The application of model-checking to verification problems in telephone switching systems

Stephan Reiff and Muffy Calder
University of Glasgow, Department of Computer Science
Glasgow, G12 8QQ, Scotland
e-mail: {sreiff,muffy}@dcs.gla.ac.uk
WWW: http://www.dcs.gla.ac.uk/{~sreiff/,~muffy/}

Abstract

Model-checking is becoming very popular for automated reasoning about hardware and software. The input can be in any language provided there is a translation to transition systems or Kripke structures; properties are then expressed in a temporal logic.

Telephone switching systems are concurrent systems which are naturally described by a composition of interacting finite automata. We describe a model of one switching system in IO-Automata, express it in PROMELA (process meta language) and then formulate certain properties in LTL (linear temporal logic). Using the model-checker SPIN, we have attempted to verify the model with respect to some properties. A particular challenge is that even a simple switching system leads to state space explosion.

1 Introduction

Telephone switching systems have developed from manual switchboards to automated systems. The latter require control software, which is usually large, unstructured and not well documented (legacy software). On the other hand, increasing user demands are forcing the provision of extensions, usually in the form of features. Implementing features is not straight forward, as in addition to the desired interactions with the legacy system and other features, undesirable ones also occur. This is commonly known as the *feature interaction problem*. A typical example for an undesirable interaction is an inconsistency between *call waiting* and *call forwarding on busy*. The former requires to provide a call waiting tone, the latter tries to forward the call – both are triggered by an incoming call on a busy line.

To improve the provision of features in telephone switching systems, we are investigating methods to detect and resolve feature interactions. We are convinced that model-checking will provide results that can be used to improve operational and legacy systems as well as their development process. But to achieve this, the model needs to be an appropriate abstraction of the operational system, i.e. realistic enough to represent the system while being simple enough to be formally verified.

We propose such a model in Section 2 and give details of one component in the subsequent section. In Section 4 we comment on the implementation in PROMELA (PROcess MEta LAnguage) – the input language for the model-checker SPIN([Hol91]) and in section 5 show examples of properties that we wish to verify. In the conclusion we comment on the achievements so far, and discuss further directions.

2 A Telephone Switching System

In this section we give a high-level operational overview of a telephone switching system. As can be seen from figure (2) the system consists of three principal groups: the network, users and the operator. The operator interactions are mainly database enquiries and manipulations and do not concern us here.

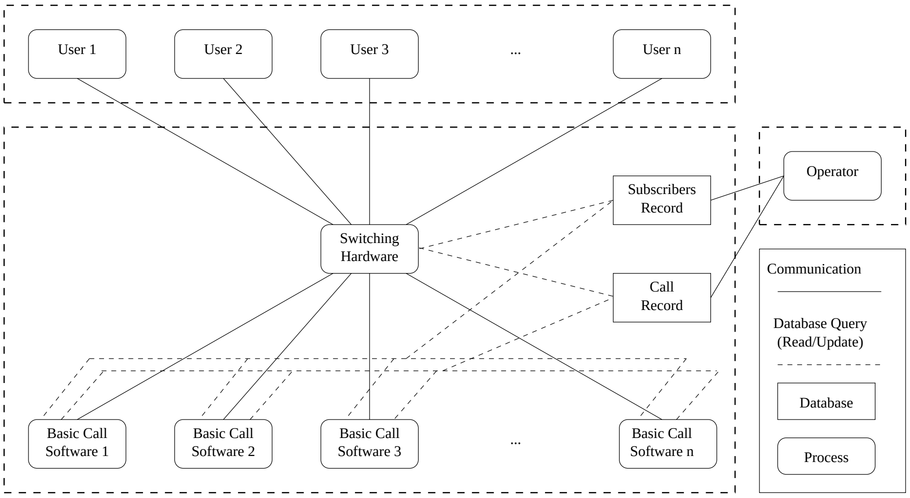

Figure 2: Overview of the System

The term 'user' will be used to describe points at which the network interacts with the world, such as terminal devices, trunks (connections to other networks) or any other communication equipment. For simplification, we only consider telephones as terminal devices and assume that there is only one switch.

The network consists of two database components (the call record and the subscribers record), which are used for control and billing purposes. Furthermore, we have a switching hardware process which passes messages between users and basic call software processes (and combinations thereof). Each user is assigned a basic call software process; they are only instantiated on demand and provide POTS (Plain Old Telephone System) call behaviour.

This model might seem rather more simplistic than a PSTN (Public Switched Telephone Network) but it does, for example, describe a PBX (Private Branch Exchange).

3 A Basic Call Software Process

We use IO-Automata to model the processes. An IO-Automata ensures that an input message is never blocked but allows us to place restrictions on when an output message is sent ([LT88]). Furthermore, we can compose such automata, allowing communication via their respective input and output messages.

Figure (3) shows the automaton for a basic call software process, where input messages are in italics, and all other messages are output messages. A message consists of two parts, an event and a numeric argument. The argument is usually an user identifier, or 0 when no argument is required. Note that once this automaton is placed in an environment there will be no non-deterministic choices, as choices are determined either by the received input actions or data from the databases. Nevertheless, any path through the automaton is possible. In the following example we describe a successful call to user 4 from another user (user 3 say).

© S. Reiff, M. Calder. Model-checking Telephone Switching Systems

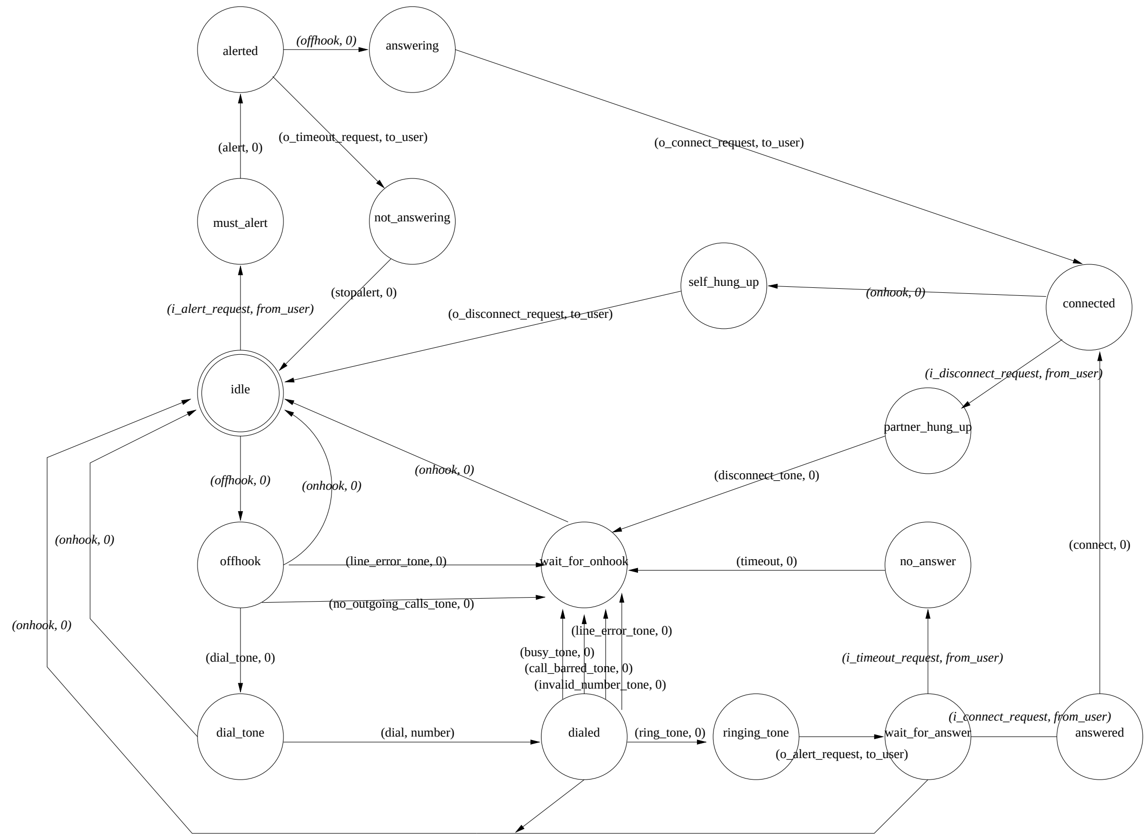

Figure 3: The Basic Call Software Process

We only consider the basic call software relating to the originating behaviour, ignoring any terminating behaviour.

Example 1 *The basic call software receives an* $(offhook, 0)$ *message to which it responds with* $(dial\_tone, 0)$*. Next a* $(dial, 4)$ *messages triggers the sending of* $(ring\_tone, 0)$ *and* $(o\_alert\_request, 4)$*. User 4 goes offhook which leads to an* $(i\_connect\_request, 4)$ *message being received, triggering* $(connect, 0)$ *being sent. Now the two users are connected. An* $(onhook, 0)$ *indicates that the connection is to be cleared down, triggering an* $(o\_disconnect\_request, 4)$ *message being sent. At this point the basic call software is in its originating state and a new call can be made.*

We note that this automaton does not describe the real behaviour in a PSTN. For example in the current BT network only the originator of a call can clear down the call (with exceptions, like 999). However, many PBXs already show symmetric behaviour.

4 The Promela Implementation

Implementing the above system in PROMELA is largely a matter of mastering the syntax correctly. Promela uses channels for communication. Each automaton is represented by a process, the number of arguments passed to each process is determined by the number of communication channels it needs (e.g. a basic call software process needs two uni-directional channels).

5 Verification of Some Properties

Properties of the system are expressed in LTL (Linear Temporal Logic) for automatic proofs in SPIN. Here will give them only informally, as the LTL formulation can only be understood with respect to details of the model implementation.

Examples of properties we wish to verify are:

- If user A calls him/herself he/she should perceive a busytone.
- If there are no faults with the lines, user A is allowed to originate calls and user B is not busy, then eventually they can be connected.

These property might sound not very interesting, but we note that once a feature such as *call forwarding* is added to the system, they might become invalid.

6 Conclusions

Our long term goal is to reason about feature interactions in legacy systems with added features. As a first step we have constructed a model of an existing telephony system (possibly legacy) and investigated the use of model-checking techniques for analysis.

Next steps include adding features and a feature manager ([MM98]) to the model, derive new properties and prove those. These properties rank from if no feature is activated, then the system behaves exactly like POTS, to totally new behaviour that is expected of activated features. We want to detect unwanted interactions as well and resolve those (hence the feature manager). Generic properties (e.g. livelock or deadlock) are of no less interest, because they are often consequences of feature interactions. In fact they may be more useful, when dealing with systems where we do not have access to feature specifications, because they are but are independent of a particular features's expected behaviour.

The work reported here provides a basis for a prototype implementation, future feature development processes, and, hopefully, decision making at runtime when an interaction is detected.

Related work on exploring user views of telephone switching systems, as in [CM98] and general work on feature interaction shows the importance of finding solutions to the problem ([KB98], [DBL97] and others).

References

[CM98] M. Calder and A. Miller. Analysing a basic call protocol using PROMELA/XSPIN. In *Proceedings* SPIN*'98: Workshop on automata theoretic verification with the* SPIN *model checker*, pages 169–181, Paris, France, November 1998.

[DBL97] P. Dini, R. Boutaba, and L. Logrippo, editors. *Proceedings of the* 4^{th} *International Workshop on Feature Interactions in Telecommunications Systems.* IOS Press (Amsterdam), June 1997.

[Hol91] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall (New York), 1991.

[KB98] K. Kimbler and L. G. Bouma, editors. *Proceedings of the* 5^{th} *International Workshop on Feature Interactions in Telecommunications and Software Systems.* IOS Press (Amsterdam), September 1998.

[LT88] N. A. Lynch and M. R. Tuttle. An introduction to Input/Output Automata. November 1988.

[MM98] D. Marples and E. H. Magill. The use of rollback to prevent incorrect operation of features in intelligent network based systems. In K. Kimbler and L. G. Bouma, editors, *Proceedings of the* 5^{th} *International Workshop on Feature Interactions in Telecommunications and Software Systems*, pages 115–134, Lund, Sweden, September 1998.