HAL
archives-ouvertes.fr

Composing Families of Timed Automata

Guillermina Cledou, José Proença, Luis Soares Barbosa

▶ **To cite this version:**

Guillermina Cledou, José Proença, Luis Soares Barbosa. Composing Families of Timed Automata. 7th International Conference on Fundamentals of Software Engineering (FSEN), Apr 2017, Teheran, Iran. pp.51-66, 10.1007/978-3-319-68972-2_4 . hal-01760866

HAL Id: hal-01760866
https://hal.inria.fr/hal-01760866

Submitted on 6 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Composing Families of Timed Automata

Guillermina Cledou*, José Proença**, and Luis Barbosa***

HASLab INESCTEC and Universidade do Minho, Portugal
mgc@inesctec.pt, {jose.proenca,lsb}@di.uminho.pt

Abstract. Featured Timed Automata (FTA) is a formalism that enables the verification of an entire Software Product Line (SPL), by capturing its behavior in a single model instead of product-by-product. However, it disregards compositional aspects inherent to SPL development. This paper introduces Interface FTA (IFTA), which extends FTA with variable interfaces that restrict the way automata can be composed, and with support for transitions with atomic multiple actions, simplifying the design. To support modular composition, a set of Reo connectors are modelled as IFTA. This separation of concerns increases reusability of functionality across products, and simplifies modelling, maintainability, and extension of SPLs. We show how IFTA can be easily translated into FTA and into networks of Timed Automata supported by UPPAAL. We illustrate this with a case study from the electronic government domain.

Keywords: Software Product Lines; Featured Timed Automata; Compositionality

1 Introduction

Software product lines (SPLs) enable the definition of families of systems where all members share a high percentage of common features while they differ in others. Among several formalisms developed to support SPLs, Featured Timed Automata (FTA) [5] model families of real-time systems in a single model. This enables the verification of the entire SPL instead of product-by-product. However, FTA still need more modular and compositional techniques well suited to SPL-based development.

To address this issue, this paper proposes Interface FTA (IFTA), a mechanism enriching FTA with (1) interfaces that restrict the way multiple automata

* 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 POCI-01-0145-FEDER-016826 and FCT grant PD/BD/52238/2013.

** Supported by FCT under grant SFRH/BPD/91908/2012.

*** Supported by the project SmartEGOV: Harnessing EGOV for Smart Governance (Foundations, Methods, Tools) / NORTE-01-0145-FEDER-000037, supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF)

interact, and (2) transitions labelled with multiple actions that simplify the design. Interfaces are synchronisation actions that can be linked with interfaces from other automata when composing automata in parallel. IFTA can be composed by combining their feature models and linking interfaces, imposing new restrictions over them. The resulting IFTA can be exported to the UPPAAL real-time model checker to verify temporal properties, using either a network of parallel automata in UPPAAL, or by flattening the composed automata into a single one. The latter is better suited for IFTA with many multiple actions.

We illustrate the applicability of IFTA with a case study from the electronic government (e-government) domain, in particular, a family of licensing services. This services are present in most local governments, who are responsible for assessing requests and issuing licenses of various types. E.g., for providing public transport services, driving, construction, etc. Such services comprise a number of common functionality while they differ in a number of features, mostly due to specific local regulations.

The rest of this paper is structured as follows. Section 2 presents some background on FTA. Section 3 introduces IFTA. Section 4 presents a set of Reo connectors modeled as IFTA. Section 5 discusses a prototype tool to specify and manipulate IFTA. Section 6 presents the case study. Section 7 discusses related work, and Section 8 concludes.

2 Featured Timed Automata

This work builds on top of *Featured Timed Automata* (FTA) an extension to *Timed Automata*, introduced by Cordy et al. [5] to verify real-time systems parameterised by a variability model. This section provides an overview of FTA and their semantics, based on Cordy et al..

Informally, a Featured Timed Automaton is an automaton whose *edges* are enriched with *clocks*, *clock constraints* (CC), *synchronisation actions*, and *feature expressions* (FE). 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 *synchronisation action* $a \in A$ is used to coordinate automata in parallel; an edge with an action a can only be taken when its dual action in a neighbor automaton is also on an edge that can be taken simultaneously. Finally, a *feature expression* (FE) is a logical constraint over a set of *features*. Each of these features denotes a unit of variability; by selecting a desired combination of features one can map an FTA into a Timed Automaton.

Figure 1 exemplifies a simple FTA with 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. Initially the automaton is in location ℓ_0, 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

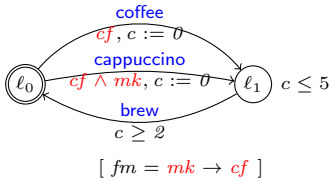


Fig. 1: Example of a Featured Timed Automata over the features cf and mk.

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 transition labelled with *brew* has a different guard: a clock constraint $c \geq 2$ that allows this transition to be taken only when the clock c is greater than 2. Finally, the lower expression $[\ fm = mk \rightarrow cf\ ]$ defines the *feature model*. I.e., how the features relate to each other. In this case the mk feature can only be selected when the cf feature is also selected.

We now formalize clock constraints, feature expressions, and the definition of FTA and its semantics.

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 $\eta \colon C \rightarrow \mathbb{R}_{\geq 0}$ that assigns each clock $c \in C$ to its current value ηc. 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 $C \setminus r$.*

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

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

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

Definition 2 (Feature Expressions (FE) and satisfaction). *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*

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

where $\diamondsuit \in \{\wedge, \vee\}$.

Definition 3 (Featured Timed Automata (FTA) [5]). *An FTA is a tuple $\mathcal{A} = (L, L_0, A, C, F, E, Inv, fm, \gamma)$ where L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, A is a finite set of synchronisation actions, C is a finite set of clocks, F is a finite set of features, E is a finite set of edges, $E \subseteq L \times CC(C) \times A \times \mathbf{2}^C \times L$, $Inv : L \to CC(C)$ is the invariant, a partial function that assigns CCs to locations, $fm \in FE(F)$ is a feature model defined as a Boolean formula over features in F, and $\gamma : E \to FE(F)$ is a total function that assigns feature expressions to edges.*

The semantics of FTA is given in terms of Featured Transition Systems (FTSs) [4]. An FTS extends Labelled Transition Systems with a set of features F, a feature model *fm*, and a total function γ that assigns FE to transitions.

Definition 4 (Semantics of FTA). *Let $\mathcal{A} = (L, L_0, A, C, F, E, Inv, fm, \gamma)$ be an FTA. The semantics of $\mathcal{A}$ is defined as an FTS $\langle S, S_0, A, T, F, fm, \gamma' \rangle$, where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $S_0 = \{\langle \ell_0, \eta_0 \rangle \mid \ell_0 \in L_0\}$ is the set of initial states, $T \subseteq S \times (A \cup \mathbb{R}_{\geq 0}) \times S$ is the transition relation, with $(s_1, \alpha, s_2) \in T$, and $\gamma' \colon T \to FE(F)$ is a total function that assigns feature expressions to transitions. The transition relation and γ are defined as follows.*

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

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

where $s_1 \xrightarrow[\varphi]{\alpha} s_2$ means that $(s_1, \alpha, s_2) \in T$ and $\gamma(s_1, \alpha, s_2) = \varphi$, for any $s_1, s_2 \in S$.

Notation: We write $L_{\mathcal{A}}$, $L_{0\mathcal{A}}$, $A_{\mathcal{A}}$, etc., to denote the locations, initial locations, actions, etc., of an FTA $\mathcal{A}$, respectively. We write $\ell_1 \xrightarrow{cc,a,c}_{\mathcal{A}} \ell_2$ to denote that $(\ell_1, cc, a, c, \ell_2) \in E_{\mathcal{A}}$, and use $\ell_1 \xrightarrow[\varphi]{cc,a,c}_{\mathcal{A}} \ell_2$ to express that $\gamma_{\mathcal{A}}(\ell_1, cc, a, c, \ell_2) = \varphi$. We omit the subscript whenever automaton $\mathcal{A}$ is clear from the context. We use an analogous notation for elements of an FTS.

3 Interface Featured Timed Automata

Multiple FTAs can be composed and executed in parallel, using *synchronising actions* to synchronise edges from different parallel automata. This section introduces *interfaces* to FTA that: (1) makes this implicit notion of communication more explicit, and (2) allows multiple actions to be executed atomically in a transition. Synchronisation actions are lifted to so-called *ports*, which correspond to actions that can be linked with actions from other automata. Hence composition of IFTA is made by linking ports and by combining their variability models.

Definition 5 (Interface Featured Timed Automata). *An IFTA is a tuple* $\mathcal{A} = (L, l_0, A, C, F, E, Inv, fm, \gamma)$ *where* L, C, F, Inv, fm, γ *are defined as in Featured Timed Automata, there exists only one initial location* l_0*,* $A = I \uplus O \uplus H$ *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, and edges in* E *contain sets of actions instead of single actions (*$E \subseteq L \times CC(C) \times \underline{\mathbf{2}^A} \times \mathbf{2}^C \times L$*).*

We call *interface* of an IFTA $\mathcal{A}$ the set $\mathbb{P}_\mathcal{A} = I_\mathcal{A} \uplus O_\mathcal{A}$ of all input and output ports of an automaton. Given a port $\mathsf{p} \in \mathbb{P}$ we write $\mathsf{p?}$ and $\mathsf{p!}$ to denote that p is an input or output port, respectively, following the same conventions as UPPAAL for actions, and write p instead of $\{\mathsf{p}\}$ when clear from context. The lifting of actions into sets of actions will be relevant for the composition of automata. **Notation:** we use i, i_1, etc., and o, o_1, etc. to refer specifically to input and output ports of an IFTA, respectively. For any IFTA $\mathcal{A}$ it is possible to infer a feature expression for each action $\mathsf{a} \in A_\mathcal{A}$ based on the feature expressions of the edges in which a appears. Intuitively, this feature expression determines the set of products requiring a.

Definition 6 (Feature Expression of an Action). *Given an IFTA* $\mathcal{A}$*, the feature expression of any action* a *is the disjunction of the feature expressions of all of its associated edges, defined as*

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

We say an IFTA $\mathcal{A}$ is *grounded*, if it has a total function associating a feature expression to each action $\mathsf{a} \in A_\mathcal{A}$ that indicates the set of products where a was originally designed to be present in. Given an IFTA $\mathcal{A}$ we can construct a *grounded* $\mathcal{A}$ by incorporating a function Γ such that, $\mathcal{A} = (L_\mathcal{A}, l_{0_\mathcal{A}}, A_\mathcal{A}, C_\mathcal{A}, Inv_\mathcal{A}, F_\mathcal{A}, fm_\mathcal{A}, \gamma_\mathcal{A}, \Gamma)$, where $\Gamma : A_\mathcal{A} \to FE(F_\mathcal{A})$ assigns a feature expression to each action of $\mathcal{A}$, and is constructed based on $\widehat{\Gamma}_\mathcal{A}$. From now on when referring to an IFTA we assume it is a grounded IFTA.

Figure 2 depicts the interfaces of 3 different IFTA. The leftmost is a payment machine that receives actions representing coins and publishes actions confirming the payment, whose actions are dependent on a feature called *pay*. The rightmost is the coffee machine from Figure 1. Finally, the middle one depicts a connector *Router* that could be used to *combine* the payment and the coffee machines. This notion of *combining* IFTA is the core contribution of this work: how to reason

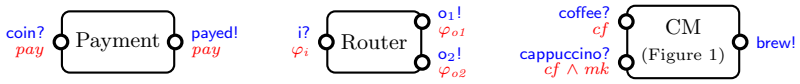


Fig. 2: Representation of 3 IFTA, depicting their interfaces (blue) and associated feature expressions.

about the modular composition of timed systems with variable interfaces. For example, let us assume the previous IFTA are connected by linking actions as follows: (payed,i), (o_1,coffee), and (o_2,cappuccino). In a real coffee machine, after a payment, the machine should allow to select only beverages supported, i.e., if the machine does not support cappuccino the user should not be able to select it and be charged. Similarly, the composed system here should not allow to derive a product with o_2 if cappuccino is not present. To achieve this, we need to impose additional restriction on the variability model of the composed system, since as it will be shown later in this section, combining the feature models of the composed IFTA through logical conjunction is not enough.

The semantics of IFTA is given in terms of FTSs, similarly to the semantics of FTA with the difference that transitions are now labelled with sets of actions. We formalize this as follows.

Definition 7 (Semantics of IFTA). *Let $\mathcal{A}$ be an IFTA, its semantics is an FTS $\mathcal{F} = (S, s_0, A, T, F, fm, \gamma)$, where S, A, F, fm, and γ are defined as in Definition 4, $s_0 = \langle \ell_0, \eta_0 \rangle$ is now the only initial state, and $T \subseteq S \times (\mathbf{2}^A \cup \mathbb{R}_{\geq 0}) \times S$ now supports transitions labelled with sets of actions.*

We now introduce two operations: *product* and *synchronisation*, which are used to define the composition of IFTA. The *product* operation for IFTA, unlike the classical product of timed automata, is defined over IFTA with disjoint sets of actions, clocks and features, performing their transitions in an interleaving fashion.

Definition 8 (Product of IFTA). *Given two IFTA $\mathcal{A}_1$ and $\mathcal{A}_2$, with disjoint actions, clocks and features, 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* $i = 1, 2$.

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

Definition 9 (Synchronisation). *Given an 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* $H' = H \cup \{a.b\}$.
– $E' = \{\ell \xrightarrow{g,\omega,r} \ell' \in E \mid a \notin \omega \text{ and } b \notin \omega\} \cup$
$\{\ell \xrightarrow{g,\omega\setminus\{a,b\},r} \ell' \mid \ell \xrightarrow{g,\omega,r} \ell' \in E \text{ and } a \in \omega \text{ and } b \in \omega\}$
– $fm' = fm \wedge (\Gamma_{\mathcal{A}}(a) \leftrightarrow \Gamma_{\mathcal{A}}(b))$.

Together, the product and the synchronisation can be used to obtain in a *compositional* way, a complex IFTA modelling SPLs built out of primitive IFTA.

Definition 10 (Composition of IFTA). *Given two disjoint IFTA,* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *and a set of bindings* $\{(a_1, b_1), \ldots, (a_n, b_n)\}$, *where* $a_k \in \mathbb{P}_1$, $b_k \in \mathbb{P}_2$, *and such that* $(a_k, b_k) \in I_1 \times O_2$ *or* $(a_k, b_k) \in I_2 \times O_1$, *for* $1 \leq k \leq n$, *the composition of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *is defined as* $\mathcal{A}_1 \bowtie_{(a_1,b_1),\ldots,(a_n,b_n)} \mathcal{A}_2 = \Delta_{a_1,b_1} \ldots \Delta_{a_n,b_n}(\mathcal{A}_1 \times \mathcal{A}_2)$.

Figure 3 exemplifies the composition of the coffee machine (CM) and Router IFTA from Figure 2. The resulting IFTA combines the feature models of the CM and Router, 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 labelled with non-binded actions can transition independently or together. Combining their feature models only through logical conjunction allows $\{cf, f_{o_2}, f_i, f_{o_1}\}$ as a valid feature selection. In such scenario, we could derive a product that can issue o_2 but that can not be captured by cappuccino. In terms of methods calls in a programming language, the derive product will have a call to a method that does not exists, leading to an error.

To study properties of IFTA operations, we define the notion of IFTA equivalence in terms of bisimulation over their underlying FTSs. We formally introduce the notion of timed bisimulation adapted to FTSs.

Definition 11 (Timed Bisimulation). *Given two FTSs* $\mathcal{F}_1$ *and* $\mathcal{F}_2$, *we say* $\mathcal{R} \subseteq S_1 \times S_2$ *is a bisimulation, if and only if, for all possible feature selections* $FS \in \mathbf{2}^{F_1 \cup F_2}$, $FS \models fm_1 \Leftrightarrow FS \models fm_2$ *and for all* $(s_1, s_2) \in R$ *we have:*

– $\forall\ t = s_1 \xrightarrow{\alpha}_1 s_1'$, $\alpha \in \mathbf{2}^A \cup \mathbb{R}_{\geq 0}$, $\exists\ t' = s_2 \xrightarrow{\alpha}_2 s_2'$ *s.t.* $(s_1', s_2') \in \mathcal{R}$ *and* $FS \models \gamma_1(t) \Leftrightarrow FS \models \gamma_2(t')$,

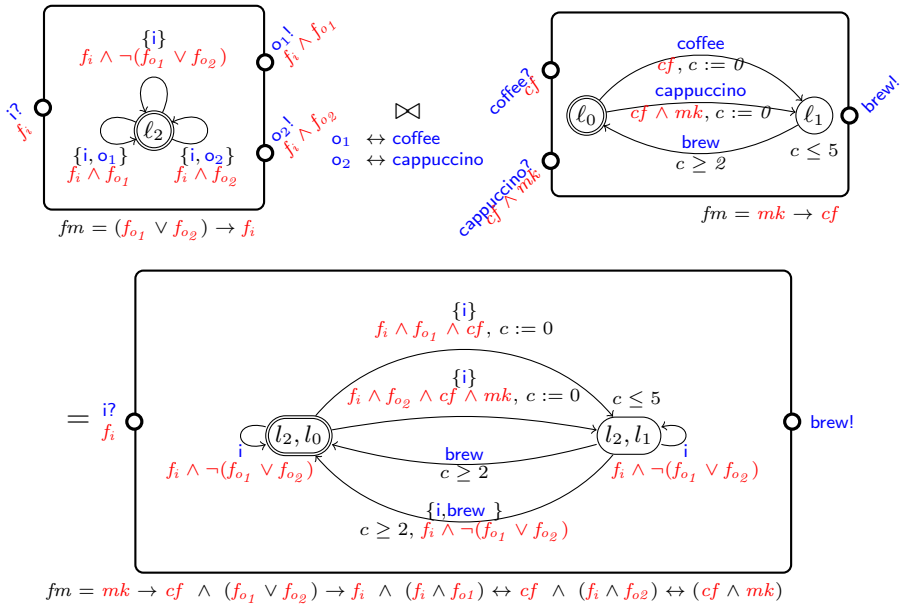


Fig. 3: Composition of a Router IFTA (top left) with the CM IFTA (top right) by binding ports (o_1, coffee) and $(o_2, \text{cappuccino})$, yielding the IFTA below.

– $\forall\ t' = s_2 \xrightarrow{\alpha}_2 s_2',\ \alpha \in \mathbf{2}^A \cup \mathbb{R}_{\geq 0},\ \exists\ t = s_1 \xrightarrow{\alpha}_1 s_1'\ s.t.\ (s_1', s_2') \in \mathcal{R}$ *and* $FS \models \gamma_1(t) \Leftrightarrow FS \models \gamma_2(t')$

where $A = A_1 \cup A_2$.

Two states $s_1 \in S_1$ and $s_2 \in S_2$ are bisimilar, written $s_1 \sim s_2$, if there exists a bisimulation relation containing the pair (s_1, s_2). Given two IFTA $\mathcal{A}_1$ and $\mathcal{A}_2$, we say they are bisimilar, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there exists a bisimulation relation containing the initial states of their corresponding FTSs.

Proposition 1 (Product is commutative and associative). *Given two IFTA* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *with disjoint set of actions and clocks,* $\mathcal{A}_1 \times \mathcal{A}_2 \sim \mathcal{A}_2 \times \mathcal{A}_1$, *and* $\mathcal{A}_1 \times (\mathcal{A}_2 \times \mathcal{A}_3) \sim (\mathcal{A}_1 \times \mathcal{A}_2) \times \mathcal{A}_3$.

The proof follows trivially by definition of product and FTSs, and because $\cup$ and $\wedge$ are associative and commutative.

The *synchronisation* operation is commutative, and it interacts well with *product*. The following proposition captures these properties.

Proposition 2 (Synchronisation commutativity). *Given two IFTA* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *the following properties hold:*

1. $\Delta_{a,b}\Delta_{c,d}\mathcal{A}_1 \sim \Delta_{c,d}\Delta_{a,b}\mathcal{A}_1$, *if* $a, b, c, d \in A_1$, a, b, c, d *different actions.*
2. $(\Delta_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \Delta_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \in A_1$ *and* $A_1 \cap A_2 = \emptyset$.

Both proof follow trivially by definition of product, synchronization and FTSs.

4 Reo connectors as IFTA

Reo is a channel-based exogenous coordination language where complex coordinators, called connectors, are compositionally built out of simpler ones, called channels [2]. Exogenous coordination facilitates anonymous communication of components. Each connector has a set of input and output ports, and a formal semantics of how data flows from the inputs to the outputs. We abstract from the notion of data and rather concentrate on how execution of actions associated to input ports enables execution of actions associated to output ports.

Table 1 shows examples of basic Reo connectors and their corresponding IFTA. For example, $\mathit{Merger}(i_1, i_2, o)$ synchronises each input port, separately, with the output port, i.e. each i_k executes simultaneously with o for $k = 1, 2$; and $\mathit{FIFO1}(i, o)$ introduces the notion of delay by executing its input while transitions to a state where time can pass, enabling the execution of its output without time restrictions.

Modelling Reo connectors as IFTA enables them with variable behavior based on the presence of ports connected through synchronisation to their ports. Thus, we can use them to coordinate components with variable interfaces. We associate a feature f_a to each port a of a connector and define its behavior in terms of these features. Table 1 shows Reo basic connectors as IFTA with variable behavior. Bold edges represent the standard behavior of the corresponding Reo connector, and thinner edges model variable behavior. For example, the *Merger* connector supports the standard behavior, indicated by the transitions $\ell_0 \xrightarrow{\{\mathsf{i}_k,\mathsf{o}\}} \ell_0$, $k = 1, 2$ and the corresponding feature expression $f_k \wedge f_o$; and a variable behavior, in which both inputs can execute independently at any time if o is not present, indicated by transitions $\ell_0 \xrightarrow{\{\mathsf{i}_k\}} \ell_0$, $k = 1, 2$ and the corresponding feature expression $f_k \wedge \neg f_o$.

The *Sync* connector behaves as the *identity* when composed with other automata. The following proposition captures this property.

Proposition 3 (*Sync* behaves as identity). *Given an IFTA $\mathcal{A}$ and a Sync connector, $\Delta_{i,a}(\mathcal{A} \times \mathit{Sync}(i, o)) \sim \mathcal{A}[o/a]$ with the following updates*

- $fm_{\mathcal{A}[o/a]} = fm_{\mathcal{A}} \wedge (f_{io} \leftrightarrow \Gamma_{\mathcal{A}}(a))$
- $\gamma_{\mathcal{A}[o/a]}(\ell \xrightarrow{g,\omega,r}_{\mathcal{A}[o/a]} \ell') = \gamma_{\mathcal{A}}(\ell \xrightarrow{g,\omega[a/o],r}_{\mathcal{A}} \ell') \wedge f_{io}$, *if* $o \in \omega$
- $F_{\mathcal{A}[o/a]} = F_{\mathcal{A}} \cup \{f_{io}\}$
- $\Gamma_{\mathcal{A}[o/a]}(o) = \Gamma_{\mathit{Sync}}(o)$

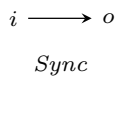
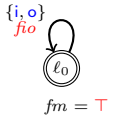
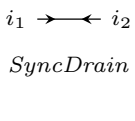
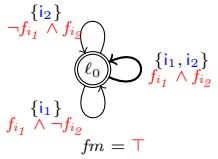
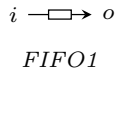
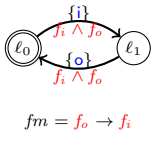
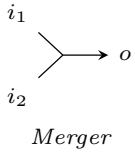
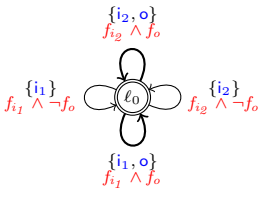
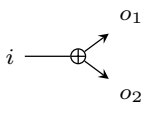
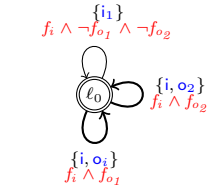
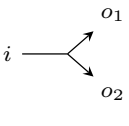
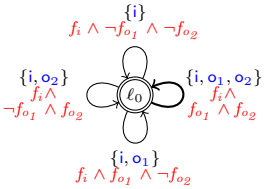
if $\{i, o\} \not\subseteq A_{\mathcal{A}}$, and $a \in A_{\mathcal{A}}$. $\mathcal{A}[o/a]$ is $\mathcal{A}$ with all occurrences of a replaced by o.

Proof. First for simplicity, let $\mathcal{A}_S = (\mathcal{A} \times \mathit{Sync}(i, o))$, and $\mathcal{A}' = \Delta_{i,a}(\mathcal{A}_S)$. Lets note that the set of edges in $\mathcal{A}'$ is defined as follows

$$E_{\mathcal{A}'} = \{(\ell_1, \ell_0) \xrightarrow{g,\omega,r}_{\mathcal{A}_S} (\ell_1', \ell_0) \mid i \notin \omega \text{ and } a \notin \omega\} \cup \tag{1}$$

$$\{(\ell_1, \ell_0) \xrightarrow{g,\omega\setminus\{i,a\},r} (\ell_1', \ell_0) \mid (\ell_1, \ell_0) \xrightarrow{g,\omega,r}_{\mathcal{A}_S} (\ell_1', \ell_0) \text{ and } i \in \omega \text{ and } a \in \omega\} \tag{2}$$

Table 1: Examples of basic Reo connectors and their corresponding IFTA.

Connector	IFTA	Connector	IFTA
			
			
			

where ℓ_0 is the initial and only location of *Sync*. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the underlying FTS of $\mathcal{A}'$ and $\mathcal{A}[o/a]$, and note that $\mathcal{R} = \{(\langle(\ell_1, \ell_0), \eta\rangle, \langle \ell_1, \eta\rangle) \mid \ell_1 \in S_{\mathcal{A}[o/a]}\}$ is a bisimulation between states of $\mathcal{F}_1$ and $\mathcal{F}_2$. Let $(\langle(\ell_1, \ell_0), \eta\rangle, \langle \ell_1, \eta\rangle) \in \mathcal{R}$. The proof for delay transitions follows trivially from the fact that $Inv(\ell_1, \ell_0) = Inv(\ell_1)$ for all $\ell_1 \in S_{\mathcal{A}[o/a]}$.

Lets consider any action transition $\langle(\ell_1, \ell_0), \eta\rangle \xrightarrow{\omega} \langle(\ell_1', \ell_0), \eta'\rangle \in T_{\mathcal{F}_1}$. If it comes from an edge in (1), then $\exists\ \ell_1 \xrightarrow{g,\omega,r} \ell_1' \in E_{\mathcal{A}}\ s.t.\ a \notin \omega$, thus $\exists\ \langle \ell_1, \eta\rangle \xrightarrow{\omega} \langle \ell_1', \eta'\rangle \in T_{\mathcal{F}_2}$; if it comes from (2), then $\exists\ \ell_1 \xrightarrow{g,\omega_1,r} \ell_1' \in E_{\mathcal{A}}\ s.t.\ a \in \omega_1$, thus $\exists\ \langle \ell_1, \eta\rangle \xrightarrow{\omega_1[o/a]} \langle \ell_1', \eta'\rangle \in T_{\mathcal{F}_2}$, where $\omega = \omega_1 \cup \{i, o\} \setminus \{i, a\} = \omega[o/a]$. Conversely, if $\exists\ \langle \ell_1, \eta\rangle \xrightarrow{\omega} \langle \ell_1', \eta'\rangle \in T_{\mathcal{F}_2}$ and $o \notin \omega$, then $\exists\ (\ell_1, \ell_0) \xrightarrow{g,\omega,r} (\ell_1', \ell_0) \in E_{\mathcal{A}_S}\ s.t.\ i \notin \omega \wedge a \notin \omega$, thus $\exists\ \langle(\ell_1, \ell_0), \eta\rangle \xrightarrow{\omega} \langle(\ell_1', \ell_0), \eta'\rangle \in T_{\mathcal{F}_1}$; if $o \in \omega$, then $\exists\ (\ell_1, \ell_0) \xrightarrow{g,\omega_1 \cup \{o\} \setminus \{a\},r} (\ell_1', \ell_0) \in E_{\mathcal{A}'}$, such that $\omega = \omega_1[o/a] = \omega_1 \cup \{o\} \setminus \{a\}$, thus $\exists\ \langle(\ell_1, \ell_0), \eta \xrightarrow{\omega} \langle(\ell_1', \ell_0), \eta'\rangle\rangle \in T_{\mathcal{F}_1}$.

In both cases, we have $\gamma_{\mathcal{F}_1}(\langle(\ell_1, \ell_0), \eta\rangle \xrightarrow{\omega} \langle(\ell_1', \ell_0), \eta'\rangle) = \gamma_{\mathcal{F}_2}(\langle \ell_1, \eta\rangle \xrightarrow{\omega} \langle \ell_1', \eta'\rangle)$. Furthermore, $fm'_{\mathcal{A}} = fm_{\mathcal{A}[o/a]}$. □

5 Implementation

We developed a prototype tool in Scala[1] consisting of a small Domain Specific Language (DSL) to specify (networks of) (N)IFTA and manipulate them. Although we do not provide the formal definitions and semantics due to space constraints, informally, a network of any kind of automata is a set of automata parallel composed (||) and synchronised over a set of shared actions.

Main features supported by the DSL include: 1) specification of (N)IFTA, 2) composition, product and synchronisation over IFTA, 3) conversion of NIFTA to networks of FTA (NFTA) with committed states (CS), and 4) conversion of NFTA to UPPAAL networks of TA (NTA) with features. Listing 1.1 shows how the *router* connector from Table 1 can be specified using the DSL. A comprehensive list of functionality and more examples, including the case study from Section 6 can be found in the tool's repository[1].

```
val router = newifta ++ (
    0 --> 0 by "i,o1"   when "vi" && "vo1",
    0 --> 0 by "i,o2"   when "vi" && "vo2",
    0 --> 0 by "i"      when "vi" && not("vo1" || "vo2")
  ) get "i" pub "o1,o2" when ("vo1" || "vo2") --> "vi"
```

Listing 1.1: Example specification of a router connector using the Scala DSL.

A NIFTA can be converted into a NFTA with committed states, which in turn can be converted into a network of UPPAAL TA, through a stepwise conversion, as follows. **NIFTA to NFTA**. Informally, this is achieved by converting each transition with set of actions into to a set of transitions with single actions. All transitions in this set must execute atomically (committed states between them) and support all combinations of execution of the actions. **NFTA to UPPAAL NTA**. First, the NFTA obtained in the previous step is translated into a network of UPPAAL TA, where features are encoded as Boolean variables, and transition's feature expressions as logical guards over Boolean variables. Second, the *feature model* of the network is solved using a SAT solver to find the set of valid feature selections. This set is encoded as a TA with an initial committed location and outgoing transitions to new locations for each element in the set. Each transition initializes the set of variables of a valid feature selection. The initial committed state ensures a feature selection is made before any other transition is taken.

When translating IFTA to FTA with committed states, the complexity of the model grows quickly. For example, the IFTA of a simple replicator with 3 output ports consists of a location and 8 transitions, while its corresponding FTA consists of 23 locations and 38 transitions. Without any support for composing variable connectors, modelling all possible cases is error prone and it quickly becomes unmanageable. This simplicity in design achieved through multi-action transitions leads to a more efficient approach to translate IFTA to UPPAAL TA, in particular by using the composition of IFTA. The IFTA resulting from composing a network of IFTA, can be simply converted to an FTA by flattening the set of actions in to a single action, and later into an UPPAAL TA.

[1] `https://github.com/haslab/ifta`

6 Case Study: Licensing Services in e-government

This section presents a case study of using IFTA to model a family of public licensing services. All services in the family support submissions and assessment of licensing requests. Some services, in addition, require a fee before submitting (pa), others allow appeals on rejected requests (apl), or both. Furthermore, services that require a fee can support credit card (cc) or PayPal payments (pp), or both. Functionality is divided in components and provided as follows. Each component can be visualized in Figure 4. We omit the explicit illustration of interfaces and rather use the notation ?,! to indicate whether an action corresponds to an input or output, respectively. In addition, we use the same action name in two different automata to indicate pairs of actions to be linked. The feature model, also omitted, is initially $\top$ for each of these IFTA.

App - Models licenses requests. An applicant must submit the required documents (subdocs), and pay a fee (payapp) if pa is present, before submitting (submit). If the request is accepted (accept) or considered incomplete (incomplete), the request is closed. If it is rejected (reject) and it is not possible to appeal ($\neg apl$), the request is closed, otherwise a clock ($tapl$) is reseted to track the appeal window time. The applicant has 31 days to appeal ($Inv_{App}(\ell_5)$), otherwise the request is canceled (cancelapp) and closed. If an appeal is submitted (appeal), it can be rejected or accepted, and the request is closed.

CC and *PP* - Handle payments through credit cards and PayPal, respectively. If a user requests to pay by credit card (paycc) or PayPal (paypp), a clock is reset to track payment elapsed time ($tocc$ and $topp$). The user has 1 day ($Inv_{CC}(\ell_1)$ and $Inv_{PP}(\ell_1)$) to proceed with the payment which can result in success (paidcc and paidpp) or cancellation (cancelcc and cancelpp).

Appeal - Handles appeal requests. When an appeal is received (appeal), a clock is reseted to track the appeal submission elapsed time (tas). Authorities have 20 days ($Inv_{Appeal}(\ell_1)$) to start assessing the request (assessapl).

Preassess - Checks if a request contains all required documents. When a request is received (submit), a clock is reseted to track the submission elapsed time (ts). Authorities have 20 days ($Inv_{Preasses}(\ell_1)$) to check the completeness of the documents and notify whether it is incomplete (incomplete) or ready to assessed (assessapp).

Assess - Analyzes requests. When a request is ready to be assessed (assess), a clock is reseted to track the processing elapsed time (tp). Authorities have 90 days to make a decision of weather accept it (accept) or reject it (reject).

We use a set of Reo connectors to integrate these IFTA. The final integrated model can be seen in Figure 5. For simplicity, we omit the feature expressions associated to ports and the resulting feature model. Broadly, we can identify two new components in this figure: *Payment* - (right of *App*) Orchestrates payment requests based on the presence of payment methods. It is composed by componets *CC*, *PP*, and a set of connectors. A *router* synchronises payment requests (payapp) with payment by *CC* or *PayPal* (paypp or paycc). A *merger* synchronises the successful response (paidpp or paidcc), while other *merger* synchronises the cancellation response (cancelpp or cancelcc) from either *CC* or *PP*.

On top of the composed feature model, we add the restriction $pa \leftrightarrow cc \vee pp$ to ensure payment is supported, if and only if, Credit card or PayPal are supported; and *Processing* - (left of *App*) Orchestrates the processing of licenses requests and appeals (if *apl* is present). It is composed by *Appeal*, *Preassess*, *Assess*, a set of trivial *sync* connectors and a *merger* that synchronises assessment requests from either *Appeal* or *Preassess* (assessapl or assessapp) with *Assess* (assess).

By using IFTA, connectors are reused and it is simple to create complex connectors out of simple ones. If in the future a new payment methods is supported, the model can be updated by simple using a three output replicator and two three inputs mergers. By composing the future model and inferring new restrictions based on how interfaces are connected, it is possible to reason about the variability of the entire network, E.g., we can check if the resulting feature model satisfies variability requirements or if the interaction of automata is consistent with the presence of features. In addition, by using the DSL we can translate this components to UPPAAL to verify properties such as: *Deadlock free* – `A[] not deadlock`; *Liveness* – a submission and an appeal will eventually result in an answer (`App.`ℓ_4 `--> App.`ℓ_0 and `App.`ℓ_6 `--> App.`ℓ_0, respectively); *Safety* – a submission must be processed within 110 days (`A[] App.`ℓ_4 `imply App.tsub <=110`).

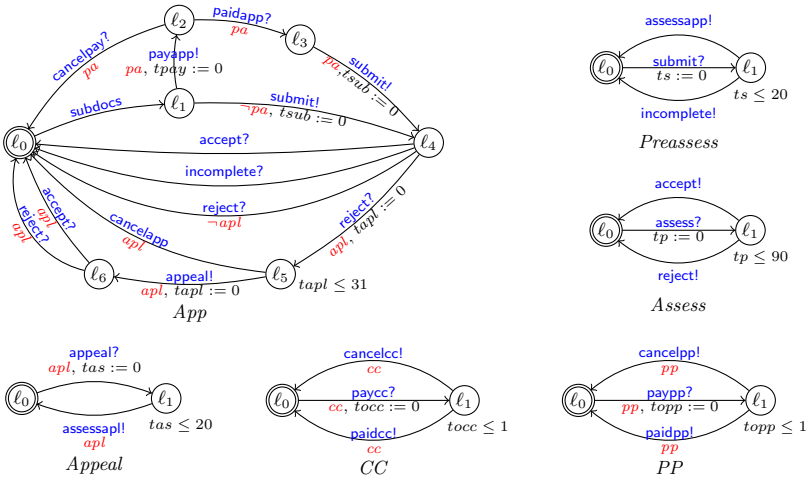


Fig. 4: IFTA modelling domain functionality.

7 Related Work

Related work is discussed following two lines: 1) compositionality and modularity of SPLs, and 2) compositionality and interfaces for automata.

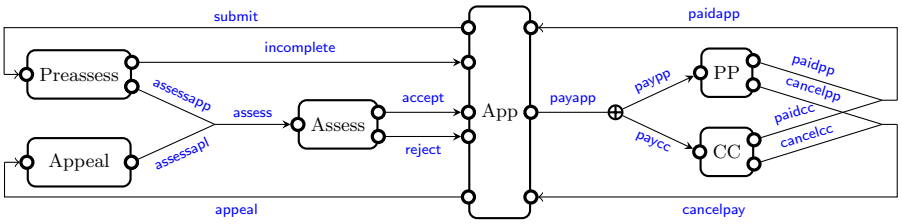


Fig. 5: IFTA for a family of Licensing Services

Compositionality and modularity of SPLs. An extension to Petri Nets, Feature Nets (FNs) [11] enables specifying the behavior of an SPL in a single model, and supports composition of FNs by applying deltas FNs to core FNs. An extension to CCS process calculus consisting on a modular approach to modelling and verifying variability of SPLs based on DeltaCCS [9]. A compositional approach for verification of software product lines [10] where new features and variability may be added incrementally, specified as finite state machines with variability information.

Interfaces and compositionality of automata. Interface automata [1] use input interfaces to support incremental design and independent implementability of components, allowing compatibility checking of interfaces for partial system descriptions, without knowing the interfaces of all components, and separate refinement of compatible interfaces, respectively. [6] presents a specification theory for I/O TA supporting refinement, consistency checking, logical and structural composition, and quotient of specifications. In [8] Modal I/O automata are used to construct a behavioral variability theory for SPL development and can serve to verify if certain requirements can be satisfied from a set of existing assets. [7] proposes a formal integration model based on Hierarchical TA for real time systems, with different component composition techniques. [3] presents a compositional specification theory to reason about components that interact by synchronisation of I/O actions.

8 Conclusions

This paper introduced IFTA, a formalism for modelling SPL in a modular and compositional manner, which extends FTA with variable interfaces to restrict the way automata can be composed, and with multi-action transitions that simplify the design. A set of Reo connectors were modeled as IFTA and used to orchestrate the way various automata connect. We discussed a prototype tool to specify and manipulate IFTA, which takes advantage of IFTA composition to translate them into TA that can be verified using the UPPAAL model checker.

Delegating coordination aspects to connectors enables separation of concerns. Each automata can be designed to be modular and cohesive, facilitating the maintenance, adaptability, and extension of an SPL. In particular, by facilitat-

ing compositional reasoning when replacing components, E.g., when checking for a refinement relation, as well as enabling changes in the coordination mechanisms without affecting core domain functionality. Using bare FTA for designing variable connectors, can be error prone and it quickly becomes unmanageable. IFTA simplifies this design by enabling the modeling of automata in isolation and composing them by explicitly linking interfaces and combining their feature models.

Future work includes studying an implementation relation, I.e, refinement, to reason about how to safely replace an IFTA with a more detailed one in a compose environment.

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. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical. Structures in Comp. Sci. 14(3), 329–366 (Jun 2004), http://dx.doi.org/10.1017/S0960129504004153
3. Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A Compositional Specification Theory for Component Behaviours, pp. 148–168. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-28869-2_8
4. 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
5. 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. SPLC '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2362536.2362549
6. 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
7. Jin, X., Ma, H., Gu, Z.: Real-time component composition using hierarchical timed automata. In: Seventh International Conference on Quality Software (QSIC 2007). pp. 90–99 (Oct 2007)
8. Larsen, K.G., Nyman, U., Wasowski, A.: Modal i/o automata for interface and product line theories. In: European Symposium on Programming. pp. 64–79. Springer (2007), http://dx.doi.org/10.1007/978-3-540-71316-6_6
9. Lochau, M., Mennicke, S., Baller, H., Ribbeck, L.: Incremental model checking of delta-oriented software product lines. Journal of Logical and Algebraic Methods in Programming 85(1, Part 2), 245 – 267 (2016), http://dx.doi.org/10.1016/j.jlamp.2015.09.004, formal Methods for Software Product Line Engineering
10. Millo, J.V., Ramesh, S., Krishna, S.N., Narwane, G.K.: Compositional Verification of Software Product Lines, pp. 109–123. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-38613-8_8
11. Muschevici, R., Proença, J., Clarke, D.: Feature nets: behavioural modelling of software product lines. Software & Systems Modeling 15(4), 1181–1206 (2016), http://dx.doi.org/10.1007/s10270-015-0475-z