Model-based protocol log generation for testing a telecommunication test harness using CLP

Kenneth Balck
Ericsson AB
Linköping, Sweden
kenneth.balck@ericsson.com

Olga Grinchtein
Ericsson AB
Stockholm, Sweden
olga.grinchtein@ericsson.com

Justin Pearson
Department of Information Technology
Uppsala University, Sweden
justin.pearson@it.uu.se

Abstract—**Within telecommunications development it is vital to have frameworks and systems to replay complicated scenarios on equipment under test, often there are not enough available scenarios. In this paper we study the problem of testing a test harness, which replays scenarios and analyses protocol logs for the Public Warning System service, which is a part of the Long Term Evolution (LTE) 4G standard. Protocol logs are sequences of messages with timestamps; and are generated by different mobile network entities. In our case study we focus on user equipment protocol logs. In order to test the test harness we require that logs have both incorrect and correct behaviour. It is easy to collect logs from real system runs, but these logs do not show much variation in the behaviour of system under test. We present an approach where we use constraint logic programming (CLP) for both modelling and test generation, where each test case is a protocol log. In this case study, we uncovered previously unknown faults in the test harness.**

I. Introduction

A major problem with software testing is how to derive test cases. Model based testing [1] solves this, by deriving test cases from a model of the system under test. In this paper we study part of the protocol of the Long Term Evolution (LTE) 4G standard [2], [3] responsible for the broadcast of public warning messages [4]. The protocol includes a number of messages with complex timing requirements between them. The main novelty is that we use constraint programming [5] to directly model the protocol and to derive protocol logs that are used to test the test harness.

The system under test is a *test harness* for a base station, a mobile phone simulator, and a simulated network entity. The test harness is used to test system functionality automatically. The test harness communicates with each component and executes a script on each component that initialises parameters and requires that certain protocol messages are sent. This allows various communication scenarios to be tried between the components. The test harness is designed to capture and analyse certain log files in the simulated environment in order to automate manual testing of the telecom system.

As yet the actual test harness has not been extensively tested. Log files can be obtained by executing tests on real systems, but it is not easy to obtain all possible behaviours, and further obtaining log files of incorrect behaviours is nearly impossible. Our goal was not to test the whole of the test harness, but was to test if it was able to analyse log files correctly. Therefore, we needed log files that exhibit incorrect behaviour. With constraint programming [5] it was easy [6] to specify both correct and incorrect behaviours, and to generate logs, as well as to have control over what incorrect behaviours were allowed. Incorrect behaviours are generated by modifying (or mutating) the original *model* of the system.

The contributions of this paper include: A constraint programming model of part of a complex real-life telecom protocol; by using constraint programming, we are able produce logs that have correct behaviour and logs that have incorrect behaviour, where the use of constraint programming allowed fine grained control over the type of incorrect behaviour that we allowed; finally we uncovered faults in test harness by finding test cases when the test harness does not find an error when there is an error in log, or wrongly indicating an error when there is none.

II. Model Based Testing

The approach of model based testing [1] is to model the system under test, and to use the model to derive test cases. This has the advantage that often test cases can be derived automatically from the model. There are two aspects of model based testing: how to specify the model, and how to derive tests cases from models. Approaches to modelling include: automata or state based approaches [7]; mathematical based descriptions using Z [8] or the B-Method [9]; or models provided in an extended high-level programming language, such as the approach used in Spec Explorer [10].

The use of constraints in automatic test case generation has a long history. In [11] constraint solving is used to derive test cases that distinguish between a piece of code and a mutation of that piece of code. More recently there has been a lot of work on using recent advances in constraint programming applied to white box testing of Java or C [12], [13].

III. Constraint Programming

Constraint Programming [5] is a framework for modelling and solving combinatorial problems. A constraint problem is given as a set of *variables* that have to be *labelled* with *values* and a set of *constraints* on these variables that are to be *satisfied*. There are many techniques used to solve constraint problems, but most techniques use some form of intelligent backtrack search that enumerate partial solutions, together with pruning algorithms that allow early detection that the current partial solution can not be extended to a total solution satisfying all the constraint of the problem. In this

978-3-9815370-2-4/DATE14/©2014 EDAA

paper we use a constraint solver (CLP(FD)) implemented in Sicstus Prolog [14]. The combination of Prolog and constraint programing allows us to easily specify both positive and negative constraints.

IV. Public Warning System for LTE

In our case study we produced logs for testing the test harness with a setup for the Public Warning System (PWS). This was carried out at an organisation in Ericsson AB, which is responsible for testing the LTE Radio Base Station. The Public Warning System is a technology that broadcast Warning Notifications to multiple users in case of disasters or other emergencies.

A. *E-UTRAN architecture*

LTE (Long Term Evolution) [2] is the global standard for the fourth generation of mobile networks (4G). Radio Access of LTE is called evolved UMTS Terrestrial Radio Access Network (E-UTRAN) [3]. A E-UTRAN consists of eNodeBs (eNBs), which is just another name for radio base stations. Our setup consists of an eNB, a simulated Mobility Management Entity (MME) that forwards PWS messages to the eNB, and some simulated User Equipment (UE). The functions of these entities are described in more detail below.

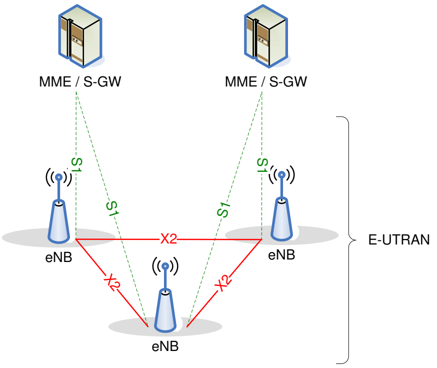

Fig. 1. This figure is from 3GPP TS 36.300

An eNB connects to User Equipment via the air interface. The eNBs may be interconnected with each other by means of the X2 interface. The eNBs are also connected by means of the S1 interface to the EPC (Evolved Packet Core), more specifically to the MME (Mobility Management Entity) by means of the S1-MME interface, and to the Serving Gateway (S-GW) by means of the S1-U interface [3]. The functions of eNBs include radio resource management; IP header compression and encryption, selection of MME at UE attachment; routing of user plane data towards S-GW; scheduling and transmission of paging messages and broadcast information; and measurement and reporting configuration for mobility and scheduling [2]. An eNB is responsible for the scheduling and transmission of PWS messages received from MME. The MME performs mobility management; security control; distribution of paging messages; ciphering and integrity protection of signaling; and provides support for PWS message transmission. S-GW is responsible for packet routing and forwarding.

B. *ETWS*

Earthquake and Tsunami warning system (ETWS) is a part of PWS that delivers Primary and Secondary Warning Notifications to the UEs within an area where Warning Notifications are broadcast [4]. We show in Figure 2 the network structure of PWS architecture.

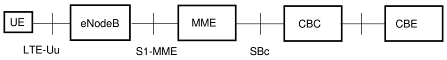

Fig. 2. This figure is from 3GPP TS 23.041

The Cell broadcast Entity (CBE) can be located at the content provider and sends messages to the Cell Broadcast Center. The Cell Broadcast Center (CBC) is part of EPC and connected to the MME.

The CBE sends emergency information to the CBC. The CBC identifies which MMEs need to be contacted and sends a Write-Replace Warning Request message containing the warning message to be broadcast to the MMEs. The MME sends a Write-Replace Warning Confirm message that indicates to the CBC that the MME has started to distribute the warning message to eNBs. The MME forwards Write-Replace Warning Request to eNBs in the delivery area. The eNB determines the cells in which the message is to be broadcast based on information received from MME [15]. If a Warning Type IE (information element) is included in a Write-Replace Warning Request message, then the eNB broadcasts a Primary Notification. If Warning Message Contents IE is included in a Write-Replace Warning Request message, then the eNB schedules a broadcast of the warning message according to the value of Repetition Period IE (`rPer`) and Number of Broadcasts Requested IE (`NumberofBroadcastRequested`) [16]. To inform UE about presence of an ETWS primary notification and/or ETWS secondary notification, a paging message is used. UE attempts to read paging at least once every defaultPagingCycle (`dPC`). If UE receives a Paging message including ETWS-indication, then it starts receiving ETWS primary notification or ETWS secondary notification according to schedulingInfoList contained in SystemInformationBlockType1 (`SIB1`). ETWS primary notification is contained in SystemInformationBlockType10 (`SIB10`) and ETWS secondary notification is contained in SystemInformationBlockType11 (`SIB11`). `SIB10` and `SIB11` are transmitted in System Information (SI) messages with different periodicity. If secondary notification contains large message, then it is divided in several segments, which are transmitted in System Information messages.

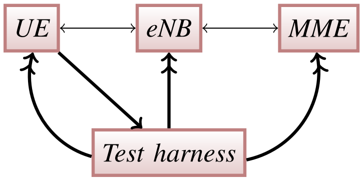

Fig. 3. Block diagram representing the test setup

V. Testing procedure

ETWS requires testing that the paging messages, `SIB1`, `SIB10` and `SIB11` are transmitted correctly by the eNB. These messages appear in a UE protocol log. To test functionality automatically, the test harness should: initiate transmission of Write-Replace Warning Request messages by the MME simulator; configure the UE simulator and initiate logging; configure the eNB; and capture and read a UE protocol log.

This is done in order to detect errors in timing constraints, message content, message ordering, and the number of transmitted messages. (see Figure 3). The arrow shown in Figure 3 from the test harness to MME indicates that the test harness initiates transmission of messages between MME and eNB. The arrow from the test harness to UE indicates initiation of logging. The arrow from UE to the test harness indicates that the test harness captures and analyses UE protocol log. The arrow from the test harness to eNB indicates that the test harness initializes some parameters in eNB. The test harness has many checks on the produced logs, which contributes to its complexity; therefore the test harness should be tested itself. Thus, number of logs with errors should be generated for testing. We used constraint programming [5] to model ETWS notifications acquisition by UE and based on solutions provided by Sicstus Prolog we generate UE protocol logs for ETWS, which consist of sequences of messages with timestamps, where different types of errors are introduced.

VI. Modelling and UE protocol log generation

Our goal is to generate an UE protocol log. To do this, we defined a model consisting of constraints on lists of timestamps and message contents. The constraints specified ordering constraints between messages; constraints on the number of messages of a certain type and content; and temporal constraints on the timestamps. Time was represented as integers with a granularity of 20ms, which was sufficient for our purposes. Several parameters can represent one message. Our model describes how UE acquires ETWS notifications sending by eNB after receiving one Write-Replace Warning Request message from MME. In order to generate incorrect logs we have extra parameters that indicate if a message should appear in a log. This was all implemented in Sicstus Prolog [14] using its CLP(FD) library.

A. Modelling of ETWS notifications acquisition by UE

We only modelled a subset of the ETWS notification system, where only one warning message is transmitted by the MME.

The model contains several lists of parameters that represent time stamps of messages. For example, we have a list $\mathtt{PagSN}$, whose elements represent timestamps of paging messages transmitted every repetition period. The length of the list is $\mathtt{nBR} = \mathtt{NumberofBroadcastRequested} + 1$.

To model ETWS notification acquisition by UE we used the following constraints. The constraint that defines time difference between two consecutive paging messages transmitted every repetition period is $(\mathtt{PagSN}_{i+1} - \mathtt{PagSN}_i = \lfloor \mathtt{rPer}/\mathtt{dPC} \rfloor \cdot \mathtt{dPC}) \vee (\mathtt{PagSN}_{i+1} - \mathtt{PagSN}_i = (\lfloor \mathtt{rPer}/\mathtt{dPC} \rfloor + 1) \cdot \mathtt{dPC})$ for every i from 1 to $\mathtt{nBR} - 1$, where $\mathtt{PagSN}_i$ is ith element in the list $\mathtt{PagSN}$. The constraint that guarantees that there is at least one paging message every repetition period is $((i-1) \cdot \mathtt{rPer} - \mathtt{dPC} < \mathtt{PagSN}_i - \mathtt{PagSN}_1) \wedge (\mathtt{PagSN}_i - \mathtt{PagSN}_1 < (i-1) \cdot \mathtt{rPer} + \mathtt{dPC})$ for every i from 2 to $\mathtt{nBR}$. Note that although this and all the following constraints are expressed in first order logic, the translation into Sicstus Prolog was mechanical. We have also list $\mathtt{PagPN}$ of timestamps for paging messages which are transmitted every $\mathtt{dPC}$. The size of the list is $\mathtt{ndPC}$ and is configured in eNB.

Timestamps for $\mathtt{SIB10}$ and $\mathtt{SIB11}$ are elements of lists of lists, since several messages can be transmitted during the same paging cycle or repetition period. The constraint that defines that there are n System Information messages with $\mathtt{SIB11}$ during every repetition period is $\mathtt{PagSN}_i < \mathtt{SIB11Time}_{i,j} < \mathtt{PagSN}_{i+1}$ for all $1 \leq i \leq \mathtt{nBR} - 1$ and $1 \leq j \leq n$, where $\mathtt{SIB11Time}_{i,j}$ is jth element of ith sublist in the list of lists $\mathtt{SIB11Time}$, and $\mathtt{SIB11Time}$ is a list of lists of timestamps of System Information messages with $\mathtt{SIB11}$. It can be that UE reads different number of $\mathtt{SIB11}$ during different repetition periods, but since we are interested in incorrect behaviour, we do not model all possible correct behaviours.

Secondary notification can come in one or several segments. $\mathtt{SIB11Segment}_{ij}$ contains the segment number of $\mathtt{SIB11}$ with timestamp $\mathtt{SIB11Time}_{i,j}$. The UE should read every segment at least once during every repetition period. That is for all $(0 \leq i < \mathtt{nSeg})$, where $\mathtt{nSeg}$ is the number of segments in a secondary notification, and for all $(1 \leq j \leq \mathtt{nBR} - 1)$ there should be some $1 \leq k \leq n$ such that $\mathtt{SIB11Segment}_{j,k} = i$. We also constraint the time difference between two consecutive $\mathtt{SIB10}$ received by UE in the same paging cycle and two consecutive $\mathtt{SIB11}$ received by UE in the same repetition period. The constraints on two consecutive $\mathtt{SIB11}$ received by UE for all i from 1 to $\mathtt{nBR} - 1$ and j from 1 to $n - 1$ are: $\mathtt{SIB11Time}_{i,j+1} - \mathtt{SIB11Time}_{i,j} > 0$ and $\mathtt{SIB11Time}_{i,j+1} - \mathtt{SIB11Time}_{i,j} \bmod \mathtt{siPer} = 0$ and $((\mathtt{SIB11Time}_{i,j+1} - \mathtt{SIB11Time}_{i,j})/\mathtt{siPer}) \bmod \mathtt{nSeg} = (\mathtt{SIB11Segment}_{i,j+1} - \mathtt{SIB11Segment}_{i,j}) \bmod \mathtt{nSeg}$. The model contains parameters that represent timestamps and content of $\mathtt{SIB1}$ messages. $\mathtt{SIB1SIB11Time}$ is a list of timestamps of $\mathtt{SIB1}$ messages during repetition periods. $\mathtt{SIB1SIB11Type}$ is list of values from 0 to 3 that indicates whether $\mathtt{SIB1}$ contains $\mathtt{schedulingInfoList}$ for $\mathtt{SIB10}$ and/or $\mathtt{SIB11}$. Then for all i from 1 to $\mathtt{nBR}$ we post a constraint: $(\mathtt{SIB1SIB11Type}_i = 0 \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagSN}_{\mathtt{nBR}})$ or $(\mathtt{SIB1SIB11Type}_i = 1 \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i < \mathtt{PagSN}_{\mathtt{nBR}})$ or $(\mathtt{SIB1SIB11Type}_i = 2 \wedge \mathtt{SIB1SIB11Time}_i < \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i < \mathtt{PagSN}_{\mathtt{nBR}})$ or $(\mathtt{SIB1SIB11Type}_i = 3 \wedge \mathtt{SIB1SIB11Time}_i < \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagSN}_{\mathtt{nBR}})$.

B. Mutation of the specification

We use three lists of parameters to indicate if messages should appear in the log. For example, we have the list $\mathtt{PagPNInd}$ of boolean variables that is the same length as $\mathtt{PagPN}$. If $\mathtt{PagPNInd}_i$ is equal to 1, then we include paging message with timestamp $\mathtt{PagPN}_i$ into protocol log. We use parameters as arguments in the constraints to introduce specific errors into the model. For example, we have a parameter whose value 0 means that every element of $\mathtt{PagPNInd}$ is equal to 1, while 1 means that elements of $\mathtt{PagPNInd}$ are generated randomly.

We introduced errors into the model in different ways by adding or removing messages, randomly changing time differences, excluding randomly chosen sublists, by changing the timing between `SIB10` and `SIB11` messages, generating the wrong number of segments, and generating random values that control which message types are generated. For every type of negation we have an argument in a Prolog rule, that allows to include or exclude errors in the model.

C. UE protocol log generation and test execution

We created protocol logs automatically by using a script that generates paging messages, `SIB1` messages and System Information messages with `SIB10` and `SIB11`, based on values of parameters from solutions found by Sicstus Prolog. Then we ran the test harness that reads our generated logs. If the test harness found a fault in the log, then it printed an error message. It is important to test the test harness, since it can provide incorrect pass/fail verdict if it contains faults. We found different types of faults in the test harness: by reducing length of the lists we found a case when the test harness does not print an error message if not all messages of certain type (paging messages) are transmitted; by generating an incorrect time difference between two messages, we found an error message, generated by the test harness, which provided incorrect information about the number of messages; and by randomly generating extra elements in the list, we found a case when an error message was displayed, but there was no error of corresponding type in the protocol logs.

VII. Conclusion

It only takes a few seconds for Sicstus Prolog to generate a log. In contrast collecting logs by executing test cases on the system can take many minutes. Further, in order to get sufficient variation in the behaviour of the collected logs many different setups should be used, changing setups can take a significant amount of time, and there is no guarantee that changing the setup will give the desired variation. Thus our approach gives us a rich set of generated protocol logs.

Also, the modelling is useful to understand what kind of incorrect behaviour can occur in logs. Generated logs can be used in the different stages of testing the test harness, from the very beginning to late in the development process. Our approach shows that use of constraint logic programming is an easy way to introduce faults into the model and helped to find real faults in the test harness.

The use of constraint programming has a lot of potential applications in model based testing of telecommunication protocols. These protocols can be complicated, and often require constraints on the data that appear in messages. With constraint programming it was easy to model the protocol here. Further, it was easy, once we had a high level declarative model, to mutate the model to generate protocol logs that have incorrect behaviour. In the future we plan to automate the mutation of the models.

Acknowledgment

The authors would like to thank Frits Vaandrager for valuable comments on an earlier draft of this paper. The second author is supported by VINNMER Program 2011-03229 funded by Swedish Governmental Agency for Innovation Systems. The third author is supported by grant 2012-4908 of the Swedish Research Council(VR).

References

[1] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[2] S. Chadchan and C. Akki, “3GPP LTE/SAE: An overview,” *International Journal of Computer and Electrical Engineering*, vol. 2, no. 5, pp. 806–814, 2010.

[3] 3GPP, “General packet radio service (GPRS) enhancements for evolved universal terrestrial radio access network (E-UTRAN) access,” 3rd Generation Partnership Project (3GPP), TS 23.401. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/23401.htm

[4] 3GPP, “Public warning system (PWS) requirements.” 3rd Generation Partnership Project (3GPP), TS 22.268. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/22268.htm

[5] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[6] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson, “On the reification of global constraints,” *Constraints*, vol. 18, no. 1, pp. 1–6, January 2013.

[7] E. Farchi, A. Hartman, and S. S. Pinter, “Using a model-based test generator to test for standard conformance,” *IBM systems journal*, vol. 41, no. 1, pp. 89–110, 2002.

[8] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, L. Gerald, A. J. Simons, S. Vilkomir, M. R. Woodard, and H. Zedan, “Using formal specifications to support testing,” *ACM Computing Surveys (CSUR)*, vol. 41, no. 2, p. 9, 2009.

[9] B. Legeard and F. Peureux, “Generation of functional test sequences from B formal specifications presentation and industrial case-study,” in *16th Annual International Conference on Automated Software Engineering, 2001. (ASE 2001).*, 2001, pp. 377–381.

[10] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, “Model-based testing of object-oriented reactive systems with spec explorer,” *Formal Methods and Testing*, pp. 39–76, 2008.

[11] R. DeMilli and A. J. Offutt, “Constraint-based automatic test data generation,” *Software Engineering, IEEE Transactions on*, vol. 17, no. 9, pp. 900–910, 1991.

[12] N. Williams, B. Marre, P. Mouy, and M. Roger, “Pathcrawler: Automatic generation of path tests by combining static and dynamic analysis,” in *5th European Dependable Computing Conference (EDCC-5)*, ser. Lecture Notes in Computer Science, M. Dal Cin, M. Kaâniche, and A. Pataricza, Eds., vol. 3463. Springer-Verlag, 2005, pp. 281–292.

[13] M. Carlier, C. Dubois, and A. Gotlieb, “Focaltest: A constraint programming approach for property-based testing,” in *Software and Data Technologies*, ser. Communications in Computer and Information Science, J. Cordeiro, M. Virvou, and B. Shishkov, Eds. Springer Berlin Heidelberg, 2013, vol. 170, pp. 140–155.

[14] M. Carlsson, G. Ottosson, and B. Carlson, “An open-ended finite domain constraint solver,” in *PLILP 1997*, ser. LNCS, H. Glaser, P. Hartel, and H. Kuchen, Eds., vol. 1292. Springer, 1997, pp. 191–206.

[15] 3GPP, “Technical realization of cell broadcast service (CBS),” 3rd Generation Partnership Project (3GPP), TS 23.041. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/23041.htm

[16] 3GPP, “Evolved universal terrestrial radio access (E-UTRA) ; S1 application protocol (S1AP),” 3rd Generation Partnership Project (3GPP), TS 36.413. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36413.htm