Specifying Features and Analysing Their Interactions in a LOTOS Environment[1]

M. Faci and L. Logrippo

University of Ottawa, Protocols Research Group,
Department of Computer Science, Ottawa, Ontario, Canada K1N 6N5
E-mail: {mfaci, luigi}@csi.uottawa.ca

Abstract. This paper presents an approach for specifying telephone features and analysing their interactions in a LOTOS environment. The approach is characterized by a flexible specification structure and an analysis method based on knowledge goals. Structurally, the specifications allow the integration of new features into existing ones by specifying each feature independently and composing its behaviour with the existing system. Analytically, the reasoning mechanism allows the specifier to analyse features, for the purpose of detecting their interactions, by defining knowledge goals and simulating the system to verify if they are reachable. A non reachable knowledge goal reveals the existence of a feature interaction, or a design error. We explain this approach by the use of two, now classical, examples of feature interactions, namely Call Waiting & Three Way Calling and Call Waiting & Call Forward on Busy.

Keywords: Telephone Features, Feature Interactions, Formal Specifications, LOTOS.

1. Motivation and Background

The plain old telephone service (POTS) is used for establishing a communication session between two users. A telephone feature, such as *call waiting* (cw), *call forward on busy* (cfb), and *three way calling* (3wc), is defined as an added functionality of POTS. Augmenting POTS with a small set of features is considered to be a technically straightforward job. Both the behaviours of POTS and the features are analysed and decisions are taken as to how to integrate the features into POTS. Conflicts between any of the features are resolved on a case by case basis. However, as more and more features need to be integrated, as is the case for present and future telephone networks [Lata89], the task becomes more difficult. Features that perform their functions satisfactorily on their own are, in some instances, prevented from doing so in the presence of other features. This problem has become known as the *feature interaction problem* [BDCG89].

Investigations into the feature interaction problem fall into one of three complementary categories[CaVe93]: Detection, avoidance, and resolution. The

1. Appeared in: L. Bouma and H. Velthuijsen (eds.). Feature Interactions in Telecommunications Systems. IOS Press, 1994, 136-151.

objective of a detection approach is to analyse a set of independently specified features and determine whether or not there are any conflicts between their joint behaviour [CaLi91], [Lee92], [BoLo93], [DaNa93]. An avoidance mechanism assumes that the causes of the interactions are known and an architectural or analytical approach is defined to prevent the manifestation of such interactions [MiTJ93]. The avoidance approach is most suitable in the early phases of specification and design of features. Finally, the objective of a resolution mechanism is to find appropriate solutions to interactions that manifest themselves at execution time [Cain92] [GrVe92].

This paper describes a method, based on the LOTOS specification language [BoBr87], [LoFH92], for detecting feature interactions at the specification level, in the context of single user single element features[CGLN93]. Central to our method are the concepts of *constraints* and *knowledge goals*. Constraints, which we have categorized as local, end-to-end, and global, are used to structure the specification so that new features can be added to the specification or removed from it with plausible ease [FaLS91]. While the concept of constraints is useful for structuring specifications, the concept of knowledge goals is useful for reasoning about telephone features, in order to detect feature interactions. It is based on the theories of *knowledge* which are being developed for understanding and reasoning about communication protocols and distributed systems [HaFa89], [HaMo90], [PaTa92]. In the knowledge-based approach, the evolution of the system can be described by the evolving knowledge of the components, about the state of other components, which collectively make up the global state. Thus, the state of knowledge in the system changes as a result of exchanging messages between the communicating components, and communication between the components depends on their knowledge about the system state. We use this concept to analyse combinations of features in order to detect the interactions between them.

In section 2, we describe our method for detecting interactions between independently specified features. In section 3, we demonstrate the application of our approach on two examples: *cw&cfb* and *cw&3wc*. We conclude with some thoughts regarding our research directions in section 4.

2. Using a LOTOS Environment to Detect Feature Interactions

Although the feature interaction problem has existed for quite many years, little attention was paid to it until it was explicitly defined [BDCG89]. Since then, a whole new field of interest is born. Due to the lack of space, we simply point the reader to some of the work of other researchers in this area. In particular, we mention the work of [HoSi88], [CaLi91], [Dwor91], [Cain92], [EKDB92], [GrVe92], [Inoue92], [Lee92], [DaNa93], and [Zave93]. Two other excellent sources of information are the special issues of IEEE Computer[Comp93] and IEEE Communications magazine [Magz93].

We begin the section by reviewing the constraint-oriented style that we had developed for specifying telephone systems in LOTOS. Then, we describe an improvement to the structure of our specifications which makes it possible to easily integrate features into a telephone system, while still using the concept of constraints. We conclude the section by describing a reasoning mechanism which allows us to analyse the joint behaviour of features, for the purpose of detecting their interactions.

2.1 LOTOS Structure of the POTS Specification: An Overview

We have previously developed a LOTOS specification structure that is well suited for specifying the behaviour of telephone systems[FaLS91]. The structure is based on the constraint-oriented specification style [VSVB91], where we identified three types of constraints:

(1) *Local constraints* are used to enforce the appropriate sequences of events at each telephone, and are different according to whether the telephone is a *Caller* or a *Called*. Therefore local constraints are represented by the processes *Caller* and *Called* and an instance of each of these is associated with each telephone existing in the system. Because these two processes are independent of each other, they are composed by the interleaving operator |||.

(2) *End-to-End* constraints are related to each connection, and enforce the appropriate sequence of actions between telephones in a connection. For example, ringing at the *Called* must necessarily follow dialling at the *Caller*. Process *Controller* enforces these constraints. Because they must apply to both *Caller* and *Called*, we have the structure *(Caller* ||| *Called)* || *Controller.* Thus the controller must participate in every action of the *Caller*, as well as in every action of the *Called*, separately.

(3) *Global constraints* are system-wide constraints. In the POTS context, we identified one main constraint, which is the fact that at any time, a number is associated with at most one connection. Because global constraints, represented by a process *GlobalConstraints*, must be satisfied simultaneously over the whole system, we have the structure *Connections* || *GlobalConstraints*.

It should be stressed that the constraint-oriented style is purely a specification style, which allows to clearly separate the logical constraints a system must abide. It does not necessarily reflect an implementation architecture. To obtain an implementation architecture from a constraint-oriented structure, style transformations may be applied [VSVB91].

2.2 Enhancing the Structure of POTS Specifications

A graphical representation of the enhanced structure of POTS specifications is shown in Fig.1. The LOTOS specification of this structure, which handles only 4 users for the purposes of illustration, is shown in Fig. 2. This structure has the following characteristics: (1) Each user is represented by two processes, a caller side and a called side and each caller process is bound to its own controller. So, in the POTS case, each

connection becomes: *(Caller(n)* ||| *Called)* || *Controller(n)*, where *n* is the subscriber number. (2) Contrary to the structure presented in [FaLS91], where the number of the called user was passed to the called process at instantiation time, in this structure, the called process is now represented as a set of alternatives, where each alternative represents the called side of a user in the system. Clearly, since the controller offers to synchronize with only one called at a time, only a single alternative will offer the same value. As we will see in section 3.1, this structure is highly flexible for integrating new features into a telephone system. (3) the global constraints process participates in every action in which any instance of the controllers participates.

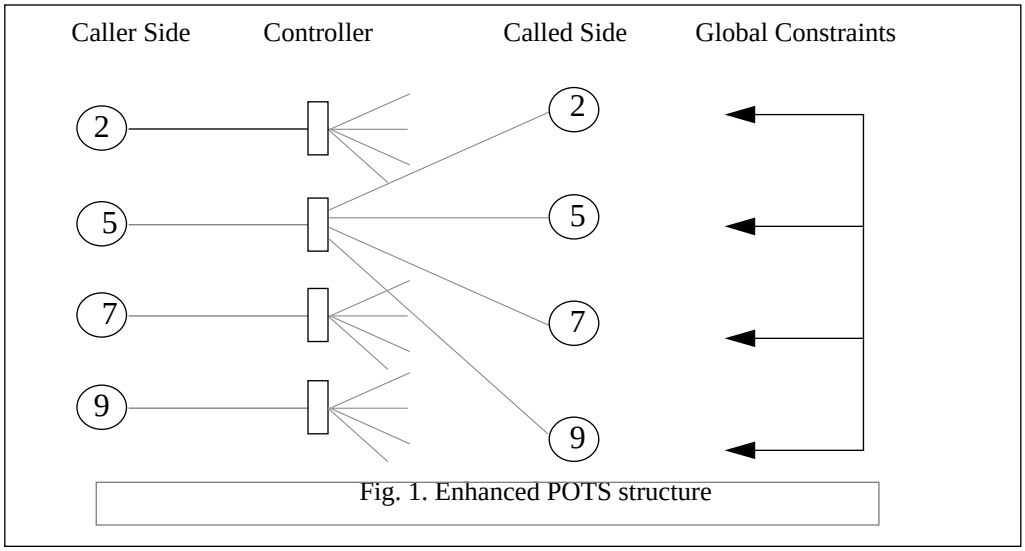

Fig. 1. Enhanced POTS structure

```
1   behaviour
2   (    ( Caller[Suser] (2) ||| Called [Suser](Users) )|| Controller [Suser] (2)
3        |||
4        ( Caller[Suser] (5) ||| Called [Suser](Users) ) || Controller [Suser] (5)
         |||
5        (Caller[Suser] (7) ||| Called [Suser](Users) ) || Controller [Suser] (7)
6        |||
7        (  Caller[Suser] (9) ||| Called [Suser](Users) ) || Controller [Suser] (9)      )
8   ||
9   GlobalConstraints[Suser](parameters)
10  where
11              process Caller[Suser](n: TelNo):noexit:= ...
12              process Called [Suser](Users: List): noexit:=
13                   Called [Suser](2) [] Called [Suser](5) [] Called [Suser](7) [] Called [Suser] (9)
14              endproc (* Called *)
15              process Controller[Suser](n: Digit): noexit:= ...
16              process GlobalConstraints [Suser](Parameters: Sets)): noexit:= ...
```

Fig. 2. LOTOS specification of POTS using the enhanced structure.

2.2.1 Local Constraints

As mentioned, *Local constraints* are used to enforce the appropriate sequences of events within each process. For example, to specify a *Caller* process, the specifier needs only to understand the events that are exchanged between the *Caller* process and its environment. At this stage, the specifier does not need to concentrate on *who* represents the environment or *which* processes will interact with the *Caller* process. These concerns are addressed at a later stage of the specification. By taking this view, we have in fact reinforced the concept of *separation of concerns*. In the case of specifications dealing with POTS [FaLS91] [BoLo93], *local constraints* were applied to the caller entity and the called entity only. Our experience has shown that the concept of local constraints can be used for specifying telephone features as well.

2.2.2 End-to-End Constraints

For a simple two-way call processing, the *end-to-end* constraints were used to synchronize the actions of two processes with respect to each other, most often the sender and the receiver. Our experience in writing the POTS specifications is that establishing a temporal order between two actions, one being offered by the caller and the other one by the called, is quite intuitive and simple. However, we recognize that expressing *end-to-end* constraints of the new structure may become more complicated, because there are more processes which may offer synchronization actions. And, it is still the specifier's task to impose a temporal order on a set of given actions. The specifier must then have some heuristics and guidelines at his/her disposal. A possible approach is to start by expressing the end-to-end constraints as *cause-effect*[Lin90] [NuPr93] rules.

2.2.3 Global Constraints

Finally, the *Global constraints* are at a higher level of abstraction than the end-to-end constraints, since they are imposed on the global behaviour of the system. In the simple two-way call processing model, the global constraints were restricted to enforcing *value* constraints between independent connections. In our new structure, global constraints gain an added importance. They enforce *control* constraints as well. Let us illustrate this with the following example, see Fig. 4. Suppose that user 2, who subscribes to *cw*, establishes a connection (A) with user 5. Also, suppose that, while 2 is talking to 5, 7 calls 2. Since 2 has *cw*, the global constraints process must manage the new connection (B), which consists of a new caller and shares the called side with the existing connection of 2. Therefore, the global constraint is responsible for switching *the control* between connection A and connection B, depending on what stage of the communication the users are in. For instance, when 7 dials 2 and 2 answers the call, the global constraint removes the communications between 5 and 2 from the set of active sessions and inserts it into the set of holding sessions. At the same time, it allows a communication session to be established between 7 and 2. When 2 and 7 finish their conversation, the global constraint reactivates the connection (A), between 2 and 5.

The structure that we have just defined exhibits the required flexibility. New features are defined in terms of their local constraints and their end-to-end constraints, while their global constraints are composed with the existing global constraints of the system. To specify a new feature, the specifier either instantiates actions that already exist in the system, such as the *dial* action, or defines new actions, such as the *CwTone* action used in the definition of the *cw* feature. In the former case, the resulting global constraints on the *dial* action is the conjunction of the existing constraints and the new constraints. In the latter case, a new alternative action is added to the behaviour of the global constraint process.

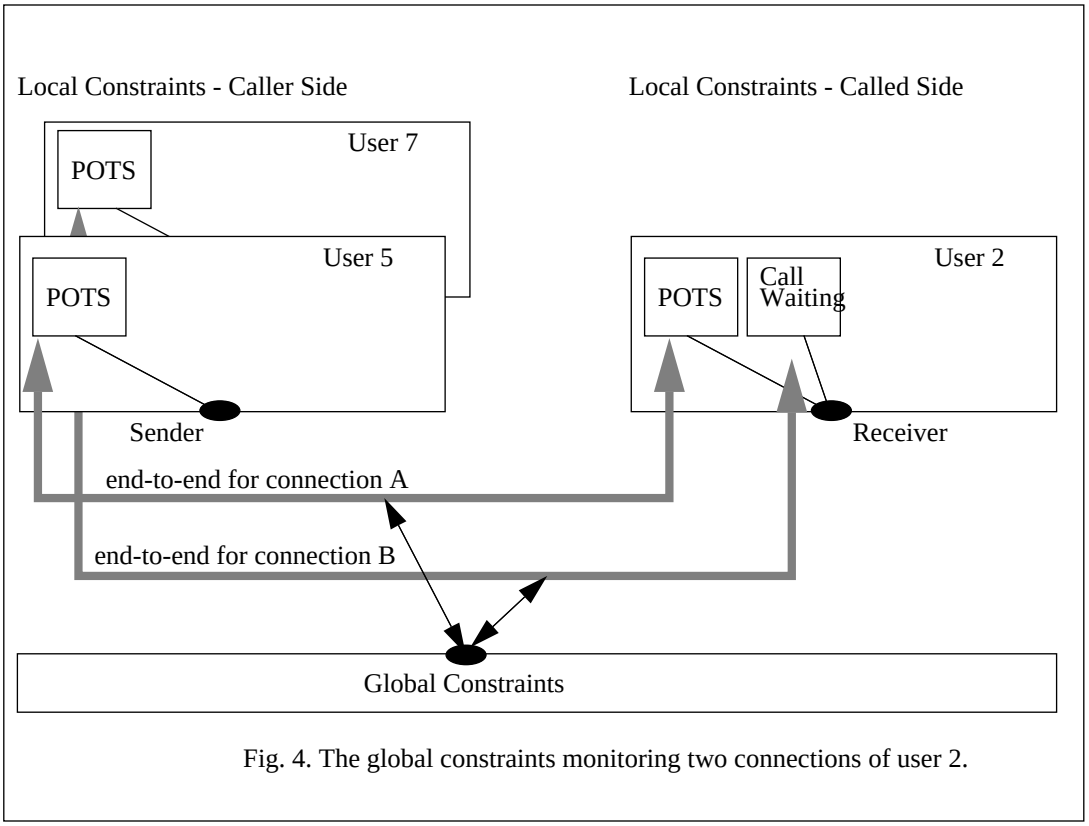

Fig. 4. The global constraints monitoring two connections of user 2.

2.3 Using Knowledge Goals to Reason about LOTOS Specifications

This section describes how to adapt the *knowledge-oriented* model of Halpern and Moses [HaMo90] and incorporate it into LOTOS specifications for telephony systems. The intuition we want to capture, by using the knowledge-based approach, is that the designer reasons about LOTOS processes in terms of how relevant information, from the local point of view (i.e., local constraints) of each process, becomes satisfied at certain points during the execution of the system. To analyze whether two features, say *cw* and *cfb*, interfere with each other, the designer defines a set of knowledge goals and verify their reachability, when both features are active. If any of the goals is unreachable, the designer concludes that a feature interaction (or design error) exists. Otherwise, no conclusion can be drawn from the analysis. In a way, this is similar to system testing. A test which fails to reveal an error does not indicate that the system under test is error free, it only means that the system is error free with respect to the assumption expressed by the test. Details of our analysis using two examples are given in section 3.

It is interesting to emphasize that each process reasons about the outside world only in terms of its local information. Therefore, a process moves from one state to another state based only on its knowledge. Similarly, it gains (or loses) new knowledge as it moves from one state to another. Also, notice that knowledge in this context is an *external* notion, in the sense that processes do not acquire knowledge on their own nor are they able to analyze the knowledge state of other processes. For the purposes of analysis, the specifier is responsible for choosing the appropriate *knowledge goals* used to reason about the system.

3. Application of the Approach

In this section, we show how to apply our method to two examples of feature interactions: *cw* vs. *cfb* and *cw* vs. *3wc*. For each example, we show how to use the enhanced structure to integrate the two features into a single LOTOS specification, and then we show how to use the reasoning mechanism to detect their interactions. Only LOTOS segments which contribute significantly to the understanding of the integration and reasoning mechanisms are given.

3.1 Specifying Features in LOTOS

We have concluded from our experiments of specifying telephone features that most features act on behalf of either the caller side or the called side. From our structural and analytical points of view, some features which seem to act on behalf of both the caller side and the called side can be given only one of the two roles. An example of this is the *automatic recall* feature (not discussed in this paper). When a user is busy, this feature automatically returns the last incoming call when the subscriber's line becomes idle. We have classified this feature as having a caller role because its first action is to initiate a connection back to a user who has initiated a call.

3.1.1 Call Waiting

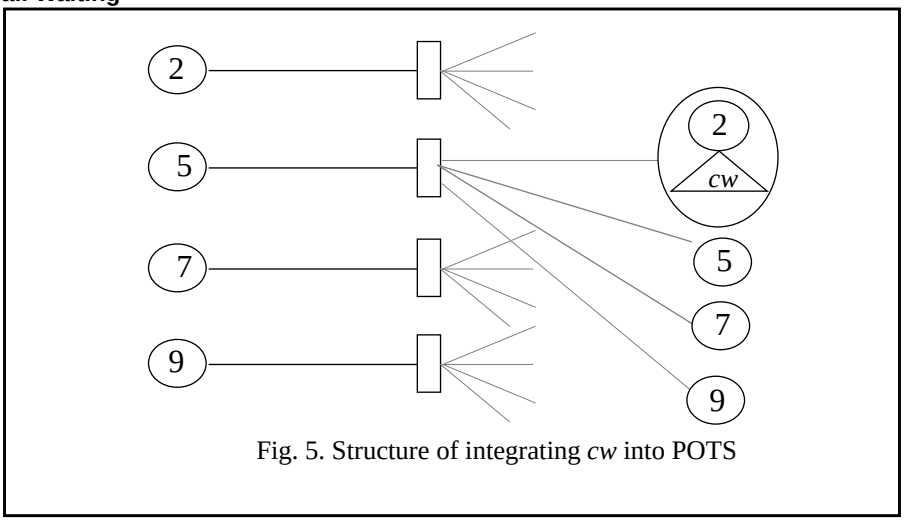

Fig. 5. Structure of integrating *cw* into POTS

Call Waiting is a feature which generates a call waiting tone, to alert a busy user that a second incoming call is waiting to be answered. The user may choose to answer the call,

using a special *flashhook* signal, or may simply continue with the original communication and ignore the call waiting tone.

This feature has a called role. Therefore, first, we specify the behaviour of the feature in the context of POTS. Second, we modify the existing *Controller* so that it handles any new actions that the new feature participates in, such as *call waiting tone*. This results in a structure of the form: *(Caller(n)* ||| *Called)* || *ControllerCw(n).* Third, we replace the existing definition of the controller with the new definition, which is now capable of handling POTS calls as well as subscribers with the call waiting feature. Finally, for each action in which the feature participates, we check that the predicates remain valid in the global constraints. Modifications of figure 1 are shown in Fig. 5 and result in the structure of Fig. 6.

```
1   behaviour
2   (    ( Caller[Suser] (2) ||| Called [Suser](Users) ) || Controller [Suser] (2)
3        |||
4        ( Caller[Suser] (5) ||| Called [Suser](Users) ) || Controller [Suser] (5)

5        |||
6        (  Caller[Suser] (7) ||| Called [Suser](Users) ) || Controller [Suser] (7)
7        |||
8        (  Caller[Suser] (9) ||| Called [Suser](Users) ) || Controller [Suser] (9)      )
9   ||
10  GlobalConstraints[Suser](parameters)
11  where
12       process Caller[Suser](n: TelNo):noexit:= ...
13       process Called [Suser](Users: List): noexit:=
14            CalledPots [Suser](2) [] CalledPots [Suser](5) [] CalledPots [Suser](7)
15            [] CalledPots [Suser] (9)
16            [] CalledCw(2)                              <----------------Added
17       endproc (* Called *)
18       process Controller[Suser](n: Digit): noexit:= ... <---------------- Modified
19       process GlobalConstraints [Suser](Parameters: Sets)): noexit:= ... <--- Modified
```

Fig. 6. LOTOS specification of *cw* within POTS.

3.1.2 Call Forward on Busy

Call Forward on Busy is a feature which allows a user, who is already involved in a conversation with a second user, to transfer his/her incoming calls to a predetermined third user. Depending on the specifier's intentions, the busy user may or may not be informed that a call transfer has occurred. This feature acts on behalf of the called side, so its specification is similar to that of call waiting. The modification of Fig. 1 results in a similar structure to that of call waiting, and is not shown.

3.1.3 Three Way Calling

Three way calling is a feature which allows a user, who is already involved in a conversation with a second user, to add a third user to the conversation. The subscriber

of the feature must put the second user on hold, using a special *flashhook* signal, while establishing a communication with the third user. Once the communication is established, a second *flashhook* brings the second user back to the conversation to form a 3 way communication.

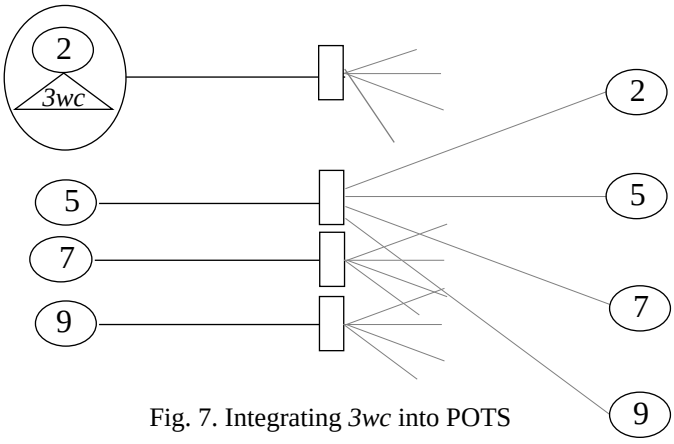

Fig. 7. Integrating *3wc* into POTS

This feature has a caller role. It is specified as follows. First, we specify the behaviour of the feature with respect to POTS. Second, we modify the existing *Controller* so that it handles any new actions that *3wc* participates in, such as *flashhook*. This results in a structure of the form: *(Twc(n)* ||| *Called)* || *Controller3wc(n),* where *n* identifies the subscriber for which the feature is to be invoked. Third, we compose this new structure with the existing connections using the ||| operator. Finally, for each action in which the feature participates, we check that the predicates in the global constraints process remain valid. Extending our specification of Fig. 1, the structures of Figs. 7 and 8 result:

```
1   behaviour
2   (    ( Caller[Suser] (2) ||| Called [Suser](Users) ) || Controller [Suser] (2)
3        |||
4        (  Caller3wc[Suser] (2) ||| Called [Suser](Users) ) || Controller3wc [Suser](2) <-------- added
5        |||
6        ( Caller[Suser] (5) ||| Called [Suser](Users) ) || Controller [Suser] (5)
         |||
7        (  Caller[Suser] (7) ||| Called [Suser](Users) ) || Controller [Suser] (7)
8        |||
9        (  Caller[Suser] (9) ||| Called [Suser](Users) ) || Controller [Suser] (9)
10  )
11  ||
12  GlobalConstraints[Suser](parameters)
13  where
14          process Caller[Suser](n: TelNo):noexit:= ...
15          process Caller[Suser](n: TelNo):noexit:= ...
16          process Called [Suser](Users: List): noexit:=
17               Called [Suser](2) [] Called [Suser](5) [] Called [Suser](7) [] Called [Suser] (9)
18          endproc (* Called *)
19          process Controller[Suser](n: Digit): noexit:= ...
20          process Controller3wc [Suser](n: Digit): noexit:= ... <-------- Added
21          process GlobalConstraints [Suser](Parameters: Sets)): noexit:= ...
```

Fig. 8. New LOTOS Structure of POTS

3.2 Analysing Features to Detect their Interactions

In this section, we present our method and show how it can be used to detect interactions between *cw*&*cfb* and *cw*&*3wc*. The method calls for the following steps to be carried out:

1• Specify each feature independently, within a POTS context;

2• Use the structure defined in section 2.2 to integrate both features into a single specification;

3• Define the knowledge goals to be reached in the reasoning phase; a knowledge goal is expressed as a LOTOS process which is composed in parallel with the specification obtained in 2 above.

4• Finally, simulate the system and check if the selected goals are reachable. A feature interaction (or design error) is detected if the selected goals are not reachable.

Assuming that points 1 and 2 above are completed successfully, let us proceed with point 3. In the two examples to follow, the designer has chosen to reason about the system in terms of subscribers' talking states. In other words, the designer knows that each feature, when active by itself within the context of POTS, can successfully reach its talking state, and the question she is attempting to answer is: can two features, if activated simultaneously by the same user, reach their talking states? as will be seen, the answer is no in both examples. Notice that if the designer fails to detect an interaction using a selected knowledge goal, it is strongly recommended that other goals be tried. So, it is the responsibility of the designer, using her design insights, to define the appropriate goals for reasoning about the system.

3.2.1 Call Waiting & Call Forward on Busy

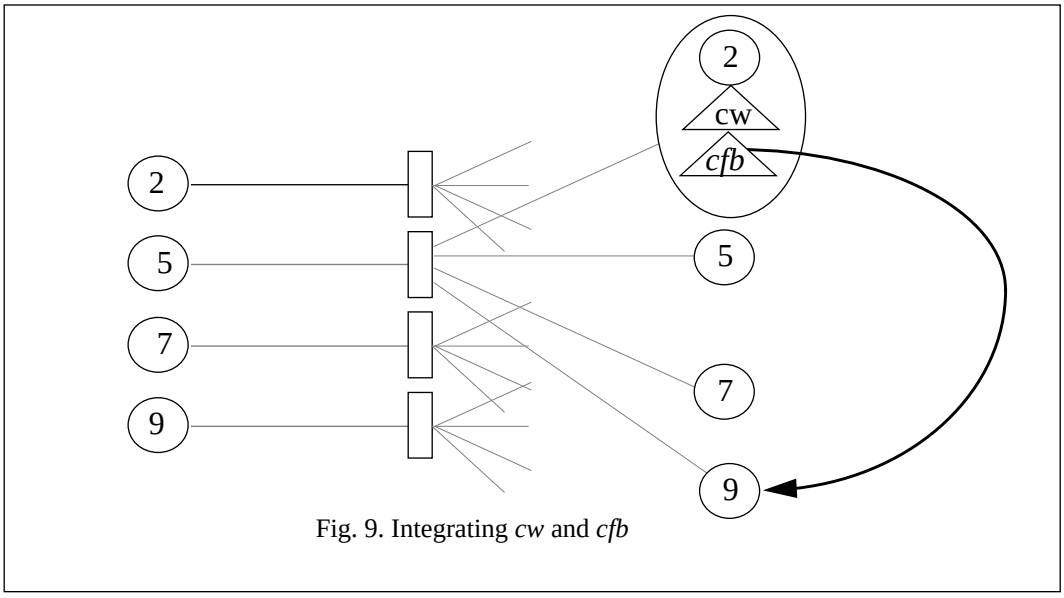

Fig. 9. Integrating *cw* and *cfb*

Fig. 10 shows the modification of POTS by extending its functionality according to *cw* and *cfb*.Let us assume that only cw is active on 2. Then, if 7 calls 2 while 2 is talking to 5, one possible scenario is that 5 is put on hold and a talking session between 7 and 2 is established. This scenario is shown on the left side of Fig. 10. If we assume instead that only cfb is active, and using the same execution scenario, a talking session between 2 and 5 remains active and a talking session is established between 7 and 9. Therefore, our knowledge goal is defined as: Talk(2, 7) *and* Talk(7, 9). Since each of these two goals is reachable when only one feature is active, then they *must* also be reachable if cw and cfb are activated simultaneously with respect to their talking sessions.

To clarify this further, we denote by POTS+CW+CFB[gpots, gcw, gcfb] the extension of POTS by the features *cw* and *cfb*, where each feature is added to POTS separately; *gpots* is the gate through which all POTS events occur, *gcw* is the gate through which all *cw* events occur, *gcfb* is the gate through which all *cfb* events occur. Let the first part of the goal Talk[gpots, gcw](2, 7) be defined by the expression (gpots !7 !dials !2; gcw !2 !flashhook), as shown in the left branch of Fig. 10; let the second part of the goal Talk[gpots, gcfb](7, 9) be defined by the expression (gpots !7 !dials !2; gcfb

!9 !rings (from 7); gcfb !9 !answers), as shown in the right branch of Fig. 10. Therefore, if a deadlock (in the sense of LOTOS) occurs in the behaviour expression: POTS+CW+CFB[gpots, gcw, gcfb] || (Talk[gpots, gcw] |[gpots]| Talk[gpots, gcfb]), then we can conclude that a feature interaction exists. The LOTOS specification and its execution tree are given as follows:

```
3   behaviour
4           gpots !7 !dials !2;
5           (   gcw !2 !flashhook; stop
6           []
7               gcfb !9 !rings; gcfb !9 !answers !7; stop
8           )
9           ||
10          (
11              gpots !7 !dials !2; gcw !2 !flashhook; stop
12              |[gpots]|
13              gpots !7 !dials !2; gcfb !9 !rings; gcfb !9 !answers !7; stop
14          )
Execution tree
1   gpots [4,11,13]
    | 1 gcw [5,11] DEADLOCK
    | 2 gcfb [7,13] DEADLOCK
```

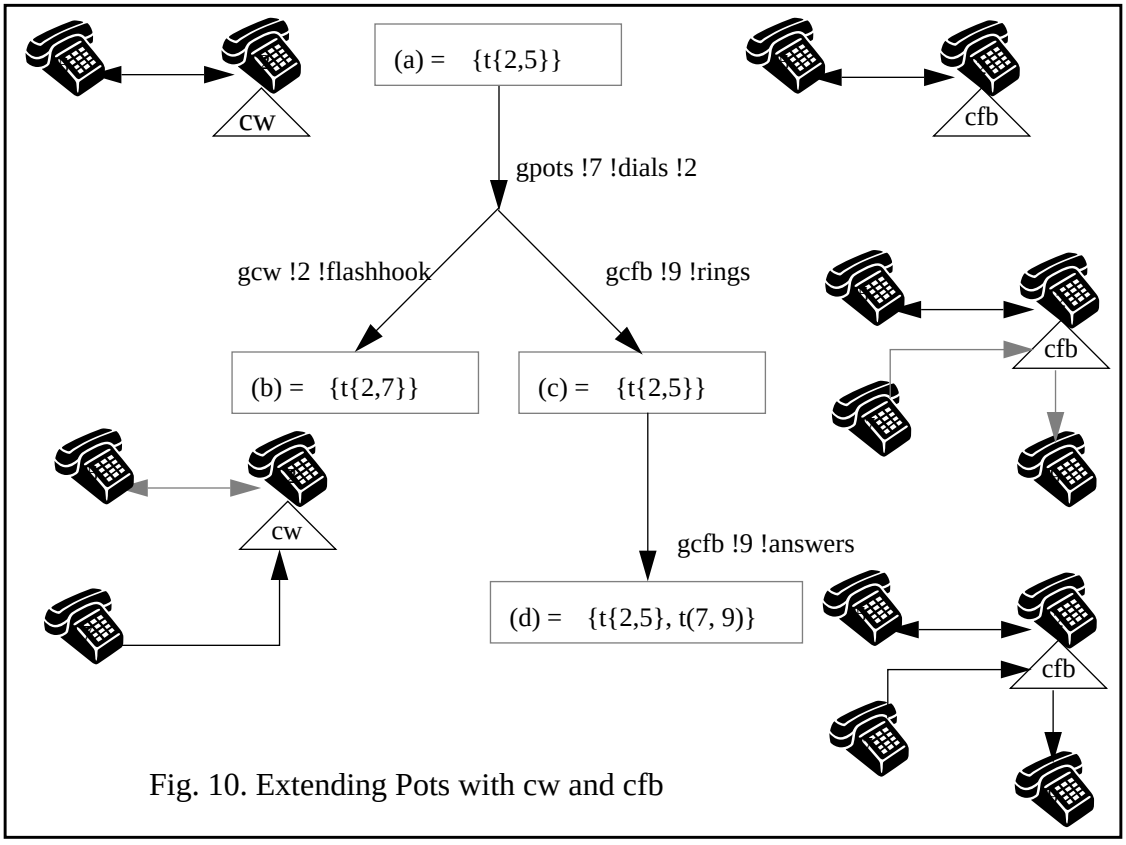

Fig. 10. Extending Pots with cw and cfb

3.2.2 Call Waiting & Three Way Calling

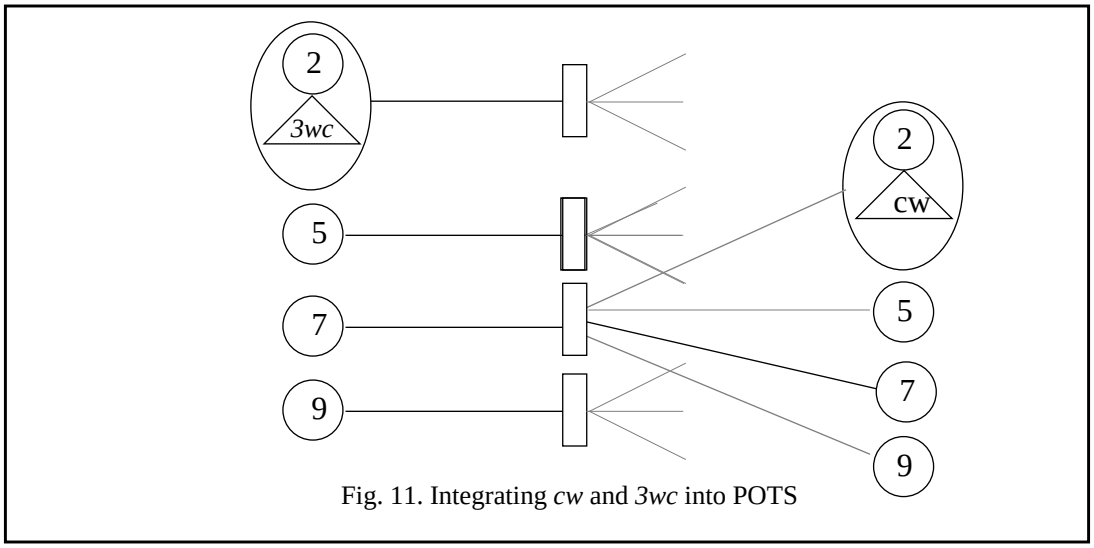

Fig. 11. Integrating *cw* and *3wc* into POTS

Our analysis for detecting the interaction between *cw* and *3wc* is similar to that of the previous example. Clearly, our objective is to know whether or not *cw* and *3wc* can be invoked simultaneously by 2 without causing a conflict between them.

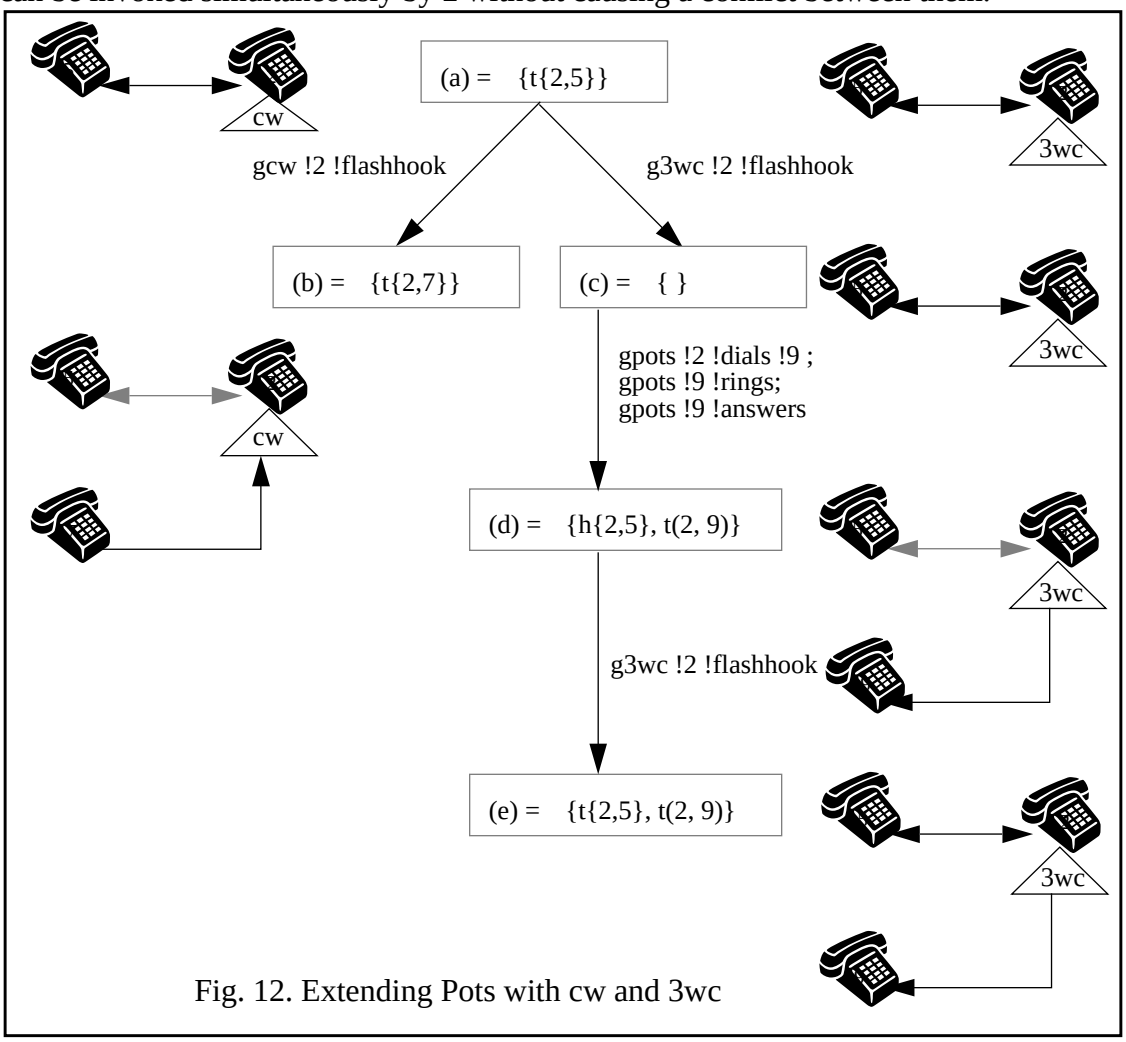

Fig. 12. Extending Pots with cw and 3wc

The left side of Fig. 12 is the same as the left side of Fig. 10. Concerning the right-hand side, assume that only 3wc is active, and that, using the same flashhook signal, 2 moves to a state where he may dial 9. To make the analysis easier, we assume that 9 is idle and answers the call. Therefore, a talking session between 2 and 9 is established. A second flashhook reestablishes the original talking session between 2 and 5. Therefore, our knowledge goal can be defined as: Talk(2, 7) *and* Talk(2, 9). As is in the previous example, we must show that the behaviour: POTS+CW+3WC[gpots, gcw, g3wc] || (Talk[gpots, gcw] |[gpots]| Talk[gpots, g3cw])) is deadlock free. In this case as well, a deadlock is reached as can be easily verified by executing the above behaviour expression. We conclude that a feature interaction exists.

4. Conclusions and Research Directions

We have proposed a method, based on a formal approach, for detecting feature interactions at the specification level, for single user single element features. Structurally, the approach uses three types of constraints: local constraints, end-to-end constraints, and global constraints. This structure allows the integration of new feature specifications into existing ones simply by classifying their roles as *caller* or *called* and expressing their global constraints as a conjunction. This structure is possible because of LOTOS's multiway synchronization mechanism, which also offers the flexibility to describe a system as a composition of constraints. Analytically, the approach is based on a reasoning mechanism which allows the specifier to analyse features, for the purpose of detecting their interactions, based on knowledge goals, where each goal is expressed as a LOTOS process. Our interpretation, in this context, is that conflicts between features correspond to deadlock situations in the LOTOS sense.

Important items for future research are adapting our approach to more realistic examples, extending the technique to other types of feature interactions, and making the technique more automatic. i.e., less dependent on designer's insight regarding where the problem might be found.

Acknowledgment. Funding sources for our work include the Natural Sciences and Engineering Research Council of Canada, the Telecommunications Research Institute of Ontario, Bellcore, Bell-Northern Research, and the Canadian Department of Communications. We like to acknowledge the many fruitful discussions that we had with Bernard Stepien and members of our LOTOS group. Also, comments from the referees have led to improvements of the content of this paper.

5. References

[BDCG89] T.F. Bowen, F.S. Dworak, C.H. Chow, N. Griffeth, G.E. Herman, and Y-J. Lin, The Feature Interaction Problem in Telecommunications Systems, 7th International Conference on Software Engineering for Telecommunication

Switching Systems, July 1989, 59-62.

[BoBr87] Bolognesi, B.,Brinksma, E. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems 14, 1987, 25-59.

[BoLo93] R. Boumezbeur, L. Logrippo, Specifying Telephone Systems in LOTOS, IEEE Communications Magazine, Aug. 1993, 38-45. E. J. Cameron, N. Griffeth, Y. Lin, M. E. Nilson, W. K. Schnure, H. Velthuijsen, A Feature Interaction Benchmark for IN and Beyond, IEEE Communications, vol. 31, No. 3, 64-69, March 1993.

[CaLi91] E. J. Cameron and Y.J. Lin, A Real-Time Transition Model for Analyzing Behavioral Compatibility of Telecommunications Services. In Proceedings of the ACM SIGSOFT 1991 Conference on Software for Critical Systems, pp. 101-111, December 1991, New Orleans, Louisiana.

[CaVe93] J. Cameron and H. Velthuijsen, Feature Interactions in Telecommunications Systems, IEEE Communications Magazine, Aug. 1993, 18-23.

[Cain92] M. Cain, Managing Run-Time Interactions Between Call-Processing Features, IEEE Communications Magazine, pp. 44-50, February 1992.

[Comp93] IEEE Computer, Special Issue on Feature Interactions in Telecomunications Systems, Aug. 1993.

[DaNa93] O. Dahl and E. Najm, Specification and Detection of IN Service Interference Using LOTOS, to appear in the proceedings of Forte ’93, Boston.

[Dwor91] F. S. Dworak, Approaches to Detecting and Resolving Feature Interactions, GLOBECOM 1991, pp. 1371-1377.

[EKDB92] M. Erradi, F. Khendek, R. Dsouli, and G. V. Bochmann, Dynamic Extension of Object-Oriented Distributed System Specifications, First International Workshop on Feature Interactions in Telecommunications Software Systems, Florida, 1992, 116-132.

[FaLS91] Faci, L. Logrippo and B. Stepien,Formal Specifications of Telephone Systems in LOTOS: The Constraint-Oriented Style Approach, Computer Networks and ISDN Systems, 21, 52-67, North Holland, 1991.

[GrVe92] N. D. Griffeth and H. Velthuijsen, The negotiating agent model for Rapid Feature Development, Proceedings of the 8th International Conference on Software Engineering for Telecommunications Systems and Services, Florence, Italy, March/April 1992.

[HaFa89] J. Y. Halpern and R. Fagin, Modelling Knowledge and action in distributed systems, Distributed Computing, 3, 159-177, 1989.

[HaMo90] J. Y. Halpern and Y. Moses, Knowledge and Comon Knowledge in a Distributed Environment, JACM, Vol. 37, No. 3, 549-587, July 1990.

[HoSi88] S. Homayoon and H. Singh, Methods of Addressing the Interactions of Intelligent Network Services With Embedded Switch Services, IEEE Communications Magazine, pp. 42-47, Dec. 1988.

[Inoue92] Y. Inoue, K. Takami, and T. Ohta, Method for Supporting Detection and Elimination of Feature Interaction in a Telecommunication System, First International Workshop on Feature Interactions in Telecommunications

Software Systems, Florida, 1992, 61-81.

[Lata89] LATA Switching Systems Generic Requirements (LSSGR), Bellcore, TR-TSY-000064, FSD 00-00-0100, July 1989.

[Lee92] A. Lee, Formal Specification and Analysis of Intelligent Network Services and their Interaction, Ph. D. Thesis, Dept. of Computer Science, University of Queensland, 1993.

[Lin90] Y. J. Lin, Analyzing Service Specifications Based upon the Logic Programming Paradigm, In Proceedings of the IEEE GLOBECOM 1990, pp. 651-655, December 1990, San Diego, California.

[LoFH92] L. Logrippo, M. Faci and M. Haj-Hussein, An Introduction to LOTOS: Learning by Examples, Computer Networks & ISDN Systems, Vol. 23, No. 5, 1992, pp. 325-342.

[Magz93] IEEE Communications Magazine, Special Issue on Feature Interactions in Telecomunications Systems, Aug. 1993.

[MiTJ93] J. Mierop, S. Tax, R. Janmaat, Service Interaction in an Object Oriented Environment, IEEE Communications Magazine, Aug. 1993.

[NuPr93] K. Nursimulu and R. L. Probert, Cause-Effect Validation of Telecommunications Service Requirements, Technical Report TR-93-15, University of Ottawa, Dept. of Computer Science, October 1993.

[PaTa92] P. Panangaden and K. Taylor, Concurrent Common Knowledge: Defining Agreement for Asynchronous Systems, Distributed Computing, 6, 73-93, 1992.

[VSVB91] C.A. Vissers, G. Scollo, M. van Sinderen, E. Brinksma, Specification Styles in Distributed Systems Design and Verification, Theoretical Computer Science 89, 1991, 179-206.

[Zave93] P. Zave, Feature Interactions and Formal Specifications in Telecommunications, IEEE Computer Magazine, Aug. 1993, 18-23.