A Discipline for Handling Feature Interaction

Egidio Astesiano and Gianna Reggio

DISI
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35 – Genova 16146 – Italy
{astes,reggio} @ disi.unige.it
http://www.disi.unige.it

Abstract. A challenging problem within the wider software evolution problem is the development of systems by features. While most of the recent work centered around the detection of feature interactions, we present an approach based on modular specification, separation of concerns and prevention of unwanted interactions. We illustrate our approach extending a formalism for the specification of reactive systems and showing its application to some aspects of the well-known case of telephone systems (POTS and variations).
The paper concentrates more on the methodological aspects, which are, at large extent, independent of the formalism. Indeed, this seems to be the case of some rather novel concepts like the distinction between pre-features (features in isolation) and features, closed and open semantics, feature composition and discipline of feature interaction, and finally the pervading role of a kind of anti-frame assumption.

Introduction

Evolution in software development has many facets. One which emerged in the last five years, especially in the area of telecommunications and networking, is the continual expansion of services. Recognizing that objected-oriented incrementality is not adequate to cope with this new problem in full generality, the concept of "feature" as unit of update has been introduced, see e.g., [11], and taken as a pivotal unit for even new paradigms, like feature-oriented programming, feature-oriented specification and so on [9].

In spite of the considerable effort (see some pointers to recent work at the end), still many issues deserve further attention and investigation, as it is admitted by the specialists of the subject, also taking into account the growing complexity of the applications concerned. Among the issues, feature composition and interaction is definitely the one attracting most attention. This is also witnessed by the success of an International Workshop on Feature Interaction, now reaching in '98 its fifth edition. In particular a lot of work is reported on the so-called feature interaction detection, possibly done automatically. According to this viewpoint, feature interaction is synonym with unexpected/unwanted results.

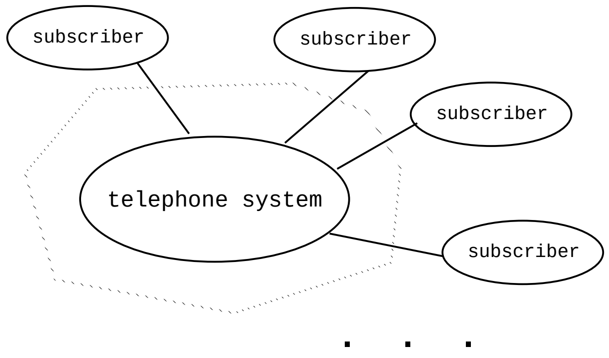
We are among those sharing the view that the problem of feature-interaction should be tackled within a wider methodological approach. This view is best expressed by Pamela Zave in [11], who calls for "an approach based on modular specifications and separation of concerns ... (aimed) to organize the specification so that it is easy to add without destroying its structure or desiderable properties". This is indeed the underlying challenging problem; again in P. Zave's words "the goal of extendible specifications is as difficult to achieve as it is easy to state". For example, in the realm of reactive and concurrent systems, the classical approach to incrementality has been based on the notion of process/ agent as unit of change; feature-driven incrementality deals instead with incrementality within a process and refers/affects the behaviour of the pre-existing processes.

Here we want to outline a specification framework supporting a feature-driven software development method with rigorous semantics, offering conceptual tools for expressing requirements on unwanted interactions. The framework intends to be adaptable to a variety of different application fields, like telecommunications, information systems; clearly, depending on the application, sensible domain specific methods should be derived.

This paper is devoted to a semiformal introductory illustration of our approach by means of a significant running example. A more formal and detailed presentation of the technicalities can be found in other papers [2].

At the end of the paper we discuss the relationship with the existing work.

Running example As running examples we consider various telephone systems modularly built by composing features. We start with POTS (Plain Old Telephone System) schematically presented in the following picture.



Each subscriber may

- hear the phone ringing,
- <u>lift/put down the receiver</u>,
- <u>dial the number of another subscriber</u>,
- hear a busy/free tone/the voice of another subscriber on the receiver,
- and speak on the microphone.

In this paper, following [3] and others, in order to illustrate our approach, we consider the telephone system as a simple reactive system consisting of the part enclosed by the dotted line in the above picture; and we consider as its interchanges with the external world (the subscribers) only those underlined in the above list. However, in the real life application it would be better to see POTS as a concurrent system, whose components are the phones and the telephone net.

We then extend the functionalities of POTS by adding the possibility of *automatic-call back* (ACB) in the case of a call to a busy subscriber and of *three-parts service* (TP), i.e., telephone communications involving three subscribers. Using a telecommunication terminology, we can extend the system by adding two "features"; in our setting also the starting system (POTS) is considered a feature.

Outline Let us now outline the main steps of our approach.

- First, a feature is treated almost in isolation and considered just a simple reactive system; thus POTS, ACB and TP are specified (Sect. 1) as generalized transition systems, whose states are sets of attribute values (as in many O-O approaches) and whose transitions, defined by a set of *activity rules*, denote action capabilities, with the labels indicating the interface with the external environment. Still a basic new concept is introduced for taking care of the possible addition of other features: transitions are grouped under *types*, essentially indicating the kind of performed activity; when composing two features only transitions with the same type will have the possibility of being merged into a more complex transition.
  We call *pre-feature specifications* those concerning features as reactive systems in isolation. With a pre-feature specification a *closed semantics* is associated, consisting of the generalized labelled transition system logically derived from the specification. This semantics is based on an underlying closed world/frame assumption: the attribute updates and transitions which are not explicitly asserted cannot happen.
- Then we consider the possibility of adding features (Sect. 2.1). We take a compositional style and speak of *composition* of pre-features; composition will be a partial, associative and commutative operation.
  As an example, POTS and ACB can be composed, resulting in another pre-feature specification POTS $\oplus$ ACB; analogously we can get POTS $\oplus$ TP. Analyzing the examples we show how in the composition the transitions can be combined, under an underlying anti-frame assumption: the fact that a rule of a pre-feature does not update an attribute does not mean that when we compose such rule with another one that attribute cannot be updated.
- The composition of pre-features provides the technical setting for analysing *interaction of features* (Sect. 2.2): roughly, PF_2 interacts with PF_1 iff in $\mathrm{PF}_1 \oplus \mathrm{PF}_2$ some parts of PF_1 are modified. We emphasize that, differently than in many other approaches, "interaction" is for us a neutral concept, namely it is not in itself good nor bad, as we show in the cases of POTS$\oplus$ACB and POTS $\oplus$ TP.

- One of the methodological keypoint of our approach, following P. Zave's quoted principle, is trying to prevent the occurrence of bad interactions qualifying the kind of interaction we admit as much as we can.

  Hence the full concept of *feature specification* comes out naturally as a pair (Sect. 3): a pre-feature specification and a set of *interaction requirements*, which are formulae in some logical formalism constraining the possibility of adding other features. As we show, interactions can be grouped in families of some kind, e.g., those concerning the atomic transitions, the overall behaviour, etc. Roughly speaking, a logic can *discipline* the interactions of some kind whenever it is powerful enough to express requirements preventing them. Thus, methodologically, we suggest to prevent bad interactions first of all by an appropriate specification, in some respect like in the specification phase of any system development. It may well happen that some bad interactions escape our qualification; but that means we need some feedback process, via testing, for adjusting our specification, not differently than in the usual development process.

- The technical, rather novel, concept supporting that aspect of the method is the *open semantics* of a feature specification. While the closed semantics of the pre-feature specification part gives just one generalized labelled transition system, the one logically deducible adopting the usual "frame-assumption", the open semantics consists of all the generalized labelled transition systems satisfying not only the interaction requirements, but also the activity rules under an *anti-frame assumption.* We have indeed to take into account that adding features can induce w.r.t. the closed semantics a variety of changes, like updates of the attributes and new transitions, as long as they are not explicitly forbidden; moreover all models have to be in a sense "logic extensions"of the closed semantics, intending that the activity rules producing the closed semantics have to be satisfied, though in a generalized sense; this last point is formalized as an inplicit *default requirement*, one for each rule. Verifying, with the help of available verification tools, that the closed semantics satisfies the interaction requirements, is a standard way of checking consistency. Technically speaking, the open semantics is a kind of ultra-loose semantics, since it consists of models over signatures extending the one of the pre-feature specification.

  What is then the relationship between the open and the closed semantics? Unless the feature specification is inconsistent (no models), the closed semantics is one of the possible models.

  Then the *complete semantics* of a consistent feature specification is the pair formed by the closed and the open semantics.

  Now the concept of composition carries over feature specifications; but two features are *compatible* only if their pre-feature parts are composable and the requirements, including the default ones, are not in conflict; this implies that the resulting composed feature is consistent, i.e., it admits some model.

1 Pre-Features

1.1 Pre-feature Specifications

First we consider a feature in isolation, i.e., without any concern about adding other features; we will use the name "pre-feature" for such an entity, to mark the difference with the complete concept of feature that we will introduce later.

We start illustrating what is for us the description of a pre-feature; then we discuss its semantics.

In our approach a pre-feature is a partial description of a reactive system, possibly intended as a description of only some "parts" of the system.

We distinguish reactive systems in *simple* and *structured or concurrent*; the latter are those having cooperating components, which are in turn reactive systems (simple or structured). Our specification technique considers differently the two cases; and in this paper we show how to handle the simple ones; the concurrent ones will be considered in some future work.

We formally model a reactive system R with a generalized labelled transition system (see [7]), shortly *glts*, which is a 4-uple

$$(STATE, LABEL, TT, \rightarrow),$$

where $STATE$, $LABEL$ and TT are sets, the *states*, the *labels* and the *transition types* of the system, and $\rightarrow \subseteq TT \times STATE \times LABEL \times STATE$ is the *transition relation*.

The states represent the intermediate (interesting) situations of the life of R and the arcs between them the possibilities of R of passing from a state to another one. It is important to note that here an arc (a transition) $tt\colon s \xrightarrow{l} s'$ has the following meaning: R in the state s has the *capability* of passing into the state s' by performing a transition, where the label l represents the interaction with the external (to R) world during such move. Thus l contains information on the conditions on the external world for the capability to become effective, and on the transformation of such world induced by the execution of the action; so transitions correspond to *action capabilities*. The transition type tt allows to classify transitions by putting together those corresponding to execute "conceptually" related activities. The reasons of the introduction of transition types will be clear later. We only anticipate that while labels, as usual, are a tool for supporting process composition, transition types will play an essential role when defining feature composition and hence handling feature interaction.

Our specification technique is described below, and exemplified with its application to POTS, whose specification as a pre-feature (simple reactive system) is given in Fig. 1.

- The POTS data part (keyword **data**) uses the specifications SET(IDENT) for the sets of subscriber identifiers and REL for binary relations on identifiers (just sets of identifier pairs). Here we neglect the part of the data specification; it is only important to recall that its semantics should result in a many-sorted first-order structure.

```
**pre-feature** POTS[$s$] =
**data** SET(IDENT), REL
**attributes**
      $ACT$: $set(ident)$
- -   the set of the active subscribers, i.e., those which have lifted the receiver
      $BUSY$: $set(ident)$   - -  the set of the subscribers hearing a busy tone
      $TRYING$: $rel$
- -   the set of the pairs of subscribers $(id, id')$ s.t. $id$ is trying to connect with $id'$
      $CONN$: $rel$   - -  the set of the pairs of connected subscribers
**interface**
      $LIFT, DOWN(ident)$
- -   a subscriber lifts/puts down the receiver
      $DIAL(ident, ident)$   - -  a subscriber dials another one
**auxiliary**
      $Allowed, Available$: $ident$
- -   a subscriber is allowed to dial iff he has lifted the receiver, is not hearing a busy
- -   tone, and is neither trying to connect nor connected with another subscriber
      $Allowed(id) =_{\mathrm{def}}$
            $id \in ACT$ **and** $id \notin BUSY$ **and**
            **not exists** $id_1$ **s.t.**
            $(id, id_1) \in TRYING$ **or** $(id, id_1) \in CONN$ **or** $(id_1, id) \in CONN$
- -   a subscriber is available to be connected iff
- -   he has not lifted the receiver and his telephone is not ringing
      $Available(id) =_{\mathrm{def}} id \notin ACT$ **and not exists** $id_1$ **s.t.** $(id_1, id) \in TRYING$
**activity**
•     **if** $Available(id)$ **then**
            START: $s \xrightarrow{LIFT(id)} s[ACT \cup \{id\}/ACT]$
•     **if** $id \notin ACT$ **and** $(id', id) \in TRYING$ **then**
            ANSW: $s \xrightarrow{LIFT(id)} s[ACT \cup \{id\}/ACT]$
                                 $[TRYING - \{(id', id)\}/TRYING]$
                                 $[CONN \cup \{(id, id')\}/CONN]$
•     **if** $id \in ACT$ **then**
            DWN: $s \xrightarrow{DOWN(id)} s[ACT - \{id\}/ACT]$
                                 $[BUSY - \{id\}/BUSY]$
                                 $[id ⩤ TRYING/TRYING]$
                                 $[id ⩤ CONN ⩥ id/CONN]$
- -   $r ⩥ id$ ($r ⩤ id$) is $r$ minus all pairs whose right (left) component is $id$
•     **if** $Allowed(id)$ **and not** $Available(id')$ **then**
            D: $s \xrightarrow{DIAL(id,id')} s[BUSY \cup \{id\}/BUSY]$
•     **if** $Allowed(id)$ **and** $Available(id')$ **then**
            D: $s \xrightarrow{DIAL(id,id')} s[TRYING \cup \{(id, id')\}/TRYING]$
```

Fig. 1. The plain old telephone system pre-feature

- We characterize an intermediate state of a glts by means of attributes (keyword **attributes**) with values in the data part; in the case of POTS we must know which are the active subscribers, those hearing a busy tone, those trying to connect to another one and those connected each other.
  An attribute, say A, is a function from the states space into the corresponding value set; as usual in the O-O style, we write A for $A(s)$.
- We describe the various kinds of interactions of the system with the external world by means of label "constructors" possibly parameterized by values in the data part; we use the keyword **interface** since they really give the interface of the system. The label kinds of POTS (and so the label constructors) are: a subscriber lifts/puts down the receiver and dials another one.
- In the auxiliary definitions part (keyword **auxiliary**) we introduce some parametric shortcuts (macro definitions) which can be used in the activity; e.g., each occurrence of $Available(id)$ in a rule stands for

$$id \notin ACT \textbf{ and not exists } id_1 \textbf{ s.t. } (id_1, id) \in TRYING.$$

- The activity of POTS, i.e., the transitions of the glts modelling it, are given in the part after the keyword **activity** by means of conditional rules having the general form

$$\textbf{if } cond \textbf{ then } \mathsf{TT}\colon s \xrightarrow{L(y_1,\ldots,y_m)} s[x_1/A_1]\ldots[x_k/A_k],$$

  where TT is the type of the transitions defined by the rule; s is the variable appearing on the first line of the specification denoting a generic state of the system; $y_1, \ldots, y_m, x_1, \ldots, x_k$ are variables; L is one of the label constructors; for $j = 1, \ldots, k$ $\_[\_/A_j]$ is the state update operation for the attribute A_j; and $cond$ is a formula built on the signatures of the used data extended with the attributes.
  Usually the rules are written in a more friendly way, not only by substituting an attribute name A for each occurrence of $A(s)$, but also by dropping in the premise of a rule the atoms of the form $x_i = t$ $(y_j = t)$ and by replacing then in such rule each occurrence of x_i (y_j) with t.
  For example, the expanded version of the first rule of POTS is

$$\begin{array}{l}\textbf{if } id \notin ACT(s) \textbf{ and } (\textbf{not exists } id' \text{ s.t. } (id', id) \in TRYING(s)) \textbf{ and}\\ \quad a = ACT(s) \cup \{id\} \textbf{ then}\\ \mathsf{START}\colon s \xrightarrow{LIFT(id)} s[a/ACT].\end{array}$$

  The first two rules of POTS give the action capabilities corresponding to the interactions with the external world informally described by "a subscriber lifts the receiver"; notice that they have two different transition types, the first corresponding to the cases when the receiver is lifted for starting a call and the second for answering to a ringing phone. This distinction will be extremely useful when composing features together to avoid merging transitions corresponding to different activities.
  The third rule describes the effect of putting down the receiver.

The last two rules with the same label and transition type describe the effect of dialling a number (we assume that in this case at a conceptual level there is just a kind of activity: to dial).
In our formalism transitions using the same label constructor may have different transition types, but it cannot happen the contrary; thus transitions with the same transition type must use the same label constructor.

Let us give two other examples of pre-feature specifications corresponding to add to a telephone system the automatic call back and communications involving three subscribers respectively.

ACB **(Automatic Call Back)** This pre-feature introduces in the telephone system the call-back possibility; precisely, a subscriber, say S, after calling another subscriber, say S1, non-available for the communication, can ask the telephone system by pressing a special button to automatically call S1 again when he and S1 will be both available; then, if S1 will lift the receiver, then the phone of S will start ringing.

The full specification is reported in Fig. 2.

TP **(Three Parts service)** This pre-feature introduces in the telephone system the possibility of having communications involving three subscribers. Precisely, once two subscribers, say S1 and S2, have established a connection, S1 (or S2) may put such connection on hold and start another connection with a third subscriber, say S3; now S1 can switch between the connection with S2 and that with S3; moreover he can also start a three-way connection among he, S2 and S3.

TP uses as data also REL3 the specification of the ternary relations over subscriber identifiers (sets of triples of identifiers).

The full specification is reported in Fig. 3.

1.2 Closed Semantics

In our setting the *closed semantics* of a pre-feature PF, denoted by $[\![\mathrm{PF}]\!]_{\mathrm{C}}$, is just the semantics associated with PF considered as a simple system specification, with no concern for the possible addition and interference with other features; thus it is essentially given by the glts determined by the rules; "essentially" because it includes also the used data structures, just a many-sorted first-order structure. In this paper, for simplicity, we call glts also these richer structures. By "glts determined by the rules" we mean the glts whose transitions are all and only those that can be logically deduced by the rules. Because of the very simple form of our rules, that means to consider just the transitions obtained by instantiating the given activity rules (of course interpreting syntactic terms within the given data structure). Notice that the closed semantics is implicitly using a "frame assumption": any update or transition which is not asserted to happen cannot happen.

pre-feature ACB$[s]$ =
data SET(IDENT), REL
attributes
    ACT: $set(ident)$
- - the set of the active subscribers, i.e., those who have lifted the receiver
    $BUSY$: $set(ident)$ - - the set of the subscribers hearing a busy tone
    TO_CALL: rel
- - the set of the pairs of subscribers (id, id') s.t. id has required the call back of id'
    $DIALLED$: rel
- - the set of pairs of subscribers (id, id') s.t. id' has been automatically dialled by id
interface
    $CALL\_BACK(ident, ident)$
- - a subscriber requires the automatic call-back of another one
    INT - - internal activity
    $LIFT(ident)$ - - a subscriber lifts the receiver
auxiliary
    $Available$: $ident$
- - a subscriber is available to be connected iff
- - he has not lifted the receiver and his telephone is not ringing
    $Available(id) =_{\mathrm{def}} id \notin ACT$ **and not exists** id_1 **s.t.** $(id_1, id) \in TRYING$
activity

- **if** $id \in BUSY$ **and not** $Available(id')$ **then**

  $$\mathsf{CB}\colon s \xrightarrow{CALL\_BACK(id,id')} s[TO\_CALL \cup \{(id, id')\}/TO\_CALL]$$

- **if** $(id, id') \in TO\_CALL$ **and** $Available(id)$ **and** $Available(id')$ **then**

  $$\mathsf{ACB}\colon s \xrightarrow{INT} s[DIALLED \cup \{(id, id')\}/DIALLED]\\ [TO\_CALL - \{(id, id')\}/TO\_CALL]$$

- **if** $id \notin ACT$ **and** $Available(id')$ **and** $(id, id') \in DIALLED$ **then**

  $$\mathsf{ANSW\_CB}\colon s \xrightarrow{LIFT(id)} s[ACT \cup \{id\}/ACT]\\ [DIALLED - \{(id, id')\}/DIALLED]\\ [TRYING \cup \{(id, id')\}/TRYING]$$

- **if** $id \notin ACT$ **and** (**not** $Available(id')$) **and** $(id, id') \in DIALLED$ **then**

  $$\mathsf{ANSW\_CB}\colon s \xrightarrow{LIFT(id)} s[ACT \cup \{id\}/ACT]\\ [DIALLED - \{(id, id')\}/DIALLED]\\ [BUSY \cup \{id\}/BUSY]$$

Fig. 2. The automatic call back pre-feature

pre-feature TP[s] =
data REL, REL3
attributes

$HOLD\_CONN$: rel
- - the set of pairs of connected subscribers, whose connection is hold-on

3_CONN: $rel3$ - - the set of the triples of subscribers connected together

$CONN$: rel - - the set of the pairs of connected subscribers

interface

$ON\_HOLD(ident, ident)$
- - a subscriber puts on hold an established connection with another subscriber

$SWITCH(ident)$
- - a subscriber switches the current connection to the hold-on one

$THREE\_WAY(ident)$
- - a subscriber starts a three-way connection with the connected subscriber
- - and with the one whose connection was hold-on

$DOWN(ident)$ - - a subscriber puts down the receiver

$DIAL(ident, ident)$ - - a subscriber dials another one

auxiliary

$Not\_Connected$: $ident$
- - a subscriber is not connected iff he is neither participating in a two-way
- - nor in a three-way connection, nor in a hold-on connection

$Not\_Connected(id)$ **iff** (**not exists** id_1, id_2 s.t.
$(id, id_1, id_2) \in 3\_CONN$ **or** $(id_1, id, id_2) \in 3\_CONN$ **or** $(id_1, id_2, id) \in 3\_CONN$
or $(id, id_1) \in CONN$ **or** $(id_1, id) \in CONN$) **or** $(id_1, id) \in HOLD\_CONN$

activity

- **if** $(id, id') \in CONN$ **or** $(id', id) \in CONN$ **then**
$$\mathsf{H}\colon s \xrightarrow{ON\_HOLD(id,id')} s[HOLD\_CONN \cup \{(id, id')\}/HOLD\_CONN]\\ [CONN - \{(id, id'), (id', id)\}/CONN]$$
- **if** $((id, id') \in CONN$ **or** $(id', id) \in CONN)$ **and** $(id, id'') \in HOLD\_CONN$ **then**
$$\mathsf{SW}\colon s \xrightarrow{SWITCH(id)} s[(CONN - \{(id, id'), (id', id)\}) \cup \{(id, id'')\}/CONN]\\ [(HOLD\_CONN - \{(id, id'')\}) \cup \{(id, id')\}/HOLD\_CONN]$$
- **if** $((id, id') \in CONN$ **or** $(id', id) \in CONN)$ **and** $(id, id'') \in HOLD\_CONN$ **then**
$$\mathsf{T}\colon s \xrightarrow{THREE\_WAY(id)} s[CONN - \{(id, id'), (id', id)\}/CONN]\\ [HOLD\_CONN - \{(id, id'')\}/HOLD\_CONN]\\ [3\_CONN \cup \{(id, id', id'')\}/3\_CONN]$$
- $$\mathsf{DWN}\colon s \xrightarrow{DOWN(id)} s[id \lhd CONN \rhd id/CONN]\\ [id \lhd HOLD\_CONN \rhd id/HOLD\_CONN]\\ [3\_CONN \boxminus id/3\_CONN]$$

- - $r \boxminus id$ is r minus all triples having id as a component

- **if** $Not\_Connected(id)$ **then** $\mathsf{D}\colon s \xrightarrow{DIAL(id,id')} s$

Fig. 3. The three-parts pre-feature

In Fig. 4 we depict a fragment (i.e., just few transitions) of the glts that together with a model for the data (sets of identifiers and binary relations over identifiers) gives the closed semantics of POTS; it shows the execution of a standard call between a subscriber id and another subscriber id'. In this paper, to keep the drawings small enough, we report the transition types only when they are really relevant, and we depict the states by reporting the values of the relevant attributes.

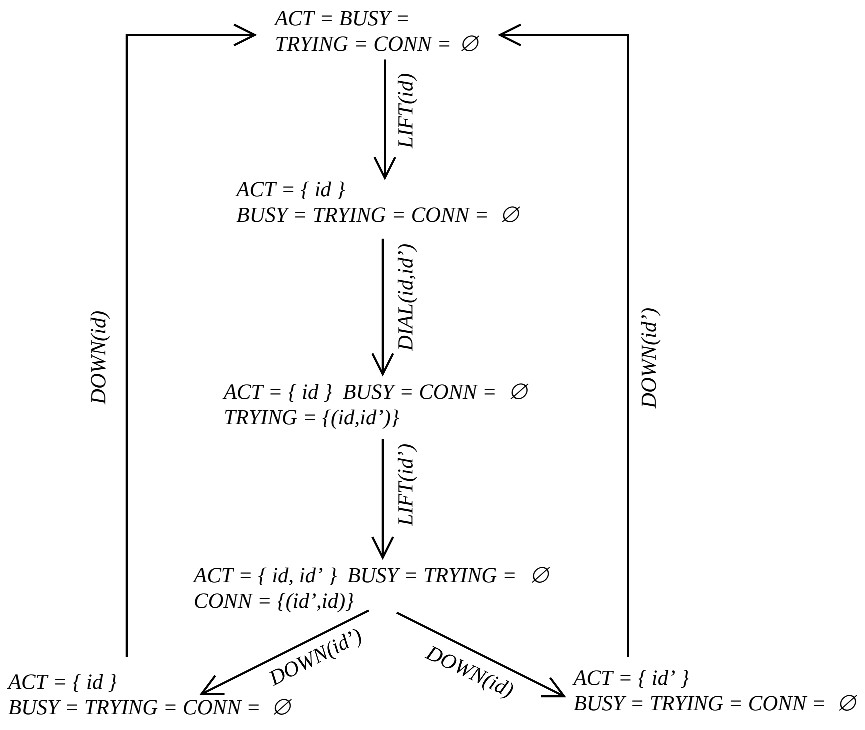


Fig. 4. A fragment of $[\![\mathrm{POTS}]\!]_{\mathrm{C}}$

The closed semantics is already useful to analyse a pre-feature. For example, if we try to analyse TP by examining $[\![\mathrm{TP}]\!]_{\mathrm{C}}$ we can see that a subscriber can put many connections on hold simultaneously; just look at the fragment of $[\![\mathrm{TP}]\!]_{\mathrm{C}}$ reported in Fig. 5. For this reason the effect of a switch action is nondeterministic. Indeed the pre-feature TP has been designed by implicitly assuming that

not exists id_1, id_2, id_3 **s.t.**

$id_1 \neq id_2 \neq id_3$ **and** $(id_1, id_2), (id_1, id_3) \in \mathit{HOLD\_CONN}$

We can guarantee that the above formula holds if we change the rule about putting on hold a call as follows

if $((id, id') \in CONN$ **or** $(id', id) \in CONN)$ **and**
not exists id_1 **s.t.** $((id_1, id) \in HOLD\_CONN$ **or** $(id, id_1) \in HOLD\_CONN)$ **then**

$$\mathsf{H}: s \xrightarrow{ON\_HOLD(id,id')} s[HOLD\_CONN \cup \{(id,id')\}/HOLD\_CONN] \\ [CONN - \{(id,id'),(id',id)\}/CONN]$$

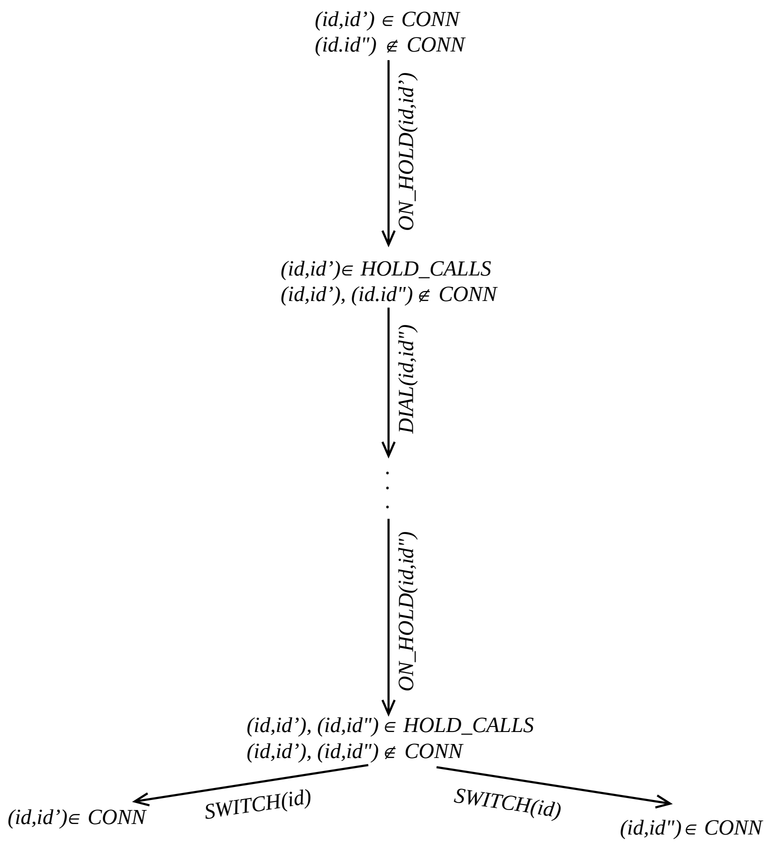


Fig. 5. A fragment of $[\![\mathrm{TP}]\!]_{\mathrm{C}}$

2 Pre-feature Composition and Interaction

2.1 Pre-feature Composition

In our approach adding pre-features is modelled by *composition*, which has nothing to do with parallel composition of processes (here is where the difference w.r.t.

a process-oriented approach comes to light). Indeed in a composition each pre-feature is seen as a part of a system roughly resulting from the union of the two parts following the principle "same name – same thing". We can compose two pre-features (they are composable) only if common names are used coherently in both.

Precisely, two pre-features PF_1 and PF_2 are *composable* iff

- any shared data type is defined in the same way;
- they use the same variables to denote the generic state of the system, the new values of shared attributes and the arguments of the shared label constructors;
- the attributes with the same name have the same result type;
- the label constructors with the same name have the same number of arguments, of the same type and in the same order;
- the auxiliary definitions with the same name coincide;
- for each rule r_1 of PF_1 and r_2 of PF_2 defining transitions with the same type
  * r_1 and r_2 define transitions using the same label constructor;
  * r_1 and r_2 do not share free variables except those denoting the new values of the attributes and the arguments of the label constructors.

It is easy to see that the three pre-features POTS, ACB and TP introduced in the previous section are pairwise composable.

Pre-features correspond to partial descriptions of simple systems; thus composing them means to put together the parts given by both to get a new simple system specification. That is easy for the data part, the attributes, the label constructors and the auxiliary definitions (just the union of the two parts); while since rules correspond to partial descriptions of action capabilities we need in the a mechanism for deciding which rules r_1 of PF_1 and r_2 of PF_2 describe parts of the same action capability, and thus have to be composed. We assume that rules defining transitions with the same type (shortly *rules with the same type*) describe parts of the same action capabilities.

Then the activity part of the composition of the two pre-features contains the rules of PF_1 with a type not present in PF_2, those of PF_2 with a type not present in PF_1 and the *pairwise compositions* of the rules of PF_1 and of PF_2 with a common type. The pairwise composition of two rules with the same type just means to make the conjuction of the premises and the union of the attribute updates (since the two pre-features are composable, the two rules use the same label constructor, the same variables for the state, for the arguments of the label constructor and for the new values of the attributes). Note that the composition of two rules may be a null rule (i.e., a rule which does not generate any transition, since its premises cannot be satisfied). Later on, when extending the composition of pre-features to the composition of pre-features, we will see that in such case the features are considered incompatible and so cannot be composed.

Most importantly we note that in some sense an *anti-frame assumption* underlies our concept of pre-feature composition; indeed the fact that a rule of a pre-feature does not update an attribute does not mean that when we compose such rule with another one that attribute cannot be updated. For example, a rule as

if $A > B$ **then** TT: $s \xrightarrow{L(y)} s[y - A/A]$

where the attribute B is not updated, can be composed with the rule

if $A > B$ **then** TT: $s \xrightarrow{L(y)} s[B - 1/B]$

where A is not updated, resulting in

if $A > B$ **then** TT: $s \xrightarrow{L(y)} s[y - A/A][B - 1/B]$

where both A and B are updated.

Notice also that the composition of two pre-features returns another pre-feature, which can then be further composed with other pre-features.

In the following, we use $\oplus$ to denote the composition operation for pre-features.

Let us illustrate the concepts introduced so far by means of the compositions of the pre-feature POTS with ACB and with TP respectively; we skip the composition of ACB with TP since it is not interesting and that of the three pre-features together for lack of room.

Composing POTS **and** ACB In this case the two pre-features do not share any transition type, and so we do not compose any pair of rules. That is exactly what we want; and it is worthwhile to note that this is due to the use of transition types; for example, if instead we compose rules sharing the same label constructor, then we would get inappropriate results, e.g., a subscriber may lift the receiver only if someone has called him back.

Hence POTS$\oplus$ACB is obtained just by adding the respective parts, keyword by keyword.

Composing POTS **and** TP Much more interesting is the composition of POTS and of TP, which is in Fig. 6, where we have omitted the attribute, interface and auxiliary part, which are obtained just gluing together those of the respective single specifications.

In this case there are some interesting rule compositions, since the two pre-features share the transition types DWN and D. Notice how the composition of the two rules of type DWN puts together the attribute updates; while that of the rules of type D puts together the premises. In both cases the updates and the premises are not disjoint but they agree on the common part and so the compositions are not null rules (both rules update the attribute *CONN*, and both rules ask for the subscriber not to be in a binary connection).

Due to the anti-frame assumption the rule of type D of TP can avoid to repeat the attribute updates (clearly also the conditions) already given in POTS.

The anti-frame assumption allows us also to simplify the rule of type DWN of TP, by dropping the update of the attribute *CONN* from it; the result of the composition will be exactly the same as in Fig. 6. Indeed, the simplified rule

$$\begin{aligned}\mathsf{DWN}: s \xrightarrow{DOWN(id)} & s[id \lhd HOLD\_CONN \rhd id/HOLD\_CONN]\\ & [3\_CONN \boxminus id/3\_CONN]\end{aligned}$$

pre-feature $\mathrm{POTS} \oplus \mathrm{TP}[s] =$
data $\mathrm{SET(IDENT), REL, REL3}$
attributes ... ...
interface ... ...
auxiliary ... ...
activity

- **if** $Available(id)$ **then**
$$\mathsf{START}\colon s \xrightarrow{LIFT(id)} s[ACT \cup \{id\}/ACT]$$
- **if** $id \notin ACT$ **and** $(id', id) \in TRYING$ **then**
$$\begin{array}{ll}\mathsf{ANSW}\colon s \xrightarrow{LIFT(id)} & s[ACT \cup \{id\}/ACT] \\ & [TRYING - \{(id', id)\}/TRYING] \\ & [CONN \cup \{(id, id')\}/CONN]\end{array}$$
- **if** $id \in ACT$ **then**
$$\begin{array}{ll}\mathsf{DWN}\colon s \xrightarrow{DOWN(id)} & s[ACT - \{id\}/ACT] \\ & [BUSY - \{id\}/BUSY] \\ & [id \lhd TRYING/TRYING] \\ & [id \lhd CONN \rhd id/CONN] \\ & [id \lhd HOLD\_CONN \rhd id/HOLD\_CONN] \\ & [3\_CONN \boxminus id/3\_CONN]\end{array}$$
- **if** $Allowed(id)$ **and** (**not** $Available(id')$) **and** $Not\_Connected(id)$ **then**
$$\mathsf{D}\colon s \xrightarrow{DIAL(id,id')} s[BUSY \cup \{id\}/BUSY]$$
- **if** $Allowed(id)$ **and** $Available(id')$ **and** $Not\_Connected(id)$ **then**
$$\mathsf{D}\colon s \xrightarrow{DIAL(id,id')} s[TRYING \cup \{(id, id')\}/TRYING]$$
- **if** $(id, id') \in CONN$ **or** $(id', id) \in CONN$ **then**
$$\begin{array}{ll}\mathsf{H}\colon s \xrightarrow{ON\_HOLD(id,id')} & s[HOLD\_CONN \cup \{(id, id')\}/HOLD\_CONN] \\ & [CONN - \{(id, id'), (id', id)\}/CONN]\end{array}$$
- **if** $((id, id') \in CONN$ **or** $(id', id) \in CONN)$ **and** $(id, id'') \in HOLD\_CONN$ **then**
$$\begin{array}{ll}\mathsf{SW}\colon s \xrightarrow{SWITCH(id)} & s[(CONN - \{(id, id'), (id', id)\}) \cup \{(id, id'')\}/CONN] \\ & [(HOLD\_CONN - \{(id, id'')\}) \cup \{(id, id')\}/HOLD\_CONN]\end{array}$$
- **if** $((id, id') \in CONN$ **or** $(id', id) \in CONN)$ **and** $(id, id'') \in HOLD\_CONN$ **then**
$$\begin{array}{ll}\mathsf{T}\colon s \xrightarrow{THREE\_WAY(id)} & s[CONN - \{(id, id'), (id', id)\}/CONN] \\ & [HOLD\_CONN - \{(id, id'')\}/HOLD\_CONN] \\ & [3\_CONN \cup \{(id, id', id'')\}/3\_CONN]\end{array}$$

Fig. 6. The composition of POTS and TP

does not assert that *CONN* cannot modify its value during a transition partly built by this rule (i.e., by a rule obtained by composing it with another one).

2.2 Pre-Feature Interaction

As already suggested, when organizing a system by features, it is of paramount importance to have a clear picture of the variations which may occur when adding features. On the basis of the concepts introduced before, we are able to single out some basic criteria for reasoning about feature interactions.

Together with many other (but not all) authors by "interaction of pre-feature PF_2 on PF_1" (or "PF_2 interacts with PF_1") we mean that the "part of $\mathrm{PF}_1 \oplus \mathrm{PF}_2$ due to PF_1 is not as specified by PF_1". We assume that "part of $\mathrm{PF}_1 \oplus \mathrm{PF}_2$ due to PF_1" is the projection on the signature of PF_1.

We may have different concepts of interaction between pre-features depending on how we compare $[\![\mathrm{PF}_1]\!]_{\mathrm{C}}$ with $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}}$ projected on PF_1; for example we may consider:

atomic interaction: we compare the set of transitions (corresponding to atomic activities) of the system described by $[\![\mathrm{PF}_1]\!]_{\mathrm{C}}$ and of the one described by $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}}$ projected onto the signature of PF_1;
behavioural interaction: we compare the behaviour of the system described by $[\![\mathrm{PF}_1]\!]_{\mathrm{C}}$, i.e., the labelled transition tree defined by the associated glts, and of the one described by $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}}$ projected onto the signature of PF_1.

Moreover when comparing the transitions/behaviours of two systems we can look at

- transition types, attributes and labels (**TAL**);
- only attributes and labels (**AL**);
- or only labels (**L**).

It is not interesting to look only at the attributes, since we are considering reactive (open) systems in a formal framework where the interface of a system towards the outside world is represented by the labels.

There are various reasons for the transitions/behaviours of $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}}$ projectd onto the signature of PF_1 to be different from those of $[\![\mathrm{PF}_1]\!]_{\mathrm{C}}$; e.g.

- the premises of a rule of PF_1 become more restrictive (conditions on the new attributes) when it is composed with a rule of PF_2;
- a rule of PF_1 when it is composed with one of PF_2 may update attributes of PF_1 previously not modified (anti-frame assumption);
- new rules about PF_1 transition types/labels modifying the PF_1 attributes are added by PF_2.
- new rules about new transition types are added by PF_2 with PF_1 labels and modifying the PF_1 attributes.

Here we do not formally define what is an "interaction", but we just try to illustrate these different views of interactions using the telephonic features introduced before.

Interactions between POTS **and** ACB

POTS and ACB do not atomically **TAL** interact, since they have disjoint sets of transition types.

[Int 1] ACB atomically **AL** interacts with POTS; indeed it is sufficient to consider the transitions of $[\![\text{POTS} \oplus \text{ACB}]\!]_C$ with label constructor *LIFT* obtained by instantiating the following rule of POTS⊕ACB (originated from ACB).

if $id \notin ACT$ **and** $Available(id')$ **and** $(id, id') \in DIALLED$ **then**

$$\begin{aligned}\mathsf{ANSW\_CB}\colon s \xrightarrow{LIFT(id)} & s[ACT \cup \{id\}/ACT] \\ & [DIALLED - \{(id, id')\}/DIALLED] \\ & [TRYING \cup \{(id, id')\}/TRYING]\end{aligned}$$

while no transition of POTS with label $LIFT(id)$ adds more pairs to the attribute *TRYING*.

[Int 2] Consider now the fragment of $[\![\text{POTS} \oplus \text{ACB}]\!]_C$ reported in Fig. 7.

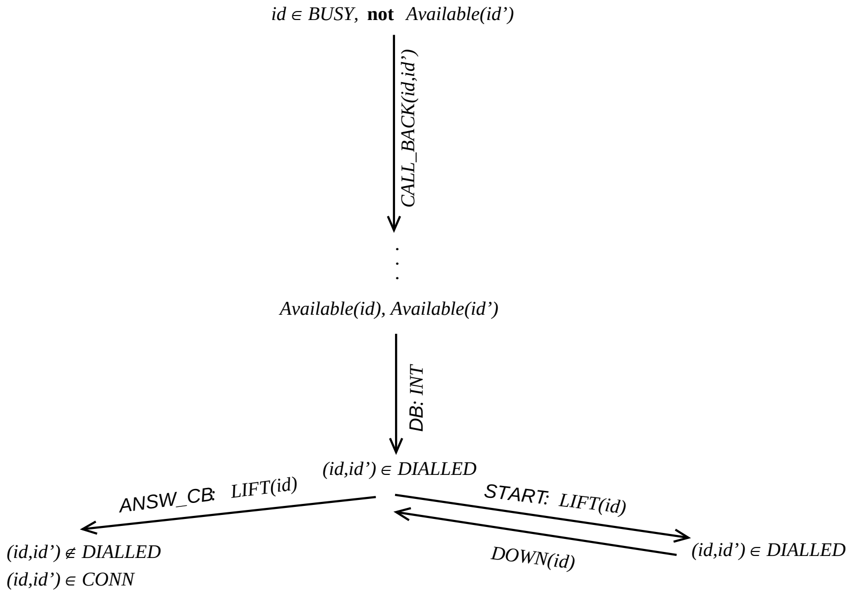


Fig. 7. A fragment of $[\![\text{POTS} \oplus \text{ACB}]\!]_C$

If the system chooses forever the right branch of the behaviour represented in the picture, then we have that forever the phone of *id* will ring (in this specification the telephone of a subscriber *id* is ringing whenever $(id, id') \in DIALLED$); and this is a typical example of interaction in a telephone system.

Technically POTS behaviourally **AL** interacts with ACB; indeed we have in $[\![\mathrm{POTS} \oplus \mathrm{ACB}]\!]_{\mathrm{C}}$ an infinite behaviour where forever $(id, id') \in \mathit{DIALLED}$ and where infinitely many times there is a transition labelled by $\mathit{LIFT}(id)$.

[Int 3] If instead the system takes the left branch in the picture, then we have in $[\![\mathrm{POTS} \oplus \mathrm{ACB}]\!]_{\mathrm{C}}$ a behaviour not present in $[\![\mathrm{POTS}]\!]_{\mathrm{C}}$, where between a state where id is busy and a state where id and id' are connected, there is always a transition labelled with $\mathit{DIAL}(id, id')$.

Technically ACB behaviourally **AL** interacts with POTS.

Interactions between POTS **and** TP

[Int 4] POTS atomically **AL** interacts with TP; indeed it is sufficient to consider the transitions labelled by *DOWN*, which in $[\![\mathrm{TP}]\!]_{\mathrm{C}}$ are present in any state, while in the composition a subscriber may put down the receiver only if he is active.

[Int 5] POTS and TP do not behaviourally **L** interact; indeed TP just adds new parts of behaviours but using mainly its private labels.

3 Features

3.1 Interaction requirements

It is not true that any case of interaction is negative and must be prohibited.

Consider, for example:

- a feature for a telephone system which offers a discount whenever some particular toll-free numbers are called (transition types/transitions which did not change the debt of a user, now decrease it) (**TAL** and **AL** atomic interaction);
- a feature cutting the telephone service for a subscriber when the debt exceeds some amount;
- a feature adding to a lift an external security block mechanism stopping the cabin to the nearest floor and opening the doors; in some case the lift has a new possibility of behaviour made by old labels and using old attributes (**AL** and **L** behavioural interaction);
- a feature adding the possibility of randomly winning a free recharge of a phonecard; now there are behaviours where the action of recharge is not preceded by the action pay (**TAL** behavioural interaction).

Let us consider again the telephone examples at the end of the previous section.

[Int 1] Good interaction (good means wanted, useful); indeed it is essential to have these new action capabilities acting differently also on the old attributes and label constructors.

[Int 2] This is really a bad unwanted interaction.

[Int 3] Good interaction.

[Int 4] Luckily the POTS interaction corrects a kind of error in TP, where not lifted receivers may be put down.

Thus we are naturally led to add to a pre-feature specification a set of constraints describing explicitly which interactions are good and which are not. We call those constraints *interaction requirements.*

How to express the interaction requirements ?

We choose to use logic formulae expressing the wanted properties of the transitions/behaviours of the composition of the feature with any other one. Now the problem is to choose an "appropriate" logic. We suggest to find a logic able to *discipline* the interaction (of some kind): for all pre-features PF_1, PF_2 s.t. PF_2 interacts with PF_1 there exists a formula of the logic ϕ s.t. $[\![\mathrm{PF}_1]\!]_{\mathrm{C}}$ satisfies ϕ and $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}}$ does not satisfy ϕ.

For example, atomic **TAL** interaction can be disciplined by the following subset of first-order formulae.

atomic safety formulae

if $tt\colon s \xrightarrow{l} s'$ **then** *cond*

where tt, s, l, s' are variables of the appropriate types, *cond* is a first-order formula on tt, s, l and s', built using only the signature of the basic data structures, the operations extracting the value of the attributes from the states and the label constructors.

atomic liveness formulae

if *cond* **then** $(\exists tt\,.)(\exists l\,.)(\exists s'\,.)tt\colon s \xrightarrow{l} s'$

where tt, s, s', l and *cond* as above, and any of the existential quantifications may be omitted.

Also in these formulae we abbreviate $A(s)$, where A is an attribute name, to A and $A(s')$ to A'.

Instead, atomic **AL** interaction may be disciplined by the subset of the safety and liveness atomic formulae, where tt does not occur in *cond* and is always existentially quantified.

The most used interaction requirements just impose that a part of a pre-feature is protected in some way by the interactions of the other features; in the telephone examples we just need to require that a transition type/an attribute/a label constructor of a feature cannot be modified by any added feature, shortly it is "stable". Such particular requirements may be expressed by using atomic safety formulae, see below; but for sake of clarity we will simply write "... **is stable**.

A **is stable** requires that any added feature cannot change the way the attribute A is updated. If the rules of the feature in which the attribute A is updated are

$$\textbf{if } cond_j \textbf{ then } \mathsf{TT}_j\colon s \xrightarrow{l_j} s[v_j/A]... \qquad (j = 1, \ldots, k)$$

A **is stable** stands for

$$\textbf{if } tt\colon s \xrightarrow{l} s' \textbf{ and } A \neq A' \textbf{ then } case_1 \textbf{ or } \ldots \textbf{ or } case_k$$

where $case_j$ is the existential closure w.r.t. all variables different from l, v_j, s of

$$cond_j \textbf{ and } tt = \mathsf{TT}_j \textbf{ and } l = l_j \textbf{ and } A' = v_j.$$

L **is stable** requires that any added feature cannot change the effects/the blocking conditions on the feature attributes of transitions labelled by L and the possibility of transitions labelled by L to have some type of that feature. If the rules of the feature in which the label constructor L appears are

$$\textbf{if } cond_j \textbf{ then } \mathsf{TT}_j : s \xrightarrow{L(y_1,\dots,y_n)} s[v_1^j/A_1]\dots \qquad (j = 1,\dots,k)$$

L **is stable** stands for

$$\textbf{if } tt\colon s \xrightarrow{L(y_1,\dots,y_n)} s' \textbf{ then } case_1 \textbf{ or } \dots \textbf{ or } case_k$$

where $case_j$ is the existential closure w.r.t. all variables different from y_1, ..., y_n, v_1^j, ..., s of

$$cond_j \textbf{ and } tt = \mathsf{TT}_j \textbf{ and } A'_1 = v_1^j \textbf{ and } \dots.$$

TT **is stable** requires that any added feature cannot change how the transitions with type TT use the feature attributes and label constructors. If the rules of the feature in which TT appears are

$$\textbf{if } cond_j \textbf{ then } \mathsf{TT}\colon s \xrightarrow{L(y_1,\dots,y_n)} s[v_1^j/A_1]\dots \qquad (j = 1,\dots,k)$$

TT **is stable** stands for

$$\textbf{if } \mathsf{TT}\colon s \xrightarrow{L(y_1,\dots,y_n)} s' \textbf{ then } case_1 \textbf{ or } \dots \textbf{ or } case_k$$

where $case_j$ is the existential closure w.r.t. all variables different from y_1, ..., y_n, v_1^j, ..., s of

$$cond_j \textbf{ and } A'_1 = v_1^j \textbf{ and } \dots.$$

As examples, we report below the expanded version of the stability interaction requirement TO_CALL **is stable** for the pre-feature ACB of Fig. 2.

if $tt\colon s \xrightarrow{l} s'$ **and** $TO\_CALL \neq TO\_CALL'$ **then**
(**exists** id, id' s.t. $id \in BUSY$ **and** (**not** $Available(id')$) **and** $tt = \mathsf{CB}$ **and**
$l = CALL\_BACK(id, id')$ **and** $TO\_CALL' = TO\_CALL \cup \{(id, id')\}$)
or
(**exists** id, id' s.t. $(id, id') \in TO\_CALL$ **and** $Available(id)$ **and** $Available(id')$
and $tt = \mathsf{ANSW\_CB}$ **and** $l = INT$ **and** $TO\_CALL' = TO\_CALL - \{(id, id')\}$
and $DIALLED' = DIALLED \cup \{(id, id')\}$))

3.2 Feature Specifications

According to the preliminary discussion, a *feature specification* is a pair consisting of a pre-feature specification and a set of interaction requirements. If F is a feature, then PF will denote its pre-feature part.

Now we have to provide a semantics not only taking into account the interaction requirements, but also supporting the composition operation, where what we call "anti-frame assumption" is playing a major role. Indeed we have to express the fact that F is a "partial description" of a system, which will result in the end, possibly after the addition of other features. To this end we introduce the rather novel concept of *open semantics*, which is the class of all simple systems (glts's modelling them) having at least all parts specified by F. These systems may have more data, attributes, label constructors and transition types than those of F, and different transitions; technically, their signature is an *extension* of the signature of F. Thus the *open semantics* of F, denoted by $[\![\mathrm{F}]\!]_{\mathrm{O}}$, is the class of all glts over some extension of the signature of PF, satisfying the interaction requirements plus some *default requirements* enforcing that the basic activity defined by the rules is respected modulo an anti-frame assumption concerning the possible update of the attributes.

The default constraint corresponding to an activity rule of the form

$$\textbf{if } \mathit{cond} \textbf{ then } \mathsf{TT}\colon s \xrightarrow{L(y_1,\ldots,y_m)} s[x_1/A_1]\ldots[x_k/A_k]$$

is

if cond **then exists** s' **s.t.**
$A_1(s') = x_1$ **and** $\ldots$ **and** $A_k(s') = x_k$ **and** $\mathsf{TT}\colon s \xrightarrow{L(y_1,\ldots,y_m)} s'$.

Notice that, because of the default requirements, either a feature specification F is inconsistent (no models, i.e., $[\![\mathrm{F}]\!]_{\mathrm{O}} = \emptyset$) or $[\![\mathrm{PF}]\!]_{\mathrm{C}} \in [\![\mathrm{F}]\!]_{\mathrm{O}}$.

Neither the closed nor the open semantics are sufficient, each one, to qualify the semantics of a feature specification. The *complete semantics* of a consistent feature specification F, denoted by $[\![\mathrm{F}]\!]$, is the pair formed by the closed and the open semantics; all inconsistent specifications are obviously considered semantically equivalent (meaningless).

The notion of composition is easily carried over to feature specifications. Two consistent feature specifications, say F_1 and F_2, with composable pre-feature parts PF_1 and PF_2, are *compatible* iff $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}} \in [\![\mathrm{F}_1]\!]_{\mathrm{O}} \cap [\![\mathrm{F}_2]\!]_{\mathrm{O}}$. Intuitively this means that not only both interaction requirements parts are satisfied (and thus they are not inconsistent), but also that the activity rules of the two were not conflicting (both default requirements are satisfied). Then the composition $\mathrm{F}_1 \oplus \mathrm{F}_2$ is consistent and thus also $[\![\mathrm{PF}_1 \oplus \mathrm{PF}_2]\!]_{\mathrm{C}} \in [\![\mathrm{F}_1 \oplus \mathrm{F}_2]\!]_{\mathrm{O}}$. It is not difficult to see that the composition is well-defined w.r.t. the semantics: if $[\![\mathrm{F}_1]\!] = [\![\mathrm{F}_1{}']\!]$ then $[\![\mathrm{F}_1 \oplus \mathrm{F}_2]\!] = [\![\mathrm{F}_1{}' \oplus \mathrm{F}_2]\!]$.

Now we show how to transform the pre-features for the telephone systems introduced before into features, by adding appropriate interaction requirements.

POTS **as a feature** A telephone system is intended to have the following basic properties

– if a subscriber lifts the receiver, then he must become active,
– a subscriber can put down the receiver only if he is active, and then becomes inactive

and is designed assuming that they hold; so we impose that any added feature preserves them, by the following two safety formulae.

if $tt\colon s \xrightarrow{LIFT(id)} s'$ **then** $id \in ACT'$

if $tt\colon s \xrightarrow{DOWN(id)} s'$ **then** $id \in ACT$ **and not** $id \in ACT'$

As an example, we give in Fig. 8 a fragment of an element ALARM of $[\![\text{POTS}]\!]_{\text{O}}$ different from $[\![\text{POTS}]\!]_{\text{C}}$, corresponding to extend POTS with a telephone alarm clock service.

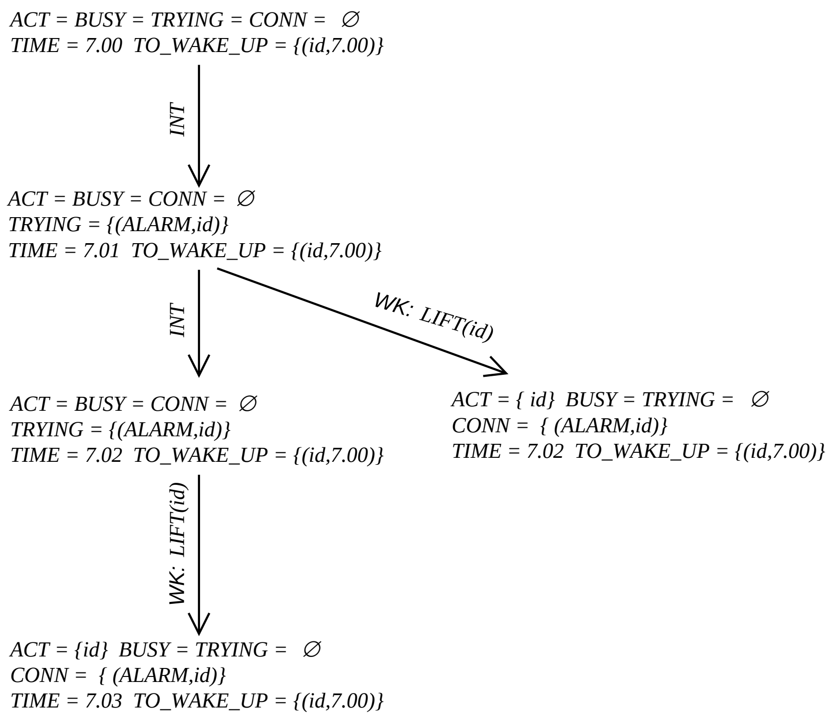

ACT = BUSY = TRYING = CONN = ∅
TIME = 7.00 TO_WAKE_UP = {(id,7.00)}
INT
ACT = BUSY = CONN = ∅
TRYING = {(ALARM,id)}
TIME = 7.01 TO_WAKE_UP = {(id,7.00)}
INT
WK: LIFT(id)
ACT = BUSY = CONN = ∅
TRYING = {(ALARM,id)}
TIME = 7.02 TO_WAKE_UP = {(id,7.00)}
ACT = { id} BUSY = TRYING = ∅
CONN = { (ALARM,id)}
TIME = 7.02 TO_WAKE_UP = {(id,7.00)}
WK: LIFT(id)
ACT = {id} BUSY = TRYING = ∅
CONN = { (ALARM,id)}
TIME = 7.03 TO_WAKE_UP = {(id,7.00)}

Fig. 8. A fragment of ALARM

ALARM has new attributes (the time and the list of the subscribers to be waken up with the corresponding time), new label constructors (*INT* corresponding to internal activity, i.e., null interaction with the external world) and new transitions (those labelled by *INT* and those of type WK).

ACB **as a feature** We first impose on ACB a set of rather standard interaction requirements, listed below, saying that in some sense the use of the attributes, label constructors and transition types for handling of the automatic call back cannot be redefined by a compatible feature. Clearly compatible features may extend the stable labels and transition types to act on new attributes; e.g., if we consider a billing feature, then an automatic call back may increase the bill.

$\mathit{TO\_CALL}$ **is stable**
$\mathit{CALL\_BACK}$ **is stable**
CB **is stable**
$\mathit{DIALLED}$ **is stable**
$\mathsf{ANSW\_CB}$ **is stable**
ACB **is stable**

If instead we want to discipline the interaction with POTS **[Int 2]** (see Sect. 2.2), then we introduce also the interaction requirement

if $tt\colon s \xrightarrow{LIFT(id)} s'$ **and** $(id, id') \in \mathit{DIALLED}$ **then** $tt = \mathsf{ANSW\_CB}$.

But now POTS and ACB considered as features are not compatible anymore, due to the last interaction requirement. If we need to compose them, then we have to modify one of the two; in this case it seems sensible to add to ACB some rules expressing that transition of type START and ANSW can be done only by subscribers that have not automatically called some other one, as

if not exists id' **s.t.** $(id, id') \in \mathit{DIALLED}$ **then**
$\quad\mathsf{START}\colon s \xrightarrow{LIFT(id)} s'$
if not exists id' **s.t.** $(id, id') \in \mathit{DIALLED}$ **then**
$\quad\mathsf{ANSW}\colon s \xrightarrow{LIFT(id)} s'$

TP **as a feature** To get a feature we add to TP the following interaction requirements:

SWITCH **is stable**
$\mathit{THREE\_WAY}$ **is stable**
$\mathit{ON\_HOLD}$ **is stable**

We do not make stable the attributes $\mathit{HOLD\_CONN}$ and $\mathit{3\_CONN}$, since it is sensible that can be used by other features, e.g., to handle urgent calls which automatically make a running call put on hold.

It is easy to see that the feature TP is compatible with POTS.

4 Conclusion and Related Work

We have illustrated a rather general framework for feature-oriented development. Somewhat differently than in other approaches, our aim is to provide a flexible discipline for handling features, more than just checking the interactions. The flexibility is provided by factorizing the specification development: features in isolation as pre-features with closed semantics, composition of pre-features and analysis of the variety of interactions, interaction requirements, and finally full feature specifications with open and complete semantics. We have also introduced some novel concepts like open semantics with the underlying anti-frame assumption, which we believe is a key issue in combining features.

In another paper [2] we provide the technical details (formal definitions, properties).

The approach we have presented can be used at two levels: one, methodological, which can be reified also using other technical formalisms, and another which adopts and exploits some specific technique [1]. We plan to turn the second level into a formalism for handling interactions within the COFI initiative (see [8] and `http://www.brics.dk/Projects/CoFI`).

In our opinion the general framework should be adapted to the particular domain-specific application, as it is supported by the work of P. Zave on telephone systems [12].

It is worthwhile mentioning that the useful graphical representations are really possible not only for the simple examples considered in the paper; indeed there is a way of presenting graphically design specifications for reactive and concurrent systems (see [10]), which is adjustable to the case of features, as we plan to do in some further work. As for the automatic generation of the graphical representations, it does not seem out of the state-of-the-art, though not yet explored by us.

Together with improving the graphical representation, our ongoing work aims first of all at dealing with features for structured systems, i.e. systems made of subsystems; in other words we want to have at hand both component and feature modularity.

Recently some papers trying to study features and their interactions on a formal ground have started to appear, but none of them presents something similar to our "interaction requirements"; moreover it seems that the role of concepts like anti-frame assumption and open semantics, not even in an implicit form, has not been noted.

Among them we recall [3], presenting feature specifications based on logical formulae conceptually similar to our rules, but their idea of "feature composition" and of "interaction" is really less flexible than our (e.g., using our terminology transitions are composed only when have the same label, and interaction is just atomic). In e.g., [4] Bredereke, trying to give a formal view of features and interactions, considers the importance of the behavioural aspects. Also [5] presents a formal based treatment of features for telecommunication systems, but at a more concrete level (i.e. more oriented towards the implementation) and so it is difficult it to fully relate to our work.

Prehofer considers both methodological aspects as in [9] and formal aspects, as in [6], where he presents an approach based on transition systems (diagrams); but differently from our work, for him to add a feature to a system means to refine graphically a part of the diagram specifying it. It is interesting to note that our framework may offer a formal counterpart to part of his work in [9] including the "lifters", i.e. feature modifiers for helping to resolve feature interactions.

References

1. E. Astesiano and G. Reggio. Labelled Transition Logic: An Outline. Technical Report DISI–TR–96–20, DISI – Università di Genova, Italy, 1996.
2. E. Astesiano and G. Reggio. Feature Interaction: Prevention Is Better than Detection: A Formal Modular Support for Feature Specification and Interaction Handling. Technical Report DISI–TR–98–14, DISI – Università di Genova, Italy, 1998.
3. J. Blom, R. Bol, and L. Kempe. Automatic Detection of Feature Interactions in Temporal Logic. Technical Report 95/61, Department of Computer Systems, Uppsala University, 1995.
4. J. Bredereke. Formal Criteria for Feature Interactions in Telecommunications Systems. In J. Norgaard and V. B. Iversen, editors, *Intelligent Networks and New Technologies*. Chapman & Hall, 1996.
5. M. Faci and L. Logrippo. Specifying Features and Analyzing their Interactions in a LOTOS Environment. In L.G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems (Proc. of the 2nd International Workshop on Feature Interactions in Telecommunications Systems, Amsterdam)*, pages 136–151. IOS Press, 1994.
6. C. Klein, C. Prehofer, and B. Rumpe. Feature Specification and Refinement with State Transition Diagrams. In P. Dini, editor, *Fourth IEEE Workshop on Feature Interactions in Telecommunications Networks and Distributed Systems*. IOS-Press, 1997.
7. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
8. P.D. Mosses. CoFI: The Common Framework Initiative for Algebraic Specification and Development. In M. Bidoit and M. Dauchet, editors, *Proc. TAPSOFT '97*, number 1214 in Lecture Notes in Computer Science, pages 115–137, Berlin, 1997. Springer Verlag.
9. C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of ECOOP'97*, number 1241 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 1997.
10. G. Reggio and M. Larosa. A Graphic Notation for Formal Specifications of Dynamic Systems. In J. Fitzgerald and C.B. Jones, editors, *Proc. FME 97 - Industrial Applications and Strengthened Foundations of Formal Methods*, number 1313 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 1997.
11. P. Zave. Feature interactions and formal specifications in telecommunications. *Computer*, 26(8):20–29, 1993.
12. P. Zave. Calls considered harmful and other observations: A tutorial on telephony. In T. Margaria, editor, *Second International Workshop on Advanced Intelligent Networks '97*, 1997.