Institut de Recherche en Informatique de Toulouse
CNRS
INPT
UPS
UT1

TECHNICAL REPORT

IRIT/RT–2013-11–FR

Safety Contracts for Timed Reactive Components in SysML

Iulia Dragomir — Iulian Ober — Christian Percebois

Université de Toulouse - IRIT
118 Route de Narbonne, 31062 Toulouse, France
{iulia.dragomir,iulian.ober,christian.percebois}@irit.fr

June 15, 2013

Abstract

A variety of system design and architecture description languages, such as SysML, UML or AADL, allows the decomposition of complex system designs into communicating timed components. In this paper we consider the contract-based specification of such components. A contract is a pair formed of an assumption, which is an abstraction of the component's environment, and a guarantee, which is an abstraction of the component's behavior given that the environment behaves according to the assumption. Thus, a contract concentrates on a specific aspect of the component's functionality and on a subset of its interface, which makes it relatively simpler to specify. Contracts may be used as an aid for hierarchical decomposition during design or for verification of properties of composites. This paper defines contracts for components formalized as a variant of timed input/output automata, introduces compositional results allowing to reason with contracts and shows how contracts can be used in a high-level modeling language (SysML) for specification and verification, based on an example extracted from a real-life system.

Keywords

component, timed input-output automata, contract, V&V, compositional reasoning, SysML

Contents

1 Motivation and approach

The development of safety critical real-time embedded systems is a complex and costly process, and the early validation of design models is of paramount importance for satisfying qualification requirements, reducing overall costs and increasing quality. Design models are validated using a variety of techniques, including design reviews [31], simulation and model-checking [24, 32]. In all these activities system requirements play a central role; for this reason processes-oriented standards such as the DO-178C [29] emphasize the necessity to model requirements at various levels of abstraction and ensure their traceability from high-level down to detailed design and coding.

Since the vast majority of systems are designed with a component-based approach, the mapping of requirements is often difficult: a requirement is in general satisfied by the collaboration of a set of components and each component is involved in satisfying several requirements. A way to tackle this problem is to have partial and abstract component specifications which concentrate on specifying how a particular component collaborates in realizing a particular requirement; such a specification is called a *contract*. A contract is defined as a pair formed of an *assumption*, which is an abstraction of the component's environment, and a *guarantee*, which is an abstraction of the component's behavior given that the environment behaves according to the assumption.

The justification for using contracts is therefore manyfold: support for requirement specification and decomposition, mapping and tracing requirements to components and even for model reviews. Last but not least, contracts can support formal verification of properties through model-checking since, given the right composability properties, they can be used to restructure the verification of a property by splitting it in two steps: (1) verify that each component satisfies its contract and (2) verify that the network of contracts correctly assembles and satisfies the property. Thus, one only needs to reason on abstractions when verifying a property, which potentially induces an important reduction of the combinatorial explosion problem.

Our interest in contracts is driven by potential applications in system engineering using SysML [30], in particular in the verification of complex industrial-scale designs for which we have reached the limits of our tools [22]. In SysML one can describe various types of communicating timed reactive components; for most of these, their semantics can be given in a variant of Timed Input/Output Automata (TIOA) [27]. For this reason, in this paper we concentrate on defining a contract framework for TIOA. The SysML layer is partially explored: we show how contracts can be used for requirement verification without providing details about the language aspects that are subject of a different paper [23].

Paper structure. In §2 we present the contract framework described in [33]. §3 introduces the formal notation for system models and their semantics. In §4 we define the contract framework and we specify the verification relations to be used and their properties, while applying the contract-based theory on a toy-example. In §5 we apply the verification mechanism on an industrial-scale system model, while §6 presents other related approaches, before concluding.

2 A meta-theory for Contract-based Reasoning

Our contract theory is an instance of the *meta-theory* proposed in [34] and later detailed in [33], which formalizes the relations that come into play in a contract theory and the properties that these relations have to satisfy in order to support reasoning with contracts. The term *meta-theory* refers to the fact that [33] does not fix the formalism used for component specification, nor the exact nature of certain relations defined on specifications (conformance, refinement under context). In order to obtain a concrete contract theory for a particular specification formalism one has to define

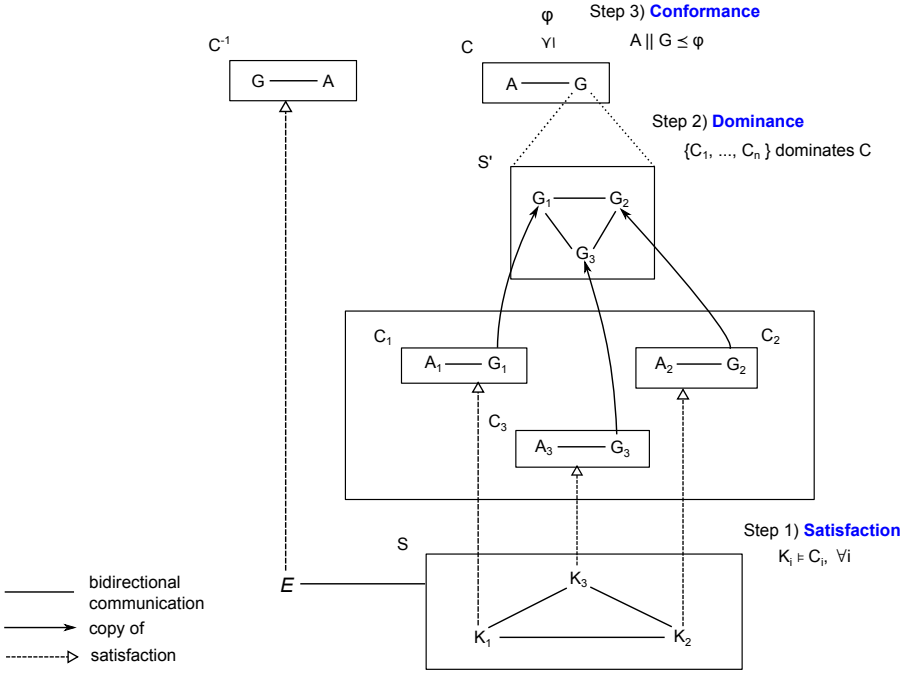

Figure 1: Contract-based reasoning for a subsystem containing three components ([33]).

these relationships such that certain properties, pre-required by the meta-theory, are satisfied. In return, [33] provides a ready-to-use contract-based reasoning methodology, described below.

The purpose of the methodology (illustrated in Figure 1) is to support reasoning with contracts in a system obtained by hierarchical composition of components. At each level of the hierarchy, n components $K_1, ..., K_n$ are combined to form a composite component $K_1 \parallel ... \parallel K_n$, where $\parallel$ denotes the usual parallel composition operator. Then verifying that the the composite satisfies a global property φ runs down to checking that the contracts implemented by $K_1, \cdots, K_n$ combine together correctly to ensure φ. This avoids the need to directly model-check the composite to establish φ and, so, alleviates the combinatorial explosion of the state space. The contracts being specified by more abstract automata, one can assume that their composition will be reduced.

The reasoning proceeds as follows: for each component K_i, a contract C_i is given which consists of an abstraction A_i of the behavior of K_i's environment, and an abstraction G_i that describes the expected behavior of K_i given that the environment acts as A_i. Figure 1 presents three components K_1, K_2 and K_3 and a corresponding set of contracts C_1, respectively C_2 and C_3. Step 1 of the reasoning is to verify that each component is a correct implementation of the contract, i.e. the component *satisfies* its contract.

Step 2 of the reasoning consists in defining a contract $C = (A, G)$ for the composition $K_1 \parallel ... \parallel K_n$ and proving that the set of contracts $\{C_1, C_2, ..., C_n\}$ *implies* C. To do so, the meta-theory of [33] introduces a hierarchy relation between contracts called *dominance* based on an idea first introduced in [28]: a set of contracts $\{C_1, C_2, ..., C_n\}$ *dominates* a contract C if and only if the composition of any valid implementations of $C_1, C_2, ..., C_n$ is an implementation of C. As we will see later, to prove dominance one will have to verify certain conditions on compositions of assumptions and guarantees. In a multi-level hierarchy, the second step can be applied recursively up to a top-level contract, i.e. a contract for the whole (sub)system.

Finally, in the third step one has to prove that the top contract $C = (A, G)$ implies the specification φ. This is done by verifying that $A \parallel G \preceq \varphi$, where $\preceq$ is a *conformance* relation. This step is sufficient for proving that the whole system satisfies φ if and only if either the assumption A is

empty as it is the case when the property is defined for the entire closed modeled system or A is a correct abstraction of the environment E with which S communicates given that S behaves like G, i.e. E satisfies the "mirror" contract $C^{-1} = (G, A)$.

The reasoning strategy presented here assumes that the system designer defines all the contracts necessary for verifying a particular requirement φ. How these contracts are produced is an interesting question but it is outside the scope of this paper. A possible approach is to apply contracts not only *a posteriori* for verification but also during design, in a top-down manner: considering that we want to design a system S that satisfies a requirement φ, we define a closed contract which conforms to φ then we refine the contract until correct implementations can be modeled while verifying at each step that dominance is satisfied. For a posteriori construction of contracts, another possibility is to concentrate on the guarantees and automatically generate the weakest assumptions needed for a component to satisfy a guarantee.

Contributions. The theoretical contribution of this paper is the instantiation of this meta-theory for a variant of Timed Input/Output Automata [27], which further required choosing the appropriate refinement relations and proving that they satisfy certain properties needed for the meta-theory to be applied. Concretely, this involved defining *refinement under context* and *conformance* relations and proving the composability properties required by the meta-theory, all of which can be found in §4.

The practical contribution of this paper is the application of the contract framework to a case study modeled in SysML, which can be found in §5. Due to space limitations we skip the syntactic details of how contracts are expressed in SysML and we concentrate on describing the example, the property of interest and the contracts involved in proving it. The relatively complex SysML language aspects are detailed in [23].

3 Timed Input/Output Automata

Many mathematical formalisms have been proposed in the literature for modelling communicating timed reactive components. We choose to build our framework based on a variant of Timed Input/Output Automata of [27] since it is one of the most general formalisms, thoroughly defined and for which several interesting compositionality results are already available.

Definition 1 (Timed Input/Output Automaton)**.** *A* timed input/output automaton $\mathcal{A}$ *is a tuple* $(X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$ *where*

- X *is a finite set of discrete variables and* Clk *is a finite set of clocks. We denote* $Y = X \cup Clk$ *the set of internal variables.*
- $Q \subseteq val(Y)$ *is a set of states where* $val(Y)$ *is the set of valuations for* Y*. A valuation is a function that associates with each variable from* Y *a value from its domain.*
- $\theta \in Q$ *is the start state.*
- I *is a set of input actions,* O *a set of output actions and* V *a set of visible actions. We denote* $E = I \cup O \cup V$*. Input actions are represented by* ? *before their name and output actions are represented by* ! *before their name.*
- H *is a set of internal actions. We denote* $A = E \cup H$*.*
- $I/O, V$ *and* H *are pairwise disjoint.*
- $D \subseteq Q \times A \times Q$ *is the set of discrete transitions. We denote by* $\mathrm{x} \xrightarrow{a} \mathrm{x}'$ *any* $(\mathrm{x}, a, \mathrm{x}') \in D$*.*

- *$\mathcal{T}$ is the set of trajectories. Each trajectory is a function $\tau : J_\tau \to Q$, where J_τ is a real interval of type $[0, t]$ or $[0, \infty)$ with $t \in \mathbb{R}^+$.*

The difference between the TIOA of [27] and our variant is that in addition to *inputs* and *outputs*, we allow for another type of *visible* actions; this is because, in [27], when composing two automata, an *output* of one matched by an *input* of the other becomes an *output* of the composite, which does not correspond to our needs when using the TIOA for defining the semantics of usual modelling languages like SysML. As we will show, in order for contract dominance to work, we still need the resulting action to be *visible* in traces, hence the necessity for an additional type of actions.

Moreover, note that in the following we will limit our attention to trajectories that are constant functions for discrete variables, and linear functions with derivative equal to 1 for clocks, while [27] allows more general functions to be used as trajectories. This restriction makes the model expressiveness equivalent to that of Alur-Dill timed automata [2], and will be important later on as it opens the possibility to automatically verify simulation relations between automata (simulation is undecidable for the TIOA of [27]). It also simplifies the presentation of examples, since trajectories are then fully determined by their domain, so we simply use the interval J to represent the trajectory. However, this hypothesis is not needed for proving the compositionality results in §4, and so we did not integrate it to the definition to preserve generality. Furthermore, to simplify the presentation, we limit the domain of a trajectory to closed intervals of type $[0, t]$ with $t \in \mathbb{R}^+$ or open intervals of type $[0, \infty)$.

For a trajectory τ we denote $\tau.ltime$ to be the supremum of its domain. A trajectory is *closed* if the domain is a closed interval.

A timed input/output automaton has to satisfy the following axioms:

A0) *(Existence of point trajectories)*
$\forall \mathrm{x} \in Q,\ \gamma(\mathrm{x}) \in \mathcal{T}$ where $\gamma(\mathrm{x}) : [0, 0] \to \mathrm{x}$ maps 0 to x.

A1) *(Prefix closure)*
$\forall \tau \in \mathcal{T}, \forall \tau'$ a prefix of τ (i.e. τ' obtained by restricting τ to $[0, t]$ with $t \in J_\tau$), $\tau' \in \mathcal{T}$.

A2) *(Suffix closure)*
$\forall \tau \in \mathcal{T}, \forall \tau'$ a suffix of τ, $\tau' \in \mathcal{T}$. τ' is a suffix if $\exists t \in J_\tau$ such that $\tau' : [0, \tau.ltime - t] \to Q$ if τ is closed, $\tau' : [0, \infty) \to Q$ if τ is open, and $\tau'(u) = \tau(t + u)$ (i.e. τ' obtained by restricting τ to $J_\tau \cap [t, \infty)$ and left-shifting it such that $J_{\tau'}$ starts in 0).

A3) *(Concatenation closure)*
Let $\tau_0\tau_1\tau_2\cdots$ be a (finite or countably infinite) sequence of trajectories in $\mathcal{T}$ such that, for each nonfinal index i, τ_i is closed and $\tau_i(\tau_i.ltime) = \tau_{i+1}(0)$. Then $\tau_0{}^\frown\tau_1{}^\frown\tau_2{}^\frown\cdots \in \mathcal{T}$, where $^\frown$ denotes the concatenation operator (i.e. the union between a first closed trajectory and a second one right-shifted such that its start time coincides to the limit time of the first one).

A4) *(Input actions enabling)*
$\forall \mathrm{x} \in Q, \forall a \in I, \exists \mathrm{x}' \in Q$ such that $\mathrm{x} \xrightarrow{?a} \mathrm{x}'$.

A5) *(Time-passage enabling)*
$\forall \mathrm{x} \in Q, \exists \tau \in \mathcal{T}$ such that $\tau(0) = \mathrm{x}$ and either

1. $\tau.ltime = \infty$, or
2. τ is closed and some $l \in H \cup V \cup O$ is enabled in $\tau(\tau.ltime)$.

3.1 Notation and running example

For compactness and understandability, in this paper we use a graphical notation, only intended for informally representing TIOA examples, which implicitly defines states, actions, transitions and trajectories (in the sense of the TIOA definition). An example is shown in Figure 2.

We are interested in automata communicating by asynchronous messages. An automaton is contained within a frame, arrows between frames represent messages that are *output* by one automaton and *input* by the other. It is assumed that each automaton has an implicit variable *queue* which stores the incoming messages. The input-enabledness axiom (A4) of TIOA is well suited here: in any state, the automaton can *input* a message and store it in the queue. These *input* transitions are not represented.

It is assumed that each automaton has a *location* variable which ranges in a finite domain, with values represented by black dots. Other variables can be used; in the example we only use integers (i, j) and clocks (x, y).

An arc between a dot s_1 and a dot s_2 is a template representing a set of *discrete transitions* (in the sense of the TIOA definition) between states q_1 and q_2 such that $q_1.location = s_1$ and $q_2.location = s_2$. On an arc we may represent several information:

- A guard condition between brackets: the meaning is that the TIOA transition exists only if the condition is satisfied in the starting state q_1.
- A message consumption guard, denoted $\downarrow m$: the meaning is that the TIOA transition exists only if in q_1 the message queue begins with a message m. The queue of q_2 must then be equal to the tail of the q_1 queue. *Note*: $\downarrow m$ is not an *input* action in the sense of TIOA. As mentioned, *input* actions are not explicitly represented since TIOAs are input-complete.
- An *output action* for a message m denoted $!m$
- A sequence of assignments of discrete and clock variables: the meaning is that the TIOA transition exists only if q_2 can be obtained from q_1 by applying these assignments.
- An *urgency* label, which implicitly constrains the set of trajectories starting in q_1. We use the notion of urgency defined for Timed Automata in [7]. This means that, by default, in any state, any trajectory with an arbitrary domain $[0, j]$ or $[0, \infty)$ is allowed. This set of trajectory is restricted by the urgency labels of transitions outgoing from q_1 as follows:
  - *lazy* transitions do not add any restrictions.
  - *eager* transitions with no clock guard restrict the set of trajectories to point trajectories only. *Eager* transitions with a clock guard restrict the set of trajectories so that they end in the smallest j where the guard is true.

The subsystem K of the running example (Figure 2) contains three components K_1, K_2 and K_3 represented as timed input/output automata: K_1 sends a message a to the environment and a message p to K_2, then awaits a message q from K_2. If q is received before a deadline (constant δ_1), K_1 emits a again (the q-a cycle represented in red), otherwise it goes back to the initial state when q is received. In addition, K_1 counts the number of a's emitted (in i), and can answer a message m with a message $n(i)$ in any state (the m-n cycle represented with green).

K_2 waits for p then sends a message b to the environment. After that, it waits for δ_2 time units and sends q to K_1; if p is received during this time, b is emitted again (the p-b cycle represented in red). K_2 counts the number of b's emitted (in j), and can answer a message u with a message $v(j)$ in any state (the u-v cycle represented with blue).

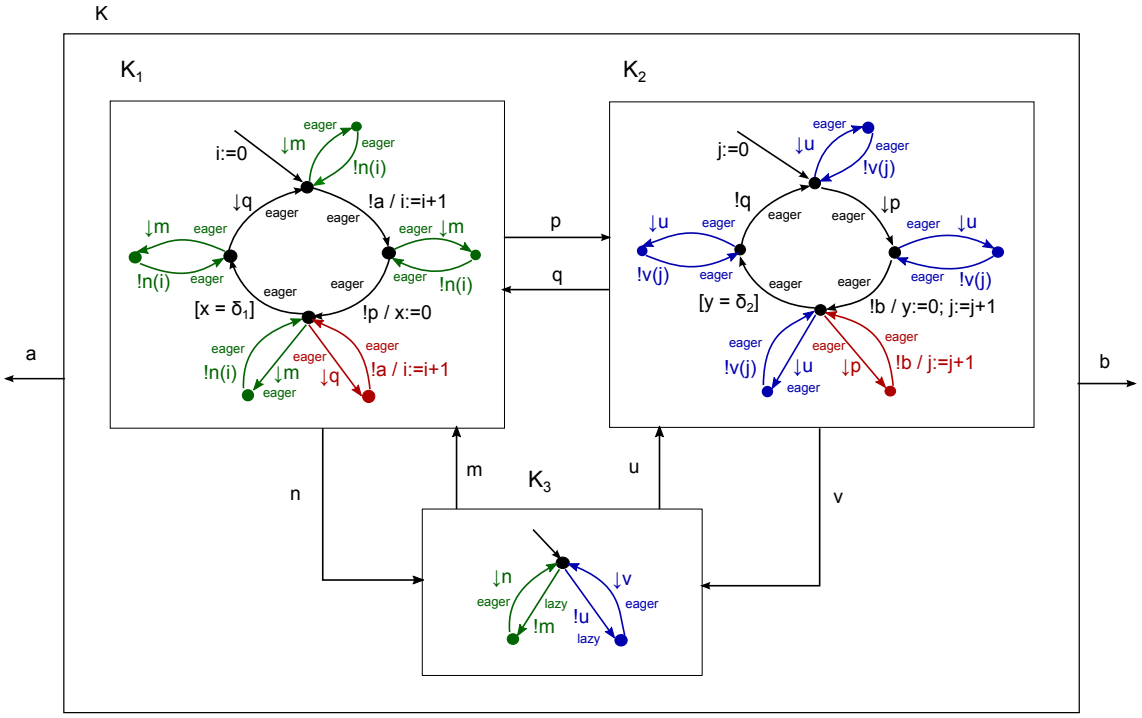

Figure 2: Running example containing three components that interchange p, q, m, n, u and v messages and send to the environment sequences of a and b.

K_3 sporadically sends m (represented in green) and u (represented in blue) to K_1 respectively K_2. The purpose of component K_3 is to show how signature refinement is taken into consideration when contracts are defined and refinement is checked.

The interesting property of this system is that, if $\delta_1 < \delta_2$, then the composition emits a sequence alternating a's and b's. This safety requirement is modelled in Figure 3. We use it as a running example to show how this can be proved using contracts.

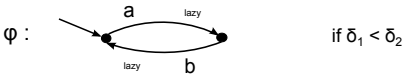

Figure 3: The requirement that the running example has to satisfy.

3.2 TIOA behaviour and composition

The behaviour of a timed input/output automaton is given by sets of execution fragments. An *execution fragment* records what happens during a particular run of an automaton including discrete changes of states and changes that occur during time passage (trajectories).

Definition 2 (Execution fragment)**.** *An* execution fragment *of a timed input-output automaton $\mathcal{A}$ is a (possibly infinite) sequence $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \cdots$ where (1) each a_i is an action in $A_{\mathcal{A}}$, (2) each τ_i is a trajectory in $\mathcal{T}$, (3) all τ_i are closed except the last trajectory which can be either open or closed and (4) if τ_i is not the last trajectory in α then $\tau_i(\tau_i.ltime) \xrightarrow{a_{i+1}} \tau_{i+1}(0)$.*

An execution fragment is *closed* if it is a finite sequence and its final trajectory is closed. For the timed input-output automaton K_1, an execution fragment is $\alpha = [0,0]\, !a\, [0,0]\, !p\, [0,\delta_1]\, \epsilon\, [0,\delta_2-$

$\delta_1]\ ?q\ [0,0]\ \downarrow q\ [0,0]$, where ϵ denotes the silent transition of clock comparison and $?q$ the input of message q. Note that we represent trajectories by their domain.

A trace is a projection of an execution fragment which preserves only the external actions and the time elapse intervals. To define it, we use an operator for *restricting* an execution trace α of $\mathcal{A}$ on a pair (A, Y) where $A \subseteq A_{\mathcal{A}}$ is a subset of actions and $Y \subseteq Y_{\mathcal{A}}$ a subset of variables. The restriction, denoted $\alpha\lceil(A, Y)$, is computed as follows: project all trajectories of α on the variables in Y (i.e. only the evolution of variables in Y is followed by the trajectory, denoted $\tau_i\lceil Y$), remove the actions not contained in A and concatenate all adjacent trajectories. The formalization and properties of this operator can be found in [27].

Definition 3 (Trace)**.** *Let α an execution fragment. Then $trace(\alpha)$ is the restriction of α to $(E_{\mathcal{A}}, \emptyset)$, denoted $trace(\alpha) = \alpha\lceil(E_A, \emptyset)$ (i.e. a_i is an action in $E_{\mathcal{A}}$ and $\tau_i : J_{\tau_i} \to \emptyset$, $J_{\tau_i} \subseteq \mathbb{R}^+$, records only the length of the time-passage).*

We denote by $traces_{\mathcal{A}}$ the set of traces of the automaton $\mathcal{A}$. The trace for the previously presented execution fragment is $trace(\alpha) = [0,0]\ !a\ [0,0]\ !p\ [0,\delta_2]\ ?q\ [0,0]$.

The following definitions present the conditions which have to be satisfied in order for two automata to compose and the *parallel composition* operator.

Definition 4 (Composition compatibility)**.** *Two timed input-output automata K_1 and K_2 are* compatible *if for $i \neq j$, $Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset$.*

Syntactically, the parallel composition operator models the output-input synchronization and interleaving of all non-matched actions including internal and visible actions of the components involved.

Definition 5 (Parallel composition)**.** *If K_1 and K_2 are two compatible timed input-output automata then their* composition *$K_1 \parallel K_2$ is defined to be the tuple $(X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$ where*

- *$X = X_1 \cup X_2$ and $Clk = Clk_1 \cup Clk_2$*
- *$Q = \{\mathrm{x}_1 \cup \mathrm{x}_2 | \mathrm{x}_1 \in Q_1, \mathrm{x}_2 \in Q_2\}$. Note that $\mathrm{x}_1 \cup \mathrm{x}_2$, which denotes the set union of functions x_1 and x_2, is well defined since the domains of x_1 and x_2 are disjoint.*
- *$\theta = \theta_1 \cup \theta_2$.*
- *$I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$, $O = (O_1 \setminus I_2) \cup (O_2 \setminus I_1)$ and $V = V_1 \cup V_2 \cup (I_1 \cap O_2) \cup (O_1 \cap I_2)$.*
- *$H = H_1 \cup H_2$.*
- *D is the set of discrete transitions where for each $\mathrm{x} = \mathrm{x}_1 \cup \mathrm{x}_2, \mathrm{x}' = \mathrm{x}'_1 \cup \mathrm{x}'_2 \in Q$ and each $a \in A$, $\mathrm{x} \xrightarrow{a} \mathrm{x}'$ if and only if for $i \in \{1,2\}$, either*
  1. *$a \in A_i$ and $\mathrm{x}_i \xrightarrow{a} \mathrm{x}'_i$, or*
  2. *$a \notin A_i$ and $\mathrm{x}_i = \mathrm{x}'_i$.*
- *$\mathcal{T}$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau\lceil X_i \in \mathcal{T}_i$, $i \in \{1,2\}$.*

The only difference with respect to the parallel composition operator defined in [27] is related to the interface of the composite timed input/output automata: the input and output sets of the composite are given by thos actions not matched between components, while all matched input-output pairs become visible actions.

Theorem 1. *The parallel composition operator is commutative and associative.*

Proof. Let K_1, K_2 and K_3 be three timed input/output automata.
1) Commutativity: $K_1 \parallel K_2 = K_2 \parallel K_1$ is true since the composition operator doesn't take into account the order of composition.
2) Associativity: $(K_1 \parallel K_2) \parallel K_3 = K_1 \parallel (K_2 \parallel K_3)$

By applying the composition operator definition, one can verify that $(K_1 \parallel K_2) \parallel K_3 = K_1 \parallel (K_2 \parallel K_3) = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$ where

- $X = X_{K_1} \cup X_{K_2} \cup X_{K_3}$.
- $Clk = Clk_{K_1} \cup Clk_{K_2} \cup Clk_{K_3}$.
- $Q = \{\mathrm{x}_1 \cup \mathrm{x}_2 \cup \mathrm{x}_3 | \mathrm{x}_1 \in Q_{K_1}, \mathrm{x}_2 \in Q_{K_2} \text{ and } \mathrm{x}_3 \in Q_{K_3}\}$.
- $\theta = \theta_1 \cup \theta_2 \cup \theta_3$.
- $I = (I_{K_1} \setminus (O_{K_2} \cup O_{K_3})) \cup (I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup (I_{K_3} \setminus (O_{K_1} \cup O_{K_2}))$.
- $O = (O_{K_1} \setminus (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \setminus (I_{K_1} \cup I_{K_2}))$.
- $V = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \cap (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \cap (I_{K_1} \cup I_{K_2}))$.
- $H = H_{K_1} \cup H_{K_2} \cup H_{K_3}$.
- D is the set of discrete transitions where for each $\mathrm{x} = \mathrm{x}_1 \cup \mathrm{x}_2 \cup \mathrm{x}_3$ and $\mathrm{x}' = \mathrm{x}'_1 \cup \mathrm{x}'_2 \cup \mathrm{x}'_3 \in Q$ and each $a \in A$, $\mathrm{x} \xrightarrow{a} \mathrm{x}'$ if and only if for $i \in \{1, 2, 3\}$, either
  1. $a \in A_{K_i}$ and $\mathrm{x}_i \xrightarrow{a} \mathrm{x}'_i$ or
  2. $a \notin A_{K_i}$ and $\mathrm{x}_i = \mathrm{x}'_i$.
- $\mathcal{T} \subseteq trajs(Q)$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau\lceil X_{K_i} \in \mathcal{T}_i, i \in \{1, 2, 3\}$.

The complete proof on the sets I, O and V of actions can be found in the Appendix A. □

4 Contracts for Timed Input/Output Automata

In this section we formalize contracts for TIOA and the relations described in §2 and we list the properties that have been proved upon these and that make contract-based reasoning possible.

Definition 6 (Component)**.** *A* component K *is a timed input-output automaton.*

Definition 7 (Environment)**.** *Let K be a component. An* environment E *for the component K is a timed input/output automaton comptible with K for which the following hold: $I_E \subseteq O_K$ and $O_E \subseteq I_K$.*

Definition 8 (Closed/open component)**.** *A component K is* closed *if $I_K = O_K = \emptyset$. A component is* open *if it not closed.*

Closed components result from the composition of components with complementary interfaces.

Definition 9 (Contract)**.** *A* contract *for a component K is a pair (A, G) of TIOA such that $I_A = O_G$ and $I_G = O_A$ (i.e. the composition pair $A \parallel G$ defines a closed component) and $I_G \subseteq I_K$ and $O_G \subseteq O_K$ (i.e. the interface of G is a subset of that of K). A is called the* assumption *over the environment of the component and G is called the* guarantee*. The* interface *of a contract is the set of actions of its guarantee.*

Figure 4 presents a set of contracts for the two components K_1 and K_2 of the running example. In the case of K_1, the assumption over the environment sends a q after at least δ_1 time units since a p is received and the component guarantees that consecutive a's are triggered by a message q from the environment. For K_2, the environment guarantees that it will wait for a q between sending two consecutive p's and the component guarantees that it waits for a p before sending a b and then for a delay of δ_2 time units before a q. Recall that the property we want to verify (Figure 3) is defined on the subset $\{a, b\}$ of actions. Thus K_3 whose action set is $\{m, n, u, v\}$ does not contribute to the interface of the subsystem K and neither to the satisfaction of this property. Its contract for this requirement is given by two empty timed input/output automata (i.e. the sets of variables and actions of the automaton are empty, and all trajectories up to ∞ are admitted). For each property that we want to verify, a set of contracts has to modelled by the user.

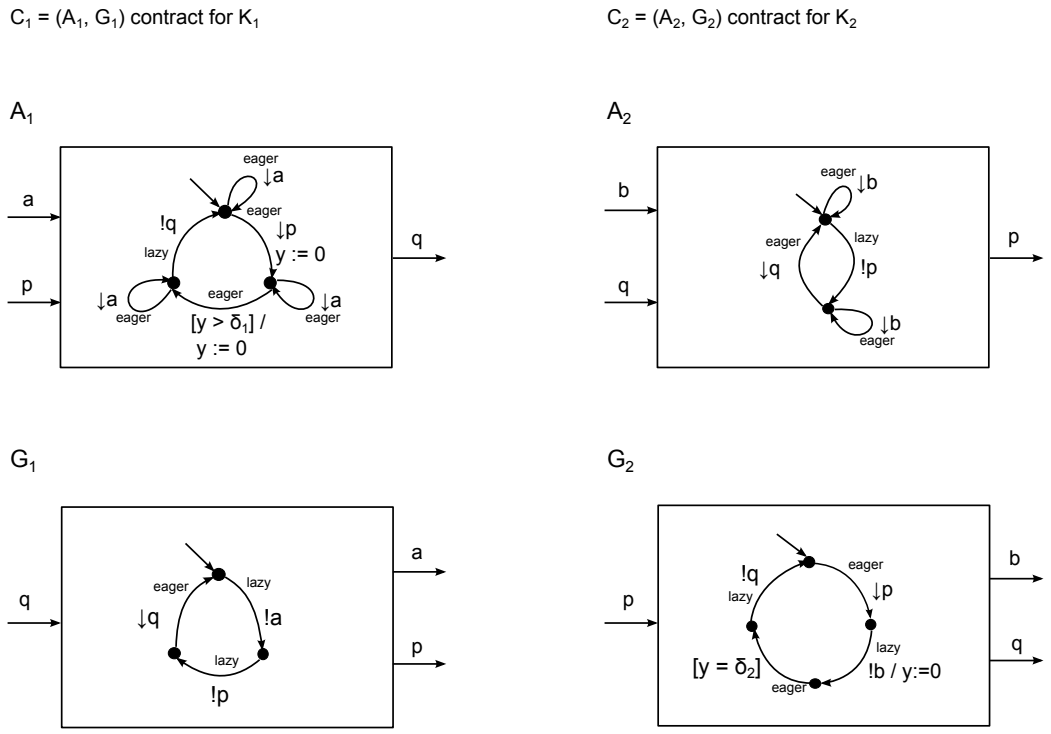

Figure 4: Contracts for the components K_1 and K_2 of the running example.

The first step of the verification, as presented in §2, is to prove that the modelled components are an implementation of the given contracts. For this we specify the notions of *conformance* and *refinement under context* introduced in [33], notions that both need to be preorder relations in order to establish compositional reasoning results.

As in [27], we use trace inclusion to check the refinement relation between components:

Definition 10 (Comparable components)**.** *Two components K_1 and K_2 are* comparable *if they have the same external interface, i.e. $E_{K_1} = E_{K_2}$.*

Definition 11 (Conformance)**.** *Let K_1 and K_2 be two comparable components.* K_1 conforms to K_2*, denoted $K_1 \preceq K_2$, if $traces_{K_1} \subseteq traces_{K_2}$.*

This relation is also used for verifying the satisfaction of the system's properties by the top contract: $A \parallel G \preceq \varphi$, where $A \parallel G$ and φ have the same interface. It can be easily shown that conformance is a preorder. The following useful compositional result, presented in the TIOA theory of [27], can be easily extended to our variant of TIOA:

Theorem 2 (Composability theorem 8.5 of [27])**.** *Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.*

The *refinement under context* relation formalizes that, given an environment compatible with two components, one is a refinement of the other in the specified context. We define this relation based on conformance. Since we want to take into account interface refinement between components and conformance imposes comparability, we have to compose each member of the conformance relation obtained from refinement under context with an additional timed input/output automaton such that they both define closed comparable systems.

Definition 12 (Refinement under context)**.** *Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that K_1* refines *K_2 in the context of E, denoted $K_1 \sqsubseteq_E K_2$, if*

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where K' and E' are defined such that both members of the conformance relation are comparable, as follows:

- $E' = (\emptyset, \{t_{E'}\}, \{\phi\}, \phi, (O_{K_1} \setminus I_E), (I_{K_1} \setminus O_E), \emptyset, \emptyset, D_{E'}, \mathcal{T}_{E'})$ *where ϕ is the function $\emptyset \mapsto \emptyset$, $t_{E'}$ the unique clock of E', $D_{E'} = \{(\phi, a, \phi) | \forall a \in E_{E'}\}$ and $\mathcal{T}_{E'} = \{[0,t] | t \in \mathbb{R}^+\} \cup \{[0,\infty)\}$ contains all possible trajectories,*
- $K' = (\emptyset, \{t_{K'}\}, \{\phi\}, \phi, ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})), ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})), (V_{K_1} \setminus E_{K_2}), \emptyset, D_{K'}, \mathcal{T}_{K'})$ *where ϕ is the function $\emptyset \mapsto \emptyset$, $t_{K'}$ the unique clock of K', $D_{K'} = \{(\phi, a, \phi) | \forall a \in E_{K'}\}$ and $\mathcal{T}_{K'} = \{[0,t] | t \in \mathbb{R}^+\} \cup \{[0,\infty)\}$.*

Informally, K' is an automaton that contains the actions defined by the interface of the concrete component K_1 and not present in its abstraction, the component K_2, such that K_1 and $K_2 \parallel K'$ are comparable. The automaton E' is defined to close the composition $K_1 \parallel E$ and $K_2 \parallel E \parallel K'$, since an environment may be defined on a subset of the component's interface. So, the interface of E' is given by the actions of K_1 and $K_2 \parallel K'$ not matched by E at composition and with reversed directionality.

The particular inclusion relations between the interfaces of K_1 and K_2 in the previous definition is due to the fact that both K_1 and K_2 can be components obtained from composition: $K_1 = K_1' \parallel K_3$ and $K_2 = K_2' \parallel K_3$, where $I_{K_2'} \subseteq I_{K_1'}$, $O_{K_2'} \subseteq O_{K_1'}$ and $V_{K_2'} \subseteq V_{K_1'}$. This happens in particular when K_2' is a contract guarantee for K_1'. Then, by composition, actions of K_3 may be matched by actions of K_1' but have no input/output correspondent in K_2'. This case also imposes the term $V_{K_1} \cap O_{K_2}$ for the inputs of K', since the additional outputs of K_2 may belong to a different component, and the term $V_{K_1} \cap I_{K_2}$ for the outputs of K'.

Notation. In the following we use $\triangleq$ to express that *by definition* the left-hand side and right-hand side terms are *equivalent*. We denote the partition of intervals on $\mathbb{R}^+$ having 0 as left end-point by $2_0^{[\mathbb{R}^+]}$, i.e. the set $\{[0,t] | t \in \mathbb{R}^+\} \cup \{[0,\infty)\}$.

Theorem 3. *Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.*

Proof. 1) *Reflexivity*: $K \sqsubseteq_E K \Leftrightarrow K \parallel E \preceq K \parallel E$ which is true from the definition of *conformance* relation.

2) *Transitivity*: $K_1 \sqsubseteq_E K_2 \wedge K_2 \sqsubseteq_E K_3 \implies K_1 \sqsubseteq_E K_3$.

$K_1 \sqsubseteq_E K_2 \triangleq K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K_2' \parallel E'$ (1)

We write the automaton $E' = E_1 \parallel E_2$ where

- $E_1 = (\emptyset, \{\phi\}, \phi, ((O_{K_1} \cap O_{K_2}) \setminus I_E), ((I_{K_1} \cap I_{K_2}) \setminus O_E), \emptyset, \emptyset, D_{E_1}, 2_0^{[\mathbb{R}^+]})$
- $E_2 = (\emptyset, \{\phi\}, \phi, ((O_{K_1} \setminus O_{K_2}) \setminus I_E), ((I_{K_1} \setminus I_{K_2}) \setminus O_E), \emptyset, \emptyset, D_{E_2}, 2_0^{[\mathbb{R}^+]})$

One can remark that the sets of input and output actions are pairwise disjoint for E_1 and E_2.
We write the automaton $K_2' = K_2'' \parallel E_3$ where

- $K_2'' = (\emptyset, \{\phi\}, \phi, (I_{K_1} \setminus I_{K_2}), (O_{K_1} \setminus O_{K_2}), (V_{K_1} \setminus E_{K_2}), \emptyset, D_{K_2''}, 2_0^{[\mathbb{R}^+]})$
- $E_3 = (\emptyset, \{\phi\}, \phi, (V_{K_1} \cap O_{K_2}), (V_{K_1} \cap I_{K_2}), \emptyset, \emptyset, D_{E_3}, 2_0^{[\mathbb{R}^+]})$

Similarly, the sets of input, output and visible actions are pairwise disjoint for K_2'' and E_3.
Then (1) $\Leftrightarrow K_1 \parallel E \parallel E_1 \parallel E_2 \preceq K_2 \parallel E \parallel K_2'' \parallel E_3 \parallel E_1 \parallel E_2$ (2)

$K_2 \sqsubseteq_E K_3 \overset{\Delta}{\Leftrightarrow} K_2 \parallel E \parallel E'' \preceq K_3 \parallel E \parallel K_3' \parallel E''$ (3)
With the previous notations we have that $E'' = E_1 \parallel E_3$.
Then (3) $\Leftrightarrow K_2 \parallel E \parallel E_1 \parallel E_3 \preceq K_3 \parallel E \parallel K_3' \parallel E_1 \parallel E_3$ (4)
Composing (4) with $K_2'' \parallel E_2$ and from Theorem 2 we get

$$K_2 \parallel E \parallel E_1 \parallel E_3 \parallel K_2'' \parallel E_2 \preceq K_3 \parallel E \parallel K_3' \parallel E_1 \parallel E_3 \parallel K_2'' \parallel E_2 \Leftrightarrow$$

$$\left.\begin{array}{r} \Leftrightarrow K_2 \parallel E \parallel K_2'' \parallel E_3 \parallel E_1 \parallel E_2 \preceq K_3 \parallel E \parallel K_3' \parallel K_2' \parallel E_1 \parallel E_2 \\ (2)\ K_1 \parallel E \parallel E_1 \parallel E_2 \preceq K_2 \parallel E \parallel K_2'' \parallel E_3 \parallel E_1 \parallel E_2 \end{array}\right\} \overset{Transitivity}{\Longrightarrow}$$

$$\Longrightarrow K_1 \parallel E \parallel E_1 \parallel E_2 \preceq K_3 \parallel E \parallel K_3' \parallel K_2' \parallel E_1 \parallel E_2 \Leftrightarrow$$
$$\Leftrightarrow K_1 \parallel E \parallel E' \preceq K_3 \parallel E \parallel K_2' \parallel K_3' \parallel E'$$

By denoting $K' = K_2' \parallel K_3'$ we have:

$$K_1 \parallel E \parallel E' \preceq K_3 \parallel E \parallel K' \parallel E' \overset{\Delta}{\Leftrightarrow} K_1 \sqsubseteq_E K_3$$

The last step consists in proving that K' is the automaton generated by the refinement under context relation. Since K_2' and K_3' are built from the hypothesis by the refinement under context relation, by composition they define the correct structure for K'. Moreover $I_{K'} = (I_{K_1} \setminus I_{K_3}) \cup (V_{K_1} \cap O_{K_3})$, $O_{K'} = (O_{K_1} \setminus O_{K_3}) \cup (V_{K_1} \cap I_{K_3})$ and $V_K = V_{K_1} \setminus E_{K_3}$. The proofs on the sets of actions are detailed in Appendix A. □

The following results, required to allow reasoning with contracts as shown in [33], hold in our framework:

Theorem 4 (Compositionality)**.** *Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$. $K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$.*

Proof. $K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \overset{\Delta}{\Leftrightarrow} K_1 \parallel E_1 \parallel E_2 \parallel E' \preceq K_2 \parallel E_1 \parallel E_2 \parallel K' \parallel E'$ (1)
with

- $E' = (\emptyset, \{\phi\}, \phi, (O_{K_1} \setminus I_{E_1 \parallel E_2}), (I_{K_1} \setminus O_{E_1 \parallel E_2}, \emptyset, \emptyset, D_{E'}, 2_0^{[\mathbb{R}^+]})$,
- $K' = (\emptyset, \{\phi\}, \phi, ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})), ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})), (V_{K_1} \setminus E_{K_2}), \emptyset, D_{K'}, 2_0^{[\mathbb{R}^+]})$.

$K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1 \overset{\Delta}{\Leftrightarrow} K_1 \parallel E_1 \parallel E_2 \parallel E'' \preceq K_2 \parallel E_1 \parallel E_2 \parallel K'' \parallel E''$ (2)
with

- $E'' = (\emptyset, \{\phi\}, \phi, (O_{K_1 \| E_1} \setminus I_{E_2}), (I_{K_1 \| E_1} \setminus O_{E_2}), \emptyset, \emptyset, D_{E''}, 2_0^{[\mathbb{R}^+]})$,
- $K'' = (\emptyset, \{\phi\}, \phi, ((I_{K_1 \| E_1} \setminus I_{K_2 \| E_1}) \cup (V_{K_1 \| E_1} \cap O_{K_2 \| E_1})), ((O_{K_1 \| E_1} \setminus O_{K_2 \| E_1}) \cup (V_{K_1 \| E_1} \cap I_{K_2 \| E_1})), (V_{K_1 \| E_1} \setminus E_{K_2 \| E_1}), \emptyset, D_{K''}, 2_0^{[\mathbb{R}^+]})$.

We prove the equivalence of relations (1) and (2) by showing the equality between the additional generated timed input/output automata involved in the composition: $E' = E''$ and $K' = K''$.

$E' = E'' = (\emptyset, \{\phi\}, \phi, (O_{K_1} \setminus (I_{E_1} \cup I_{E_2}), (I_{K_1} \setminus (O_{E_1} \cup O_{E_2})), \emptyset, \emptyset, D_{E'}, 2_0^{[\mathbb{R}^+]})$
$K' = K'' = (\emptyset, \{\phi\}, \phi, ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})), ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_1})), V_{K_1} \setminus E_{K_2}, \emptyset, D_{K'}, 2_0^{[\mathbb{R}^+]})$
The equality between the sets of actions is shown in Appendix A. □

Theorem 5 (Soundness of circular reasoning)**.** *Let K be a component, E its environment and $C = (A, G)$ the contract for K such that K and G are compatible with each of E and A. If*

1. *$traces_G$ is closed under limits,*
2. *$traces_G$ is closed under time-extension,*
3. *$K \sqsubseteq_A G$ and*
4. *$E \sqsubseteq_G A$*

then $K \sqsubseteq_E G$.

Proof. $K \sqsubseteq_A G \overset{\Delta}{\Leftrightarrow} K \parallel A \parallel A^* \preceq G \parallel A \parallel G^* \parallel A^*$
We write $A^* = A_1 \parallel A_2$ with

- $A_1 = (\emptyset, \{\phi\}, \phi, (O_K \setminus I_E), (I_K \setminus O_E), \emptyset, \emptyset, D_{A_1}, 2_0^{[\mathbb{R}^+]})$,
- $A_2 = (\emptyset, \{\phi\}, \phi, (I_E \setminus I_A), (O_E \setminus O_A), \emptyset, \emptyset, D_{A_2}, 2_0^{[\mathbb{R}^+]})$.

This partition of the sets of interfaces is complete due to the relation between the interfaces of the components $A_G = A_A \subseteq A_E \subseteq A_K$ obtained from the definitions of contract and environment.

Similarly, we write $G^* = G_1 \parallel G_2$ with

- $G_1 = (\emptyset, \{\phi\}, \phi, (I_K \setminus O_E), (O_K \setminus I_E), (V_K \setminus E_G), \emptyset, D_{G_1}, 2_0^{[\mathbb{R}^+]})$,
- $G_2 = (\emptyset, \{\phi\}, \phi, (O_E \setminus I_G), (I_E \setminus O_G), \emptyset, \emptyset, D_{G_2}, 2_0^{[\mathbb{R}^+]})$.

Then
3) $\overset{\Delta}{\Leftrightarrow} K \parallel A \parallel A_1 \parallel A_2 \preceq G \parallel A \parallel G_1 \parallel G_2 \parallel A_1 \parallel A_2$
4) $\overset{\Delta}{\Leftrightarrow} E \parallel G \parallel G_2 \preceq A \parallel G \parallel A_2 \parallel A_3 \parallel G_2$
with $A_3 = (\emptyset, \{\phi\}, \phi, \emptyset, \emptyset, (V_E \setminus E_A), \emptyset, D_{A_3}, 2_0^{[\mathbb{R}^+]})$.

With this notation the conclusion becomes: $K \parallel E \parallel A_1 \preceq G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$.

We prove this relation in two steps: every closed trace of $K \parallel E \parallel A_1$ is a trace of $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$ and every non-closed trace of $K \parallel E \parallel A_1$ is a trace of $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$. Let β be a trace. By $\beta \lceil (B, \emptyset)$ we denote the projection of β on the set of actions B.

1) Every closed trace of $K \parallel E \parallel A_1$ is also a trace of $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$. Proof by *induction*.

Step 1) Let $\beta \in trajs(\emptyset)$ a trace of $K \parallel E \parallel A_1$. From axiom A0) we have that $\exists \tau_\alpha$ a point trajectory of G such that $\alpha.ltime = 0$. Since $traces_G$ are closed under time-extension $\Rightarrow \alpha^\frown \beta = \beta \in traces_G$ (1)

From the definition of G_1 and G_2 we can similarly deduce that $\beta \in traces_{G_1}$ (2) and $\beta \in traces_{G_2}$ (3)

From hypothesis we have that $\beta \in traces_{K\|E\|A_1}$ which implies that $\beta \in traces_{E\|A_1}$ (4)

(1), (2), (3) and (4) $\implies \beta \in traces_{G\|E\|G_1\|G_2\|A}$

Step 2) Let β' a trace of $K \parallel E \parallel A_1$ and $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$. Let $\beta = \beta' \frown \beta''$ a trace of $K \parallel E \parallel A_1$.

$?\beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

a) $\beta = \beta' a \tau$ where a is an output action of K and τ a point trajectory.
From hypothesis we have that $\beta' \in traces_{G\|E\|G_1\|G_2\|A_1} \Rightarrow$

$$\left.\begin{array}{r} \beta' \in traces_{E\|G\|G_2} \\ E \parallel G \parallel G_2 \preceq A \parallel G \parallel A_2 \parallel A_3 \parallel G_2 \end{array}\right\} \Rightarrow \beta' \in traces_{A\|G\|A_2\|A_3\|G_2}$$

$\Rightarrow \beta' \lceil (E_A, \emptyset) \in traces_A$ and $\beta' \lceil (E_{A_2}, \emptyset) \in traces_{A_2}$.
$I_E \cup I_{A_1} = O_K$. We have two cases:

i) $a \in I_{A_1}$. From hypothesis we have $\beta \in traces_{K\|E\|A_1} \Rightarrow \beta' \lceil (E_{K\|A_1}, \emptyset) \in traces_{K\|A_1}$.
$a \notin I_A \Rightarrow \beta \lceil (E_A, \emptyset) = \beta' \lceil (E_A, \emptyset) \in traces_A$.
$a \notin I_{A_2} \Rightarrow \beta \lceil (E_{A_2}, \emptyset) = \beta' \lceil (E_{A_2}, \emptyset) \in traces_{A_2}$

$$\left.\begin{array}{r} \Rightarrow \beta \in traces_{K\|A\|A_1\|A_2} \\ K \parallel A \parallel A_1 \parallel A_2 \preceq G \parallel A \parallel G_1 \parallel G_2 \parallel A_1 \parallel A_2 \end{array}\right\} \Rightarrow$$

$\Rightarrow \beta \in traces_{G\|A\|G_1\|G_2\|A_1\|A_2} \Rightarrow \beta \lceil (E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$ (5)
$\beta \in traces_{K\|E\|A_1} \Rightarrow \beta \lceil (E_{E\|A_1}, \emptyset) \in traces_{E\|A_1}$ (6)

(5) and (6) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

ii) $a \in I_E$, $I_E = I_A \cup I_{A_2}$. We have two cases:

ii.1) $a \in I_A$. Let α an execution of A such that $trace(\alpha) = \beta' \lceil (E_A, \emptyset)$. From the axiom A4) we have that $\exists \mathrm{x}'$ state such that $(\alpha(\alpha.ltime), a, \mathrm{x}')$ discrete transition $\Rightarrow$
$\Rightarrow \beta' \lceil (E_A, \emptyset) \frown a \lceil (E_A, \emptyset) \frown \tau \lceil (E_A, \emptyset) = \beta \lceil (E_A, \emptyset) \in traces_A$.
$a \notin I_{A_2} \Rightarrow \beta \lceil (E_{A_2}, \emptyset) = \beta' \lceil (E_{A_2}, \emptyset) \in traces_{A_2}$
From hypothesis we have $\beta \in traces_{K\|E\|A_1} \Rightarrow \beta \lceil (E_{K\|A_1}, \emptyset) \in traces_{K\|A_1}$

$$\left.\begin{array}{r} \Rightarrow \beta \in traces_{K\|A\|A_1\|A_2} \\ K \parallel A \parallel A_1 \parallel A_2 \preceq G \parallel A \parallel G_1 \parallel G_2 \parallel A_1 \parallel A_2 \end{array}\right\} \Rightarrow$$

$\Rightarrow \beta \in traces_{G\|A\|G_1\|G_2\|A_1\|A_2} \Rightarrow \beta \lceil (E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$ (7)

(6) and (7) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

ii.2) $a \in I_{A_2}$. Let α an execution of A_2 such that $trace(\alpha) = \beta' \lceil (E_{A_2}, \emptyset)$. From the axiom A4) we have that $\exists \mathrm{x}'$ state such that $(\alpha(\alpha.ltime), a, \mathrm{x}')$ discrete transition $\Rightarrow$
$\beta' \lceil (E_{A_2}, \emptyset) \frown a \lceil (E_{A_2}, \emptyset) \frown \tau \lceil (E_{A_2}, \emptyset) = \beta \lceil (E_{A_2}, \emptyset) \in traces_{A_2}$.
$a \notin I_A \Rightarrow \beta \lceil (E_A, \emptyset) = \beta' \lceil (E_A, \emptyset) \in traces_A$
From hypothesis we have $\beta \in traces_{K\|E\|A_1} \Rightarrow \beta \lceil (E_{K\|A_1}, \emptyset) \in traces_{K\|A_1}$

$$\left.\begin{array}{r} \Rightarrow \beta \in traces_{K\|A\|A_1\|A_2} \\ K \parallel A \parallel A_1 \parallel A_2 \preceq G \parallel A \parallel G_1 \parallel G_2 \parallel A_1 \parallel A_2 \end{array}\right\} \Rightarrow$$

$\Rightarrow \beta \in traces_{G\|A\|G_1\|G_2\|A_1\|A_2} \Rightarrow \beta\lceil(E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$ (8)

(6) and (8) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

b) $\beta = \beta' a\tau$ where a is an output action of K and τ a point trajectory.
$\beta' \in traces_{G\|E\|G_1\|G_2\|A_1} \Rightarrow \beta'\lceil(E_G, \emptyset) \in traces_G$, $\beta'\lceil(E_{G_1}, \emptyset) \in traces_{G_1}$ and $\beta'\lceil(E_{G_2}, \emptyset) \in traces_{G_2}$
$I_G \cup I_{G_2} = O_E$. We have two cases:

i) $a \in I_G$. Let α be an execution of G such that $trace(\alpha) = \beta'\lceil(E_G, \emptyset)$.
From axiom A4) we have that $\exists$x′ state with $(\alpha(\alpha.ltime), a, \mathrm{x}')$ discrete transition $\Rightarrow$
$\beta'\lceil(E_G, \emptyset)^\frown a\lceil(E_G, \emptyset)^\frown\tau\lceil(E_G, \emptyset) = \beta\lceil(E_G, \emptyset) \in traces_G$ (9)
$a \notin I_{G_2} \Rightarrow \beta\lceil(E_{G_2}, \emptyset) = \beta'\lceil(E_{G_2}, \emptyset) \in traces_{G_2}$ (10)
$a \notin I_{G_1} \Rightarrow \beta\lceil(E_{G_1}, \emptyset) = \beta'\lceil(E_{G_1}, \emptyset) \in traces_{G_1}$ (11)

(6), (9), (10) and (11) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

ii) $a \in I_{G_2}$. Let α be an execution of G such that $trace(\alpha) = \beta'\lceil(E_{G_2}, \emptyset)$.
From axiom A4) we have that $\exists$x′ state with $(\alpha(\alpha.ltime), a, \mathrm{x}')$ discrete transition $\Rightarrow$
$\beta'\lceil(E_{G_2}, \emptyset)^\frown a\lceil(E_{G_2}, \emptyset)^\frown\tau\lceil(E_{G_2}, \emptyset) = \beta\lceil(E_{G_2}, \emptyset) \in traces_{G_2}$ (12)
$a \notin I_G \Rightarrow \beta\lceil(E_G, \emptyset) = \beta'\lceil(E_G, \emptyset) \in traces_G$ (13)
$a \notin I_{G_1} \Rightarrow \beta\lceil(E_{G_1}, \emptyset) = \beta'\lceil(E_{G_1}, \emptyset) \in traces_{G_1}$ (14)

(6), (12), (13) and (14) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

c) $\beta = \beta' a\tau$ where a is an output action of A_1 and τ is a point trajectory.
$\beta' \in traces_{G\|E\|G_1\|G_2\|A_1} \Rightarrow \beta'\lceil(E_G, \emptyset) \in traces_G$, $\beta'\lceil(E_{G_1}, \emptyset) \in traces_{G_1}$ and $\beta'\lceil(E_{G_2}, \emptyset) \in traces_{G_2}$
$a \notin I_G \Rightarrow \beta\lceil(E_G, \emptyset) = \beta'\lceil(E_G, \emptyset) \in traces_G$ (15)
$a \in I_{G_1} (= O_{A_1})$. Let α an execution of G_1 such that $trace(\alpha) = \beta'\lceil(E_{G_1}, \emptyset)$. From the axiom A4) we have that $\exists$x′ state such that $(\alpha(\alpha.ltime), a, \mathrm{x}')$ discrete transition $\Rightarrow$
$\Rightarrow \beta'\lceil(E_{G_1}, \emptyset)^\frown a\lceil(E_{G_1}, \emptyset)^\frown\tau\lceil(E_{G_1}, \emptyset) = \beta\lceil(E_{G_1}, \emptyset) \in traces_{G_1}$ (16)
$a \notin I_{G_2} \Rightarrow \beta\lceil(E_{G_2}, \emptyset) = \beta'\lceil(E_{G_2}, \emptyset) \in traces_{G_2}$ (17)

(6), (15), (16) and (17) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

d) $\beta = \beta' a\tau$ where a is a visible action of K and τ is a point trajectory.
$\beta \in traces_{K\|E\|A_1} \Rightarrow \beta\lceil(E_K, \emptyset) \in traces_K$
$a \in V_K \Rightarrow a \notin E_A$, $a \notin E_{A_1}$ and $a \notin E_{A_2}$ and from hypothesis $\beta'\lceil(E_{A\|A_1\|A_2}, \emptyset) \in traces_{A\|A_1\|A_2} \Rightarrow \beta\lceil(E_{A\|A_1\|A_2}, \emptyset) \in traces_{A\|A_1\|A_2}$

$$\left.\begin{array}{r} \Rightarrow \beta \in traces_{K\|A\|A_1\|A_2} \\ K \parallel A \parallel A_1 \parallel A_2 \preceq G \parallel A \parallel G_1 \parallel G_2 \parallel A_1 \parallel A_2 \end{array}\right\} \Rightarrow$$

$\Rightarrow \beta \in traces_{G\|A\|G_1\|G_2\|A_1\|A_2} \Rightarrow \beta\lceil(E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$ (18)
(6) and (18) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

e) $\beta = \beta' a\tau$ where a is a visible action of E and τ is a point trajectory.
$\beta' \in traces_{G\|E\|G_1\|G_2} \Rightarrow \beta'\lceil(E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$
Since $a \in V_E \Rightarrow a \notin E_G$, $a \notin E_{G_1}$ and $a \notin E_{G_2} \Rightarrow \beta\lceil(E_{G\|G_1\|G_2}, \emptyset) = \beta'\lceil(E_{G\|G_1\|G_2}, \emptyset) \in traces_{G\|G_1\|G_2}$ (19)
(6) and (19) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$

2) Every non-closed trace of $K \parallel E \parallel A_1$ is also a trace of $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$. Let β be a non-closed trace of $K \parallel E \parallel A_1$. Then β is the limit of a sequence $\beta_1\beta_2\cdots$ of closed

traces of $K \parallel E \parallel A_1$. We have shown that $\forall i$ β_i closed trace of $K \parallel E \parallel A_1$ is also a trace of $G \parallel E \parallel G_1 \parallel G_2 \parallel A_1$.

$$\left.\begin{array}{r} \beta_i \in traces_{G\|E\|G_1\|G_2\|A_1} \Rightarrow \beta_i\lceil(E_G,\emptyset) \in traces_G, \forall i \\ \text{restriction is a continuous operation} \end{array}\right\} \Rightarrow$$

$$\left.\begin{array}{r} \Rightarrow \beta\lceil(E_G,\emptyset) = lim_i \beta_i\lceil(E_G,\emptyset) \\ traces_G \text{ are closed under limits} \end{array}\right\} \Rightarrow \beta\lceil(E_G,\emptyset) \in traces_G(20)$$

Similarly, $\beta\lceil(E_{G_1},\emptyset) \in traces_{G_1}$ (21) and $\beta\lceil(E_{G_2},\emptyset) \in traces_{G_2}$ (22)
(6), (20), (21) and (22) $\Rightarrow \beta \in traces_{G\|E\|G_1\|G_2\|A_1}$ □

Note. This theorem is similar to *Theorem 8.8* from [27]. While we prove that $K \sqsubseteq_E G$, *Theorem 8.8* states that $K \parallel E \preceq G \parallel A$. Even if the latter is a stronger result with respect to state space reduction at verification, it doesn't guarantee the circular reasoning property needed for contract refinement. The proof follows the same reasoning as for *Theorem 8.8*.

The definitions of closure under limits and closure under time-extension for a set of traces are those given in [27]. Closure under limits informally means that any infinite sequence whose finite prefixes are traces of G is also an trace of G (a property satisfied by all automata occurring in practical examples), while closure under time-extension denotes that any trace can be extended with time passage to infinity. By making these hypotheses on G, G can only express safety properties on K and cannot impose stronger constraints on time passage than K.

We derive the *satisfaction* relation from refinement under context.

Definition 13 (Contract satisfaction)**.** *A component* K satisfies *(*implements*) a contract* $C = (A, G)$*, denoted* $K \models C$*, if and only if* $K \sqsubseteq_A G$*.*

One can easily verify that the three components K_1 (Step 1.1), K_2 (Step 1.2) and K_3 (Step 1.3) satisfy their contracts presented in Figure 4. We remark that the empty timed input/output automaton is the weakest component that refinement under context always satisfies.

We have introduced the notions and relations in order to verify that a component is a correct implementation of a contract. The second step of our contract-based verification consists in defining a refinement relation between contracts in order to discard implementations from now on. This is realized with the help of the *dominance*:

Definition 14 (Contract dominance [33])**.** *Let* C *be a contract with the interface* $\mathcal{P}$ *and* $\{C_i\}_{i=1}^n$ *a set of contracts with the interface* $\{\mathcal{P}_i\}_{i=1}^n$ *and* $\mathcal{P} \subseteq \bigcup_{i=1}^n \mathcal{P}_i$*. Then* $\{C_i\}_{i=1}^n$ dominates C *if and only if for any set of components* $\{K_i\}_{i=1}^n$ *such that* $\forall i,\ K_i \models C_i$*, we have* $(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C$*.*

Figure 5 contains a top contract for the subsystem K in the running example that contains three components K_1, K_2 and K_3. This contract guarantees that if an a message followed by a b message are sent to the environment then at least a δ_2 delay will elapse between 2 cycles. We have to prove that $\{C_1, C_2, C_3\}$ dominates C.

Based on theorems 3, 4 and 5, the following theorem which is a variant of *Theorem 2.3.5* from [33] holds:

Theorem 6 (Sufficient condition for dominance)**.** $\{C_i\}_{i=1}^n$ *dominates* C *if,* $\forall i$*,* $traces_{A_i}$*,* $traces_{G_i}$*,* $traces_A$ *and* $trace_G$ *are closed under limits and under time-extension and*

$$\left\{\begin{array}{l} G_1 \parallel ... \parallel G_n \sqsubseteq_A G \\ A \parallel G_1 \parallel ... \parallel G_{i-1} \parallel G_{i+1} \parallel ... \parallel G_n \sqsubseteq_{G_i} A_i,\ \forall i \end{array}\right.$$

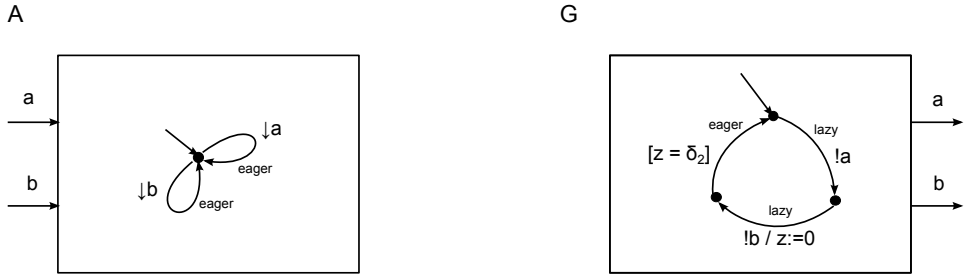

Figure 5: Contract C for the component K which is dominated by the three contracts C_1, C_2 and C_3.

Proof. Let K_i, $i = \overline{1,n}$, a set of components such that:
H1) $K_i \sqsubseteq_{A_i} G_i$
H2) $G_1 \parallel G_2 \parallel \cdots \parallel G_n \sqsubseteq_A G$
H3) $A \parallel G_1 \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i$

We have to prove that $K_1 \parallel K_2 \parallel \cdots \parallel K_n \sqsubseteq_A G$

Proof by *induction* on j where $j = \overline{0,n}$ is the number of guarantees replaced by their corresponding component:

Base step.

$j = 0$. Then the conclusion becomes $G_1 \parallel G_2 \parallel \cdots G_n \sqsubseteq_A G$ which is H2) thus true.

$j = 1$.

$$\left.\begin{array}{r} \text{From H1)} \Rightarrow K_1 \sqsubseteq_{A_1} G_1 \\ \text{From H3)} \Rightarrow A \parallel G_2 \parallel \cdots \parallel G_n \sqsubseteq_{G_1} A_1 \end{array}\right\} \overset{Theorem\ 5}{\Rightarrow}$$

$\Rightarrow K_1 \sqsubseteq_{A \parallel G_2 \parallel \cdots \parallel G_n} G_1$ (1)

$$\left.\begin{array}{r} (1) \overset{Theorem\ 4}{\Rightarrow} K_1 \parallel G_2 \parallel \cdots \parallel G_n \sqsubseteq_A G_1 \parallel G_2 \parallel \cdots \parallel G_n \\ \text{From H2)} \Rightarrow G_1 \parallel \cdots \parallel G_n \sqsubseteq_A G \end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow K_1 \parallel G_2 \parallel \cdots \parallel G_n \sqsubseteq_A G$ (2)

$$\left.\begin{array}{r} (1) \overset{Theorem\ 4}{\Rightarrow} A \parallel K_1 \parallel G_2 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A \parallel \\ \parallel G_1 \parallel G_2 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n, \forall i > 1 \\ \text{From H3)} A \parallel G_1 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i \end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow A \parallel K_1 \parallel G_2 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > 1$ (3)

$j = 2$.

$$\left.\begin{array}{r} \text{From H1)} \Rightarrow K_2 \sqsubseteq_{A_2} G_2 \\ \text{From (3) for } j = 2 \Rightarrow A \parallel K_1 \parallel G_3 \parallel \cdots \parallel G_n \sqsubseteq_{G_2} A_2 \end{array}\right\} \overset{Theorem\ 5}{\Rightarrow}$$

$\Rightarrow K_2 \sqsubseteq_{A \parallel K_1 \parallel G_3 \parallel \cdots \parallel G_n} G_2$ (4)

$$\left.\begin{array}{r} (4) \overset{Theorem\ 4}{\Rightarrow} K_1 \parallel K_2 \parallel G_3 \parallel \cdots \parallel G_n \sqsubseteq_A K_1 \parallel G_2 \parallel \cdots \parallel G_n \\ \text{From (2) } K_1 \parallel G_2 \parallel \cdots \parallel G_n \sqsubseteq_A G \end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow K_1 \parallel K_2 \parallel G_3 \parallel \cdots \parallel G_n \sqsubseteq_A G$ (5)

$$\left.\begin{array}{r}(4) \overset{Theorem\ 4}{\Rightarrow} A \parallel K_1 \parallel K_2 \parallel G_3 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A \parallel \\ \parallel K_1 \parallel G_2 \parallel G_3 \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n, \forall i > 2 \\ \text{From (3)} \Rightarrow A \parallel K_1 \parallel G_2 \parallel \cdots G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > 1\end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow A \parallel K_1 \parallel K_2 \parallel G_3 \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > 2$ (5)

Induction step. Let j be fixed. Then the hypotheses are
$K_1 \parallel K_2 \parallel \cdots \parallel K_j \parallel G_{j+1} \parallel \cdots \parallel G_n \sqsubseteq_A G$ (6)
$A \parallel K_1 \parallel K_2 \parallel \cdots \parallel K_j \parallel G_{j+1} \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > j$ (7)

The conclusion for $j+1$ is:
(8) $K_1 \parallel \cdots \parallel K_j \parallel K_{j+1} \parallel G_{j+2} \parallel G_n \sqsubseteq_A G$ and
(9) $A \parallel K_1 \parallel \cdots \parallel K_{j+1} \parallel G_{j+2} \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > j+1.$

$$\left.\begin{array}{r}\text{From H1)} \Rightarrow K_{j+1} \sqsubseteq_{A_{j+1}} G_{j+1} \\ \text{From (7)} \Rightarrow A \parallel K_1 \parallel K_2 \parallel \cdots \parallel K_j \parallel G_{j+2} \parallel \cdots \parallel G_n \sqsubseteq_{G_{j+1}} A_{j+1}\end{array}\right\} \overset{Theorem\ 5}{\Rightarrow}$$

$\Rightarrow K_{j+1} \sqsubseteq_{A \parallel K_1 \parallel \cdots \parallel K_j \parallel G_{j+2} \parallel \cdots \parallel G_n} G_{j+1}$ (10)

$$\left.\begin{array}{r}(10) \overset{Theorem\ 4}{\Rightarrow} K_1 \parallel \cdots \parallel K_j \parallel K_{j+1} \parallel G_{j+2} \parallel \cdots \parallel G_n \sqsubseteq_A K_1 \parallel \cdots \parallel K_j \parallel \\ \parallel G_{j+1} \parallel G_{j+2} \parallel \cdots \parallel G_n \\ \text{From (6)} K_1 \parallel \cdots \parallel K_j \parallel G_{j+1} \parallel \cdots \parallel G_n \sqsubseteq_A G\end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow K_1 \parallel \cdots \parallel K_j \parallel K_{j+1} \parallel G_{j+2} \parallel G_n \sqsubseteq_A G$, which is (8).

$$\left.\begin{array}{r}(10) \overset{Theorem\ 4}{\Rightarrow} A \parallel K_1 \parallel \cdots \parallel K_{j+1} \parallel G_{j+2} \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A \parallel \\ \parallel K_1 \parallel \cdots \parallel K_j \parallel G_{j+1} \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n, \forall i > j+1 \\ \text{From (7) for } i = j+1 \Rightarrow A \parallel K_1 \parallel \cdots \parallel K_j \parallel G_{j+1} \parallel \cdots \parallel G_{i-1} \parallel \\ \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > j+1\end{array}\right\} \overset{Transitivity}{\Rightarrow}$$

$\Rightarrow A \parallel K_1 \parallel \cdots \parallel K_{j+1} \parallel G_{j+2} \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n \sqsubseteq_{G_i} A_i, \forall i > j+1$, which is (9)

Conclusion. Thus for $j = n$ we have the dominance relation: $K_1 \parallel K_2 \parallel \cdots \parallel K_n \sqsubseteq_A G$ □

Note. This theorem is a particular case of *Theorem 2.3.5* from [33] where a more general composition operator (i.e. *glue*) between components is used and which demands that circular reasoning is sound. The previous proof is similar to that of *Theorem 2.3.5* by adapting it to our notation and taking into consideration the compositional results of theorems 3, 4 and 5.

Verifying dominance consists in checking several refinement under context relations on abstract timed input/output automata that are easier to handle. In our running example we prove dominance between $\{C_1, C_2, C_3\}$ and C by checking the following conditions of Step 2:

(2.1) $G_1 \parallel G_2 \sqsubseteq_A G$,

(2.2) $A \parallel G_1 \sqsubseteq_{G_2} A_2$ and

(2.3) $A \parallel G_2 \sqsubseteq_{G_1} A_1$.

		Mono-lithic	Step 1			Step 2			Step 3
			1.1	1.2	1.3	2.1	2.2	2.3	
max(i) = max(j) = 5	Number of states	41504	364	3372	-[2]	158	148	231	51
	Number of transitions	79249	604	5070	-	239	229	422	65
	Time (sec)	6.86	0.1	0.42	-	0.04	0.04	0.03	0.02
max(i) = max(j) = 10	Number of states	400711	364	10702	-	158	148	231	51
	Number of transitions	827591	604	15925	-	239	229	422	65
	Time (sec)	78.84	0.1	1.21	-	0.04	0.04	0.03	0.02
max(i) = max(j) = 100	Number of states	∞	364	809542	-	158	148	231	51
	Number of transitions	∞	604	1190290	-	239	229	422	65
	Time (sec)	∞	0.1	123.39	-	0.04	0.04	0.03	0.02

Table 1: Verification results for without/with the contract-based methodology.

We have dropped A_3 and G_3, since they are the empty components and do not impact component composition.

The last step (Step 3) in the verification of a system model is to prove that the top contract satisfies the global property, i.e. $A \parallel G \preceq \varphi$, which is true for the running example.

Table 1 presents some quantitative measures (number of transitions and of states explored and time needed for verification in seconds) for each verification step of the running example after bounding the counters i and j and the queue from the additional defined environment to the component (i.e. such a queue has a maximal length of 1)[1]. The first column corresponds to the verification of the property on the whole system without contracts. For steps 1 to 3, verification of refinement is performed using observer simulation, details are given in the following section. It is interesting to note that the longest verification step with contracts is an order of magnitude smaller than the monolithic verification in this case, the explosion being caused by messages exchanged with K_3 which are abstracted away from the contracts.

5 Application to a SysML model: the ATV Solar Wing Generation System case study

The contract-based reasoning method previously described is partially supported by the OMEGA-IFx Toolset [9] for SysML models. The details of the SysML language extended with contracts are left aside for space reasons and can be found in [23]. In the following we present a case study extracted from the industrial-grade system model of the Automated Transfer Vehicle (ATV) and we show how contracts can be used for property verification. The ATV, developed by Astrium Space Transportation for the European Space Agency, is a spacecraft put into orbit by the European heavy launcher Ariane-5 with the aim of supplying the International Space Station. This case

[1] We were using a IA64 computing server with 16GB of memory.

[2] Since an empty timed input/output automaton is the weakest component always satisfied, this verification step has not been performed.

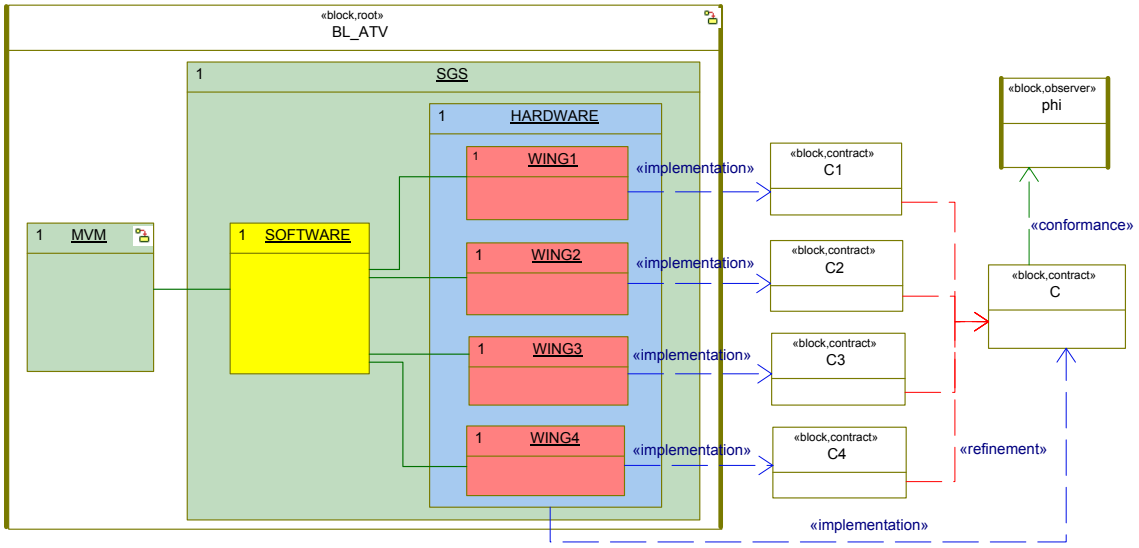

Figure 6: Architecture of the SGS system including contracts (simplified view).

study consists of the Solar Wing Generation System (SGS) [22] responsible for the deployment and management of the solar wings of the vehicle. The SysML model used in the following, provided by Astrium Space Transportation, was obtained by reverse engineering the actual SGS system for the purpose of this study.

The ATV system model illustrated in Figure 6 summarizes the three main components involved in the case study and the bidirectional communications between them: the mission and vehicle management (*MVM*) part that initiates the two functionalities of the SGS (wing deployment and rotation), the *SOFTWARE* part of the *SGS* that based on commands received from the *MVM* executes the corresponding procedures and the *HARDWARE* part that consists of the four wings. We focus here on the wing deployment mode on which we want to verify the following property φ:

Property φ: After 10 minutes from system start-up, all four wings are deployed.

The system explicitly models the redundancy of the hardware equipments which aims to ensure fault tolerance. There are 56 possible failures (14 per wing) grouped in 3 classes depending on their target (thermal knives, hold-down and release system and solar array driving group). The following hypothesis is made: throughout any execution of the system, at most one hardware fault may occur (1-fault tolerance). We are interested in verifying φ by taking into consideration this hypothesis. But applying model-checking directly on the system leads to combinatorial explosion and the verification does not finish. To give an idea about the complexity of the model at runtime, the system contains 96 objects that communicate through 661 port instances and 504 connectors. We proceed in the following to prove property φ with contract-based reasoning.

Since the property φ is expressed with respect to the behavior of the four wings that are contained in the *HARDWARE* block, with regard to the methodology of Figure 1, the subsystem S can be identified in our case study with *HARDWARE* and the components K_i are represented by *WINGi*, $i = \overline{1,4}$. The environment of the subsystem is given by the parts with which it communicates: bidirectional communication is directly established between *SOFTWARE* and *HARDWARE*, while *SOFTWARE* depends on the behavior of *MVM*. So, the environment E of Figure 1 is represented here by the composition of *MVM* and *SOFTWARE*.

The first step of the methodology consists in defining a contract $C_i = (A_i, G_i)$ for each *WINGi*, and next proving that *WINGi* satisfies C_i, $i = \overline{1,4}$. This step checks the validity of the dependency

relations illustrated in blue in Figure 6 between the wings and their corresponding contracts. In order to model a contract, first we need to identify the environment of the component to which the contract is associated and to build an abstraction from the point of view of the component. Thus, for *WINGi* the environment is given by the environment of the subsystem *HARDWARE* and all *WINGj* with $j \neq i$. We propose the following abstraction *WAj* for *WINGj*: the wing has a not deployed status for at most 400 seconds and a deployed status after 130 seconds, while all other received requests are consumed. The assumption A_i is then given by the parallel composition of *MVM*, *SOFTWARE* and *WAj* with $j \neq i$. This abstraction of the environment is sufficient to drastically reduce the state space of the verification model, since the exponential explosion in the original model is mainly due to the parallelism of the hardware pieces which are abstracted to the three leaf parts *WAj*. We want to guarantee that even if *WINGi* exhibits a failure it ends up being deployed after 400 seconds.

Contract C_i = (A_i, G_i) where

- A_i = *MVM* $\parallel$ *SOFTWARE* $\parallel$ ($\parallel_{j \neq i}$*WAj*).
- G_i: the wing answers to requests about its status with *not deployed* from startup up to 400 seconds or with *deployed* after 130 seconds and ignores all other requests. Between 130 and 400 seconds it can answer either, non-deterministically.

Since A_i is partially given by the concrete environment (*MVM* $\parallel$ *SOFTWARE*) and C_i has to define a closed system, we have to manually model the behavior of G_i for all received requests. This constraint imposes to add as consuming transitions in every state all requests corresponding to wing deployment process. Furthermore, one can remark that this guarantee is stronger than the projection of the property φ on *WINGi*. The abstraction *WAj* can also be subject to one failure since this case was not excluded from its behavior. Then the fault tolerance property that we verify via contracts is stronger than the initial hypothesis: we guarantee that the system is 4-fault tolerant if faults occur in separate wings.

The second step consists in defining a global contract $C = (A, G)$ for *HARDWARE* and proving that the contract is dominated by $\{C_1, C_2, C_3, C_4\}$. In Figure 6 it is represented by the red dependency relations between contracts. We use as assumption A the concrete environment of *HARDWARE*. The guarantee G is the composition of the four G_i within one component. All A_i, G_i, A and G as defined satisfy the conditions needed to apply Theorem 6.

Contract C = (A, G) where

- A = *MVM* $\parallel$ *SOFTWARE*
- G : for each wing status interrogation it answers as *not deployed* for at most 400 seconds and as *deployed* after at least 130 seconds, while all other requests are ignored.

The last step consists in verifying that the composition $A \parallel G$ conforms to φ, illustrated by the green dependency in Figure 6. Verifying that the environment satisfies the "mirror" contract is trivial since the assumption A is the environment itself.

The proofs of all three steps have been automatically verified within the OMEGA-IFx Toolset which translates SysML models in communicating timed automata [9]. Since trace inclusion is undecidable, we use a stronger relation named *observer simulation* whose satisfaction implies the satisfaction of trace inclusion. So, the components that play the role of guarantees are transformed into observers and we verify that *error* states (from which no more actions can be performed) are not reached during model-checking. The observer verification algorithm is implemented in IFx Toolset. For the time being, the transformation of guarantees into observers is done manually, but an automated tool is under development.

Table 2: Average verification time for each contract C_i per induced failure group.

	Average Verification Time (s)			
Type of induced Failure	Wing 1	Wing 2	Wing 3	Wing 4
Thermal Knife	13993	6869	18842	11412
Hold-down and release system	12672	6516	16578	9980
Solar array driving group	11527	5432	13548	6807

For each step of the verification methodology we have manually modeled the contracts: assumptions as blocks that we had to connect via ports with the other components and guarantees as observers. The first step gave 4 possible configurations with one concrete wing and 3 abstract ones that were each verified with respect to all 14 possible failures. The average time in seconds needed for the verification of the satisfaction relation for each contract with respect to each class of failures is presented in Table 2. Even thought the system model looks symmetrical, however some hardware pieces not represented here do not have a symmetrical behavior and due to their interconnections with the wings the state space of system's abstraction for $WING1$ and $WING3$ is greater than the one of $WING2$ and $WING4$. For the second step, only one model is created on which we verified all 5 proof obligations given by Theorem 6: the automatic validation of the global guarantee G and the automatic validation of assumptions A_i. Modeling the assumptions A_i as observers shows the symmetry of the *MVM* and *SOFTWARE* behavior. This means that only one verification is in fact sufficient for proving all 4 relations, verification that was realized in 9 seconds. The verification of the guarantee G needed 1 second. Finally, the same model was used for verifying φ that took 1 second.

6 Related work

Contract-based specification and reasoning for communicating components has been subject to intensive research recently. As mentioned in the beginning, our contract theory for TIOA is an instance of the meta-theory of [33], which has previously been applied for a number of other specification formalisms: Labelled Transition Systems (with priorities) [33], Modal Transition Systems [34], BIP framework [3, 33] which is seen as a generalization for Input/Output Automata and Heterogeneous Rich Components [6]. To the best of our knowledge, this is the first application to Timed Input/Output Automata.

An alternative meta-theory is described in [5], similar in many points with [33]. The main differences concern (1) the stronger pre-requirements that are placed upon the specification theory on which it is instantiated and (2) the method for specifying and reasoning with contracts. Regarding item (1), [5] requires the specification theory to support, in addition to a parallel composition operator and a refinement operator, also a *logical conjunction* operator and a *quotient* operator. Such a specification theory was previously defined by the same authors for a variant of TIOA in [17, 18, 16] and implemented in the ECDAR tool [19]. However, several aspects of this theory make it unsuitable for representing the semantics of timed components described in SysML or UML. The synchronization between an input of one component and an output of another component becomes an output of the composite, which equates to considering outputs as broadcasts and which is not consistent with the UML/SysML semantics. Moreover, the formalism forbids non-determinism due to the timed game semantics [8] and does not handle *silent* transitions, which is problematic for representing the semantics of complex components performing internal computation steps.

Regarding the definition of contracts, the framework of [5] does not support signature refinement, i.e. the ability of a contract to concentrate only on *some* of the inputs/outputs of the component while abstracting away the others, which is explicitly handled in our framework. Regarding

the method for reasoning with contracts, it relies on a (derived) *contract composition* operator, which can be used to compute the "strongest" contract $C_1 \boxtimes C_2$ satisfied by the composition of two components that satisfy C_1 and respectively C_2. Since the contracts of [5] require an environment to refine the contract assumption regardless of how the component behaves, the method does not support circular arguments. Moreover, the contract composition operator is partial (i.e. it can be *undefined* for certain pairs of contracts) since it is based on the quotient operator which is itself partial. This case is not explicitly discussed in [5] and it is not clear how a user can identify the cause and mitigate it. In contrast to this, in the method proposed by [33], one of the proof obligations which constitute the sufficient condition for dominance of Step 2 may also be falsified, but when this happens it can be used to identify the cause and correct the contracts or the components. On a more general note, we consider that the reasoning methodology, explicitly described in [33], is an important asset in the application of the meta-theory to concrete domains.

A partial solution for the signature refinement of contracts is provided by the meta-theory of [4]: a port is defined on a subset of the component's signature and contracts are defined on ports. However contracts are not to be used individually at design: a specification consists of a set of port contracts and a component such that the union of port contracts signatures is equal to the signature of the component. Thus, the set of port contracts are not dissociated between them and from the component, all elements being carried at each design step. Even more, refinement signature between port contracts is not possible, contrary to the meta-theory of [33]. As for the previous meta-theory, the method for reasoning with contracts relies on a specification composition operator that implies the composition of component. Component composition is partial and, depending on the level of abstraction, may lead to combinatorial explosion. This meta-theory does not support circular arguments and does not provide a mechanism for formalizing and proving requirement satisfaction directly on contracts.

A related line of research concerns specification theories, some of which are based on earlier work around interface theories [20, 21, 28, 12]. The aim is to provide a specification algebra complete with powerful operators for logical composition of specifications, for synthesis of missing components (quotient), the final goal being to provide substitutability results allowing for compositional verification. Some of the specification theories developed in this context are based on variants of TIOA close to the one that we are using, like in the case of [14, 16]. As mentioned above, a contract-based framework uses a specification theory as starting point; the notion of contract added on top serves essentially two purposes which are not tackled by specification theories: to separate assumptions from guarantees (in specification or interface theories these two are merged) and to support (possibly circular) assume-guarantee reasoning.

Assume-guarantee reasoning is another long-standing line of research, although classical approaches deal with logical specifications [15, 25, 1]. The more recent approach of [13] deals with specifications in the form of sets of I/O traces, and also deals with the problem of signature refinement which allows a contract to concentrate on a subset of the component's actions (although for untimed specifications). The reasoning approach is similar with the one proposed in [5], in particular with regard to contract composition. As such, the remarks made above with respect to the reasoning methodology remain valid for [13].

In addition, contracts in UML/SysML have until now been explored for the specification of composition compatibility of components via interfaces [35] and for the verification of pre/post conditions of operations as presented by [26]. Recent work covers the use of pre/post condition contracts for modeling transformation of models [11] and for modeling the execution semantics of UML elements [10]. To the best of our knowledge, our work is the first on using assume/guarantee behavioral contracts for the verification of UML/SysML model requirements.

7 Conclusions

We have presented a contract framework for Timed Input/Output Automata and results which allow contract-based reasoning for verifying timed safety properties of systems of TIOA components. We have illustrated the method on a case study extracted from an industrial-scale system model and we have showed how contract-based reasoning can alleviate the problem of combinatorial explosion for the verification of large systems.

The present work is a step further towards introducing contracts in SysML and providing a full solution to that problem. In [23] we defined a suitable syntax for contracts in SysML and a set of well-formedness rules that system models must satisfy for reasoning with contracts. For the moment, some steps of the method applied on SysML remain manual like modeling individual systems for each contract satisfaction relation or for each dominance obligation proof. Future work includes: (1) formalizing the semantic mapping between SysML components and contracts and their TIOA counterparts and (2) providing means for automatic verification by automated generation of proof obligations.

References

[1] Martín Abadi and Gordon D. Plotkin. A Logical View of Composition. *Theor. Comput. Sci.*, 114(1):3–30, 1993.

[2] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[3] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3 –12, sept. 2006.

[4] Sebastian Bauer, Rolf Hennicker, and Axel Legay. Component Interfaces with Contracts on Ports. In *Formal Aspects of Component Software*, volume 7684 of *LNCS*, pages 19–35. Springer, 2013.

[5] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from Specifications to Contracts in Component-Based Design. In Juan Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58. Springer Berlin Heidelberg, 2012.

[6] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, pages 142–147. IEEE, 2008.

[7] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Information and Computation*, 163, 2000.

[8] Timothy Bourke, Alexandre David, KimG. Larsen, Axel Legay, Didier Lime, Ulrik Nyman, and Andrzej Wasowski. New results on timed specifications. In Till Mossakowski and Hans-Jorg Kreowski, editors, *Recent Trends in Algebraic Development Techniques*, volume 7137 of *Lecture Notes in Computer Science*, pages 175–192. Springer Berlin Heidelberg, 2012.

[9] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The IF Toolset. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 131–132. Springer Berlin / Heidelberg, 2004.

[10] Eric Cariou, Cyril Ballagny, Alexandre Feugas, and Franck Barbier. Contracts for model execution verification. In *ECMFA'11*, pages 3–18, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] Eric Cariou, Nicolas Belloir, Franck Barbier, and Nidal Djemam. Ocl contracts for the verification of model transformations. *ECEASST*, 24, 2009.

[12] Taolue Chen, Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. A Compositional Specification Theory for Component Behaviours. In Helmut Seidl, editor, *Programming Languages and Systems*, volume 7211 of *Lecture Notes in Computer Science*, pages 148–168. Springer Berlin Heidelberg, 2012.

[13] Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. Assume-Guarantee Reasoning for Safe Component Behaviours. In Corina S. Pasareanu and Gwen Salaun, editors, *Formal Aspects of Component Software*, volume 7684 of *Lecture Notes in Computer Science*, pages 92–109. Springer Berlin Heidelberg, 2013.

[14] Chris Chilton, Marta Kwiatkowska, and Xu Wang. Revisiting Timed Specification Theories: A Linear-Time Perspective. In Marcin Jurdzinski and Dejan Nickovic, editors, *Formal Modeling and Analysis of Timed Systems*, volume 7595 of *Lecture Notes in Computer Science*, pages 75–90. Springer Berlin Heidelberg, 2012.

[15] Edmund M. Clarke, David E. Long, and Kenneth L. McMillan. Compositional Model Checking. In *LICS*, pages 353–362. IEEE Computer Society, 1989.

[16] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Mikael H. Møller, Ulrik Nyman, Anders P. Ravn, Arne Skou, and Andrzej Wasowski. Compositional verification of real-time systems using ECDAR. *STTT*, 14(6):703–720, 2012.

[17] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Methodologies for Specification of Real-Time Systems Using Timed I/O Automata. In Frank S. Boer, Marcello M. Bonsangue, Stefan Hallerstede, and Michael Leuschel, editors, *Formal Methods for Components and Objects*, volume 6286 of *Lecture Notes in Computer Science*, pages 290–310. Springer Berlin Heidelberg, 2010.

[18] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. 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*, HSCC '10, pages 91–100, New York, NY, USA, 2010. ACM.

[19] Alexandre David, Kim.G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *Automated Technology for Verification and Analysis*, volume 6252 of *Lecture Notes in Computer Science*, pages 365–370. Springer Berlin Heidelberg, 2010.

[20] Luca de Alfaro and Thomas Henzinger. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM*, pages 109–120. Press, 2001.

[21] Luca de Alfaro, Thomas Henzinger, and Mariëlle Stoelinga. Timed Interfaces. In Alberto Sangiovanni-Vincentelli and Joseph Sifakis, editors, *Embedded Software*, volume 2491 of *Lecture Notes in Computer Science*, pages 108–122. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-45828-X_9.

[22] Iulia Dragomir, Iulian Ober, and David Lesens. A case study in formal system engineering with SysML. In *Engineering of Complex Computer Systems (ICECCS), 2012 17th IEEE International Conference on*, july 2012.

[23] Iulia Dragomir, Iulian Ober, and Christian Percebois. Integrating verifiable Assume/Guarantee contracts in UML/SysML. Technical report, IRIT, 2013. Available at `http://www.irit.fr/~Iulian.Ober/docs/TR-Syntax.pdf`.

[24] E. Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

[25] Orna Grumberg and David E. Long. Model Checking and Modular Verification. In *CONCUR*, volume 527 of *LNCS*, pages 250–265. Springer, 1991.

[26] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, 1969.

[27] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata - Second Edition*. Morgan & Claypool Publishers, 2010.

[28] Kim Larsen, Ulrik Nyman, and Andrzej Wasowski. Interface Input/Output Automata. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin / Heidelberg, 2006. 10.1007/11813040_7.

[29] RTCA Inc. Software Considerations in Airborne Systems and Equipment Certification. Document RTCA/DO-178C, 2011.

[30] OMG. Object Management Group – Systems Modeling Language (SysML), v1.1. Available at http://www.omg.org/spec/SysML/1.1/, 2008.

[31] D.L. Parnas and D.M. Weiss. Active Design Reviews: Principles and Practices. In Meir M. Lehman, Horst Hünke, and Barry W. Boehm, editors, *Proceedings, 8th International Conference on Software Engineering (ICSE), London, UK, August 28-30, 1985*. IEEE Computer Society, 1985.

[32] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[33] Sophie Quinton. *Design, vérification et implémentation de systèmes à composants*. PhD thesis, Université de Grenoble, 2011.

[34] Sophie Quinton and Susanne Graf. Contract-Based Verification of Hierarchical Systems of Components. *Software Engineering and Formal Methods, IEEE International Conference on*, pages 377–381, 2008.

[35] Torben Weis, Christian Becker, Kurt Geihs, and Noël Plouzeau. A uml meta-model for contract aware components. In *UML'01*, pages 442–456. Springer-Verlag, 2001.

A Proofs

For proofs on the sets of actions we use the following theorems from the algebra of sets:

1. $A \setminus A = \emptyset$
2. $A \setminus \emptyset = A$
3. $\emptyset \setminus A = \emptyset$
4. $B \setminus (A \setminus B) = B$
5. $(A \setminus B) \setminus A = \emptyset$
6. $A \setminus (A \setminus B) = A \cap B$
7. $(A \setminus B) \setminus B = A \setminus B$
8. $(A \setminus B) \setminus C = A \setminus (B \cup C)$
9. $A \setminus (B \setminus C) = (A \setminus B) \cup (A \cap C)$
10. $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$
11. $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$
12. $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
13. $(A \cap B) \setminus C = A \cap (B \setminus C) = (A \setminus C) \cap B$
14. $(A \setminus B) \cup C = (A \cup C) \setminus (B \setminus C)$
15. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
16. $A \cup (A \cap B) = A$
17. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
18. $A \cap (A \cup B) = A$

where A, B and C are sets.

Theorem 1. The parallel composition operator is commutative and associative.

Proof. In the following we give the proofs (computations) for the input, output and visible sets of actions.

$I_{(K_1 \| K_2) \| K_3} = (I_{K_1 \| K_2} \setminus O_{K_3}) \cup (I_{K_3} \setminus O_{K_1 \| K_2}) = (((I_{K_1} \setminus O_{K_2}) \cup (I_{K_2} \setminus O_{K_1})) \setminus O_{K_3}) \cup (I_{K_3} \setminus ((O_{K_1} \setminus I_{K_2}) \cup (O_{K_2} \setminus I_{K_1}) = ((I_{K_1} \setminus O_{K_2}) \setminus O_{K_3}) \cup ((I_{K_2} \setminus O_{K_1}) \setminus O_{K_3}) \cup ((I_{K_3} \setminus (O_{K_1} \setminus I_{K_2})) \cap (I_{K_3} \setminus (O_{K_2} \setminus I_{K_1}))) = (I_{K_1} \setminus (O_{K_2} \cup O_{K_3})) \cup (I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup (((I_{K_3} \cap I_{K_2}) \cup (I_{K_3} \setminus O_{K_1})) \cap ((I_{K_3} \cap I_{K_1}) \cup (I_{K_3} \setminus O_{K_2}))) = (I_{K_1} \setminus (O_{K_2} \cup O_{K_3})) \cup (I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup ((I_{K_3} \setminus O_{K_1}) \cap (I_{K_3} \setminus O_{K_2})) = (I_{K_1} \setminus (O_{K_2} \cup O_{K_3})) \cup (I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup (I_{K_3} \setminus (O_{K_1} \cup O_{K_2}))$

$I_{K_1 \| (K_2 \| K_3)} = (I_{K_1} \setminus O_{K_2 \| K_3}) \cup (I_{K_2 \| K_3} \setminus O_{K_1}) = (I_{K_1} \setminus ((O_{K_2} \setminus I_{K_3}) \cup (O_{K_3} \setminus I_{K_2}))) \cup (((I_{K_2} \setminus O_{K_3}) \cup (I_{K_3} \setminus O_{K_2})) \setminus O_{K_1}) = ((I_{K_1} \setminus (O_{K_2} \setminus I_{K_3})) \cap (I_{K_1} \setminus (O_{K_3} \setminus I_{K_2}))) \cup (((I_{K_2} \setminus O_{K_3}) \setminus O_{K_1}) \cup ((I_{K_3} \setminus O_{K_2}) \setminus O_{K_1})) = (((I_{K_1} \cap I_{K_3}) \cup (I_{K_1} \setminus O_{K_2})) \cap ((I_{K_1} \cap I_{K_2}) \cup (I_{K_1} \setminus O_{K_3}))) \cup ((I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup ((I_{K_3} \setminus (O_{K_1} \cup O_{K_2}))) = ((I_{K_1} \setminus O_{K_2}) \cap (I_{K_1} \setminus O_{K_3})) \cup ((I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup ((I_{K_3} \setminus (O_{K_1} \cup O_{K_2}))) = (I_{K_1} \setminus (O_{K_2} \cup O_{K_3})) \cup (I_{K_2} \setminus (O_{K_1} \cup O_{K_3})) \cup (I_{K_3} \setminus (O_{K_1} \cup O_{K_2}))$

$O_{(K_1 \| K_2) \| K_3} = (O_{K_1 \| K_2} \setminus I_{K_3}) \cup (O_{K_3} \setminus I_{K_1 \| K_2}) = (((O_{K_1} \setminus I_{K_2}) \cup (O_{K_2} \setminus I_{K_1})) \setminus I_{K_3}) \cup (O_{K_3} \setminus ((I_{K_1} \setminus O_{K_2}) \cup (I_{K_2} \setminus O_{K_1}) = ((O_{K_1} \setminus I_{K_2}) \setminus I_{K_3}) \cup ((O_{K_2} \setminus I_{K_1}) \setminus I_{K_3}) \cup ((O_{K_3} \setminus (I_{K_1} \setminus$

$O_{K_2})) \cap (O_{K_3} \setminus (I_{K_2} \setminus O_{K_1}))) = (O_{K_1} \setminus (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup (((O_{K_3} \cap O_{K_2}) \cup (O_{K_3} \setminus I_{K_1})) \cap ((O_{K_3} \cap O_{K_1}) \cup (O_{K_3} \setminus I_{K_2}))) = (O_{K_1} \setminus (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup ((O_{K_3} \setminus I_{K_1}) \cap (O_{K_3} \setminus I_{K_2})) = (O_{K_1} \setminus (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \setminus (I_{K_1} \cup I_{K_2}))$

$O_{K_1 \| (K_2 \| K_3)} = (O_{K_1} \setminus I_{K_2 \| K_3}) \cup (O_{K_2 \| K_3} \setminus I_{K_1}) = (O_{K_1} \setminus ((I_{K_2} \setminus O_{K_3}) \cup (I_{K_3} \setminus O_{K_2}))) \cup (((O_{K_2} \setminus I_{K_3}) \cup (O_{K_3} \setminus I_{K_2})) \setminus I_{K_1}) = ((IO_{K_1} \setminus (I_{K_2} \setminus O_{K_3})) \cap (O_{K_1} \setminus (I_{K_3} \setminus O_{K_2}))) \cup (((O_{K_2} \setminus I_{K_3}) \setminus I_{K_1}) \cup ((O_{K_3} \setminus I_{K_2}) \setminus I_{K_1})) = (((O_{K_1} \cap O_{K_3}) \cup (O_{K_1} \setminus I_{K_2})) \cap ((O_{K_1} \cap O_{K_2}) \cup (O_{K_1} \setminus I_{K_3}))) \cup ((O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup ((O_{K_3} \setminus (I_{K_1} \cup I_{K_2}))) = ((O_{K_1} \setminus I_{K_2}) \cap (O_{K_1} \setminus I_{K_3})) \cup ((O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup ((O_{K_3} \setminus (I_{K_1} \cup I_{K_2}))) = (O_{K_1} \setminus (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \setminus (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \setminus (I_{K_1} \cup I_{K_2}))$

$V_{(K_1 \| K_2) \| K_3} = V_{K_1 \| K_2} \cup V_{K_3} \cup (O_{K_1 \| K_2} \cap I_{K_3}) \cup (I_{K_1 \| K_2} \cap O_{K_3}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap I_{K_2}) \cup (O_{K_2} \cap I_{K_1}) \cup (((O_{K_1} \setminus I_{K_2}) \cup (O_{K_2} \setminus I_{K_1})) \cap I_{K_3}) \cup (((I_{K_1} \setminus O_{K_2}) \cup (I_{K_2} \setminus O_{K_1})) \cap O_{K_3}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap I_{K_2}) \cup (O_{K_2} \cap I_{K_1}) \cup ((O_{K_1} \setminus I_{K_2}) \cap I_{K_3}) \cup ((O_{K_2} \setminus I_{K_1}) \cap I_{K_3}) \cup ((I_{K_1} \setminus O_{K_2}) \cap O_{K_3}) \cup ((I_{K_2} \setminus O_{K_1}) \cap O_{K_3}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap I_{K_2}) \cup (O_{K_2} \cap I_{K_1}) \cup (O_{K_1} \cap (I_{K_3} \setminus I_{K_2})) \cup (O_{K_2} \cap (I_{K_3} \setminus I_{K_1})) \cup (I_{K_1} \cap (O_{K_3} \setminus O_{K_2})) \cup (I_{K_2} \cap (O_{K_3} \setminus O_{K_1})) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap I_{K_2}) \cup (O_{K_2} \cap I_{K_1}) \cup (O_{K_1} \cap I_{K_3}) \cup (O_{K_2} \cap I_{K_3}) \cup (I_{K_1} \cap O_{K_3}) \cup (I_{K_2} \cap O_{K_3}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \cap (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \cap (I_{K_1} \cup I_{K_2}))$

$V_{K_1 \| (K_2 \| K_3)} = V_{K_1} \cup V_{K_2 \| K_3} \cup ((O_{K_1} \cap I_{K_2 \| K_3}) \cup (O_{K_2 \| K_3} \cap I_{K_1})) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap O_{K_3}) \cup (((I_{K_2} \setminus O_{K_3}) \cup (I_{K_3} \setminus O_{K_2})) \cap O_{K_1}) \cup (((O_{K_2} \setminus I_{K_3}) \cup (O_{K_3} \setminus I_{K_2})) \cap I_{K_1}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap O_{K_3}) \cup ((I_{K_2} \setminus O_{K_3}) \cap O_{K_1}) \cup ((I_{K_3} \setminus O_{K_2}) \cap O_{K_1}) \cup ((O_{K_2} \setminus I_{K_3}) \cap I_{K_1}) \cup ((O_{K_3} \setminus I_{K_2}) \cap I_{K_1}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap O_{K_3}) \cup (I_{K_2} \cap (O_{K_1} \setminus O_{K_3})) \cup (I_{K_3} \cap (O_{K_1} \setminus O_{K_2})) \cup (O_{K_2} \cap (I_{K_1} \setminus I_{K_3})) \cup (O_{K_3} \cap (I_{K_1} \setminus I_{K_2})) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap O_{K_3}) \cup (I_{K_2} \cap O_{K_1}) \cup (I_{K_3} \cap O_{K_1}) \cup (O_{K_2} \cap I_{K_1}) \cup (O_{K_3} \cap I_{K_1}) = V_{K_1} \cup V_{K_2} \cup V_{K_3} \cup (O_{K_1} \cap (I_{K_2} \cup I_{K_3})) \cup (O_{K_2} \cap (I_{K_1} \cup I_{K_3})) \cup (O_{K_3} \cap (I_{K_1} \cup I_{K_2}))$

□

Theorem 3. Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Proof. $E'' = E_1 \parallel E_3$:

$I_{E''} = ((O_{K_1} \cap O_{K_2}) \setminus I_E) \cup (V_{K_1} \cap O_{K_2}) = ((O_{K_1} \cap O_{K_2}) \cup (V_{K_1} \cap O_{K_2})) \setminus (I_E \setminus (V_{K_1} \cap O_{K_2})) = ((V_{K_1} \cup O_{K_1}) \cap O_{K_2}) \setminus I_E = O_{K_2} \setminus I_E$

$O_{E''} = ((I_{K_1} \cap I_{K_2}) \setminus O_E) \cup (V_{K_1} \cap I_{K_2}) = ((I_{K_1} \cap I_{K_2}) \cup (V_{K_1} \cap I_{K_2})) \setminus (O_E \setminus (V_{K_1} \cap I_{K_2})) = ((V_{K_1} \cup I_{K_1}) \cap I_{K_2}) \setminus O_E = I_{K_2} \setminus O_E$

$V_{E''} = \emptyset$

$K' = K'_2 \parallel K'_3$:

$I_{K'} = I_{K'_2 \| K'_3} = (I_{K'_2} \setminus O_{K'_3}) \cup (I_{K'_3} \setminus O_{K'_2})$

$= (((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})) \setminus ((O_{K_2} \setminus O_{K_3}) \cup (V_{K_2} \cap I_{K_3}))) \cup (((I_{K_2} \setminus I_{K_3}) \cup (V_{K_2} \cap O_{K_3})) \setminus ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})))$

$= ((((I_{K_1} \setminus I_{K_2}) \setminus (O_{K_2}) \setminus O_{K_3})) \cap ((I_{K_1} \setminus I_{K_2}) \setminus (V_{K_2} \cap I_{K_3}))) \cup (((V_{K_1} \cap O_{K_2}) \setminus (O_{K_2} \setminus O_{K_3})) \cap ((V_{K_1} \cap O_{K_2}) \setminus (V_{K_2} \cap I_{K_3})))) \cup ((((I_{K_2} \setminus I_{K_3}) \setminus (O_{K_1} \setminus O_{K_2})) \cap ((I_{K_2} \setminus I_{K_3}) \setminus (V_{K_1} \cap I_{K_2}))) \cup (((V_{K_2} \cap O_{K_3}) \setminus (O_{K_1} \setminus O_{K_2})) \cap ((V_{K_2} \cap O_{K_3}) \setminus (V_{K_1} \cap I_{K_2}))))$

$= (((((I_{K_1} \setminus I_{K_2}) \setminus O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{K_3})) \cap (((I_{K_1} \setminus I_{K_2}) \setminus V_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \setminus I_{K_3}))) \cup ((V_{K_1} \cap (O_{K_2} \setminus (O_{K_2} \setminus O_{K_3}))) \cap (((V_{K_1} \cap O_{K_2}) \setminus V_{K_2}) \cup ((V_{K_1} \cap O_{K_2}) \setminus I_{K_3})))) \cup (((((I_{K_2} \setminus I_{K_3} \setminus O_{K_1}) \cup ((I_{K_2} \setminus I_{K_3}) \cap O_{K_2})) \cap (((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1}) \cup ((I_{K_2} \setminus I_{K_3}) \setminus I_{K_2}))) \cup ((O_{K_3} \cap (V_{K_2} \setminus (O_{K_1} \setminus O_{K_2}))) \cap (((V_{K_1} \cap O_{K_3}) \setminus V_{K_1}) \cup ((V_{K_2} \cap O_{K_3}) \setminus I_{K_2}))))$

$= ((((I_{K_1} \setminus (I_{K_2} \cup O_{K_2})) \cup ((I_{K_1} \cap O_{K_3}) \setminus I_{K_2})) \cap (((I_{K_1} \setminus (I_{K_2} \cup V_{K_2})) \cup ((I_{K_1} \setminus I_{K_2}) \setminus I_{K_3}))) \cup ((V_{K_1} \cap O_{K_2} \cap O_{K_3}) \cap ((V_{K_1} \cap (O_{K_2} \setminus V_{K_2})) \cup (V_{K_1} \cap (O_{K_2} \setminus I_{K_3}))))) \cup ((((I_{K_2} \setminus (I_{K_3} \cup O_{K_1})) \cup ((I_{K_2} \cap O_{K_2}) \setminus I_{K_3})) \cap ((I_{K_2} \setminus (I_{K_3} \cup V_{K_1})) \cup \emptyset)) \cup ((O_{K_3} \cap ((V_{K_2} \setminus O_{K_1}) \cup (V_{K_2} \cap O_{K_2}))) \cap ((V_{K_1} \setminus V_{K_1}) \cap O_{K_3}) \cup ((V_{K_2} \setminus I_{K_2}) \cap O_{K_3}))))$

$= (((((I_{K_1} \setminus I_{K_2}) \cap (I_{K_1} \setminus O_{K_2})) \cup \setminus) \cap (((I_{K_1} \setminus V_{K_2}) \setminus I_{K_2}) \cup (I_{K_1} \setminus (I_{K_2} \cup I_{K_3})))) \cup ((V_{K_1} \cap$

$O_{K_2} \cap O_{K_3}) \cap ((V_{K_1} \cap O_{K_2}) \cup (V_{K_1} \cap O_{K_2})))) \cup (((((I_{K_2} \setminus I_{K_3}) \cap (I_{K_2} \setminus O_{K_1})) \cup \emptyset) \cap ((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1})) \cup ((O_{K_3} \cap (V_{K_2} \setminus O_{K_1})) \cap (\emptyset \cup (V_{K_2} \cap O_{K_3}))))$
$= (((((I_{K_1} \setminus I_{K_2}) \cap I_{K_1}) \cup \emptyset) \cap ((I_{K_1} \setminus I_{K_2}) \cup (I_{K_1} \setminus (I_{K_2} \cup I_{K_3})))) \cup ((V_{K_1} \cap O_{K_2} \cap O_{K_3}) \cap (V_{K_1} \cap O_{K_2}))) \cup ((((I_{K_2} \setminus I_{K_3}) \cap I_{K_2}) \cap ((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1})) \cup ((V_{K_2} \cap (O_{K_3} \setminus O_{K_1})) \cap (V_{K_2} \cap O_{K_3})))$
$= (((I_{K_1} \setminus I_{K_3}) \cap ((I_{K_1} \setminus I_{K_2}) \cup (I_{K_1} \setminus (I_{K_2} \cup I_{K_3})))) \cup ((V_{K_1} \cap O_{K_2} \cap O_{K_3}) \cap (V_{K_1} \cap O_{K_2}))) \cup (((I_{K_2} \setminus I_{K_3}) \cap ((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1})) \cup ((V_{K_2} \cap (O_{K_3} \setminus O_{K_1})) \cap (V_{K_2} \cap O_{K_3})))$
$= ((((I_{K_1} \setminus I_{K_2}) \cap (I_{K_1} \setminus I_{K_2})) \cup ((I_{K_1} \setminus I_{K_2}) \cap ((I_{K_1} \setminus I_{K_2}) \setminus I_{K_3}))) \cup (V_{K_1} \cap O_{K_2} \cap O_{K_3})) \cup (((I_{K_2} \setminus I_{K_3}) \cap ((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1})) \cup (((V_{K_2} \cap O_{K_3}) \setminus O_{K_1}) \cap (V_{K_2} \cap O_{K_3})))$
$= (((I_{K_1} \setminus I_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \setminus I_{K_3})) \cup (V_{K_1} \cap O_{K_2} \cap O_{K_3})) \cup (((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1}) \cup ((V_{K_2} \cap O_{K_3}) \setminus O_{K_1}))$
$= (((I_{K_1} \setminus I_{K_2}) \setminus (I_{K_3} \setminus (I_{K_1} \setminus I_{K_2}))) \cup (V_{K_1} \cap O_{K_2} \cap O_{K_3})) \cup (((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1}) \cup (V_{K_2} \cap (O_{K_3} \setminus O_{K_1})))$
$= ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2} \cap O_{K_3})) \cup (((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1}) \cup (V_{K_2} \cap (O_{K_3} \setminus O_{K_1})))$
$= (I_{K_1} \setminus I_{K_2}) \cup ((I_{K_2} \setminus I_{K_3}) \setminus V_{K_1}) \cup (V_{K_1} \cap O_{K_2} \cap O_{K_3}) \cup (V_{K_2} \cap (O_{K_3} \setminus O_{K_1}))$
$= ((I_{K_1} \cup ((I_{K_2} \setminus V_{K_1}) \setminus I_{K_3})) \setminus (I_{K_2} \setminus ((I_{K_2} \setminus V_{K_1}) \setminus I_{K_3}))) \cup (((V_{K_1} \cap O_{K_3}) \cap O_{K_2}) \cup ((V_{K_2} \cap O_{K_3}) \setminus O_{K_1}))$
$= (((I_{K_1} \cup (I_{K_2} \setminus V_{K_1})) \setminus (I_{K_3} \setminus I_{K_1})) \setminus ((I_{K_2} \cap I_{K_3}) \cup (I_{K_2} \setminus (I_{K_2} \setminus V_{K_1})))) \cup (((V_{K_1} \cap O_{K_3}) \cup ((V_{K_1} \cap O_{K_3}) \cap O_{K_2})) \setminus (O_{K_1} \setminus ((V_{K_1} \cap O_{K_3}) \cap O_{K_2})))$
$= ((((I_{K_1} \cup I_{K_2}) \setminus (V_{K_1} \setminus I_{K_1})) \setminus (I_{K_3} \setminus I_{K_1})) \setminus ((I_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap V_{K_1}))) \cup ((((V_{K_1} \cap O_{K_3}) \cup (V_{K_2} \cap O_{K_3})) \cap ((V_{K_2} \cap O_{K_3}) \cup O_{K_2}))) \setminus ((O_{K_1} \setminus (V_{K_1} \cap O_{K_3}) \cup (O_{K_1} \setminus O_{K_2})))$
$= ((((I_{K_1} \cup I_{K_2}) \setminus V_{K_1}) \setminus (I_{K_3} \setminus I_{K_1}) \setminus (I_{K_2} \cap (I_{K_3} \cup V_{K_1}))) \cup ((((V_{K_1} \cup V_{K_2}) \cap O_{K_3}) \cap ((V_{K_2} \cap O_{K_3}) \cup O_{K_2}))) \setminus (O_{K_1} \cup (O_{K_1} \setminus O_{K_2})))$
$= ((I_{K_1} \setminus (I_{K_3} \setminus I_{K_1})) \setminus (I_{K_2} \cap (I_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap O_{K_3}) \cap ((V_{K_2} \cap O_{K_3}) \cup O_{K_2})) \setminus (O_{K_1} \setminus (O_{K_2} \setminus O_{K_1})))$
$= (I_{K_1} \setminus (I_{K_2} \cap (I_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap O_{K_3} \cap V_{K_2}) \cup (V_{K_1} \cap O_{K_3} \cap O_{K_2})) \setminus O_{K_1})$
$= ((I_{K_1} \setminus I_{K_2}) \cup (I_{K_1} \setminus (I_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap (V_{K_2} \cap O_{K_3})) \cup (V_{K_1} \cap (O_{K_2} \cap O_{K_3})) \setminus O_{K_1})$
$= ((I_{K_1} \setminus I_{K_2}) \cup ((I_{K_1} \setminus I_{K_3}) \cap (I_{K_1} \setminus V_{K_1}))) \cup ((V_{K_1} \cap ((V_{K_2} \cap O_{K_3}) \cup (O_{K_2} \cap O_{K_3}))) \setminus O_{K_1})$
$= ((I_{K_1} \setminus I_{K_2}) \cup ((I_{K_1} \setminus I_{K_3}) \cap I_{K_1})) \cup ((V_{K_1} \cap (O_{K_3} \cap (O_{K_2} \cup V_{K_2}))) \setminus O_{K_1})$
$= ((I_{K_1} \setminus I_{K_2}) \cup (I_{K_1} \setminus I_{K_3})) \cup ((V_{K_1} \cap O_{K_3}) \setminus O_{K_1})$
$= (I_{K_1} \setminus I_{K_3}) \cup ((V_{K_1} \setminus O_{K_1}) \cap O_{K_3})$
$= (I_{K_1} \setminus I_{K_3}) \cup (V_{K_1} \cap O_{K_3})$

$O_{K'} = O_{K'_2 \| K'_3} = (O_{K'_2} \setminus I_{K'_3}) \cup (O_{K'_3} \setminus I_{K'_2})$
$= (((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})) \setminus ((I_{K_2} \setminus I_{K_3}) \cup (V_{K_2} \cap O_{K_3}))) \cup (((O_{K_2} \setminus O_{K_3}) \cup (V_{K_2} \cap I_{K_3})) \setminus ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})))$
$= ((((O_{K_1} \setminus O_{K_2}) \setminus (I_{K_2}) \setminus I_{K_3})) \cap ((O_{K_1} \setminus O_{K_2}) \setminus (V_{K_2} \cap O_{K_3}))) \cup (((V_{K_1} \cap I_{K_2}) \setminus (I_{K_2} \setminus I_{K_3})) \cap ((V_{K_1} \cap I_{K_2}) \setminus (V_{K_2} \cap O_{K_3})))) \cup ((((O_{K_2} \setminus O_{K_3}) \setminus (I_{K_1} \setminus I_{K_2})) \cap ((O_{K_2} \setminus O_{K_3}) \setminus (V_{K_1} \cap O_{K_2}))) \cup (((V_{K_2} \cap I_{K_3}) \setminus (I_{K_1} \setminus I_{K_2})) \cap ((V_{K_2} \cap I_{K_3}) \setminus (V_{K_1} \cap O_{K_2}))))$
$= (((((O_{K_1} \setminus O_{K_2}) \setminus I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{K_3})) \cap (((O_{K_1} \setminus O_{K_2}) \setminus V_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \setminus O_{K_3}))) \cup ((V_{K_1} \cap (I_{K_2} \setminus (I_{K_2} \setminus I_{K_3}))) \cap (((V_{K_1} \cap I_{K_2}) \setminus V_{K_2}) \cup ((V_{K_1} \cap I_{K_2}) \setminus O_{K_3})))) \cup ((((O_{K_2} \setminus O_{K_3} \setminus I_{K_1}) \cup ((O_{K_2} \setminus O_{K_3}) \cap I_{K_2})) \cap (((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1}) \cup ((O_{K_2} \setminus O_{K_3}) \setminus O_{K_2}))) \cup ((I_{K_3} \cap (V_{K_2} \setminus (I_{K_1} \setminus I_{K_2}))) \cap (((V_{K_1} \cap I_{K_3}) \setminus V_{K_1}) \cup ((V_{K_2} \cap I_{K_3}) \setminus O_{K_2}))))$
$= ((((O_{K_1} \setminus (O_{K_2} \cup I_{K_2})) \cup ((O_{K_1} \cap I_{K_3}) \setminus O_{K_2})) \cap ((O_{K_1} \setminus (O_{K_2} \cup V_{K_2})) \cup ((O_{K_1} \setminus O_{K_2}) \setminus O_{K_3}))) \cup ((V_{K_1} \cap I_{K_2} \cap I_{K_3}) \cap ((V_{K_1} \cap (I_{K_2} \setminus V_{K_2})) \cup (V_{K_1} \cap (I_{K_2} \setminus O_{K_3}))))) \cup ((((O_{K_2} \setminus (O_{K_3} \cup I_{K_1})) \cup ((O_{K_2} \cap I_{K_2}) \setminus O_{K_3})) \cap ((O_{K_2} \setminus (O_{K_3} \cup V_{K_1})) \cup \emptyset)) \cup ((I_{K_3} \cap ((V_{K_2} \setminus I_{K_1}) \cup (V_{K_2} \cap I_{K_2}))) \cap ((V_{K_1} \setminus V_{K_1}) \cap I_{K_3}) \cup ((V_{K_2} \setminus O_{K_2}) \cap I_{K_3}))))$
$= ((((((O_{K_1} \setminus O_{K_2}) \cap (O_{K_1} \setminus I_{K_2})) \cup \setminus) \cap (((O_{K_1} \setminus V_{K_2}) \setminus O_{K_2}) \cup (O_{K_1} \setminus (O_{K_2} \cup O_{K_3})))) \cup ((V_{K_1} \cap I_{K_2} \cap I_{K_3}) \cap ((V_{K_1} \cap I_{K_2}) \cup (V_{K_1} \cap I_{K_2})))) \cup (((((O_{K_2} \setminus O_{K_3}) \cap (O_{K_2} \setminus I_{K_1})) \cup \emptyset) \cap ((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1})) \cup ((I_{K_3} \cap (V_{K_2} \setminus I_{K_1})) \cap (\emptyset \cup (V_{K_2} \cap I_{K_3}))))$
$= (((((O_{K_1} \setminus O_{K_2}) \cap O_{K_1}) \cup \emptyset) \cap ((O_{K_1} \setminus O_{K_2}) \cup (O_{K_1} \setminus (O_{K_2} \cup O_{K_3})))) \cup ((V_{K_1} \cap I_{K_2} \cap I_{K_3}) \cap$

$(V_{K_1} \cap I_{K_2})))) \cup ((((O_{K_2} \setminus O_{K_3}) \cap O_{K_2}) \cap ((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1})) \cup ((V_{K_2} \cap (I_{K_3} \setminus I_{K_1})) \cap (V_{K_2} \cap I_{K_3})))$
$= (((O_{K_1} \setminus O_{K_3}) \cap ((O_{K_1} \setminus O_{K_2}) \cup (O_{K_1} \setminus (O_{K_2} \cup O_{K_3})))) \cup ((V_{K_1} \cap I_{K_2} \cap I_{K_3}) \cap (V_{K_1} \cap I_{K_2}))) \cup (((O_{K_2} \setminus O_{K_3}) \cap ((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1})) \cup ((V_{K_2} \cap (I_{K_3} \setminus I_{K_1})) \cap (V_{K_2} \cap I_{K_3})))$
$= ((((O_{K_1} \setminus O_{K_2}) \cap (O_{K_1} \setminus O_{K_2})) \cup ((O_{K_1} \setminus O_{K_2}) \cap ((O_{K_1} \setminus O_{K_2}) \setminus O_{K_3}))) \cup (V_{K_1} \cap I_{K_2} \cap I_{K_3})) \cup (((O_{K_2} \setminus O_{K_3}) \cap ((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1})) \cup (((V_{K_2} \cap I_{K_3}) \setminus I_{K_1}) \cap (V_{K_2} \cap I_{K_3})))$
$= (((O_{K_1} \setminus O_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \setminus O_{K_3})) \cup (V_{K_1} \cap I_{K_2} \cap I_{K_3})) \cup (((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1}) \cup ((V_{K_2} \cap I_{K_3}) \setminus I_{K_1}))$
$= (((O_{K_1} \setminus O_{K_2}) \setminus (O_{K_3} \setminus (O_{K_1} \setminus O_{K_2}))) \cup (V_{K_1} \cap I_{K_2} \cap I_{K_3})) \cup (((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1}) \cup (V_{K_2} \cap (I_{K_3} \setminus I_{K_1})))$
$= ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2} \cap I_{K_3})) \cup (((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1}) \cup (V_{K_2} \cap (I_{K_3} \setminus I_{K_1})))$
$= (O_{K_1} \setminus O_{K_2}) \cup ((O_{K_2} \setminus O_{K_3}) \setminus V_{K_1}) \cup (V_{K_1} \cap I_{K_2} \cap I_{K_3}) \cup (V_{K_2} \cap (I_{K_3} \setminus I_{K_1}))$
$= ((O_{K_1} \cup ((O_{K_2} \setminus V_{K_1}) \setminus O_{K_3})) \setminus (O_{K_2} \setminus ((O_{K_2} \setminus V_{K_1}) \setminus O_{K_3}))) \cup (((V_{K_1} \cap I_{K_3}) \cap I_{K_2}) \cup ((V_{K_2} \cap I_{K_3}) \setminus I_{K_1}))$
$= (((O_{K_1} \cup (O_{K_2} \setminus V_{K_1})) \setminus (O_{K_3} \setminus O_{K_1})) \setminus ((O_{K_2} \cap O_{K_3}) \cup (O_{K_2} \setminus (O_{K_2} \setminus V_{K_1})))) \cup (((V_{K_1} \cap I_{K_3}) \cup ((V_{K_1} \cap I_{K_3}) \cap I_{K_2})) \setminus (I_{K_1} \setminus ((V_{K_1} \cap I_{K_3}) \cap I_{K_2})))$
$= ((((O_{K_1} \cup O_{K_2}) \setminus (V_{K_1} \setminus O_{K_1})) \setminus (O_{K_3} \setminus O_{K_1})) \setminus ((O_{K_2} \cap O_{K_3}) \cup (O_{K_2} \cap V_{K_1}))) \cup ((((V_{K_1} \cap I_{K_3}) \cup (V_{K_2} \cap I_{K_3})) \cap ((V_{K_2} \cap I_{K_3}) \cup I_{K_2}))) \setminus ((I_{K_1} \setminus (V_{K_1} \cap I_{K_3}) \cup (I_{K_1} \setminus I_{K_2})))$
$= ((((O_{K_1} \cup O_{K_2}) \setminus V_{K_1}) \setminus (O_{K_3} \setminus O_{K_1}) \setminus (O_{K_2} \cap (O_{K_3} \cup V_{K_1}))) \cup ((((V_{K_1} \cup V_{K_2}) \cap I_{K_3}) \cap ((V_{K_2} \cap I_{K_3}) \cup I_{K_2}))) \setminus (I_{K_1} \cup (I_{K_1} \setminus I_{K_2})))$
$= ((O_{K_1} \setminus (O_{K_3} \setminus O_{K_1})) \setminus (O_{K_2} \cap (O_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap I_{K_3}) \cap ((V_{K_2} \cap I_{K_3}) \cup I_{K_2})) \setminus (I_{K_1} \setminus (I_{K_2} \setminus I_{K_1})))$
$= (O_{K_1} \setminus (O_{K_2} \cap (O_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap I_{K_3} \cap V_{K_2}) \cup (V_{K_1} \cap I_{K_3} \cap I_{K_2})) \setminus I_{K_1})$
$= ((O_{K_1} \setminus O_{K_2}) \cup (O_{K_1} \setminus (O_{K_3} \cup V_{K_1}))) \cup (((V_{K_1} \cap (V_{K_2} \cap I_{K_3})) \cup (V_{K_1} \cap (I_{K_2} \cap I_{K_3})) \setminus I_{K_1})$
$= ((O_{K_1} \setminus O_{K_2}) \cup ((O_{K_1} \setminus O_{K_3}) \cap (O_{K_1} \setminus V_{K_1}))) \cup ((V_{K_1} \cap ((V_{K_2} \cap I_{K_3}) \cup (I_{K_2} \cap I_{K_3}))) \setminus I_{K_1})$
$= ((O_{K_1} \setminus O_{K_2}) \cup ((O_{K_1} \setminus O_{K_3}) \cap O_{K_1})) \cup ((V_{K_1} \cap (I_{K_3} \cap (I_{K_2} \cup V_{K_2}))) \setminus I_{K_1})$
$= ((O_{K_1} \setminus O_{K_2}) \cup (O_{K_1} \setminus O_{K_3})) \cup ((V_{K_1} \cap I_{K_3}) \setminus I_{K_1})$
$= (O_{K_1} \setminus O_{K_3}) \cup ((V_{K_1} \setminus I_{K_1}) \cap I_{K_3})$
$= (O_{K_1} \setminus O_{K_3}) \cup (V_{K_1} \cap I_{K_3})$

$V_{K'} = V_{K'_2 \| K'_3} = V_{K'_2} \cup V_{K'_3} \cup (O_{K'_2} \cap I_{K'_3}) \cup (I_{K'_2} \cap O_{K'_3})$
$= V_{K'_2} \cup V_{K'_3} \cup (((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})) \cap ((I_{K_2} \setminus I_{K_3}) \cup (V_{K_2} \cap O_{K_3}))) \cup (((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})) \cap ((O_{K_2} \setminus O_{K_3}) \cup (V_{K_2} \cap I_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup (((O_{K_1} \setminus O_{K_2}) \cap (I_{K_2} \setminus I_{K_3})) \cup ((V_{K_1} \cap I_{K_2}) \cap (I_{K_2} \setminus I_{K_3})) \cup ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_2} \cap O_{K_3})) \cup ((V_{K_1} \cap I_{K_2}) \cap (V_{K_2} \cap O_{K_3}))) \cup (((I_{K_1} \setminus I_{K_2}) \cap (O_{K_2} \setminus O_{K_3})) \cup ((V_{K_1} \cap O_{K_2}) \cap (O_{K_2} \setminus O_{K_3})) \cup ((I_{K_1} \setminus I_{K_2}) \cap V_{K_2} \cap I_{K_3})) \cup ((V_{K_1} \cap O_{K_2}) \cap (V_{K_2} \cap I_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup (\emptyset \cup (((V_{K_1} \cap I_{K_2}) \cap I_{K_2}) \setminus I_{K_3}) \cup ((O_{K_1} \cap V_{K_2} \cap O_{K_3}) \setminus O_{K_2}) \cup \emptyset) \cup (\emptyset \cup (((V_{K_1} \cap O_{K_2}) \cap O_{K_2}) \setminus O_{K_3}) \cup ((V_{K_2} \cap I_{K_1} \cap I_{K_3}) \setminus I_{K_2}) \cup \emptyset)$
$= V_{K'_2} \cup V_{K'_3} \cup (((V_{K_1} \cap I_{K_2}) \setminus I_{K_3}) \cup \emptyset) \cup (((V_{K_1} \cap O_{K_2}) \setminus O_{K_3}) \cup \emptyset)$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap (I_{K_2} \setminus I_{K_3})) \cup (V_{K_1} \cap (O_{K_2} \setminus O_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap ((I_{K_2} \setminus I_{K_3}) \cup (O_{K_2} \setminus O_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap ((I_{K_2} \cup (O_{K_2} \setminus O_{K_3})) \setminus (I_{K_3} \setminus (O_{K_2} \setminus O_{K_3}))))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap ((I_{K_2} \cup O_{K_2}) \setminus (O_{K_3}) \setminus I_{K_2})) \setminus I_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap (((I_{K_2} \cup O_{K_2}) \setminus O_{K_3}) \setminus I_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap ((I_{K_2} \cup O_{K_2}) \setminus (I_{K_3} \cup O_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap ((E_{K_2} \setminus V_{K_2}) \setminus (E_{K_3} \setminus V_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap (E_{K_2} \setminus V_{K_2})) \setminus (E_{K_3} \setminus V_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup (((V_{K_1} \cap E_{K_2}) \setminus V_{K_2}) \setminus (E_{K_3} \setminus V_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap E_{K_2}) \setminus (V_{K_2} \cup (E_{K_3} \setminus V_{K_3})))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap E_{K_2}) \setminus ((V_{K_2} \cup E_{K_3}) \setminus (V_{K_3} \setminus V_{K_2})))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap E_{K_2}) \setminus ((V_{K_2} \cup E_{K_3}) \setminus \emptyset))$

$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap E_{K_2}) \cap (V_{K_2} \cup E_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup (((V_{K_1} \cap E_{K_2}) \setminus V_{K_2}) \cap ((V_{K_1} \cap E_{K_2}) \setminus E_{K_3}))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \cap (E_{K_2} \setminus V_{K_2}) \cap ((V_{K_1} \setminus E_{K_3}) \cap E_{K_2}))$
$= V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \cap (V_{K_1} \setminus E_{K_3}) \cap E_{K_2} \cap (E_{K_2} \setminus V_{K_2}))$
$= V_{K'_2} \cup V_{K'_3} \cup ((V_{K_1} \setminus E_{K_3}) \cap (E_{K_2} \setminus V_{K_2}))$
$= (V_{K'_2} \cup V_{K'_3} \cup (V_{K_1} \setminus E_{K_3})) \cap (V_{K'_2} \cup V_{K'_3} \cup (E_{K_2} \setminus V_{K_2}))$
$= ((V_{K_1} \setminus E_{K_2}) \cup (V_{K_2} \setminus E_{K_3}) \cup (V_{K_1} \setminus E_{K_3})) \cap ((V_{K_1} \setminus E_{K_2}) \cup (E_{K_2} \setminus V_{K_2}) \cup (V_{K_2} \setminus E_{K_3}))$
$= ((V_{K_1} \setminus (E_{K_2} \cap E_{K_3})) \cup (V_{K_2} \setminus E_{K_3})) \cap (((V_{K_1} \cup (E_{K_2} \setminus V_{K_2})) \setminus (E_{K_2} \setminus (E_{K_2} \setminus V_{K_2})) \cup (V_{K_2} \setminus E_{K_3}))$
$= ((V_{K_1} \setminus E_{K_3}) \cup (V_{K_2} \setminus E_{K_3})) \cap (((V_{K_1} \cup (E_{K_2} \setminus V_{K_2})) \setminus (E_{K_2} \cap V_{K_2})) \cup (V_{K_2} \setminus E_{K_3}))$
$= ((V_{K_1} \cup V_{K_2}) \setminus E_{K_3}) \cap (((V_{K_1} \cup (E_{K_2} \setminus V_{K_2})) \setminus V_{K_2}) \cup (V_{K_2} \setminus E_{K_3}))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((V_{K_1} \cup (E_{K_2} \setminus V_{K_2}) \cup (V_{K_2} \setminus E_{K_3})) \setminus (V_{K_2} \setminus (V_{K_2} \setminus E_{K_3})))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((V_{K_1} \cup (((E_{K_2} \setminus V_{K_2}) \cup V_{K_2}) \setminus (E_{K_3} \setminus (E_{K_2} \setminus V_{K_2})))) \setminus (V_{K_2} \cap E_{K_3}))$
$= (V_{K_1} \setminus E_{K_3}) \cap (V_{K_1} \cup (E_{K_2} \setminus ((E_{K_3} \setminus E_{K_2}) \cup (E_{K_3} \cap V_{K_2})))) \setminus (V_{K_2} \cap E_{K_3}))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((V_{K_1} \cup (E_{K_2} \setminus (\emptyset \cup (V_{K_2} \cap E_{K_3}))) \setminus (V_{K_2} \cap E_{K_3}))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((V_{K_1} \cup (E_{K_2} \setminus (E_{K_3} \cap V_{K_2}))) \setminus (E_{K_3} \cap V_{K_2}))$
$= (V_{K_1} \setminus E_{K_3}) \cap (((E_{K_2} \cup V_{K_1}) \setminus ((E_{K_3} \cap V_{K_2}) \setminus V_{K_1}))) \setminus (E_{K_3} \cap V_{K_2}))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((E_{K_2} \cup V_{K_1}) \setminus (((E_{K_3} \cap V_{K_2}) \setminus V_{K_1}) \cup (E_{K_3} \cap V_{K_2})))$
$= (V_{K_1} \setminus E_{K_3}) \cap ((E_{K_2} \cup V_{K_1}) \setminus (E_{K_3} \cap V_{K_2}))$
$= ((V_{K_1} \cup E_{K_2}) \cap (V_{K_1} \setminus E_{K_3})) \setminus (E_{K_3} \setminus V_{K_2}) = ((V_{K_1} \cap (V_{K_1} \cup E_{K_2})) \setminus (E_{K_3} \setminus (V_{K_1} \cup E_{K_2}))) \setminus (E_{K_3} \setminus V_{K_2}) = (V_{K_1} \setminus ((E_{K_3} \setminus V_{K_1}) \cap (E_{K_3} \setminus E_{K_2}))) \setminus (E_{K_3} \setminus V_{K_2}) = V_{K_1} \setminus (E_{K_3} \setminus V_{K_2}) = (V_{K_1} \setminus E_{K_3}) \setminus (V_{K_2} \setminus V_{K_1}) = V_{K_1} \setminus E_{K_3}$ □

Theorem 4. Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$. $K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$.

Proof. $E' = E'' = (\emptyset, \{\phi\}, \phi, (O_{K_1} \setminus (I_{E_1} \cup I_{E_2}), (I_{K_1} \setminus (O_{E_1} \cup O_{E_2})), \emptyset, \emptyset, D_{E'}, 2_0^{[\mathbb{R}^+]})$ from their construction and:

$I_{E'} = O_{K_1} \setminus I_{E_1 \parallel E_2} = O_{K_1} \setminus ((I_{E_1} \setminus O_{E_1}) \cup (I_{E_2} \setminus O_{E_1})) = (O_{K_1} \setminus (I_{E_1} \setminus O_{E_2})) \cap (O_{K_1} \setminus (I_{E_2} \setminus O_{E_1})) = ((O_{K_1} \cap O_{E_2}) \cup (O_{K_1} \setminus I_{E_1})) \cap ((O_{K_1} \cap O_{E_1}) \cup (O_{K_1} \setminus I_{E_2})) = (\emptyset \cup (O_{K_1} \setminus I_{E_1})) \cap (\emptyset \cup (O_{K_1} \setminus I_{E_2})) = (O_{K_1} \setminus I_{E_1}) \cap (O_{K_1} \setminus I_{E_2}) = O_{K_1} \setminus (I_{E_1} \cup I_{E_2})$

$I_{E''} = O_{K_1 \parallel E_1} \setminus I_{E_2} = ((O_{K_1} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_1})) \setminus I_{E_2} = ((O_{K_1} \setminus I_{E_1}) \setminus I_{E_2}) \cup ((O_{E_1} \setminus I_{K_1}) \setminus I_{E_2}) = (O_{K_1} \setminus (I_{E_1} \cup I_{E_2})) \cup (O_{E_1} \setminus (I_{K_1} \cup I_{E_2})) = (O_{K_1} \setminus (I_{E_1} \cup I_{E_2})) \cup \emptyset = O_{K_1} \setminus (I_{E_1} \cup I_{E_2})$

$O_{E'} = I_{K_1} \setminus O_{E_1 \parallel E_2} = I_{K_1} \setminus ((O_{E_1} \setminus I_{E_2}) \cup (O_{E_2} \setminus I_{E_1})) = (I_{K_1} \setminus (O_{E_1} \setminus I_{E_2})) \cap (I_{K_1} \setminus (O_{E_2} \setminus I_{E_1})) = ((I_{K_1} \cap I_{E_2}) \cup (I_{K_1} \setminus O_{E_1})) \cap ((I_{K_1} \cap I_{E_1}) \cup (I_{K_1} \setminus O_{E_2})) = (\emptyset \cup (I_{K_1} \setminus O_{E_1})) \cap (\emptyset \cup (I_{K_1} \setminus O_{E_2})) = (I_{K_1} \setminus O_{E_1}) \cap (I_{K_1} \setminus O_{E_2}) = I_{K_1} \setminus (O_{E_1} \cup O_{E_2})$

$O_{E''} = I_{K_1 \parallel E_1} \setminus O_{E_2} = ((I_{K_1} \setminus O_{E_1}) \setminus O_{E_2}) \cup ((I_{E_1} \setminus O_{K_1}) \setminus O_{E_2}) = (I_{K_1} \setminus (O_{E_1} \cup O_{E_2})) \cup (I_{E_1} \setminus (O_{K_1} \cup O_{E_2})) = (I_{K_1} \setminus (O_{E_1} \cup O_{E_2})) \cup \emptyset = I_{K_1} \setminus (O_{E_1} \cup O_{E_2})$

$V_{E'} = \emptyset = V_{E''}$

$K' = K'' = (\emptyset, \{\phi\}, \phi, ((I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})), ((O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_1})), V_{K_1} \setminus E_{K_2}, \emptyset, D_{K'}, 2_0^{[\mathbb{R}^+]})$ from the construction and:

$I_{K'} = (I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})$

$I_{K''} = (I_{K_1 \parallel E_1} \setminus I_{K_2 \parallel E_1}) \cup (V_{K_1 \parallel E_1} \cap O_{K_2 \parallel E_1})$
$= (((I_{K_1} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_1})) \setminus ((I_{K_2} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_2}))) \cup ((V_{K_1} \cup V_{E_1} \cup (O_{K_1} \cap I_{E_1}) \cup (I_{K_1} \cap O_{E_1})) \cap O_{K_2 \parallel E_1})$
$= (((I_{K_1} \setminus O_{E_1}) \setminus (I_{K_2} \setminus O_{E_1})) \cap ((I_{K_1} \setminus O_{E_1}) \setminus (I_{E_1} \setminus O_{K_2}))) \cup (((I_{E_1} \setminus O_{K_1}) \setminus (I_{K_2} \setminus O_{E_1})) \cap ((I_{E_1} \setminus O_{K_1}) \setminus (I_{E_1} \setminus O_{K_2}))) \cup (V_{K_1} \cap O_{K_2 \parallel E_1}) \cup (V_{E_1} \cap O_{K_2 \parallel E_1}) \cup ((O_{K_1} \cap I_{E_1}) \cap O_{K_2 \parallel E_1}) \cup ((I_{K_1} \cap O_{E_1}) \cap O_{K_2 \parallel E_1}$
$= ((I_{K_1} \setminus (O_{E_1} \cup (I_{K_2} \setminus O_{E_1}))) \cap (I_{K_1} \setminus (O_{E_1} \cup (I_{E_1} \setminus O_{K_2})))) \cup ((I_{E_1} \setminus (O_{K_2} \cup (I_{K_2} \setminus O_{E_1}))) \cap$

$(I_{E_1} \setminus (O_{K_1} \cup (I_{E_1} \setminus O_{K_2})))) \cup (V_{K_1} \cap ((O_{K_2} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_2}))) \cup (V_{E_1} \cap ((O_{K_2} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_2}))) \cup ((O_{K_1} \cap I_{E_1} \cap ((O_{K_2} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_2}))) \cup ((O_{E_1} \cap I_{K_1}) \cap ((O_{K_2} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_2})))$
$= ((I_{K_1} \setminus ((O_{E_1} \cup I_{K_2}) \setminus (O_{E_1} \setminus O_{E_1}))) \cap (I_{K_1} \setminus ((O_{E_1} \cup I_{E_1}) \setminus (O_{K_2} \setminus O_{E_1})))) \cup ((I_{E_1} \setminus ((O_{K_2} \cup I_{K_2}) \setminus (O_{E_1} \setminus O_{K_2}))) \cap (I_{E_1} \setminus ((O_{K_1} \cup I_{E_1}) \setminus (O_{K_2} \setminus O_{K_1})))) \cup (V_{K_1} \cap (O_{K_2} \setminus I_{E_1})) \cup (V_{K_1} \cap (O_{E_1} \setminus I_{K_2})) \cup \emptyset \cup ((O_{K_1} \cap I_{E_1}) \cap (O_{K_2} \setminus I_{E_1})) \cup ((O_{K_1} \cap I_{E_1}) \cap (O_{E_1} \setminus I_{K_2})) \cup ((O_{E_1} \cap I_{K_1}) \cap (O_{K_2} \setminus I_{E_1})) \cup ((O_{E_1} \cap I_{K_1}) \cap (O_{E_1} \setminus I_{K_1}))$
$= ((I_{K_1} \setminus (O_{E_1} \cup I_{K_2})) \cap (I_{K_1} \setminus ((O_{E_1} \cup I_{E_1}) \setminus O_{K_2}))) \cup ((I_{E_1} \setminus ((O_{K_2} \cup I_{K_2}) \setminus O_{E_1})) \cap (I_{E_1} \setminus ((O_{K_1} \cup I_{E_1}) \setminus (O_{K_2} \setminus O_{K_1})))) \cup (V_{K_1} \cap O_{K_2}) \cup \emptyset \cup ((O_{K_1} \cap I_{E_1} \cap O_{K_2}) \setminus I_{E_1}) \cup ((O_{K_1} \cap I_{E_1} \cap O_{E_1}) \setminus I_{K_2}) \cup ((O_{E_1} \cap I_{K_1} \cap O_{K_2}) \setminus I_{K_2}) \cup ((O_{E_1} \cap I_{K_1} \cap O_{E_1}) \setminus I_{K_2}$
$= ((I_{K_1} \setminus (O_{E_1} \cup I_{K_2})) \cap ((I_{K_1} \cap O_{K_2}) \cup (I_{K_1} \setminus (O_{E_1} \cup I_{E_1})))) \cup (((I_{E_1} \cap O_{E_1}) \cup (I_{E_1} \setminus (O_{K_2} \cup I_{K_2}))) \cap ((I_{E_1} \cap ((O_{K_2} \setminus O_{K_1})) \cup (I_{E_1} \setminus (O_{K_1} \cup I_{E_1})))) \cup (V_{K_1} \cap O_{K_2}) \cup \emptyset \cup \emptyset \cup \emptyset \cup ((O_{E_1} \cap I_{K_1}) \setminus I_{K_2})$
$= ((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cap (\emptyset \cup ((I_{K_1} \setminus O_{E_1}) \cap (I_{K_1} \setminus I_{E_1})))) \cup (((\emptyset \cup ((I_{E_1} \setminus O_{K_2}) \cap (I_{E_1} \setminus I_{K_2}))) \cap ((I_{E_1} \cap (O_{K_2} \setminus O_{K_1})) \cup ((I_{E_1} \setminus O_{K_1}) \cap (I_{E_1} \setminus I_{E_1})))) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= (((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cap ((I_{K_1} \setminus O_{E_1}) \cap I_{K_1})) \cup (((I_{E_1} \setminus O_{K_2}) \cap I_{E_1}) \cap ((I_{E_1} \cap (O_{K_2} \setminus O_{K_1})) \cup ((I_{E_1} \setminus O_{K_1}) \cap \emptyset))) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= (((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cap (I_{K_1} \setminus O_{E_1})) \cup ((I_{E_1} \setminus O_{K_2}) \cap (I_{E_1} \cap (O_{K_2} \setminus O_{K_1}))) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= ((((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cap I_{K_1}) \setminus O_{E_1}) \cup ((I_{E_1} \cap (O_{K_2} \setminus O_{K_1}) \cap I_{E_1}) \setminus O_{K_2}) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= ((((I_{K_1} \setminus I_{K_2}) \cap I_{K_1}) \setminus O_{E_1}) \setminus O_{E_1}) \cup ((I_{E_1} \cap (O_{K_2} \setminus O_{K_1})) \setminus O_{K_2}) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= ((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cup (I_{E_1} \cap ((O_{K_2} \setminus O_{K_1}) \setminus O_{K_2}))) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= ((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cup (I_{E_1} \cap \emptyset) \cup (V_{K_1} \cap O_{K_2}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1})$
$= ((I_{K_1} \setminus I_{K_2}) \setminus O_{E_1}) \cup ((I_{K_1} \setminus I_{K_2}) \cap O_{E_1}) \cup (V_{K_1} \cap O_{K_2}) = (I_{K_1} \setminus I_{K_2}) \cup (V_{K_1} \cap O_{K_2})$

$O_{K'} = (O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_1})$

$O_{K''} = (O_{K_1 \| E_1} \setminus O_{K_2 \| E_1}) \cup (V_{K_1 \| E_1} \cap I_{K_2 \| E_1}$
$= (((O_{K_1} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_1})) \setminus ((O_{K_2} \setminus I_{E_1}) \cup (O_{E_1} \setminus I_{K_2}))) \cup ((V_{K_1} \cup V_{E_1} \cup (I_{K_1} \cap O_{E_1}) \cup (O_{K_1} \cap I_{E_1})) \cap I_{K_2 \| E_1})$
$= (((O_{K_1} \setminus I_{E_1}) \setminus (O_{K_2} \setminus I_{E_1})) \cap ((O_{K_1} \setminus I_{E_1}) \setminus (O_{E_1} \setminus I_{K_2}))) \cup (((O_{E_1} \setminus I_{K_1}) \setminus (O_{K_2} \setminus I_{E_1})) \cap ((O_{E_1} \setminus I_{K_1}) \setminus (O_{E_1} \setminus I_{K_2}))) \cup (V_{K_1} \cap I_{K_2 \| E_1}) \cup (V_{E_1} \cap I_{K_2 \| E_1}) \cup ((I_{K_1} \cap O_{E_1}) \cap I_{K_2 \| E_1}) \cup ((O_{K_1} \cap I_{E_1}) \cap I_{K_2 \| E_1}$
$= ((O_{K_1} \setminus (I_{E_1} \cup (O_{K_2} \setminus I_{E_1}))) \cap (O_{K_1} \setminus (I_{E_1} \cup (O_{E_1} \setminus I_{K_2})))) \cup ((O_{E_1} \setminus (I_{K_2} \cup (O_{K_2} \setminus I_{E_1}))) \cap (O_{E_1} \setminus (I_{K_1} \cup (O_{E_1} \setminus I_{K_2})))) \cup (V_{K_1} \cap ((I_{K_2} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_2}))) \cup (V_{E_1} \cap ((I_{K_2} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_2}))) \cup ((I_{K_1} \cap O_{E_1} \cap ((I_{K_2} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_2}))) \cup ((I_{E_1} \cap O_{K_1}) \cap ((I_{K_2} \setminus O_{E_1}) \cup (I_{E_1} \setminus O_{K_2})))$
$= ((O_{K_1} \setminus ((I_{E_1} \cup O_{K_2}) \setminus (I_{E_1} \setminus I_{E_1}))) \cap (O_{K_1} \setminus ((I_{E_1} \cup O_{E_1}) \setminus (I_{K_2} \setminus I_{E_1})))) \cup ((O_{E_1} \setminus ((I_{K_2} \cup O_{K_2}) \setminus (I_{E_1} \setminus I_{K_2}))) \cap (O_{E_1} \setminus ((I_{K_1} \cup O_{E_1}) \setminus (I_{K_2} \setminus I_{K_1})))) \cup (V_{K_1} \cap (I_{K_2} \setminus O_{E_1})) \cup (V_{K_1} \cap (I_{E_1} \setminus O_{K_2})) \cup \emptyset \cup ((I_{K_1} \cap O_{E_1}) \cap (I_{K_2} \setminus O_{E_1})) \cup ((I_{K_1} \cap O_{E_1}) \cap (I_{E_1} \setminus O_{K_2})) \cup ((I_{E_1} \cap O_{K_1}) \cap (I_{K_2} \setminus O_{E_1})) \cup ((I_{E_1} \cap O_{K_1}) \cap (I_{E_1} \setminus O_{K_1}))$
$= ((O_{K_1} \setminus (I_{E_1} \cup O_{K_2})) \cap (O_{K_1} \setminus ((I_{E_1} \cup O_{E_1}) \setminus I_{K_2}))) \cup ((O_{E_1} \setminus ((I_{K_2} \cup O_{K_2}) \setminus I_{E_1})) \cap (O_{E_1} \setminus ((I_{K_1} \cup O_{E_1}) \setminus (I_{K_2} \setminus I_{K_1})))) \cup (V_{K_1} \cap I_{K_2}) \cup \emptyset \cup ((I_{K_1} \cap O_{E_1} \cap I_{K_2}) \setminus O_{E_1}) \cup ((I_{K_1} \cap O_{E_1} \cap I_{E_1}) \setminus O_{K_2}) \cup ((I_{E_1} \cap O_{K_1} \cap I_{K_2}) \setminus O_{K_2}) \cup ((I_{E_1} \cap O_{K_1} \cap I_{E_1}) \setminus O_{K_2}$
$= ((O_{K_1} \setminus (I_{E_1} \cup O_{K_2})) \cap ((O_{K_1} \cap I_{K_2}) \cup (O_{K_1} \setminus (I_{E_1} \cup O_{E_1})))) \cup (((O_{E_1} \cap I_{E_1}) \cup (O_{E_1} \setminus (I_{K_2} \cup O_{K_2}))) \cap ((O_{E_1} \cap ((I_{K_2} \setminus I_{K_1})) \cup (O_{E_1} \setminus (I_{K_1} \cup O_{E_1})))) \cup (V_{K_1} \cap I_{K_2}) \cup \emptyset \cup \emptyset \cup \emptyset \cup ((I_{E_1} \cap O_{K_1}) \setminus O_{K_2})$
$= ((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cap (\emptyset \cup ((O_{K_1} \setminus I_{E_1}) \cap (O_{K_1} \setminus O_{E_1})))) \cup ((\emptyset \cup ((O_{E_1} \setminus I_{K_2}) \cap (O_{E_1} \setminus O_{K_2}))) \cap ((O_{E_1} \cap (I_{K_2} \setminus I_{K_1})) \cup ((O_{E_1} \setminus I_{K_1}) \cap (O_{E_1} \setminus O_{E_1})))) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= (((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cap ((O_{K_1} \setminus I_{E_1}) \cap O_{K_1})) \cup (((O_{E_1} \setminus I_{K_2}) \cap O_{E_1}) \cap ((O_{E_1} \cap (I_{K_2} \setminus I_{K_1})) \cup ((O_{E_1} \setminus I_{K_1}) \cap \emptyset))) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= (((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cap (O_{K_1} \setminus I_{E_1})) \cup ((O_{E_1} \setminus I_{K_2}) \cap (O_{E_1} \cap (I_{K_2} \setminus I_{K_1}))) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$

$= ((((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cap O_{K_1}) \setminus I_{E_1}) \cup ((O_{E_1} \cap (I_{K_2} \setminus I_{K_1}) \cap O_{E_1}) \setminus I_{K_2}) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= ((((O_{K_1} \setminus O_{K_2}) \cap O_{K_1}) \setminus I_{E_1}) \setminus I_{E_1}) \cup ((O_{E_1} \cap (I_{K_2} \setminus I_{K_1})) \setminus I_{K_2}) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= ((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cup (O_{E_1} \cap ((I_{K_2} \setminus I_{K_1}) \setminus I_{K_2}))) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= ((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cup (O_{E_1} \cap \emptyset) \cup (V_{K_1} \cap I_{K_2}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1})$
$= ((O_{K_1} \setminus O_{K_2}) \setminus I_{E_1}) \cup ((O_{K_1} \setminus O_{K_2}) \cap I_{E_1}) \cup (V_{K_1} \cap I_{K_2}) = (O_{K_1} \setminus O_{K_2}) \cup (V_{K_1} \cap I_{K_2})$

$V_{K'} = V_{K_1} \setminus E_{K_2}$

$V_{K''} = V_{K_1 \| E_1} \setminus E_{K_2 \| E_1} = (V_{K_1} \cup V_{E_1} \cup (I_{K_1} \cap O_{E_1}) \cup (I_{E_1} \cap O_{K_1})) \setminus (E_{K_2} \cup E_{E_1}) = (V_{K_1} \setminus (E_{K_2} \cup E_{E_1})) \cup (V_{E_1} \setminus (E_{K_2} \cup E_{E_1})) \cup ((I_{K_1} \cap O_{E_1}) \setminus (E_{K_2} \cup E_{E_1})) \cup ((O_{K_1} \cap I_{E_1}) \setminus (E_{K_2} \cup E_{E_1})) = ((V_{K_1} \setminus E_{K_2}) \setminus E_{E_1}) \cup \emptyset \cup \emptyset \cup \emptyset = V_{K_1} \setminus E_{K_2}$ □

Institut de Recherche en Informatique de Toulouse
CNRS
INPT
UPS
UT1

ABSTRACT

A variety of system design and architecture description languages, such as SysML, UML or AADL, allows the decomposition of complex system designs into communicating timed components. In this paper we consider the contract-based specification of such components. A contract is a pair formed of an assumption, which is an abstraction of the component's environment, and a guarantee, which is an abstraction of the component's behavior given that the environment behaves according to the assumption. Thus, a contract concentrates on a specific aspect of the component's functionality and on a subset of its interface, which makes it relatively simpler to specify. Contracts may be used as an aid for hierarchical decomposition during design or for verification of properties of composites. This paper defines contracts for components formalized as a variant of timed input/output automata, introduces compositional results allowing to reason with contracts and shows how contracts can be used in a high-level modeling language (SysML) for specification and verification, based on an example extracted from a real-life system.

KEYWORDS

component, timed input-output automata, contract, V&V, compositional reasoning, SysML

UPS-IRIT, 118 route de Narbonne, 31062 Toulouse CEDEX 9
+33 (0) 561 55 67 65 info@irit.fr www.irit.fr