A Refinement Relation for Families of Timed Automata

Guillermina Cledou, José Proença, and Luís S. Barbosa

HASLab INESC TEC – Univ. Minho, Portugal
mgc@inesctec.pt, {jose.proenca,lsb}@di.uminho.pt

Abstract. Software Product Lines (SPLs) are families of systems that share a high number of common assets while differing in others. In component-based systems, components themselves can be SPLs, i.e., each component can be seen as a family of variations, with different interfaces and functionalities, typically parameterized by a set of *features* and a *feature model* that specifies the valid combinations of features. This paper explores how to safely replace such families of components with more refined ones. We propose a notion of refinement for Interface Featured Timed Automata (IFTA), a formalism to model families of timed automata with support for multi-action transitions. We separate the notion of IFTA refinement into *behavioral* and *variability* refinement, i.e., the refinement of the underlying timed automata and feature model. Furthermore, we define behavioral refinement for the semantic level, i.e., transition systems, as an alternating simulation between systems, and lift this definition to IFTA refinement. We illustrate this notion with examples throughout the text and show that refinement is a pre-order and compositional.

Keywords: Software Product Lines; Refinement; Timed Automata

1 Introduction

A Software Product Line (SPL) is a set of software systems that share a high number of *features* while differing in others, where concrete configurations are derived from a core of common assets in a prescribed way. A feature is referred as a characteristic of the system visible to the user. A concrete configuration of the SPL results in a particular product and is given by a selection of features. The set of all valid feature selections, i.e., the products that can be derived from the SPL, is determined by a *feature model*.

As in the development of any complex system, common and variable assets of an SPL, such as software components, can be designed and developed by different engineers agreeing on a common specification of what their interfaces should be. In this sense, being able to reason about how standalone components, and in this case *families* of components, *implemented* separately satisfy a given *specification* becomes crucial. In this paper, we propose a notion of *refinement* for real timed software product lines that are modeled as *Interface Featured Timed Automata* (IFTA), a formalism to model families of timed automata. We introduced

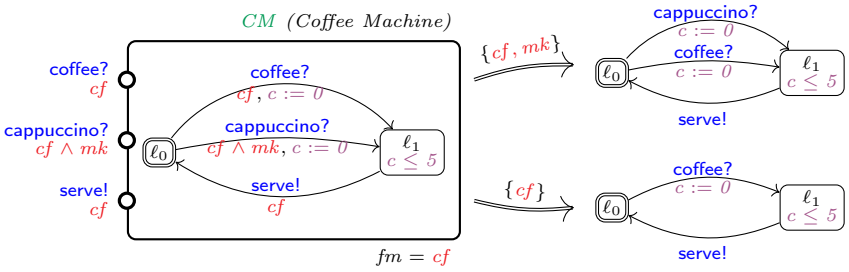

Fig. 1. Example of an IFTA representing a family of coffee machines (left), and its two projections into concrete products, represented as Timed Automata (right).

IFTA in [7] as an extension to Featured Timed Automata, in turn introduced by Cordy et. al. [8], which incorporates interfaces in order to reason about variability during composition and prepare the way to reason about refinement. Figure 1 shows an example of an IFTA representing a family of coffee machines (left), and its corresponding projections into its concrete products (right), a coffee machine that serves coffee and cappuccino (top right) and a coffee machines that serves only coffee (bottom right), both represented as Timed Automata (TA). Projections are obtained by selecting a valid feature selection. The necessary background on (Interface Featured) Timed Automata is introduced in Section 2. Briefly, an IFTA is a TA with: logic guards over transitions, restricting the set of products where the transitions are present; a logic guard associated to the automaton representing the feature model; interface actions representing communication points with other automata; and inferred logic guards associated to interfaces, indicating the set of products where the interface action was designed to be present in.

Refinement allows us to compare two models of the same system presented at different levels of abstraction. The most abstract one is referred to as the *specification*, while the most detailed one is referred to as an *implementation* of the system. If an implementation refines the specification, it agrees with the requirements of the specification in the sense that one may replace the implementation in any context where the specification is used, and still obtain an equivalent system. However, since we are dealing with families of components, we need to reason about how a *set of implementations* refine a *set of specifications*. Figure 2 exemplifies this problem. The figure shows two composed systems (top): one (top left), composed by an IFTA C, representing a context (here left undefined), and an IFTA CM corresponding to the coffee machine in Figure 1; and the other (top right), which is a refinement of the system on the left, is composed by the same context C, and a new family of coffee machines CM' (defined in Figure 4). Because refinement is compositional (up to some pre-conditions), as will be discussed in Section 3, it suffices to verify if CM' refines CM (in addition to such pre-conditions). However, both IFTA, CM and CM', are actually families of components which model different concrete automata, as depicted

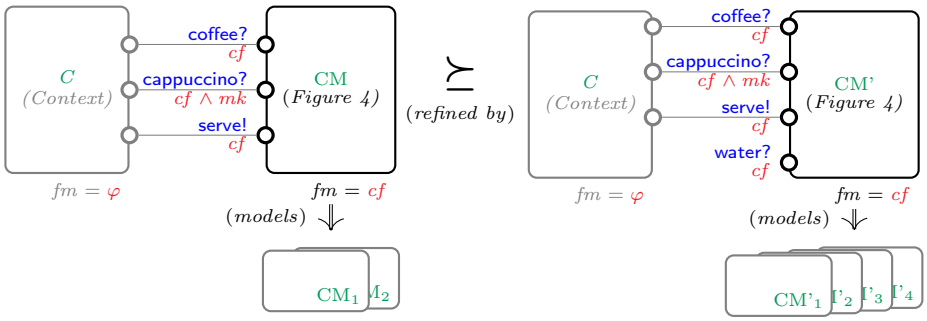

Fig. 2. Example scenario when reasoning about refinement of families of components. A system composed by two IFTA, C and CM (left), is refined by a more detailed system composed by the same IFTA C and a new IFTA CM' (right).

in Figure 2 (bottom). Thus, we need to consider if each of the new automaton CM'_i, for $i = 1, \ldots, 4$ refines an automaton CM_j, for $j = 1, 2$.

In order to simplify this reasoning and allow greater flexibility we separate the notion of refinement into *variability refinement* – which deals with feature model refinement, i.e., when is a set of features considered a refinement of another one; and *behavioral refinement* – which captures timed automata refinement, i.e., when a specific system refines another one. Refinement of timed automata is defined in terms of refinement of timed transition systems, their semantic representation.

There are not many publications in the literature that explore the notion of a refinement relation between two feature models. In [10] the authors reason about four kinds of relations between feature models. However, we believe neither of these aligns with the notion of refinement. Intuitively, a feature model refines another one if it preserves its variability, i.e., allows the same set of products, and such that new variability can only be defined in terms of new features.

There exist various notions of automata refinement in the literature, differing on requirements made over the set of actions of the systems being compared, properties inherent to the models used to specify the systems, and properties that the relation should preserve, among others. Commonly, when dealing with closed systems, i.e., systems that do not interact with the environment through inputs or outputs, refinement is defined as a simulation relation [4]. The advantage is that it preserves all safety properties from the specification. However, when dealing with open systems, as in our case, simulation is a too strict relation, since it requires the implementation to have the same or less inputs than the specification. On the one hand, this means that a refinement can not incorporate new behavior in terms of new inputs, which would not be a problem since it would imply no behavioral changes in the resulting system, provided that we can guarantee that the new inputs are not used. On the other hand, it allows the refinement to have less inputs than the specification. But, in the case

of reactive systems, we can not replace a system for another that reacts to less inputs than the original one. This would limit the behavior of the system, since there will be output actions that are now not capture by the system, but are left unattended. Then, when dealing with open systems it is common to define refinement in terms of an *alternating simulation* relation [3,2,1,9], in which the implementation must simulate all input behavior of the specification, while the latter must simulate all output behavior from the implementation. For example, de Alfaro et. al. [2] introduce Interface Automata, without time, and define the notion of refinement in terms of alternating simulation, extended to support internal steps, i.e., internal actions from both automata which are independent from each other. In [9] David et. al. provide a complete specification theory for Timed I/O Automata where they define refinement, logical conjunction, structural composition, and a quotient operator. However, their theory is based on input enabled automata.

Our notion of refinement can be seen as an adaptation of [2] for families of timed systems with support for multi-action transitions. We as well define refinement as an alternating simulation, however, we relax some of the requirements as discussed in Section 3.2. We show that refinement is a pre-order and congruent with respect to IFTA operations, meaning refinement is compositional.

The rest of this document is structured as follows. Section 2 presents the required theory to understand IFTA. Section 3 proposes a refinement relation for IFTA. Finally, Section 4 concludes and hints on future work.

2 Interface Featured Timed Automata

Interface Featured Timed Automata is a mechanism introduced in [7] to enrich Featured Timed Automata (FTA) [8] with (1) interfaces that restrict the way multiple automata interact, and (2) transitions labelled with multiple actions that simplify the design of synchronous coordination. Interfaces are input-output synchronisation actions that can be linked to interfaces of other automata when composing automata in parallel.

First, we recall some basic notions of timed automata and variability, and the definition of IFTA. Then, we deconstruct IFTA into another formalisms, namely, Interface Transition Systems (ITS), and Interface Featured Transition Systems (IFTS), upon which we base the definition of refinement proposed here. Finally, we explain IFTA operations and their properties.

2.1 IFTA Preliminaries

Informally, an IFTA is an automaton whose *edges* are enriched with *clocks*, *clock constraints* (CC), *feature expressions* (FE), and *multiple synchronisation actions*. A *clock* $c \in C$ is a logical entity that captures the (continuous and dense) time that has passed since it was last reset. When a timed automaton evolves over time, all clocks are incremented simultaneously. A *clock constraint* is a logic condition over the value of a clock. A *feature expression* (FE) is a

logical constraint over a set of *features* F. Each feature denotes a unit of variability; by selecting a desired combination of features one can map an IFTA into an (Interface) Timed Automaton. The *synchronization actions* can be *input* or *output* actions, and represent the *interface* of the automaton, i.e., the actions through which an automaton can communicate with other automata. Each *synchronization action* has associated an inferred feature expression that expresses the valid set of products in which such action was designed to be present. Finally, an IFTA has a special feature expression representing its *feature model*, which imposes restrictions over possible combinations of features.

For example, consider the IFTA CM from Figure 1 (left). It has two locations, ℓ_0 and ℓ_1, with a clock c and two features cf and mk, standing for the support for brewing *coffee* and for including *milk* in the coffee. There are two input actions, coffee?, and cappuccino?, and one output action, serve!, standing, respectively, for the selection of coffee, cappuccino, and the action of serving the beverage. Initially the automaton is in location ℓ_0, indicated by a double-edge node (following UPPAAL[1] real time model checker notation), and it can evolve either by waiting for time to pass (incrementing the clock c) or by taking one of its two transitions to ℓ_1. The top transition, for example, is labelled by the action coffee? and is only active when the feature cf is present. Taking this transition triggers the reset of the clock c back to 0, evolving to the state ℓ_1. Here it can again wait for the time to pass, but for at most 5 time units, determined by the invariant $c \leq 5$ in ℓ_1. The synchronization actions are lifted to the interface of the automaton, depicted with ❍, and associated to the corresponding inferred feature expression. Finally, the lower expression $fm = cf$ defines the *feature model*, i.e., how the features relate to each other. We model this as restrictions, thus, in this case cf is a mandatory feature, however nothing is expressed about mk, meaning it can either be present or absent.

Formally, clock constraints, feature expressions, and IFTA can be defined as follows.

Definition 1 (Clock Constraints (CC), valuation, and satisfaction). *A* clock constraint *over a set of clocks C, written $g \in CC(C)$ is defined by*

$$g ::= c < n \mid c \leq n \mid c = n \mid c > n \mid c \geq n \mid g \wedge g \mid \top \qquad \text{(clock constraint)}$$

where $c \in C$, and $n \in \mathbb{N}$.

A clock valuation *η for a set of clocks C is a function $\eta\colon C \to \mathbb{R}_{\geq 0}$ that assigns each clock $c \in C$ to its current value $\eta(c)$. We use $\mathbb{R}^C$ to refer to the set of all clock valuations over a set of clocks C. Let $\eta_0(c) = 0$ for all $c \in C$ be the initial clock valuation that sets to 0 all clocks in C. We use $\eta + d$, $d \in \mathbb{R}_{\geq 0}$, to denote the clock assignment that maps all $c \in C$ to $\eta(c) + d$, and let $[r \mapsto 0]\eta$, $r \subseteq C$, be the clock assignment that maps all clocks in r to 0 and agrees with η for all other clocks in $C \setminus r$.*

[1] `http://www.uppaal.org`

The satisfaction *of a clock constraint g by a clock valuation η, written $\eta \models g$, is defined as follows*

$$\begin{array}{ll} \eta \models \top & \textit{always} \\ \eta \models c \sqcap n & \textit{if } \eta(c) \sqcap n \\ \eta \models g_1 \wedge g_2 & \textit{if } \eta \models g_1 \wedge \eta \models g_2 \end{array} \qquad \text{(clock satisfaction)}$$

where $\sqcap \in \{<, \leq, =, >, \geq\}$.

Definition 2 (Feature Expressions (FE), satisfaction, and semantics). *A feature expression φ over a set of features F, written $\varphi \in FE(F)$, is defined by*

$$\varphi ::= f \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg \varphi \mid \top \qquad \text{(feature expression)}$$

where $f \in F$ is a feature. The other logical connectives can be encoded as usual: $\bot = \neg\top$; $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$; and $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$.

Given a feature selection $FS \subseteq F$ over a set of features F, and a feature expression $\varphi \in FE(F)$, FS satisfies *φ, noted $FS \models \varphi$, if*

$$\begin{array}{ll} FS \models \top & \textit{always} \\ FS \models f & \Leftrightarrow f \in FS \\ FS \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow FS \models \varphi_1 \textit{ and } FS \models \varphi_2 \\ FS \models \varphi_1 \vee \varphi_2 & \Leftrightarrow FS \models \varphi_1 \textit{ or } FS \models \varphi_2 \\ FS \models \neg \varphi & \Leftrightarrow FS \not\models \varphi \end{array} \qquad \text{(FE satisfaction)}$$

The semantics of a feature expression φ with respect to a set of features F, denoted $[\![\varphi]\!]^F$, is the set of valid feature selections over F that satisfy φ, formally,

$$[\![\varphi]\!]^F = \{FS \subseteq F \mid FS \models \varphi\} \qquad \text{(FE semantics)}$$

Definition 3 (Interface Featured Timed Automata). *An IFTA is a tuple $\mathcal{A} = (L, l_0, A, C, E, Inv, F, fm, \gamma)$ where L is a finite set of locations, l_0 is the initial location, $A = I \uplus O \uplus H$ is a finite set of actions, where I is a set of input ports, O is a set of output ports, and H is a set of hidden (internal) actions, C is a finite set of clocks, $E \subseteq L \times CC(C) \times \mathbf{2}^A \times \mathbf{2}^C \times L$ is the set of edges, $Inv : L \rightarrow CC(C)$ is the invariant, a total function that assigns clock constraints to locations, F is a finite set of features, $fm \in FE(F)$ is a feature model defined as a Boolean formula over features in F, and $\gamma : E \rightarrow FE(F)$ is a total function that assigns feature expressions to edges.*

Notation: when not clear from the context, we will use $L_{\mathcal{A}}, l_{0_{\mathcal{A}}}, A_{\mathcal{A}}, \ldots$ to refer to the elements of an IFTA $\mathcal{A}$, and when using automata names with subscripts such as $\mathcal{A}_1, \mathcal{A}_2, \ldots$, we will simply use $L_1, L_2, l_{0_1}, l_{0_2}, \ldots$. For simplicity, sometimes we write $l \xrightarrow{g,\omega,r}_{\mathcal{A}} l'$ instead of $(l, g, \omega, r, l') \in E_{\mathcal{A}}$, and use $l \xrightarrow[\varphi]{g,\omega,r}_{\mathcal{A}} l'$ to express that $(l, g, \omega, r, l') \in E_{\mathcal{A}}$ and $\gamma_{\mathcal{A}}(l, g, \omega, r, l') = \varphi$.

The *interface* of an IFTA $\mathcal{A}$ is the set $\mathbb{P}_{\mathcal{A}} = I_{\mathcal{A}} \uplus O_{\mathcal{A}}$ of all input and output ports of $\mathcal{A}$. Given a port $\mathsf{p} \in \mathbb{P}_{\mathcal{A}}$ we write $\mathsf{p?}$ and $\mathsf{p!}$ to denote that p is an input or output port, respectively, and write p instead of $\{\mathsf{p}\}$ when clear from context.

Notice that at this point, the definition of IFTA only incorporates the notion of feature expressions associated to transitions through function γ, but does not incorporate the notion of feature expressions associated to interfaces. Before doing this, we define the notion of feature expression of an action. Given an IFTA $\mathcal{A}$, it is possible to infer for each action $\mathtt{a} \in A_{\mathcal{A}}$ a feature expression based on the feature expressions of the edges in which $\mathtt{a}$ appears. Intuitively, this feature expression determines the set of products requiring $\mathtt{a}$. The formal definition follows.

Definition 4 (Feature Expression of an Action). *Given an IFTA $\mathcal{A}$, the inferred feature expression of any action $\mathtt{a}$ is the disjunction of the feature expressions of all of its associated edges, defined as*

$$\widehat{\Gamma}_{\mathcal{A}}(\mathtt{a}) = \bigvee \{\gamma_{\mathcal{A}}(l \xrightarrow{g,\omega,r}_{\mathcal{A}} l') \mid \mathtt{a} \in \omega\} \qquad \text{(FE of an action)}$$

Now we can associate feature expressions to the actions of an IFTA. In order to do this, we incorporate a new function Γ to the definition of an IFTA $\mathcal{A}$, and we say that $\mathcal{A}$ is *grounded*. Thus, given an IFTA $\mathcal{A}$ we can construct a *grounded* $\mathcal{A} = (L_{\mathcal{A}}, l_{0_{\mathcal{A}}}, A_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}}, Inv_{\mathcal{A}}, F_{\mathcal{A}}, fm_{\mathcal{A}}, \gamma_{\mathcal{A}}, \Gamma)$, where $\Gamma : A_{\mathcal{A}} \to FE(F_{\mathcal{A}})$ is a total function that assigns a feature expression to each action of $\mathcal{A}$, and is constructed based on $\widehat{\Gamma}_{\mathcal{A}}$. By doing this association only once, we are *fixing* the feature expressions associated to each action, such that it represents the set of products where each action was originally design to be present in.

The need for this function and for fixing it instead of using directly $\widehat{\Gamma}$ has to do with the way we define the composition of IFTA and the properties that we expect from it. We discuss this in Section 2.3.

2.2 Semantics

The above definition of IFTA, introduced in [7], is built on top of Featured Transition Systems [5] extended with multi-action transitions. This section discusses the decomposition of an IFTA into *Interface Featured Transition Systems* (IFTS) and *Interface Transition Systems* (ITS). These two formalisms can be seen as an infinite transition system semantics for IFTA, and as an IFTS without variability, respectively. The notion of refinement will be presented in Section 3 based on the semantics of IFTA as an IFTS.

We define an ITS as a regular transition system with multi-action transitions and with an interface, i.e., we distinguish between input, output and internal actions. An IFTS is then defined by extending ITS with variability, by incorporating features and a feature model.

Definition 5 (Interface Transition System). *An ITS is a tuple $S = (St, s_0, A, T)$, where St is the set of states, s_0 is the initial state, $A = I \uplus O \uplus H$ is the set of actions where I, O, and H are the set of input, output, and hidden actions, respectively, and $T \subseteq St \times (\mathbf{2}^A \cup \mathbb{R}_{\geq 0}) \times St$ is the transition relation.*

Definition 6 (Interface Featured Transition System). *An IFTS is a tuple $S = (St, s_0, A, T, F, fm, \gamma, \Gamma)$, where St, s_0, A, T are defined as in ITS, F is a set of features, fm is the feature model, $\gamma : T \to FE(F)$, is a total function that assigns feature expressions to transitions, and $\Gamma : A \to FE(F)$, is a total function that assigns feature expressions to actions.*

Notation: As before, when not clear from the context, we will use $St_S, s_{0_S}, A_S, \ldots$ to refer to the elements of an I(F)TS S.

We may now present the formal definition of semantics of a *grounded* IFTA in terms of an IFTS.

Definition 7 (Semantics of an IFTA as an IFTS). *The semantics of a grounded IFTA $\mathcal{A} = (L, l_0, A, C, E, Inv, F, fm, \gamma, \Gamma)$ written $[\![\mathcal{A}]\!]$, is an IFTS $S = (St, s_0, A, T, F, fm, \gamma', \Gamma)$, where $St \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = \langle \ell_0, \eta_0 \rangle$ is the initial state, $T \subseteq St \times (\mathbf{2}^A \cup \mathbb{R}_{\geq 0}) \times St$ is the transition relation, and $\gamma' : T \to FE(F)$ is the total function that assigns feature expressions to transitions in T, both defined as follows.*

$$\langle \ell, \eta \rangle \xrightarrow[\top]{d} \langle \ell, \eta + d \rangle \;\; \textit{if} \;\; \eta \models Inv(\ell) \;\; \textit{and} \;\; (\eta + d) \models Inv(\ell), \quad \textit{for} \;\; d \in \mathbb{R}_{\geq 0} \tag{1}$$

$$\langle \ell, \eta \rangle \xrightarrow[\varphi]{\omega} \langle \ell', \eta' \rangle \quad \textit{if} \;\; \exists \; \ell \xrightarrow[\varphi]{g, \omega, r} \ell' \in E \;\; \textit{s.t.} \;\; \eta \models g, \;\; \eta \models Inv(l), \;\; \eta' = [r \mapsto 0]\eta, \;\; \textit{and} \;\; \eta' \models Inv(\ell') \tag{2}$$

Given a feature selection FS it is possible to *project* an IFTS into an ITS. Only transitions and actions satisfied by FS are preserved by the projection.

Definition 8 (IFTS Projection). *The projection of an IFTS S over a set of features FS is an ITS $S \downarrow_{FS} = (St_S, s_{0_S}, A, T)$, where A and T are defined as*

$$T = \{\; t \in T_S \;\; | \; FS \models \gamma_S(t) \;\; \}$$
$$A = \{\; a \in A_S \;\; | \; FS \models \Gamma_S(a) \;\; \}$$

2.3 Operations on IFTA

Two IFTA can be composed by combining their feature models and linking interfaces, imposing new restrictions over them. The composition is built on top of two operations: *product* and *synchronisation*. The *product* operation for IFTA, unlike the classical product of timed automata, is defined over grounded IFTA with disjoint sets of actions, clocks and features, performing their transitions in an interleaving or synchronous-step fashion.

Definition 9 (Product of IFTA). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$, be two different grounded IFTA with disjoint actions, clocks and features; then, the product of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted $\mathcal{A}_1 \times \mathcal{A}_2$, is*

$$\mathcal{A} = (L_1 \times L_2, \ell_{0_1} \times \ell_{0_2}, A, C_1 \cup C_2, F_1 \cup F_2, E, Inv, fm_1 \wedge fm_2, \gamma, \Gamma)$$

where A, E, Inv, γ and Γ are defined as follows

– $A = I \uplus O \uplus H$, *where* $I = I_1 \cup I_2$, $O = O_1 \cup O_2$, *and* $H = H_1 \cup H_2$.
– E *and* γ *are defined by the rules below, for any* $\omega_1 \subseteq A_1$, $\omega_2 \subseteq A_2$.

$$\frac{\ell_1 \xrightarrow[\varphi_1]{g_1,\omega_1,r_1}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_1]{g_1,\omega_1,r_1} \langle \ell_1', \ell_2 \rangle} \qquad \frac{\ell_2 \xrightarrow[\varphi_2]{g_2,\omega_2,r_2}_2 \ell_2{}'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_2]{g_2,\omega_2,r_2} \langle \ell_1, \ell_2' \rangle}$$

$$\frac{\ell_1 \xrightarrow[\varphi_1]{g_1,\omega_1,r_1}_1 \ell_1' \quad \ell_2 \xrightarrow[\varphi_2]{g_2,\omega_2,r_2}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_1 \wedge \varphi_2]{g_1 \wedge g_2, \omega_1 \cup \omega_2, r_1 \cup r_2} \langle \ell_1', \ell_2' \rangle}$$

– $Inv(\ell_1, \ell_2) = Inv_1(\ell_1) \wedge Inv_2(\ell_2)$.
– $\forall_{\ a \in \mathbb{P}_{\mathcal{A}}} \ \cdot \ \Gamma(a) = \Gamma_i(a)$ *if* $a \in A_i$, *for* $i = 1, 2$.

Both top transitions represent the interleaving of both automata. The bottom transition represents the synchronous execution of transitions from $\mathcal{A}_1$ and $\mathcal{A}_2$, for every combination of outgoing transitions from a state $\ell_1 \in L_1$ and $\ell_2 \in L_2$.

The *synchronisation* operation over an IFTA $\mathcal{A}$ connects and synchronises two actions a and b in $A_{\mathcal{A}}$. The resulting automaton has transitions without neither a and b, nor both a and b. The latter become internal transitions.

Definition 10 (Synchronisation). *Given a grounded IFTA* $\mathcal{A} = (L, \ell_0, A, C, F, E, Inv, fm, \gamma, \Gamma)$ *and two actions* $a, b \in A$, *the synchronisation of* a *and* b *is given by* $\Delta_{a,b}(\mathcal{A}) = (L, \ell_0, A', C, F, E', Inv, fm', \gamma, \Gamma)$ *where* A', E' *and* fm' *are defined as follows*

– $A' = I' \uplus O' \uplus H'$, *where* $I' = I \setminus \{a, b\}$, $O' = O \setminus \{a, b\}$, *and* $H' = H \cup \{a, b\}$.
– $E' = \{\ell \xrightarrow{g,\omega,r} \ell' \in E \mid a \notin \omega \text{ and } b \notin \omega\} \cup$
$\{\ell \xrightarrow{g,\omega \setminus \{a,b\},r} \ell' \mid \ell \xrightarrow{g,\omega,r} \ell' \in E \text{ and } a \in \omega \text{ and } b \in \omega\}$
– $fm' = fm \wedge (\Gamma_{\mathcal{A}}(a) \leftrightarrow \Gamma_{\mathcal{A}}(b))$.

The resulting feature model imposes new restrictions over the set of features based on the actions being synchronised. Intuitively, if two actions a and b are synchronised, they depend on each other. Thus, we require that they should both be present or both absent in any valid set of features. This is done based on Γ which gives us the original set of products in which a and b where design to be present in.

Together, the product and the synchronisation can be used to obtain in a *compositional* way, a complex IFTA built out of primitive ones. The composition of IFTA is made by linking ports and by combining their variability models. Thus, we define the composition of two IFTA as their product, followed by the explicit binding of actions through synchronization. The composition is defined for interface actions synchronized on an input-output fashion only.

Definition 11 (Composition of IFTA). *Given two grounded IFTA,* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *with disjoint set of actions, features and clocks; and a possible empty set of*

bindings $\{(a_1, b_1), \ldots, (a_n, b_n)\}$*, such that for each pair* a_i *and* b_i*, for* $1 \leq i \leq n$*, we have that*

$$(a_i, b_i) \in I_1 \times O_2 \text{ or } (a_i, b_i) \in O_1 \times I_2 \qquad (io\text{-only})$$

then, their composition is a new grounded IFTA defined as follows

$$\mathcal{A}_1 \bowtie_{(a_1,b_1),\ldots,(a_n,b_n)} \mathcal{A}_2 = \Delta_{a_1,b_1} \ldots \Delta_{a_n,b_n}(\mathcal{A}_1 \times \mathcal{A}_2)$$

Figure 3 exemplifies the composition of the coffee machine CM (top right) from Figure 1, and a new IFTA R, representing a *router* (top left), which receives an input i?, and executes simultaneously one of its outputs, if they are present, or receives i? and does nothing if neither output are present. The composition is done by linking the ports $o_1!$ with coffee?, and $o_2!$ with cappuccino?. The resulting IFTA combines the feature models of both IFTA, imposing additional restrictions given by the binded ports, e.g., the binding ($o_1!$, coffee?) imposes that $o_1!$ will be present, if and only if, coffee? is present, which depends on the feature expressions of each port, i.e., $(f_i \wedge f_{o1}) \leftrightarrow cf$. In the composed IFTA, transitions with binded actions transition together, while transitions with non-binded actions (i? and serve!) can transition independently or together.

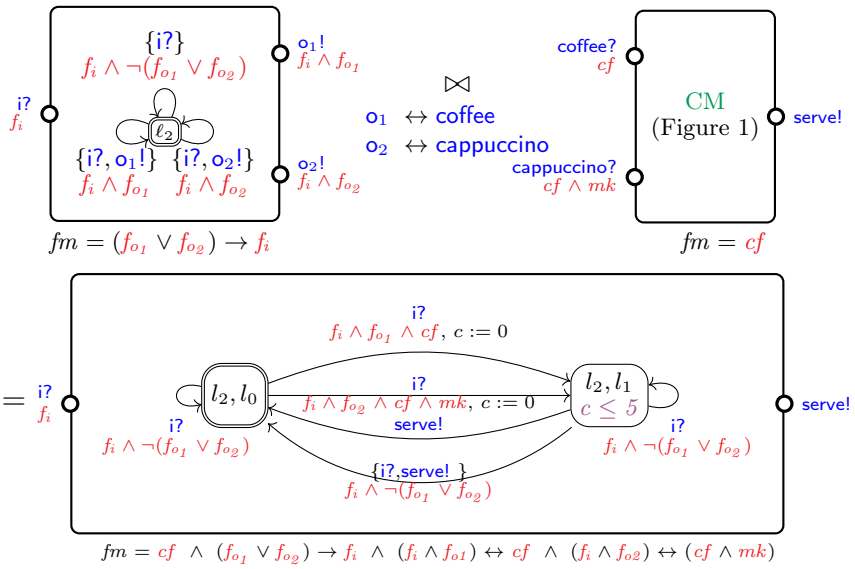

Fig. 3. Composition of an IFTA R (top left), representing a router coordination component, with the IFTA CM (top right), defined in Figure 1, by binding ports (o_1, coffee) and (o_2, cappuccino), yielding the IFTA at the bottom.

Notice that because we define composition as the product followed by the synchronization, the product will produce many transitions that are later *cut* by

the synchronization when linking actions. Then, the order in which actions are linked, and therefore, the order in which transitions are cut by the synchronization operation affects the inferred feature expression of actions. Thus, if we were to use $\widehat{\Gamma}$ instead of Γ, synchronization would not be commutative. By *fixing* the feature expression of an action before doing the product, we avoid this issue and the synchronization remains commutative.

By allowing each IFTA to have its own feature model and taking into account variability during composition, we can reason about how composing families of timed automata in parallel affects the presence of interfaces and the variability of the composed system.

Operations over IFTA satisfy the usual properties up to *strong bisimulation* ($\sim$) and are discussed in [7]. We recall them in the following theorem.

Theorem 1. *Given any IFTA $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$, and actions $a, b, c, d \in A_{\mathcal{A}_1}$, such that a, b, c, d are different actions, the following properties hold.*

$$\mathcal{A}_1 \times \mathcal{A}_2 \sim \mathcal{A}_2 \times \mathcal{A}_1 \qquad (\times\text{-commutativity})$$
$$\mathcal{A}_1 \times (\mathcal{A}_2 \times \mathcal{A}_3) \sim (\mathcal{A}_1 \times \mathcal{A}_2) \times \mathcal{A}_3 \qquad (\times\text{-associativity})$$
$$\Delta_{a,b}\Delta_{c,d}\mathcal{A}_1 \sim \Delta_{c,d}\Delta_{a,b}\mathcal{A}_1 \qquad (\Delta\text{-commutativity})$$
$$(\Delta_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \Delta_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2) \qquad (\Delta \text{ interacts well with } \times)$$

3 Refinement

As mentioned in the introduction, there are two different aspects to be taken into account when discussing a notion of refinement for IFTA. The first concerns refinement of the feature model, which we call *variability refinement*. The second one is refinement of timed automata obtained by projection onto a feature selection, which we call *behavioural refinement*. Thus, refinement of an IFTA will be defined as a refinement of both its feature model and its projections.

3.1 Variability Refinement

Thum et al. [10] recognize four type of relations between two feature models fm_1 and fm_2, based on their set of products, even when their set of features, may not coincide: fm_1 *refactors or is equivalent* to fm_2 if they model the same set of products; fm_1 *specializes* fm_2 if the set of products of fm_1 is a subset of the products of fm_2; fm_1 *generalizes* fm_2 if the set of products of fm_1 is a superset of the products of fm_2; and fm_1 and fm_2 are *arbitrarily* related otherwise.

However, in order to reason about refinement of families of timed automata we also would like to relate feature models in terms of a refinement relation. Intuitively, a feature model fm_1 refines a feature model fm_2 if, when considering the set of features of fm_2, fm_1 expresses exactly the same set of products as expressed by fm_2. Thus, fm_1 can add new variability or details only in terms of new features. Formally, if we consider feature models with only terminal features [10], i.e., no abstract features, we define feature model refinement as follows.

Definition 12 (Feature model refinement). *Given two feature models* $fm_i \in FE(F_i)$ *over a set of features* F_i, $i = 1, 2$, fm_1 *refines* fm_2, *denoted* $fm_1 \sqsubseteq fm_2$, *if and only if,*

$$F_1 \supseteq F_2 \qquad \text{(preserves features)}$$

$$[\![fm_1]\!]^{F_1}|_{F_2} = [\![fm_2]\!]^{F_2} \qquad (fm_1 \text{ refines } fm_2)$$

where $[\![fm]\!]^F|_{F'} = \{FS \cap F' \mid FS \in [\![fm]\!]^F\}$.

For example, if we consider the coffee machines CM and CM' from Figure 4, we have that $[\![fm_{CM}]\!] = \{\{cf\}, \{cf, mk\}\}$ and $[\![fm_{CM'}]\!] = \{\{cf\}, \{cf, wt\}, \{cf, mk\}, \{cf, mk, wt\}\}$. When we restrict $fm_{CM'}$ to only features in F_{CM}, we have $[\![fm_{CM'}]\!]|_{F_{CM}} = \{\{cf\}, \{cf, \cancel{wt}\}, \{cf, mk\}, \{cf, mk, \cancel{wt}\}\} = [\![fm_{CM}]\!]$, where $\cancel{wt}$ means that feature wt is removed from the set. Thus, $fm_{CM'} \sqsubseteq fm_{CM}$. However, let us assume that the feature model of CM' is $fm_{CM'} = cf \wedge (wt \rightarrow mk)$ instead of just cf, then we have $[\![fm_{CM'}]\!]|_{F_{CM'}} = \{\{cf, mk\}, \{cf, mk, \cancel{wt}\}\}$. In this case $fm_{CM'} \not\sqsubseteq fm_{CM}$, since it does not preserves variability by not allowing a coffee machine that only serves coffee.

Theorem 2 ($\sqsubseteq$ is a partial order). *For any feature model* fm_i, *for* $i = 1, 2, 3$, $fm_1 \sqsubseteq fm_1$; *if* $fm_1 \sqsubseteq fm_2$ *and* $fm_2 \sqsubseteq fm_3$, *then* $fm_1 \sqsubseteq fm_3$; *and if* $fm_1 \sqsubseteq fm_2$ *and* $fm_2 \sqsubseteq fm_1$, *then* $fm_1 \equiv fm_2$.

Proof. Immediate by unfolding definitions and set-theoretic properties.

3.2 Behavioral Refinement

Intuitively, an automata $\mathcal{A}$ that refines an automata $\mathcal{B}$ should be able to replace $\mathcal{B}$ in every environment that $\mathcal{B}$ appears in, yielding an equivalent system. Refinement allows to check if a given *implementation* agrees with a *specification*. We consider implementations as automata that are more detailed specifications. Our notion of refinement is similar to the one in [2], where there is an alternating simulation between both automata: $\mathcal{A}$ must simulate all input behavior of $\mathcal{B}$, while $\mathcal{B}$ must simulate all output behavior from $\mathcal{A}$. Thus, $\mathcal{A}$ can allow more legal inputs, and fewer outputs, than $\mathcal{B}$.

Similarly to [9] we define refinement at the semantic level, i.e., IFTS, and then we define refinement of IFTA in terms of IFTS refinement. However, first we define the notion of refinement for Interface Transition Systems, separating the notion of behavioral refinement from variability refinement.

Our notion of refinement can be seen as an extension of [2] for timed systems and multi-action transitions. Here as well, the definition of refinement must consider the fact that both automata have internal actions which are independent from each other. Since we are dealing with timed transition systems, the definition of refinement must consider that internal steps can incorporate delays. Thus, we define a transition relation that captures all transition steps that can be done from a state s to a state s', by any combination of internal and delay steps.

Definition 13. *Given an ITS S and states $s, s' \in St_S$, we write $s \overset{d}{\Longrightarrow}_S s'$ if there is a sequence of transition steps from T_S, such that*

$$\exists\ s \xrightarrow{d_0 \cup \tau}_S s_1 \ldots s_n \xrightarrow{d_n \cup \tau}_S s' \text{ and } d = \sum_{i=0}^{n} d_i$$

where $d_i \in \mathbb{R}_{\geq 0}$, and τ represents any internal action. For simplicity, we write $s \overset{\omega}{\Longrightarrow}^d_S s'$ if there is a sequence of transition steps from T_S, such that

$$\exists\ s \overset{d}{\Longrightarrow}_S s_n \xrightarrow{\omega}_S s'$$

In this context, ITS refinement is defined as follows.

Definition 14 (Refinement of ITS). *Given two ITS, S and T, such that $I_T \subseteq I_S$ and $O_S \subseteq O_T$, S refines T, denoted $S \preceq T$, if and only if, $\exists\ \mathcal{R} \subseteq St_S \times St_T$, such that $(s_0, t_0) \in \mathcal{R}$ and for each $(s,t) \in \mathcal{R}$, we have*

1. *$s \overset{d}{\Longrightarrow}_S s', d \in \mathbb{R}_{\geq 0}$ then $t \overset{d}{\Longrightarrow}_T t'$ and $(s', t') \in \mathcal{R}$, for some $t' \in St_T$*
2. *$s \overset{OI_s}{\Longrightarrow}^d_S s', d \in \mathbb{R}_{\geq 0}, O \neq \emptyset$ then $t \overset{OI_s}{\Longrightarrow}^d_T t'$ and $(s', t') \in \mathcal{R}$ for some $t' \in St_T$*
3. *$t \overset{IO}{\Longrightarrow}^d_T t', d \in \mathbb{R}_{\geq 0}, I \neq \emptyset$ then $s \overset{IO}{\Longrightarrow}^d_S s'$ and $(s', t') \in \mathcal{R}$ for some $s' \in St_S$*

where I_s is either $\emptyset$, or has only inputs shared by both automata, $I_s \subseteq I_T$.

Condition 1 expresses that any delay d allowed by s, possibly through internal steps, must be a delay allowed from t, possibly through internal steps. Condition 2 expresses that any transition with output O, with a possible empty set of inputs $I_s \subseteq I_T$ that are *shared* by both systems, which can be taken from s after a delay d, possibly through internal steps, must simulate a (sequence of) transition(s) from t. In case there is a multi-action transition with outputs and inputs, such that the inputs include new inputs in I_S, is considered as new behavior incorporated by the new inputs, and as such, it is ignored. Condition 3 expresses that any transition with inputs I, with a possible empty set of outputs O, which can be taken from t after a delay d, possibly through internal steps, must be simulated by a (sequence of) transition(s) from s.

In comparison with de Alfaro et. al., we relax some of the requirements made over the states being compared, s and t. In particular, when considering input labeled transitions (condition 3), de Alfaro et. al. defines that s and t are in a refinement relationship, only if, whenever in t is possible to receive and input, s may receive the same input. Here, we require that whenever in t is possible to receive an input within certain time, possibly through a sequence of internal steps, s may receive the same input within the same time, possibly through a series of internal steps.

3.3 IFTA refinement

Before considering refinement for families of timed automata, let us consider refinement for IFTS. Informally, given two IFTS, S and T, S refines T if for each product in S, the projection of S onto such product refines the projection of T onto the same product. However, depending on the relation existing between the set of products of S and T, this can lead to different notions of refinement. Ideally, S should preserve the variability of T, i.e., S should allow exactly the products in T, although it may also increase the set of features and allow more products when considering the new features. Formally, the refinement of IFTS and IFTA are defined as follows.

Definition 15 (Refinement of IFTS). *Given two IFTS, S and T, S refines T, denoted $S \preceq T$, if and only if,*

$$fm_S \sqsubseteq fm_T \qquad \text{(variability refinement)}$$

$$\forall\ FS \in [\![fm_S]\!]^{F_S}\ \cdot\ S\downarrow_{FS}\ \preceq\ T\downarrow_{FS} \qquad \text{(behaviour refinement)}$$

Definition 16 (Refinement of IFTA). *Given two grounded IFTA $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ refines $\mathcal{B}$, denoted $\mathcal{A} \preceq \mathcal{B}$, if and only if, $[\![\mathcal{A}]\!] \preceq [\![\mathcal{B}]\!]$.*

Figure 4 shows an implementation of a family of coffee machines, CM' (right), which refines the IFTA CM (left). The new automaton introduces a new input, water? that depends on a new feature wt which represents the support for serving water. In addition, CM' ensures that coffee is served faster than in *CM*, as indicated by the invariant $c \leq 3$.

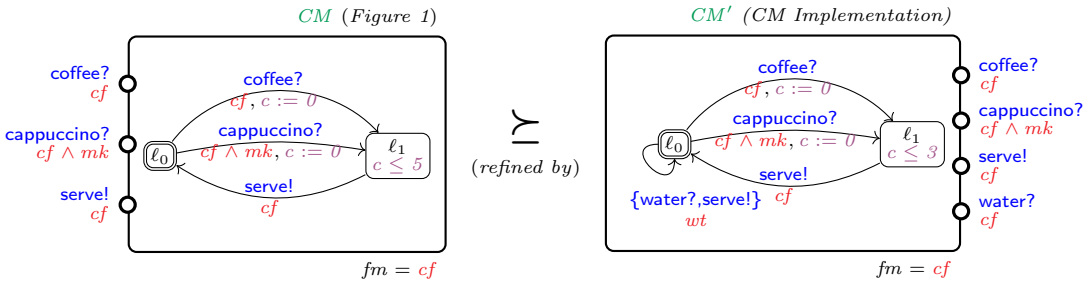

Fig. 4. Example of a family of coffee machines CM' with new variability, interfaces and time restrictions, refining the family CM introduced in Figure 1.

Figure 5 shows a more complex example of refinement incorporating internal actions. The IFTA on the left, P_1, specifies a payment system using PayPal, and is part of a larger system composed of various automata, which models a family of licensing services introduced in [7]. The IFTA on the right, P_2, represents a more detailed implementation of P_1. The specification requires that whenever

the user makes a payment through PayPal, the system will issue an error or a success signal in less than ten units of time. The implementation deals with the actual login into PayPal and confirmation of the payment. In P_2, after the user requests to issue a payment through PayPal, the user must login within 5 units of time, or an error is issued. The log in can be successful or it can issue an error in less than one unit of time. In case the user logs in successfully, a confirmation of the payment must be issued in less than one unit of time after which the system issues a signal of error or success. Both, P_1 and P_2, share the same feature model. In addition, P_2 guarantees that whenever a payment is made through PayPal, the system will issue an error or success signal in less than seven units of time, satisfying the requirements of P_1. Thus, $P_2 \preceq P_1$.

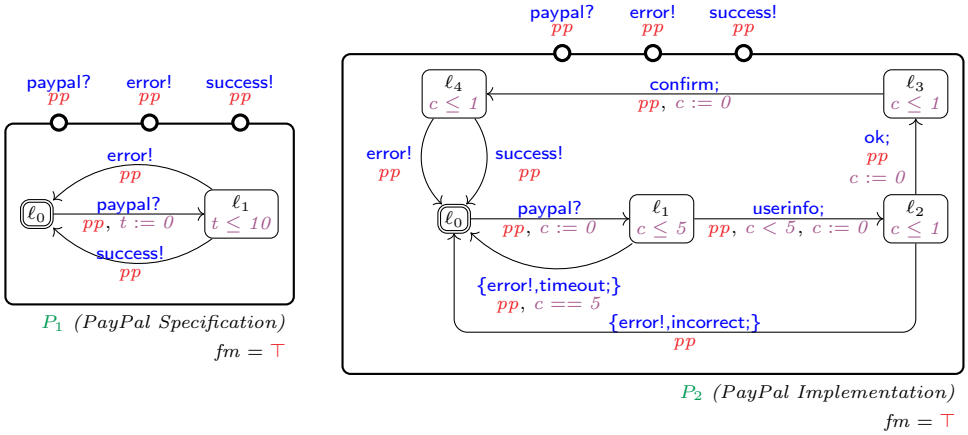

Fig. 5. An example of IFTA refinement with internal actions, where $P_2 \preceq P_1$.

Refinement of IFTA is a pre-order and it is compositional. The latter allows decomposition of refinement proofs, improving efficiency in refinement checking. In order to be compositional, refinement must be congruent with respect to IFTA operations, product and synchronization. The former is straightforward, however stronger pre-conditions are required to ensure congruence with respect to synchronization.

The problem arises with feature model refinement. Intuitively, by definition of refinement, in the implementation an input can be present in more products and an output can be present in less products, than in the specification. Thus, it is natural that the feature expressions associated to the input and output that we want to synchronize in the implementation differ from the feature expressions in the specification. Thus, if an implementation refines a specification, after synchronization, the feature model of the implementation does not necessarily refine the feature model of the specification.

Intuitively, a possible solution is to require that an implementation can only replace the specification if it does not add new interface bindings and maintains all bindings already in the specification. This means that, for each valid product in the implementation, the corresponding automata in the implementation can be synchronized over a given set of input and outputs, if and only if, the corresponding automata in the specification can be synchronized over the same inputs and outputs. The following theorems capture the pre-order and compositional properties of IFTA refinement.

Theorem 3 ($\preceq$ is a pre-order). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, $A_1 \preceq A_1$, and if $\mathcal{A}_1 \preceq \mathcal{A}_2$ and $\mathcal{A}_2 \preceq \mathcal{A}_3$, then $\mathcal{A}_1 \preceq \mathcal{A}_3$.*

Proof. $A_1 \preceq A_1$ is trivial by definition of $\preceq$. In the case of transitivity, feature model refinement follows immediately from Theorem 2. Behavioral refinement follows by induction on the structure of IFTA for each case of Definition 14.

Theorem 4 ($\preceq$ is congruent w.r.t. $\times$). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{B}$, such that $\mathcal{A}_i$ and $\mathcal{B}$ have disjoint set of actions and features, $i = 1, 2$, if $\mathcal{A}_1 \preceq \mathcal{A}_2$, then $\mathcal{A}_1 \times \mathcal{B} \preceq \mathcal{A}_2 \times \mathcal{B}$.*

Proof. Feature model refinement follows by definition of semantics of a feature expression. Behavioral refinement follows by induction on the structure of IFTA for each case of Definition 14.

Theorem 5 ($\preceq$ is congruent w.r.t. Δ). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$, and actions i, o such that $(i, o) \in I_i \times O_i$ for $i = 1, 2$, if $\mathcal{A}_1 \preceq \mathcal{A}_2$, then $\Delta_{i,o}\mathcal{A}_1 \preceq \Delta_{i,o}\mathcal{A}_2$, only if, $fm_1 \rightarrow ((\Gamma_1(i) \leftrightarrow \Gamma_1(o)) \leftrightarrow (\Gamma_2(i) \leftrightarrow \Gamma_2(o)))$.*

Proof. Feature model refinement follows by definition of semantics of a feature expression, and by the precondition on Γ_i for $i \in \{1, 2\}$. Behavioral refinement follows by induction on the structure of IFTA for each case of Definition 14.

To meet strict space limitations, detailed proofs of all results are omitted in the paper, but will appear in [6].

Let us consider again the IFTA CM composed with the router R (Figure 3). If we want to check if we can replace CM by CM' (Figure 4) in the system composed by CM and R, because refinement is compositional, instead of checking $CM' \bowtie_{(\mathsf{coffee},\mathsf{o}_1),(\mathsf{cappuccino},\mathsf{o}_2)} R \preceq CM \bowtie_{(\mathsf{coffee},\mathsf{o}_1),(\mathsf{cappuccino},\mathsf{o}_2)} R$ it suffices to check the following conditions:

1. $fm_1 \rightarrow ((\Gamma_{CM'}(\mathsf{coffee}) \leftrightarrow \Gamma_{CM'}(\mathsf{o}_1)) \leftrightarrow (\Gamma_{CM}(\mathsf{coffee}) \leftrightarrow \Gamma_{CM}(\mathsf{o}_1)))$
2. $fm_1 \rightarrow ((\Gamma_{CM'}(\mathsf{cappuccino}) \leftrightarrow \Gamma_{CM'}(\mathsf{o}_2)) \leftrightarrow (\Gamma_{CM}(\mathsf{cappuccino}) \leftrightarrow \Gamma_{CM}(\mathsf{o}_2)))$
3. CM' $\preceq$ CM

Conditions 1 and 2 correspond to the precondition for Δ congruence. In our example, both conditions are satisfied. However, let us assume now that we have an IFTA CM" which differs from CM only by changing the feature expression associated to the transition labeled with cappuccino!, from $cf \wedge mk$ to $(cf \wedge mk) \vee instc$, where $instc$ represents the support for instant cappuccino. In

this case, when we try to replace CM by CM", condition 2 does not hold. This is because in CM, cappuccino appears only when *cf* and *mk* are both present, however in CM" cappuccino can appear when *cf* and *instc* are present but not *mk*. Thus, the resulting composed system with CM" and R, models a concrete automaton that enables a synchronization between $o_2!$ and cappuccino? that was not possible before.

4 Conclusions

We proposed a refinement relation for families of timed automata which are modeled as Interface Featured Timed Automata. Since each IFTA can be seen as: 1) a feature model, which determines a set of valid feature combinations; and 2) a set of concrete automata, where each of the concrete automata is determined by a valid set of features; we separated the notion of IFTA refinement into *variability refinement* and *behavioral refinement*. Furthermore, we decompose IFTA into other formalisms on which we based such notions of refinement, namely Interface Featured Transition System (IFTS) and Interface Transition Systems (ITS).

The refinement relation proposed here is a pre-order and congruent with respect to IFTA product and synchronization, meaning refinement is compositional. However, in order to be congruent with respect to synchronization, stronger conditions must be made over the synchronization actions used in the composition. In particular, the implementation can only replace the specification in a composed environment, if it does not add new interface connections and maintains all connections of the specification. Although the requirement of not allowing new connections and maintain existed ones is reasonable, it can be too strict. For example, in alignment with the notion of ITS refinement, which allows to incorporate new behavior through new inputs, it will be desirable to incorporate new behavior in terms of new features, in a way that the example of CM", introduced in Section 3.3, can be considered a refinement of CM. In fact, in [2], de Alfaro et. al. only require that no new connections with the environment are made, while some connections can be lost, however, this is not sufficient to ensure that IFTA refinement is compositional. In this sense, as a future work we would like to explore and formalize other notions of refinement and how these can affect the properties that we can expect from the relation. For example, in the case of behavioral refinement, we could have defined that $\mathcal{A}$ refines $\mathcal{B}$, if and only if, for every feature selection FS in $fm_{\mathcal{B}}$ (instead of $fm_{\mathcal{A}}$), $[\![\mathcal{A}]\!] \downarrow_{FS} \preceq [\![\mathcal{B}]\!] \downarrow_{FS}$. The advantage is that we can now incorporate behavior in terms of new features, aligning better with the notion of refinement. However, on the one hand, this requires that $fm_{\mathcal{A}}$ contains at least all feature selections allowed by $fm_{\mathcal{B}}$, meaning $fm_{\mathcal{A}}$ can not incorporate mandatory features; and on the other hand, it can be too flexible, since we can not account for how the system will behave for new variability.

Currently, we are working on defining a notion of refinement over IFTS that takes advantage of the variability to perform a refinement checking on the entire family instead of a product by product approach. In addition, previously we

developed a prototype tool[2] to specify IFTA, compose them and translate them to other formalisms, including UPPAAL Timed Automata to verify properties, and we plan to extend it to support refinement checking.

Acknowledgements. The first author is supported by the European Regional Development Fund (ERDF) through the Operational Programme for Competitiveness and Internationalisation (COMPETE 2020), and by National Funds through the Portuguese funding agency, FCT, within project TRUST, POCI-01-0145-FEDER-016826. In addition the first and second author are supported by FCT grants PD/BD/52238/2013 and SFRH/BPD/91908/2012, respectively.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes 26(5), 109–120 (Sep 2001), http://doi.acm.org/10.1145/503271.503226
2. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, T. (eds.) Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute on Engineering Theories of Software Intensive Systems Marktoberdorf, Germany 3–15 August 2004. pp. 83–104. Springer Netherlands, Dordrecht (2005), http://dx.doi.org/10.1007/1-4020-3532-2_3
3. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR'98 Concurrency Theory: 9th International Conference Nice, France, September 8–11, 1998 Proceedings. pp. 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg (1998), https://doi.org/10.1007/BFb0055622
4. Baier, C., Katoen, J.P., Larsen, K.G.: Principles of model checking (2008)
5. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. International Conference on Software Engineering, ICSE pp. 321–330 (2011), http://dl.acm.org/citation.cfm?id=1985838
6. Cledou, G.: A Virtual Factory for Smart City Service Integration (forthcoming). Ph.D. thesis, Universidades do Minho, Aveiro and Porto (Joint MAP-i Doctoral Programme) (2018)
7. Cledou, G., Proença, J., S. Barbosa, L.: Composing families of timed automata. In: 7th IPM International Conference on Fundamentals of Software Engineering (2017), (In print)
8. Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A.: Behavioural modelling and verification of real-time software product lines. In: Proceedings of the 16th International Software Product Line Conference-Volume 1. pp. 66–75. ACM (2012)
9. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: A complete specification theory for real-time systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control. pp. 91–100. HSCC '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1755952.1755967
10. Thum, T., Batory, D., Kastner, C.: Reasoning about edits to feature models. In: Proceedings of the 31st International Conference on Software Engineering. pp. 254–264. ICSE '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICSE.2009.5070526

[2] https://github.com/haslab/ifta