, P. F. I C. J. C. J. F. A. (2013). Formalizing geographical models using specification and description language: The wildfire example. In *Proceedings of the 2013 Winter Simulation Conference* (pp. 1961-1972).

Holzmann, G. J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, *23*(5), 279-295. https://doi.org/10.1109/32.588521

Holzmann, G. J. (2003). *Spin Model Checker, The: Primer and Reference Manual*. Addison Wesley.

Igor, K. M., Lazic, H., & Veith, J. W. (2017). A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages - POPL 2017*. https://doi.org/10.1145/3009837.3009860

Mazzanti, F., & Ferrari, A. (2018). Ten Diverse Formal Models for a CBTC Automatic Train Supervision System. *Electronic Proceedings in Theoretical Computer Science*, *268*, 104-149. https://doi.org/10.4204/EPTCS.268.4

Meet.jit.si. (2017). open source video conferencing solution with meet.jit.si. Retrieved from https://meet.jit.si/

Ober, I., Graf, S., & Ober, I. (2004). Validation of UML Models via a Mapping to Communicating Extended Timed Automata. In *Proceedings of SPIN'04 Workshop* (pp. 127-145). Barcelona, Spain: volume 2989 of LNCS. Springer. https://doi.org/10.1007/978-3-540-24732-6_9

P. Srisuresh, M. H. (1999). *IP Network Address Translator (NAT) Terminology and Considerations Specification. (RFC 2663)*. https://doi.org/10.17487/rfc2663

Prigent, A., Cassez, F., Dhaussy, P., & Roux, O. (2002). Extending the Translation from SDL to Promela. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software* (pp. 79-94). https://doi.org/10.1007/3-540-46017-9_8

Rosenberg, J. (2010). *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. https://doi.org/10.17487/rfc5245

Talky.io. (2013). *Truly simple video chat and screen sharing for groups with talky.io.* Retrieved from https://talky.io/

Vlaovic, B., Vreze, A., & Brezocnik, Z. (2017). Applying Automated Model Extraction for Simulation and Verification of Real-Life SDL Specification with Spin. *IEEE Access*, *5*(March), 5046-5058. https://doi.org/10.1109/ACCESS.2017.2685238

Vlaovič, B., Vreže, A., Brezočnik, Z., & Kapus, T. (2007). Automated generation of Promela model from SDL specification. *Computer Standards and Interfaces*, *29*(4), 449-461. https://doi.org/10.1016/j.csi.2006.10.001

Vreže, A., Vlaovič, B., & Brezočnik, Z. (2009). Sdl2pml - Tool for automated generation of Promela model from SDL specification. *Computer Standards and Interfaces*, *31*(4), 779-786. https://doi.org/10.1016/j.csi.2008.09.005

Copyrights