Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 168 (2007) 175–190
www.elsevier.com/locate/entcs

Component-Based Specification of Collaborative Objects

Abdessamad Imine[1]

LORIA, INRIA - Lorraine
Campus Scientifique, 54506 Vandœuvre-Lès-Nancy Cedex, France

Abstract

A collaborative object represents a data type (such as a text document or a spreadsheet) designed to be shared by multiple geographically separated users. In order to improve performance and availability of data in such a distributed context, each user has a local copy of the shared objects, upon which he may perform updates. Locally executed updates are then transmitted to the other users. So, the updates are applied in different orders at different copies of the collaborative object. This replication potentially leads, however, to divergent (*i.e.* different) copies. The Operational Transformation (OT) approach provides an interesting solution for copies divergence. Indeed, every collaborative object has an algorithm which transforms the remote update according to local concurrent ones. But this OT algorithm needs to fulfill two conditions in order to ensure the convergence. Proving the correctness of OT algorithms is very complex and error prone without the assistance of a theorem prover. In the present work, we propose a compositional method for specifying complex collaborative objects. The most important feature of our method is that designing an OT algorithm for the composed collaborative object can be done by reusing the OT algorithms of component collaborative objects. By using our method, we can start from correct small collaborative objects which are relatively easy to handle and incrementally combine them to build more complex collaborative objects.

Keywords: CSCW, groupware systems, component-based design, formal methods

1 Introduction

Distributed collaborative systems allow two or more users (sites) to simultaneously manipulate objects (*i.e.* text, image, graphic, etc.) without the need for physical proximity and enable them to synchronously observe each other's changes. In order to achieve an unconstrained group work, the shared objects are *replicated* at the local memory of each participating user. Every operation is executed locally first and then *broadcasted* for execution at other sites. So, the operations are applied in different orders at different *replicas* (or copies) of the object. This potentially leads to *divergent* (or different) replicas – an undesirable situation for replication-based collaborative systems [10].

[1] Email: imine@loria.fr

doi:10.1016/j.entcs.2006.08.027

Operational Transformation (OT) is an approach which has been proposed to overcome the divergence problem, especially for building real-time groupware [1,8]. This approach consists of an algorithm which transforms an operation (previously executed by some other site) according to local concurrent ones in order to achieve convergence. It has been used in several group editors [1,6,8,7,11,9], and it is employed in other replication-based groupwares such as a generic synchronizer [5]. The advantages of this approach are: (i) it is independent of the replica state and depends only on concurrent operations; (ii) it enables an unconstrained concurrency, *i.e.* no global order on operations is required; (iii) it ensures a good responsiveness in real-time interaction context. However, if OT algorithms are not correct then the consistency of shared data is not ensured. Thus, it is critical to verify such algorithms in order to avoid the loss of data when broadcasting operations. According to [6], the OT algorithm of every collaborative object needs to fulfill two *convergence conditions* $TP1$ and $TP2$ that will be detailed in Section 2. Finding such an OT algorithm and proving that it satisfies $TP1$ and $TP2$ is not an easy task. This proof is often difficult – even impossible – to produce by hand and unmanageably complicated. In [4], we proposed a formal framework for modeling and analyzing the OT algorithms with algebraic specifications. For checking the convergence conditions we used a theorem prover. Using our formal approach we have detected bugs in well-known OT algorithms.

Until now the OT approach has been used to only deal with simple collaborative objects, such as a string object. When we consider a complex object (such as a filesystem or an XML document that are composite of several primitive objects) the formal design of its OT algorithm becomes very tedious because of the large number of updates and synchronization situations to be considered if we start from scratch. As continuation of [4], we propose in the present work a compositional method for specifying complex collaborative objects. The most important feature of our method is that designing an OT algorithm for the composed collaborative object can be done by reusing the OT algorithms of component collaborative objects. By using our method, we can start from correct small collaborative objects (*i.e.* they satisfy convergence conditions) which are relatively easy to handle and incrementally combine them to build more complex collaborative objects that are also correct.

This paper is organized as follows: in Section 2 we give the basic concepts of the OT approach. The ingredients of our formalization for specifying the collaborative object and OT algorithm are given in Section 3. In Section 4, we present two constructions for composing collaborative objects in algebraic framework. Finally, we give conclusions and present future work.

2 Operational Transformation Approach

Due to high communication latencies in wide-area and mobile wireless networks the replication of collaborative objects is commonly used in distributed collaborative systems. But this choice is not without problem as we will see in next sub-section.

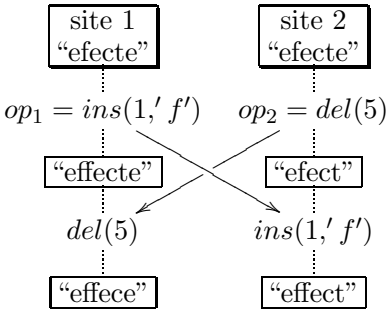


Fig. 1. Incorrect integration.

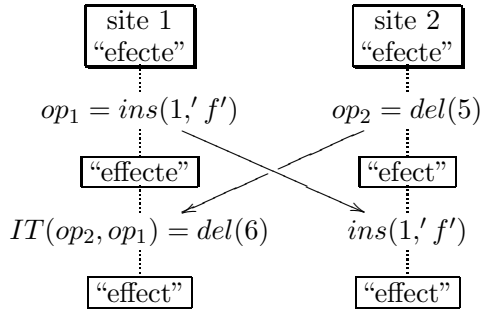


Fig. 2. Integration with transformation.

2.1 Convergence Problems

One of the significant issues when building distributed collaborative systems with a replicated architecture and an arbitrary communication of messages between users is the *consistency maintenance* (or *convergence*) of all replicas. To illustrate this problem, consider the following example:

Example 2.1 Consider the following group text editor scenario (see Figure 1): there are two users (sites) working on a shared document represented by a sequence of characters. These characters are addressed from 0 to the end of the document. Initially, both copies hold the string “`efecte`”. User 1 executes operation $op_1 = Ins(1, \text{“}\texttt{f}\text{”})$ to insert the character “`f`” at position 1. Concurrently, user 2 performs $op_2 = Del(5)$ to delete the character “`e`” at position 5. When op_1 is received and executed on site 2, it produces the expected string “`effect`”. But, when op_2 is received on site 1, it does not take into account that op_1 has been executed before it and it produces the string “`effece`”. The result at site 1 is different from the result of site 2 and it apparently violates the intention of op_2 since the last character “`e`”, which was intended to be deleted, is still present in the final string.

To maintain convergence, an OT approach has been proposed in [1]. It consists of application-dependent transformation algorithm such that for every possible pair of concurrent updates, the application programmer has to specify how to merge these updates regardless of reception order. We denote this algorithm by a function IT, called *inclusion transformation* [8].

Example 2.2 In Figure 2, we illustrate the effect of IT on the previous example. When op_2 is received on site 1, op_2 needs to be transformed in order to include the effects of op_1: $IT((Del(5), Ins(1, \text{“}\texttt{f}\text{”})) = Del(6)$. The deletion position of op_2 is incremented because op_1 has inserted a character at position 1, which is before the character deleted by op_2. Next, op_2' is executed on site 1. In the same way, when op_1 is received on site 2, it is transformed as follows: $IT(Ins(1, \text{“}\texttt{f}\text{”}), Del(5)) = Ins(1, \text{“}\texttt{f}\text{”})$; op_1 remains the same because “`f`” is inserted before the deletion position of op_2.

Intuitively we can write the transformation IT as follows:

```
IT(Ins(p1,c1),Ins(p2,c2)) = if (p1 < p2) return Ins(p1,c1)
                            else return Ins(p1+1,c1)
                            endif;
```

2.2 Transformation Properties

Notation $[op_1; op_2; \ldots; op_n]$ represents an operation sequence. We denote $Do(X, st) = st'$ when an operation (or an operation sequence) X is executed on a replica state st and produces a replica state st'.

Using an OT algorithm requires to satisfy two properties [6], called *transformation properties*. Given two operations op_1 and op_2, let $op'_2 = IT(op_2, op_1)$ and $op'_1 = IT(op_1, op_2)$, the conditions are as follows:

- **Property TP1**: $Do([op_1; op'_2], st) = Do([op_2; op'_1], st)$, for every state st.
- **Property TP2**: $IT(IT(op, op_1), op'_2) = IT(IT(op, op_2), op'_1)$.

TP1 defines a *state identity* and ensures that if op_1 and op_2 are concurrent, the effect of executing op_1 before op_2 is the same as executing op_2 before op_1. This condition is necessary but not sufficient when the number of concurrent operations is greater than two. As for **TP2**, it ensures that transforming op along equivalent and different operation sequences will give the same result. In [7], the authors have proved that conditions **TP1** and **TP2** are sufficient to ensure the convergence property for *any number* of concurrent operations which can be executed in *arbitrary order*.

Proving the correctness of OT algorithms, *w.r.t* **TP1** and **TP2** is very complex and error prone even on a simple string object. Consequently, the design of OT algorithms must be assisted by an automatic theorem prover [4].

3 Describing Individual Collaborative Objects

3.1 Basic Notions

In this sub-section we present terminology and notation that are used in the following sections. We assume that the reader is familiar with algebraic specifications. For more background on this topic see [12,3].

A *many-sorted signature* Σ is a pair (S, F) where S is a set of *sorts* and F is a $S^* \times S$-sorted set (of *function symbols*). Here, S^* is the set of finite (including empty) sequences of elements of S. Saying that $f : s_1 \times \ldots \times s_n \rightarrow s$ is in $\Sigma = (S, F)$ means that $s_1 \ldots s_n \in S^*$, $s \in S$, and $f \in F_{s_1 \ldots s_n, s}$. A Σ-*algebra* A interprets sorts as sets and operations as appropriately typed functions. A signature morphism $\Phi : \Sigma \rightarrow \Sigma'$ is a pair (f,g), such that $f : S \rightarrow S'$ and $g : \Sigma \rightarrow \Sigma'_{f^*,f}$ an $(S^* \times S)$-sorted function. Usually, we ignore the distinction between f and g and drop all subscripts, writing $\Phi(s)$ for $f(s)$ and $\Phi(\sigma)$ for $g(\sigma)$ such that $\sigma \in F_{s_1 \ldots s_n, s}$.

Let X be a family of sorted variables and let $T_\Sigma(X)$ be the algebra of Σ-terms. An *equation* is a formula of the form $l = r$ where $l, r \in T_\Sigma(X)_s$ for some sort $s \in S$. A *conditional* equation is a formula of the following form: $\bigwedge_{i=1}^{n} a_i = b_i \implies l = r$, where a_i, $b_i \in T_\Sigma(X)_{s_i}$. An *algebraic specification* is a pair (Σ, E) where Σ is a many-sorted signature and E is a set of (conditional) Σ-equations, called *axioms*

of (Σ, E). A (Σ, E)-model is a Σ-algebra A that satisfies all the axioms in E. We write $A \models^{\Sigma} E$ to indicate that A is a (Σ, E)-model. Given a signature morphism $\Phi : \Sigma \to \Sigma'$ and a Σ'-algebra A', the *reduct* of A' to Σ, denoted $\Phi(A')$, represents carriers $A'_{\Phi(s)}$ for $s \in S$ and operations $\sigma_{\Phi(s)}$ for $\sigma \in \Sigma_{s_1 \ldots s_n, s}$. Given a Σ-equation e of the form $l = r$. Then $\Phi(e)$ is $\overline{\Phi}(l) = \overline{\Phi}(r)$ where $\overline{\Phi} : T_{\Sigma}(X) \to T_{\Sigma'}(X')$ and $X' = \Phi(X)$. An important property of these translations on algebras and equations under signature morphisms is called *satisfaction condition*, which expresses the invariance of satisfaction under change of notation:

Theorem 3.1 (Satisfaction Condition [2]). *Given a signature morphism* $\Phi : \Sigma \to \Sigma'$*, a* Σ'*-algebra* A' *and a* Σ*-equation* e*,* $\Phi(A') \models^{\Sigma} e$ *iff* $A' \models^{\Sigma'} \Phi(e)$ □

An *observational signature* is a many-sorted signature $\Sigma = (S, S_{obs}, F)$ where $S_{obs} \subseteq S$ is the set of *observable* sorts. An *Observational Specification* is a pair (Σ, E) where Σ is an observational signature and E is a set of axioms. We assume that axioms are conditional equations with observable conditions. A *context* is a term with exactly one occurrence of a distinguished variable, say z. Observable contexts are contexts of observable sort. Let $C_{\Sigma}(s, s')$ be the set of contexts of sort s' that contain a distinguished variable of sort s. We write $c[t]$ for the replacement of distinguished variable z by the term t. A Σ-algebra A *behaviorally satisfies* an equation $l = r$, denoted $A \models^{\Sigma}_{obs} l = r$, iff $A \models^{\Sigma} c[l] = c[r]$ for every observable context c. A model of an observational specification $SP = (\Sigma, E)$ is a Σ-algebra A that behaviorally satisfies every axioms in E. We write $A \models^{\Sigma}_{obs} SP$ or $A \models^{\Sigma}_{obs} E$. Also we write $E \models^{\Sigma}_{obs} e$ iff $A \models^{\Sigma}_{obs} E$ implies $A \models^{\Sigma}_{obs} e$ where e is a (conditional)-equation.

3.2 Component Specifications

Using Observational semantics we consider a Collaborative Object (CO) as a *black box* with a hidden (or non-observable) state [3]. We only specify the interactions between a user and an object. In the following, we give our formalization:

Definition 3.2 ***(CO Signature).*** Given S the set of all sorts, $S_b = \{\texttt{State}, \texttt{Meth}\}$ is the set of *basic sorts* and $S_d = S \setminus S_b$ is the set of *data sorts*. A *CO signature* $\Sigma = (S, S_{obs}, F)$ is an observational signature where the sort `State` is the unique non-observable sort. The set of function symbols F is defined as follows:

(1) $F_{\texttt{Meth State,State}} = \{Do\}$, $F_{\texttt{Meth Meth,Meth}} = \{IT\}$, $F_{\texttt{Meth State,Bool}} = \{Poss\}$, and $F_{\omega,s} = \emptyset$ for all other cases where $\omega \in S_b^*$ and $s \in S_b$.

(2) A function symbol $f : s_1 \times s_2 \times \ldots \times s_n \to \texttt{Meth}$ is called a *method* if $s_1 \cdot s_2 \cdot \ldots \cdot s_n \in S_d^*$.

(3) A function symbol $f : s_1 \times s_2 \times \ldots \times s_n \to s$ is called an *attribute* if: (i) $s_1 \cdot s_2 \cdot \ldots \cdot s_n$ contains only one `State` sort; and (ii) $s \in S_d$.

We use Σ, Σ', Σ_1, Σ_2, ..., as variables ranging over CO signatures. □

The states of a collaborative object are accessible using the function *Do* which given a method and a state gives the resulting state provided that the execution of

this method is possible. For this we use a boolean function $Poss$ that indicates the conditions under which a method is enabled. The OT algorithm is denoted by the function symbol IT which takes two methods as arguments and produces another method.

Definition 3.3 ***(Σ-Morphism).*** Given CO signatures Σ and Σ', then a Σ-*morphism* $\Phi : \Sigma \to \Sigma'$ is a signature morphism such that:

(i) $\Phi(s) = s$ for all $s \in S_d$;

(ii) $\Phi(f) = f$ for all $f \in \Sigma_{\omega,s}$ where $\omega \in S_d^*$ and $s \in S_d$;

(iii) $\Phi(S_b) = S'_b$ (where $S'_b = \{\mathit{State'}, \mathit{Meth'}\}$, $\Phi(\mathit{State}) = \mathit{State'}$ and $\Phi(\mathit{Meth}) = \mathit{Meth'}$). □

The three conditions stipulate that Σ-morphisms preserve `State` sort, observable sorts and functions.

Definition 3.4 ***(Collaborative Component Specification).*** A *collaborative component specification* is a tuple $\mathcal{C} = (\Sigma, M, A, T, E)$ where:

(i) Σ is a CO signature;

(ii) M is a set of method symbols, *i.e.* $M = \{m \mid m \in \Sigma_{\omega,\mathtt{Meth}}$ *and* $\omega \in S_d^*\}$;

(iii) A *is a set of attribute symbols, i.e.* $A = \{a \mid a \in \Sigma_{\omega,s}$ *where* ω *contains exactly one* `State` *sort and* $s \in S_d^*\}$;

(iv) T *is the set of axioms corresponding to the transformation function;*

(v) E *is the set of all axioms.*

We let $\mathcal{C}$, $\mathcal{C}'$, $\mathcal{C}_1$, $\mathcal{C}_2$, ..., denote collaborative component specifications. □

In the following, we assume that all used (conditional) equations are universally quantified.

Example 3.5 The following component specification `CCHAR` models a memory cell (or a buffer) which stores a character value:

```
spec CCHAR =
sort:
  Char Meth State
opns:
  Do : Meth State -> State
  putchar : Char -> Meth
  getchar : State -> Char
  IT : Meth Meth -> Meth
axioms:
  (1) getchar(Do(putchar(c),st)) = c;
  (2) IT(putchar(c1),putchar(c2)) = putchar(maxchar(c1,c2));
```

`CCHAR` has one method `putchar` and one attribute `getchar`. Axiom `(2)` gives how to transform two concurrent `putchar` in order to achieve the data convergence. For that, we use function `maxchar` that computes the maximum of two character values. Note we could have used another way to enforce convergence.

As the previous specification `CNAT` and `CCOLOR` model a memory cell which stores respectively a natural number value and a color value:

```
spec CNAT =
sort:
  Nat Meth State
opns:
```

```
  Do : Meth State -> State
  putnat : Nat -> Meth
  getnat : State -> Nat
  IT : Meth Meth -> Meth
axioms:
  (1) getnat(Do(putchar(n),st)) = n;
  (2) IT(putnat(n1),putnat(n2)) = putnat(minnat(n1,n2));

spec CCOLOR =
sort:
  Color Meth State
opns:
  Do : Meth State -> State
  putcolor : Color -> Meth
  getcolor : State -> Color
  IT : Meth Meth -> Meth
axioms:
  (1) getcolor(Do(putcolor(cl),st)) = cl;
  (2) IT(putcolor(cl1),putcolor(cl1)) = putcolor(mincolor(cl1,cl2));
```

To get data convergence we have used in `CNAT` (resp. `CCOLOR`) another function `minnat` (resp. `mincolor`) that computes the minimum value.

The sorts `Char`, `Nat` and `Color` are built-in. □

For a concise presentation and without loss of generality, we shall omit the observable-sorted arguments from methods and attributes. We could suppose we have one function for each of its possible arguments. For instance, method `putchar(`c`)` may be replaced by $\texttt{putchar}_c$ for every $c \in$ `CHAR`.

Definition 3.6 ***((M,A)-Complete).*** Given a component specification $\mathcal{C} = (\Sigma, M, A, T, E)$. The set E is (M,A)-*complete* iff all equations involving M have the form:

$$C \implies a(Do(m, x)) = t$$

with x is a variable of sort ***State***, $a \in A$, $m \in M$, $t \in T_{\Sigma \setminus M}(\{x\})$ and C is a finite set of visible pairs $t_1 = t'_1$, $t_2 = t'_2$, …, $t_n = t'_n$ where t_1, $t'_1 \in T_\Sigma(X)_{s_1}$, t_2, $t'_2 \in T_\Sigma(X)_{s_2}$, …, t_n, $t'_n \in T_\Sigma(X)_{s_n}$. □

In Example 3.5, component specification `CCHAR` is (M,A)-complete as the only axiom involving methods (*i.e.*, axiom `(1)`) has the required form. `CNAT` and `CCOLOR` are also (M,A)-complete. In the remaining of this paper, we restrict our intention to component specification which are (M,A)-complete.

As a component specification has a an observational signature with one non-observable sort, `State`, then the observable contexts have the following form: $a(Do(m_n, \ldots, Do(m_1, s))$ where m_1,…, m_n are methods and a is an attribute.

Definition 3.7 ***(Specification morphisms).*** Given two collaborative component specifications $\mathcal{C} = (\Sigma, M, A, T, E)$ and $\mathcal{C}' = (\Sigma', M', A', T', E')$, a *specification morphism* $\Phi : \mathcal{C} \to \mathcal{C}'$ is a signature morphism $\Phi : \Sigma \to \Sigma'$ such that: (i) $\Phi(M) \subseteq M'$; (ii) $\Phi(A) \subseteq A'$; (iii) $E' \models^{\Sigma'}_{obs} \Phi(e)$ for each $e \in E$. □

Definition 3.7 provides a support for reusing component specification through the notion of specification morphism. Moreover, it exploits the fact that the source component specification is (M,A)-complete by only requiring the satisfaction of finite number of equations (see condition (*iii*)).

3.3 Convergence Properties

Before stating the properties that a component specification $\mathcal{C} = (\Sigma, M, A, T, E)$ has to satisfy for ensuring convergence, we introduce some notations. Let m_1, m_2, ..., m_n and s be terms of sorts `Meth` and `State` respectively:

(i) applying a method sequence on a state is denoted as:

$$(s)[m_1; m_2; \ldots; m_n] \equiv Do(m_n, \ldots, Do(m_2, Do(m_1, s)) \ldots)$$

(ii) $Legal([m_1; m_2; \ldots; m_n], s) \equiv Poss(m_1, s) \wedge Poss(m_2, (s)m_1) \wedge \ldots \wedge Poss(m_n, (s)[m_1; m_2; \ldots; m_{n-1}])$.

(iii) $IT^*(m, []) = m$ and $IT^*(m, [m_1; m_2; \ldots; m_{n-1}]) = IT^*(IT(m, m_1), [m_2; \ldots; m_{n-1}])$ where [] is an empty method sequence.

TP1 expresses a state identity between two method sequences. As mentioned before, we use an observational approach for comparing two states. Accordingly, we define the condition **TP1** by the following state property (where the variables st, m_1 and m_2 are universally quantified):

$$CP1 \equiv (Legal(seq_1, s) = true \wedge Legal(seq_2, s) = true) \implies (s)seq_1 = (s)seq_2$$

where $seq_1 = [m_1; IT(m_2, m_1)]$ and $seq_2 = [m_2; IT(m_1, m_2)]$.

Let $M' \subseteq M$ be a set of methods, we denote $CP1|_{M'}$ as the restriction of $CP1$ to M'. Let $M_1, M_2 \subseteq M$ be two disjoint sets of methods, we define $CP1|_{M_1,M_2}$ as:

$$CP1|_{M_1,M_2} \equiv (Legal(seq_i, s) = true \wedge Legal(seq_j, s) = true) \implies (s)seq_i = (s)seq_j$$

where $seq_i = [m_i; IT(m_j, m_i)]$ and $seq_j = [m_j; IT(m_i, m_j)]$ such that $m_i \in M_i$ and $m_j \in M_j$ for all $i \neq j \in \{1, 2\}$.

TP2 stipulates a *method identity* between two equivalent sequences. Given three methods m_1, m_2 and m_3, transforming m_3 with respect to two method sequences $[m_1; IT(m_2, m_1)]$ and $[m_2; IT(m_1, m_2)]$ must give the same method. We define **TP2** by the following property:

$$CP2 \equiv IT^*(m_3, [m_1; IT(m_2, m_1)]) = IT^*(m_3, [m_2; IT(m_1, m_2)])$$

Let $M' \subseteq M$ be a set of methods, we denote $CP2|_{M'}$ as the restriction of $CP2$ to M'. Let $M_1, M_2 \subseteq M$ be two disjoint sets of methods, we define $CP2|_{M_1,M_2}$ as:

$$CP2|_{M_1,M_2} \equiv IT^*(m, [m'; IT(m'', m')]) = IT^*(m, [m''; IT(m', m'')])$$

such that $m' \in M_i$, $m'' \in M_j$ and $m \in M_k$ for all $i, j, k \in \{1, 2\}$ with $k \neq i$ or $k \neq j$.

The following definition gives the conditions under which a component specification ensures the data convergence:

Definition 3.8 ***(Consistency).*** $\mathcal{C}$ is said *consistent* iff $\mathcal{C} \models_{obs} CP1 \wedge CP2$. □

4 Composing Collaborative Objects

The concurrent interaction in a collaborative context is an essential feature of the object paradigm. More attention is paid for allowing concurrent executions to achieve convergence. This section describes how to build complex collaborative objects from components using two kinds of composition. For that, we are inspired of work of Goguen et al [2].

4.1 Static Concurrent Composition

As first construction, we propose a composition of several components without interference (or synchronization) between them, in the sense that every component can change state independently.

Definition 4.1 ***(Static composition).*** Given n component specifications $\mathcal{C}_i = (\Sigma_i, M_i, A_i, T_i, E_i)$ with $i \in \{1, \ldots, n\}$ and $n > 1$. A component specification $\mathcal{C} = (\Sigma, M, A, T, E)$ is said *static concurrent composition* of $\mathcal{C}_i$ iff there exists specification morphisms $\Phi_i : \mathcal{C}_i \to \mathcal{C}$ such that for each $i \neq j \in \{1, \ldots, n\}$: (i) $\Sigma = \bigcup_i \Phi_i(\Sigma_i)$; (ii) $M = \bigcup_i \Phi_i(M_i)$; (iii) $A = \bigcup_i \Phi_i(A_i)$; (iv) $T = \bigcup_i \Phi_i(T_i) \cup \bigcup_{i,j} T_{ij}$ where:

$$T_{ij} = \{IT(\Phi_i(m_i), \Phi_j(m_j)) = \Phi_i(m_i) \,|\, m_i \in M_i \text{ and } m_j \in M_j\}$$

(v) $E = \bigcup_i \Phi_i(E_i) \cup IE$ where IE is called the *interaction part* between $\mathcal{C}_i$, and $IE = \bigcup_{i,j}(T_{ij} \cup A_{ij})$ such that:

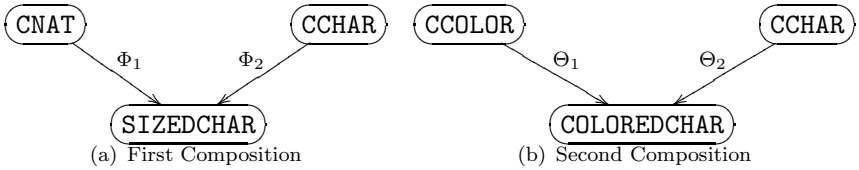
$$A_{ij} = \{\Phi_i(a_i)(Do(\Phi_j(m_j), x)) = \Phi_i(a_i)(x) \,|\, a_i \in A_i \text{ and } m_j \in M_j\}$$

We denote the static concurrent composition as $\mathcal{C} = \bigoplus_i \mathcal{C}_i$. □

Example 4.2 Consider the character memory cell `CCHAR` $= (\Sigma_1, M_1, A_1, T_1, E_1)$, the natural memory cell `CNAT` $= (\Sigma_2, M_2, A_2, T_2, E_2)$ and the color memory cell `CCOLOR` $= (\Sigma_3, M_3, A_3, T_3, E_3)$ of Example 3.5. The static concurrent composition of `CCHAR` and `CNAT` is the following composite specification `SIZEDCHAR` $= (\Sigma, M, A, T, E)$ (see Figure 3(a)):

```
spec SIZEDCHAR =
sort:
 Char Nat Meth State
opns:
 Do : Meth State -> State
 putchar : Char -> Meth
 putnat  : Nat  -> Meth
 getchar : State -> Char
 getnat  : State -> Nat
 IT : Meth Meth -> Meth
axioms:
 (1) getchar(Do(putchar(c),st)) = c;
 (2) getnat(Do(putnat(n),st)) = n;
 (3) getchar(Do(putnat(n),st))=getchar(st);
 (4) getnat(Do(putchar(c),st)) = getnat(n);
```

```
(5) IT(putchar(c1),putchar(c2)) = putchar(maxchar(c1,c2));
(6) IT(putnat(n1),putnat(n2)) = putchar(minnat(n1,n2));
(7) IT(putchar(c1),putnat(n1)) = putchar(c1);
(8) IT(putnat(n1),putchar(c1)) = putnat(n1);
```



(a) First Composition (b) Second Composition

Fig. 3. Static Concurrent Composition

where T_{ij} contains axioms `(9)-(10)` and A_{ij} contains the axioms `(4)-(5)` for all $i, j \in \{1, 2\}$ and $i \neq j$. Note that the specification morphisms $\Phi_1 : \mathcal{C}_1 \rightarrow \mathcal{C}$ and $\Phi_2 : \mathcal{C}_2 \rightarrow \mathcal{C}$ are just the inclusion morphisms. For instance this concurrent composition may be associated to an object that has a character value and an attribute for modifying the font size. In the same way, the static concurrent composition of `CCHAR` and `CCOLOR` is the following specification `COLOREDCHAR` $= (\Sigma, M, A, T, E)$ that models an object with a character and a color (see Figure 3(b)):

```
spec COLOREDCHAR =
sort:
 Char Color Meth State
opns:
 Do : Meth State -> State
 putchar : Char -> Meth
 putcolor  : Color  -> Meth
 getchar : State -> Char
 getcolor  : State -> Color
 IT : Meth Meth -> Meth
axioms:
 (1) getchar(Do(putchar(c),st)) = c;
 (2) getcolor(Do(putcolor(cl),st)) = cl;
 (3) getchar(Do(putcolor(cl),st))=getchar(st);
 (4) getcolor(Do(putchar(c),st)) = getcolor(st);
 (5) IT(putchar(c1),putchar(c2)) = putchar(maxchar(c1,c2));
 (6) IT(putcolor(cl1),putcolor(cl2)) = putcolor(mincolor(n1,n2));
 (7) IT(putchar(c1),putcolor(cl1)) = putchar(c1);
 (8) IT(putcolor(cl1),putchar(c1)) = putcolor(cl1);
```

□

We define the true concurrency (or the commutativity of methods) between n component specifications combined into one specification according to Definition 4.1:

Definition 4.3 ***(Independent Components).*** Let $\mathcal{C} = \bigoplus_i \mathcal{C}_i$ for $i \in \{1, \ldots, n\}$ and $n > 1$. The component specifications $\mathcal{C}_i$ are said *independent* iff:

$$\mathcal{C} \models^{\Sigma}_{obs} (s)[\Phi_i(m_i); \Phi_j(m_j)] = (s)[\Phi_j(m_j); \Phi_i(m_i)]$$

such that $m_i \in M_i$ and $m_j \in M_j$ with $i \neq j \in \{1, \ldots, n\}$. □

In Example 4.2, components `CCHAR` and `CNAT` are independent because the following equation:

`Do(putchar(c),Do(putnat(n),s))=Do(putnat(n),Do(putchar(c),s))`

is an observable consequence of `SIZEDCHAR`.

The following theorem means that the static concurrent composition preserves the convergence properties of individual components:

Theorem 4.4 *Given component specifications $\mathcal{C}_i$ and $\mathcal{C} = \bigoplus_i \mathcal{C}_i$ for $i \in \{1, \ldots, n\}$ and $n > 1$, such that $\mathcal{C}_i$ are independent. If $\mathcal{C}_i$ are consistent then $\mathcal{C} \models^{\Sigma}_{obs} CP1|_{\Phi_i(M_i)} \wedge CP2|_{\Phi_i(M_i)}$.* □

The proof of this theorem is given in Appendix A.

In Example 4.2, it is easy to see that `CCHAR`, `CNAT` and `CCOLOR` are consistent. The composite specifications `SIZEDCHAR` or `COLOREDCHAR` preserves also the convergence properties of their components.

The following theorem is very important in the sense that it stipulates that the consistency property (see Definition 3.8) can be obtained by composition. Indeed, composing consistent components produces a consistent composite object provided that the components are independent between them.

Theorem 4.5 *The concurrent composition of n consistent and independent component specifications is also consistent, where $n > 1$.* □

The proof of this theorem is given in Appendix A.

In Example 4.2, as `CCHAR`, `CNAT` and `CCOLOR` are independent between them, then `SIZEDCHAR` and `COLOREDCHAR` are consistent.

4.2 Static Concurrent Composition with Synchronization

As second construction, we allow the component specifications to interact. It is possible to get situations where the component specifications share some methods and attributes. These components are said *synchronized* by their shared part. Such a shared part is just a way for component objects to communicate.

Definition 4.6 ***(Shared Component).*** Let $\mathcal{C}_i$ be component specifications for $i \in \{0, \ldots, n\}$ and $n > 1$. The component $\mathcal{C}_0$ is called a *shared component* of $\mathcal{C}_1$, …, $\mathcal{C}_n$ iff there exists a family of specification morphisms $\Theta_i : \mathcal{C}_0 \to \mathcal{C}_i$ such that for every methods $m \in M_0$ and $m' \in M_i \setminus \Theta_i(M_0)$ and for every attributes $a \in A_0$ and $a' \in A_i \setminus \Theta_i(A_0)$ we have: (i) $\mathcal{C}_i \models^{\Sigma_i}_{obs} IT(\Theta_i(m), m') = \Theta_i(m)$; (ii) $\mathcal{C}_i \models^{\Sigma_i}_{obs} IT(m', \Theta_i(m)) = m'$; (iii) $\mathcal{C}_i \models^{\Sigma_i}_{obs} a'(Do(\Theta_i(m), x)) = a'(x)$; (iv) $\mathcal{C}_i \models^{\Sigma_i}_{obs} \Theta_i(a)(Do(m', x)) = \Theta_i(a)(x)$; (v) $\mathcal{C}_i \models^{\Sigma_i}_{obs} (s)[\Theta_i(m); m'] = (s)[m'; \Theta_i(m)]$. □

Conditions (i)-(iv) give how the interaction between $\mathcal{C}_i$ and $\mathcal{C}_0$ is defined. In fact, $\mathcal{C}_i$ and $\mathcal{C}_0$ do not have to interfere between them and their methods must be commutative according to condition (v).

Definition 4.7 ***(Synchronized static composition).*** Given component specifications $\mathcal{C}_0$, $\mathcal{C}_1$, …, $\mathcal{C}_n$, such that $\mathcal{C}_0$ is a shared component of $\mathcal{C}_i$ with $i \in \{1, \ldots, n\}$ and $n > 1$. A component $\mathcal{C}$ is said a *synchronized composition* of $\mathcal{C}_i$ iff there exist specification morphisms $\Theta_i : \mathcal{C}_0 \to \mathcal{C}_i$ and $\Phi_i : \mathcal{C}_i \to \mathcal{C}$ for $i \in \{1, \ldots, n\}$ such that: (i) $\Theta_i \circ \Phi_i = \Theta_j \circ \Phi_j$ for $i \neq j$, and; (ii) $\mathcal{C} = \bigoplus_i \mathcal{C}_i$. We denote the synchronized composition as $\mathcal{C} = \bigoplus_i^{\mathcal{C}_0} \mathcal{C}_i$. □

The signature of the synchronized composition is built by taking all operations from the signatures of the components. It should be noted that no duplicates are

made of any methods or attributes from a shared component (condition (i) of the above definition).

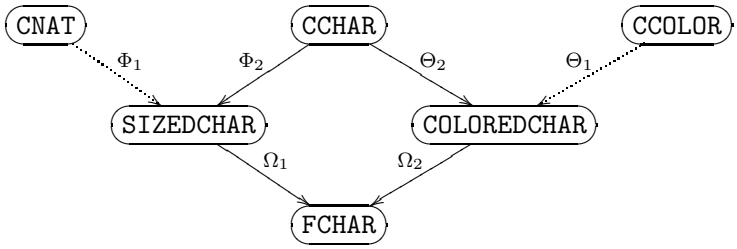


Fig. 4. Synchronized Concurrent Composition

Example 4.8 Let come back to the component specifications illustrated in Example 4.2. Recall that `SIZEDCHAR` is a character object with a size attribute and `COLOREDCHAR` is a character object with a color attribute. These objects have a shared part namely `CCHAR` (see Figure 4). Thus the composition of `SIZEDCHAR` and `COLOREDCHAR` is made by synchronization to common object `CCHAR`. We obtain the following specification:

```
spec FCHAR =
sort:
 Char Color Nat Meth State
opns:
 Do : Meth State -> State
 putchar : Char -> Meth
 putnat : Nat -> State
 putcolor  : Color  -> Meth
 getchar : State -> Char
 getnat : State -> Nat
 getcolor  : State -> Color
 IT : Meth Meth -> Meth
axioms:
 (1)  getchar(Do(putchar(c),st)) = c;
 (2)  getnat(Do(putnat(n),st)) = n;
 (3)  getcolor(Do(putcolor(cl),st)) = cl;
 (4)  getchar(Do(putcolor(cl),st))=getchar(st);
 (5)  getchar(Do(putnat(n),st)) = getchar(st);
 (6)  getnat(Do(putchar(c),st)) = getnat(st);
 (7)  getcolor(Do(putchar(c),st)) = getcolor(st);
 (8)  getnat(Do(putcolor(cl),st)) = getnat(st);
 (9)  getcolor(Do(putnat(n),st)) = getcolor(st);
 (10) IT(putchar(c1),putchar(c2)) = putchar(maxchar(c1,c2));
 (11) IT(putnat(n1),putnat(n2)) = putnat(maxnat(n1,n2));
 (12) IT(putcolor(cl1),putcolor(cl2)) = putcolor(maxcolor(cl1,cl2));
 (13) IT(putchar(c1),putcolor(cl1)) = putchar(c1);
 (14) IT(putcolor(cl1),putchar(c1)) = putcolor(cl1);
 (15) IT(putchar(c1),putnat(n1)) = putchar(c1);
 (16) IT(putnat(n1),putchar(c1)) = putnat(n1);
 (17) IT(putcolor(cl1),putnat(n1)) = putcolor(cl1);
 (18) IT(putnat(n1),putcolor(cl1)) = putnat(n1);
```

where the interaction part IE contains the axioms `(4)-(9)` and `(13)-(18)`. Note that Ω_1 and Ω_2 are just inclusion morphisms and $\Phi_2 \circ \Omega_1 = \Theta_2 \circ \Omega_2$. □

Unlike the previous composition (see Definition 4.3), the component specifications have a common part from which they are synchronized. Thus, the true concurrency is defined between the disjoint parts of these components.

Definition 4.9 ***(Independent Components).*** Let $\mathcal{C} = \bigoplus_i^{\mathcal{C}_0} \mathcal{C}_i$ for $i \in \{1, \ldots, n\}$ and $n > 1$. The component specifications $\mathcal{C}_i$ are said *independent* iff:

$$\mathcal{C} \models^{\Sigma}_{obs} (s)[\Phi_i(m_i); \Phi_j(m_j)] = (s)[\Phi_j(m_j); \Phi_i(m_i)]$$

such that $m_i \in M_i \setminus \Theta_i(M_0)$ and $m_j \in M_j \setminus \Theta_j(M_0)$ with $i \neq j \in \{1, \ldots, n\}$. □

In Example 4.8, SIZEDCHAR and COLOREDCHAR have as a shared part CCHAR and they are independent according to the above definition because:

```
Do(putcolor(c),Do(putnat(n),s))=Do(putnat(n),Do(putcolor(c),s))
```

is an observable consequence of FCHAR.

The convergence properties of individual components is preserved by the synchronized composition:

Theorem 4.10 *Given component specifications $\mathcal{C}_i$ and $\mathcal{C} = \bigoplus_i^{\mathcal{C}_0} \mathcal{C}_i$ for $i \in \{1, \ldots, n\}$ and $n > 1$, such that $\mathcal{C}_i$ are independent. If $\mathcal{C}_i$ are consistent then $\mathcal{C} \models_{obs}^{\Sigma} CP1|_{\Phi_i(M_i)} \wedge CP2|_{\Phi_i(M_i)}$.*

Proof. The proof of this theorem is similar to Theorem 4.4. □

Composing consistent components synchronized by a common component produces a consistent composite object provided that these components are independent between them.

Theorem 4.11 *The synchronized concurrent composition of n consistent and independent component specifications is also consistent, where $n > 1$.* □

The proof of this theorem is given in Appendix A.

In Example 4.8, CCHAR, CNAT and CCOLOR are consistent and independent. By composition, SIZEDCHAR = CNAT$\oplus$CCHAR and COLOREDCHAR = CCOLOR$\oplus$CCHAR are consistent. Finally, by synchronized composition, FCHAR = SIZEDCHAR$\oplus^{CCHAR}$COLOREDCHAR is also consistent.

5 Conclusion

This work is a first step toward to give a compositional method for specifying and verifying complex collaborative objects. In this respect, we have proposed two constructions for composing collaborative objects: (i) the first construction has as a basic semantic property to combine several components without allowing these components to interact; (ii) as for the second one it enables components to communicate by means of a shared part. Moreover, we have provided sufficient conditions for preserving $TP1$ and $TP2$ by both constructions.

Many features are planned to be investigated effectively with large systems. We plan to deal with composition of arbitrary number of collaborative objects by using a dynamic composition such that the objects are created and deleted dynamically. Next, we intend to study the semantic properties of compositions given in this paper. Finally, we want to implement these compositions in our tool VOTE [4].

References

[1] Ellis, C. A. and S. J. Gibbs, *Concurrency Control in Groupware Systems*, in: *Proceedings SIGMOD Conference*, 1989, pp. 399–407.

[2] Goguen, J. and R. Diaconsecu, *Towards an algebraic semantics for the object paradigm*, in: H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification*, 1994, pp. 1–29.

[3] Goguen, J. and G. Malcolm, *A hidden agenda*, Theoretical Computer Science **245** (2000), pp. 55–101.

[4] Imine, A., M. Rusinowitch, G. Oster and P. Molli, *Formal design and verification of operational transformation algorithms for copies convergence*, Theoretical Computer Science **351** (2006), pp. 167–183.

[5] Molli, P., G. Oster, H. Skaf-Molli and A. Imine, *Using the transformational approach to build a safe and generic data synchronizer*, in: *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work* (2003), pp. 212–220.

[6] Ressel, M., D. Nitsche-Ruhland and R. Gunzenhauser, *An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors*, in: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, Boston, MA, USA, 1996, pp. 288–297.

[7] Suleiman, M., M. Cart and J. Ferrié, *Concurrent Operations in a Distributed and Mobile Collaborative Environment*, in: *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA* (1998), pp. 36–45.

[8] Sun, C., X. Jia, Y. Zhang, Y. Yang and D. Chen, *Achieving Convergence, Causality-preservation and Intention-preservation in real-time Cooperative Editing Systems*, ACM Transactions on Computer-Human Interaction (TOCHI) **5** (1998), pp. 63–108.

[9] Sun, D., S. Xia, C. Sun and D. Chen, *Operational transformation for collaborative word processing*, in: *Proceedings CSCW*, 2004, pp. 437–446.

[10] Tanenbaum, A. S., "Distributed operating systems," Prentice-Hall, Inc., 2002.

[11] Vidot, N., M. Cart, J. Ferrié and M. Suleiman, *Copies convergence in a distributed real-time collaborative environment*, in: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, Philadelphia, PA, USA, 2000.

[12] Wirsing, M., *Algebraic Specification*, Handbook of theoretical computer science (vol. B): formal models and semantics (1990), pp. 675–788.

A Proofs

Proof. (Theorem 4.4) We consider two cases:

(1) *Proof of* $\mathcal{C} \models_{obs} CP1|_{\Phi_i(M_i)}$*:* We have to show that for every visible contexts c_i and c we have:

$$\mathcal{C}_i \models^{\Sigma_i} c_i[CP1|_{M_i}] \text{ implies } \mathcal{C} \models^{\Sigma} c[CP1|_{\Phi_i(M_i)}]$$

with $i \in \{1, \ldots, n\}$. Note that c_i has the form $a_i((st)[m_{i_1}; m_{i_2}; \ldots; m_{i_p}])$ with $p \geq 0$, m_{i_1}, m_{i_2}, …, $m_{i_p} \in M_i$, and $a_i \in A_i$. In the same way, c is $a((st)[m_1; m_2; \ldots; m_q])$ with $q \geq 0$, m_1, m_2, …, $m_q \in M$, and $a \in A$.

Two cases are to be considered:

(a) $c = \Phi_i(c_i)$: In this case we have

$$\mathcal{C}_i \models c_i[CP1|_{M_i}] \text{ implies } \mathcal{C} \models \Phi_i(c_i)[CP1|_{\Phi_i(M_i)}]$$

which follows from Theorem 3.1 and the fact that $\Phi_i(c_i[CP1|_{M_i}]) = \Phi_i(c_i)[CP1|_{\Phi_i(M_i)}]$.

(b) $c \neq \Phi_i(c_i)$: It means that a is not the image of a_i by Φ_i. Without loss of generality, assume that in c the methods m_1, m_2, …, m_l (for $0 \leq l < q$) are

the images by Φ_i of some methods in M_i. Thus $c[CP1|_{\Phi_i(M_i)}]$ is rewritten as follows:

$$a((st)[\Phi_i(m); IT(\Phi_i(m'), \Phi_i(m)); m_1; \ldots; m_l; \ldots; m_q]) = a((st)[\Phi_i(m'); IT(\Phi_i(m), \Phi_i(m')); m_1; \ldots; m_l; \ldots; m_q])$$

with m, $m' \in M_i$. As the component specifications $\mathcal{C}_i$ ($i \in \{1, \ldots, n\}$) are independent, methods m_1, m_2, …, m_l are commutative with m_{l+1}, …, m_q, and we get

$$a((st)[m_{l+1}; \ldots; m_q; \Phi_i(m); IT(\Phi_i(m'), \Phi_i(m)); m_1; \ldots; m_l;]) = a((st)[m_{l+1}; \ldots; m_q; \Phi_i(m'); IT(\Phi_i(m), \Phi_i(m')); m_1; \ldots; m_l])$$

which holds by using condition (v) of Definition 4.1.

(2) *Proof of* $\mathcal{C} \models_{obs} CP2|_{\Phi_i(M_i)}$*:* As $CP2$ does not contain `State` terms, then we have to show:

$$\mathcal{C}_i \models^{\Sigma_i} CP2|_{M_i} \text{ implies } \mathcal{C} \models^{\Sigma} CP2|_{\Phi_i(M_i)}$$

which also follows from Theorem 3.1 using the fact $\Phi_i(CP2|_{M_i}) = CP2|_{\Phi_i(M_i)}$. □

Proof. (Theorem 4.5) Let $\mathcal{C}$ be the concurrent composition of $\mathcal{C}_1$, $\mathcal{C}_2$, …, $\mathcal{C}_n$ where $n \geq 2$. By definition, $\mathcal{C}$ is consistent iff $\mathcal{C} \models^{\Sigma}_{obs} CP1 \wedge CP2$.

(1) *Proof of* $\mathcal{C} \models^{\Sigma}_{obs} CP1$*:* The convergence property $CP1$ can be formulated as follows:

$$CP1 \equiv \Phi_i(CP1|_{M_i}) \wedge CP1|_{\Phi_i(M_i),\Phi_j(M_j)}$$

with $i \neq j \in \{1, \ldots, n\}$. Since $\mathcal{C}_i$ are consistent then $\mathcal{C} \models^{\Sigma}_{obs} \Phi_i(CP1|_{M_i})$ follows from Theorem 4.4 and the fact that $\Phi_i(CP1|_{M_i}) = CP1|_{\Phi_i(M_i)})$. By using Definition 4.1, $\mathcal{C} \models^{\Sigma}_{obs} CP1|_{\Phi_i(M_i),\Phi_j(M_j)}$ is rewritten as follows:

$$\mathcal{C} \models_{obs} (s)[\Phi_i(m_i); \Phi_j(m_j)] = (s)[\Phi_j(m_j); \Phi_i(m_i)]$$

(where $m_i \in M_i$ and $m_j \in M_j$) that is true because the component specifications are independent.

(2) *Proof of* $\mathcal{C} \models^{\Sigma}_{obs} CP2$. The convergence property $CP2$ can be expressed as follows:

$$CP2 \equiv \Phi_i(CP2|_{M_i}) \wedge CP2|_{\Phi_i(M_i),\Phi_j(M_j)}$$

with $i \neq j \in \{1, \ldots, n\}$. Because $\mathcal{C}_i$ are consistent and using Theorem 4.4 we have $\mathcal{C} \models^{\Sigma}_{obs} \Phi_i(CP2|_{M_i})$. As for $\mathcal{C} \models^{\Sigma}_{obs} CP2|_{\Phi_i(M_i),\Phi_j(M_j)}$, it is rewritten as follows:

$$IT^*(\Phi_{k_1}(m), [\Phi_{i_1}(m'); IT(\Phi_{j_1}(m''), \Phi_{i_1}(m'))]) = IT^*(\Phi_{k_1}(m), [\Phi_{j_1}(m''); IT(\Phi_{i_1}(m'), \Phi_{j_1}(m''))])$$

where $m' \in M_{i_1}$, $m'' \in M_{j_1}$ and $m \in M_{k_1}$ for all $i_1, j_1, k_1 \in \{i, j\}$ with $k_1 \neq i_1$ or $k_1 \neq j_1$. Two cases are possible:

(a) $i_1 = j_1$ and $k_1 \neq i_1$: As $\mathcal{C}_{k_1}$ and $\mathcal{C}_{i_1}$ are independent

$$IT^*(\Phi_{k_1}(m), [\Phi_{i_1}(m'); \Phi_{i_1}(IT(m'', m'))]) = \Phi_{k_1}(m)$$
$$\text{and}$$
$$IT^*(\Phi_{k_1}(m), [\Phi_{i_1}(m''); \Phi_{i_1}(IT(m', m''))]) = \Phi_{k_1}(m)$$

(b) $i_1 \neq j_1$ and ($k_1 = i_1$ or $k_1 = j_1$): Consider the case where $k_1 = i_1$ (the case $k_1 = j_1$ is similar). As $\mathcal{C}_{i_1}$ and $\mathcal{C}_{j_1}$ are independent

$$IT^*(\Phi_{i_1}(m), [\Phi_{i_1}(m'); \Phi_{j_1}(m'')]) = IT(\Phi_{i_1}(IT(m, m')), \Phi_{j_1}(m'')) = \Phi_{i_1}(IT(m, m')) \text{ and}$$
$$IT^*(\Phi_{i_1}(m), [\Phi_{j_1}(m''); \Phi_{i_1}(m')]) = \Phi_{i_1}(IT(m, m'))$$

□

Proof. (Theorem 4.11) Let $\mathcal{C}$ be the synchronized composition of $\mathcal{C}_1$, $\mathcal{C}_2$, …, $\mathcal{C}_n$ ($n > 1$) which have $\mathcal{C}_0$ as a shared component, *i.e.* there exist specifications $\Theta_i : \mathcal{C}_0 \to \mathcal{C}_i$ and $\Phi_i : \mathcal{C}_i \to \mathcal{C}$ for $i \in \{1, \ldots, n\}$. Note that $\Phi_i(M_i) \cap \Phi_j(M_j) = \Phi_i(\Theta_i(M_0)) = \Phi_j(\Theta_j(M_0))$ for $i \neq j \in \{1, \ldots, n\}$. Let $\overline{M_i} = M_i \setminus \Theta_i(M_0)$. By definition, $\mathcal{C}$ is consistent iff $\mathcal{C} \models^{\Sigma}_{obs} CP1 \wedge CP2$.

(1) *Proof of* $\mathcal{C} \models^{\Sigma}_{obs} CP1$*:* The convergence property $CP1$ can be formulated as follows:

$$CP1 \equiv \Phi_i(CP1|_{M_i}) \wedge CP1|_{\Phi_i(\overline{M_i}), \Phi_j(\overline{M_j})}$$

with $i \neq j \in \{1, \ldots, n\}$. Since $\mathcal{C}_i$ are consistent then $\mathcal{C} \models^{\Sigma}_{obs} \Phi_i(CP1|_{M_i})$ follows from Theorem 4.10 and the fact that $\Phi_i(CP1|_{M_i}) = CP1|_{\Phi_i(M_i)}$). By using Definition 4.1, $\mathcal{C} \models^{\Sigma}_{obs} CP1|_{\Phi_i(\overline{M_i}), \Phi_j(\overline{M_j})}$ is rewritten as follows:

$$\mathcal{C} \models^{\Sigma}_{obs} (s)[\Phi_i(m_i); \Phi_j(m_j)] = (s)[\Phi_j(m_j); \Phi_i(m_i)]$$

(where $m_i \in \overline{M_i}$ and $m_j \in \overline{M_j}$) that is true because the component specifications are independent.

(2) *Proof of* $\mathcal{C} \models_{obs} CP2$. It is similar to the proof given for Theorem 4.5.

□