Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 117 (2005) 353–373
www.elsevier.com/locate/entcs

Communicating Concurrent Objects in HiddenCCS

Gabriel Ciobanu Dorel Lucanu

"A.I.Cuza" University, Faculty of Computer Science
Berthelot 16, 700483 Iaşi, Romania
E-mail: {gabriel,dlucanu}@info.uaic.ro

Abstract

In this paper we add value-passing communication to hiddenCCS, a new formalism proposed in [2] for synchronizing concurrent objects. We use hidden algebra to specify object-oriented systems, and CCS process algebra to describe the coordination aspects. The new specification formalism extends the object specification with synchronization and communication elements associated with methods and attributes of the objects, and use a CCS description of the interaction patterns. The operational semantics of hiddenCCS specifications is based on labeled transition systems which can be specified in rewriting logic. We use Maude as a platform for verification of the communicating concurrent objects specified in hiddenCCS. Triple Modular Redundancy is used as an example of a hiddenCCS specification and its verification in Maude.

Keywords: Algebraic specification, concurrent systems, process algebra, object- oriented specification, hidden algebra, rewriting logic, Maude, CCS, linear temporal logic, verification techniques.

1 Introduction

The motivating idea of this paper is to study the coordination of some local goals given by various computing components in a concurrent system. In our approach, hidden algebra is used to specify the local goals as concurrent objects, and CCS is used to describe the coordination of the synchronizing and communicating objects.

State-based formalisms such as hidden algebra provide specification techniques for capturing complex data and states; however they are weak for capturing the interaction aspects of communicating concurrent systems. On the other hand, the process algebra and other concurrent calculi can support

doi:10.1016/j.entcs.2004.06.018

dynamic interaction and mobility; however they generally are not adequate to model data and states of complex concurrent systems. We have proposed an integration of hidden algebra and CCS, introducing hiddenCCS in [2]. In hiddenCCS we use hidden algebra to specify objects, and CCS to describe their synchronization. In this paper we extend the hiddenCCS approach, adding value-passing communication between distributed objects. We keep the name hiddenCCS for this extension. Synchronization and communication are presented as two forms of interaction between objects. From an object-oriented point of view, we preserve the properties and the expressive power of hidden algebra specification; from a process algebra point of view, we describe the possible patterns of interaction and preserve the expressive power of CCS.

Hidden algebra takes as basic the notion of equational behavioral satisfaction: this means that hidden specifications characterize how objects behave in response to a given set of experiments. Hidden algebra is able to handle several features of large systems, including local states, nondeterminism, as well as the usual features of the object-oriented programming paradigm [8]. CCS is a concurrency calculus used to model the interaction among objects; a CCS process expresses the capability of the system to interact with other systems running concurrently. We use CCS to specify the communication requirements, describing interaction patterns between concurrent objects.

We extend the algebraic specifications of hidden algebra with two elements of interaction, namely with synchronization and communication elements. Synchronization elements link two objects whenever one is asking for a resource, and the other can offer such a resource. Communication given by a method accessing an attribute is similar to a value-passing interaction from the object having the attribute to the object having the method. The formal operational semantics of hiddenCCS integrates model semantics of hidden algebra and CCS reduction rules by using these elements of interaction. The resulting labeled transition systems are translated into rewriting logic specifications using the Maude implementations of CCS and Hennessy-Milner logic. We use the Maude implementation of the linear temporal logic to verify properties of the hiddenCCS specifications.

The structure of the paper is as follows. Section 2 briefly presents hidden algebra. Section 3 briefly presents CCS. Section 4 includes the main contributions: we introduce the new hiddenCCS specifications, and present some theoretical results. Then we show how the operational semantics of the hiddenCCS specifications is described in rewriting logic and how the Maude system is used to verify temporal properties of hiddenCCS specifications. Conclusions and references end the paper.

2 Specification of Objects in Hidden Algebra

We briefly present the main concepts and notations of the hidden algebra. A detailed presentation of hidden algebra can be found in [8,14].

A *fixed data hidden-signature* Σ consists of:

- two disjoint sets: $V(\Sigma)$ of *visible* sorts, and $H(\Sigma)$ of *hidden* sorts;
- a many sorted $(V \cup H)$-signature Σ;
- an $\Sigma\restriction_V$-algebra $D(\Sigma)$ called *data algebra*.

We simply write V, H, and D whenever Σ is understood from the context. Given a hidden-signature Σ, a *hidden Σ-model* is a Σ-algebra M such that $M\restriction_{\Sigma\restriction_V} = D$. This means that the interpretation of each sort $s \in V \cup H$ is a distinct set M_s, and the interpretation of a symbol $f \in \Sigma_{s_1 \cdots s_n, s}$ is a function $[\![f]\!]_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$. We denote by $M\restriction_{\Sigma\restriction_V}$ the algebra M restricted only to the visible sorts and visible operations. A *hidden Σ-homomorphism* $h : M \to M'$ is a Σ-homomorphism such that $h\restriction_{\Sigma\restriction_V} = \mathrm{id}_D$. Given a hidden-signature Σ and a subsignature $\Gamma \subseteq \Sigma$ such that $\Gamma\restriction_V = \Sigma\restriction_V$, a *$\Gamma$-context for sort s* is a term in $\mathcal{T}_\Gamma(\{\_ : s\} \uplus Z)$ having exactly one occurrence of a special variable $\_$ of sort s. Z is an infinite set of distinct variables. $\mathcal{C}_\Gamma[\_ : s]$ denotes the set of all Γ-contexts for the sort s. If $c \in \mathcal{C}_\Gamma[\_ : s]$, then the sort of c, viewed as a term, is called the *result sort* of the context c. A Γ-context with visible result sort is called a *Γ-experiment*. If $c \in \mathcal{C}_\Gamma[\_ : s]$ with the result sort s' and $t \in \mathcal{T}_\Sigma(X)_s$, then $c[t]$ denotes the term in $\mathcal{T}_\Sigma(var(c) \cup X)$ obtained from c by substituting t for $\_$. Furthermore, for each hidden Σ-model M, c defines a map $[\![c]\!]_M : M_s \to [M^{var(c)} \to M_{s'}]$ defined by $[\![c]\!]_M(a)(\vartheta) = a_\vartheta(c)$, where $a_\vartheta(c)$ is the variable assignment $\{\_ \mapsto a\} \cup \{z \mapsto \vartheta(z) \mid z \in var(c)\}$. We call $[\![c]\!]_M$ the *interpretation* of the context c in M.

Given a hidden-signature Σ, $\Gamma \subseteq \Sigma$ such that $\Gamma\restriction_V = \Sigma\restriction_V$, and a hidden Σ-model M, the *Γ-behavioral equivalence* on M, denoted by $\equiv_\Sigma^\Gamma$, is defined as follows:

$$\begin{array}{l} \text{for any sort } s \in V \cup H \text{ and any } a, a' \in M_s, \\ \quad a \equiv_\Sigma^\Gamma a' \text{ iff } [\![c]\!]_M(a)(\vartheta) = [\![c]\!]_M(a')(\vartheta) \\ \qquad\qquad \text{for all } \Gamma\text{-experiments } c \text{ and} \\ \qquad\qquad\quad \text{all } (V \cup H)\text{-sorted maps } \vartheta : var(c) \to M. \end{array}$$

Given an equivalence $\sim$ on M, an operation $f \in \Sigma_{s_1 \ldots s_n, s}$ is *congruent wrt* $\sim$ iff $[\![f]\!]_M(a_1, \ldots, a_n) \sim [\![f]\!]_M(a'_1, \ldots, a'_n)$, whenever $a_i \sim a'_i$ for $i = 1, \ldots, n$. An operation $f \in \Sigma$ is *Γ-behaviorally congruent wrt M* iff it is congruent wrt $\equiv_\Sigma^\Gamma$. A *hidden Γ-congruence* on M is a $(V \cup H)$-equivalence on M that is an identity on visible sorts, and each operation in Γ is congruent wrt it.

Theorem 2.1 *[8,14] Given a hidden-signature Σ, a subsignature $\Gamma \subseteq \Sigma$ such that $\Gamma\restriction_V = \Sigma\restriction_V$, and a hidden Σ-model M, then Γ-behavioral equivalence is the largest hidden Γ-congruence on M.*

A hidden Σ-model M Γ-*behaviorally satisfies* a Σ-equation e of the form $(\forall X)t = t'$ `if` C, where C is a set of pairs of $\Sigma(X)$-terms, if and only if for all $\vartheta : X \to M$, $\vartheta(t) \equiv^\Gamma_\Sigma \vartheta(t')$ whenever $\vartheta(u) \equiv^\Gamma_\Sigma \vartheta(v)$ for all $(u, v) \in C$. We write $M \models^\Gamma_\Sigma e$. If E is a set of Σ-equations, we write $M \models^\Gamma_\Sigma E$ iff $M \models^\Gamma_\Sigma e$ for all e in E.

A *behavioral specification* is a triplet $\mathcal{B} = (\Sigma, \Gamma, E)$ consisting of a hidden signature Σ, a subsignature $\Gamma \subseteq \Sigma$ such that $\Gamma\restriction_V = \Sigma\restriction_V$ and a set E of Σ-equations. We often denote the constituents of $\mathcal{B}$ by $\Sigma(\mathcal{B}), \Gamma(\mathcal{B})$ and $E(\mathcal{B})$, respectively. The operations in $\Gamma \setminus (\Sigma\restriction_V)$ are called *behavioral*. A hidden Σ-model M *behaviorally satisfies* the specification $\mathcal{B}$ iff M Γ-behaviorally satisfies E, that is $M \models^\Gamma_\Sigma E$. We write $M \models \mathcal{B}$ and we say that M is a $\mathcal{B}$-*model*. For any equation e, we write $\mathcal{B} \models e$ iff $M \models \mathcal{B}$ implies $M \models e$. An operation $f \in \Sigma$ is *behaviorally congruent* wrt $\mathcal{B}$ iff f is Γ-behaviorally congruent wrt each $\mathcal{B}$-model M.

Behavioral specifications can model concurrent objects. $\mathcal{B}$ specifies a *simple object* iff:

(i) $H(\mathcal{B})$ has a unique element h called *state sort*;

(ii) each operation $f \in \Sigma \setminus \Sigma\restriction_V$ is either:
  (a) a hidden (generalized) constant modeling an initial state, or
  (b) a *method* $g : hv_1 \cdots v_n \to h$ with $v_i \in V$ for $i = 1, \ldots, n$, or
  (c) an *attribute* $q : hv_1 \cdots v_n \to v$ with $v, v_1, \cdots, v_n \in V$.

In other words, the framework for simple objects is the monadic fixed-data hidden algebra [14]. A *concurrent connection* $\mathcal{B}_1 \| \cdots \| \mathcal{B}_n$ is defined as in [7] where the (composite) state sort is implemented as tupling. If h_i is the state sort of $\mathcal{B}_i$, then a composite state is a tuple $\langle st_1, \ldots, st_n\rangle$: `Tuple` where the state st_i is of sort h_i. Projection operations i^*, $i = 1, \ldots, n$, are defined by projection equations $i^*(\langle st_1, \ldots, st_n\rangle = st_i$ together with "tupling equation" $\langle 1^*st, \ldots, n^*st\rangle = st$, where st is of sort `Tuple`. We assume that all specifications $\mathcal{B}_1, \ldots, \mathcal{B}_n$ share the same data algebra. For each component $\mathcal{B}_i$ and f in $\mathcal{B}_i$, we consider an operation $f.\mathcal{B}_i$ defined by:

$$f.\mathcal{B}_i(\langle st_1, \ldots, st_n\rangle, \overrightarrow{x}) = f(st_i, \overrightarrow{x}) \text{ if } f \text{ is an attribute, and}$$

$$f.\mathcal{B}_i(\langle st_1, \ldots, st_n\rangle, \overrightarrow{x}) = \langle st_1, \ldots, f(st_i, \overrightarrow{x}), \ldots, st_n\rangle \text{ if } f \text{ is a method,}$$

where by $\overrightarrow{x}$ we denote a sequence of the form $x_1, \ldots, x_n$.

Proposition 2.2 *[8] 1. If g is a method of $\mathcal{B}_i$, g' a method of $\mathcal{B}_j$, and $i \neq j$,* *then*

$$g.\mathcal{B}_i(g'.\mathcal{B}_j(st, \overrightarrow{y}), \overrightarrow{x}) = g'.\mathcal{B}_j(g.\mathcal{B}_i(st, \overrightarrow{x}), \overrightarrow{y})$$

2. If q is an attribute of $\mathcal{B}_i$, g a method of $\mathcal{B}_j$, and $i \neq j$, then

$$q.\mathcal{B}_i(g.\mathcal{B}_j(st, \overrightarrow{y}), \overrightarrow{x}) = q.\mathcal{B}_i(st, \overrightarrow{x}).$$

The first part of Proposition 2.2 allows us to consider the method $g.\mathcal{B}_i \| g'.\mathcal{B}_j$ such that $g.\mathcal{B}_i \| g'.\mathcal{B}_j(st, \overrightarrow{x}, \overrightarrow{y})$ means the concurrent execution of $g.\mathcal{B}_i(st, \overrightarrow{x})$ and $g'.\mathcal{B}_j(st, \overrightarrow{y})$. If $g.\mathcal{B}_i$ and $g'.\mathcal{B}_j$ are in Γ, then $g.\mathcal{B}_i \| g'.\mathcal{B}_j$ is in $\Gamma(\mathcal{B}_1 \| \ldots \| \mathcal{B}_n)$ as well. Note that the new added functions $f.\mathcal{B}_i$ could be non-behavioral in $\mathcal{B}$ even if f is behavioral in $\mathcal{B}_i$.

By *object specification* we mean either a simple object specification, or a conservative extension of a concurrent connection of object specifications. We extend an object specification by adding the following elements of interaction:

(i) *Elements of synchronization* given by pairs $(a, \overline{a})$ denoting the *necessity* and *availability* of the shared name a. Both a and $\overline{a}$ denote behavioral methods in different object components; the invocation of these methods expresses the necessity and the availability of a (it does not matter which one). An element a is called *closed synchronization* for $\mathcal{B}$ iff both a and $\overline{a}$ are present. If only one (either a or $\overline{a}$) is present, then we have an *open synchronization*. Let $Synch(\mathcal{B})$ denote the set of the synchronization elements of $\mathcal{B}$.

(ii) *Elements of communication* given by pairs $(a, \overline{a})$ denoting the *receiving* and *sending* of a value along the communication channel a. Now a denotes a behavioral method and $\overline{a}$ denotes a behavioral attribute; the value supplied by the attribute is passed to the second component as an argument of the calling method. A communication channel a is called *closed communication* for $\mathcal{B}$ iff both a and $\overline{a}$ are present. If only one element of a communication pair (either a or $\overline{a}$) is present, then we have an *open communication*. Let $Comm(\mathcal{B})$ denote the set of the communication elements of $\mathcal{B}$.

We use the BOBJ language (http://www.cs.ucsd.edu/groups/tatami/bobj/) to express the behavioral specifications. We enrich the BOBJ syntax by adding the synchronization capabilities by `synch` attributes of the corresponding methods, and the communication capabilities by `comm: ~a` and `comm: index -> a`. The index in `comm: index -> a` is used to identify the receiving component linked to the corresponding `comm: ~a`. Our extension of the BOBJ language uses `~a` to denote $\overline{\mathtt{a}}$. We consider methods receiving along at most a single communication channel. However the attributes may send values along more than one communication channel.

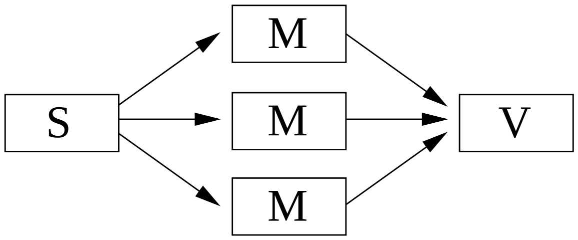

Fig. 1. Triple Modular Redundancy

Example 2.3 Triple Modular Redundancy.
The following example exhibits how the synchronization and communication elements are used. Triple Modular Redundancy (TMR) is a popular technique in fault tolerance. Three copies of a module M are put together using a splitter S and a voter V (see Figure 1). The splitter sends the input value to each copy of the system module M. There is no fixed order in which the splitter sends the input value to the three copies of M. A faulty M either behaves correctly or may output an arbitrary value. The voter accepts the results from each of these copies of M, and outputs the majority value. The behavioral specification of the system is:

```
dth DATA is
  sort Data .
  sort ModuleId .
  ops 0 1 : -> Data .
  ops 1 2 3 : -> ModuleId .
  op vop : Data -> Data .   *** equations defining vop are missing
end

bth MOD1 is
  sort Mod1 .
  inc DATA .
  op put : Mod1 Data -> Mod1 .    [comm: 1 -> mi1]
  op hop : Mod1 -> Mod1 .         [synch: w1]
  op get : Mod1 -> Data .         [comm: ~mo1]
  var M : Mod1 . var D : Data .
  eq get(put(M, D)) = D .
end

*** MOD2 and MOD3 are defined in a similar way as MOD1

bth SPLITTER is
  sort Splitter .
  inc DATA .
  op rec : Splitter Data -> Splitter .  [comm: 1 -> in]
  op split : Splitter -> Data .         [comm: ~mi1, ~mi2, ~mi3]
  op rack : Splitter -> Splitter .      [synch: ack]
  var S : Splitter . var D : Data .
  eq split(rec(S, D)) = D .
```

```
end

bth VOTER is
  sort Voter .
  inc DATA .
  op rec1 : Voter Data -> Voter .   [comm: 1 -> mo1]
  op rec2 : Voter Data -> Voter .   [comm: 1 -> mo2]
  op rec3 : Voter Data -> Voter .   [comm: 1 -> mo3]
  op val : Voter -> Data .          [comm: out]
  var V : Voter . vars D1 D2 D3 : Data .
  eq val(rec3(rec2(rec1(V, D1), D2), D3)) =
     if (D1 == D2) then D1 else D3 fi .
  op sack : Voter -> Voter .        [synch: ~ack]
end

bth TMR is
  inc (SPLITTER || MOD1 || MOD2 || MOD3 || VOTER) * (sort Tuple to Tmr) .
end
```

MOD1 specifies a simple TMR module that receives an input data from somewhere and executes an operation (method) hop over these values. Normally we expect the result of the operation hop(_) to be vop(get(hop(_))), but unfortunately the system may have faults and therefore the result is unpredictable. This nondeterministic property is implicitly expressed by the fact that we have no equation regarding the effect of hop. The comm attributes are introduced to describe the communication elements of the module. For instance, comm: 1 -> mi1 specifies that the 1st visible argument of the method put is the value passed by someone else using the communication channel ~mi1. This value is bound to the 1st argument of the method. comm: ~mo1 specifies that the module sends along the communication channel mo1 the value returned by the attribute get for the current state. We have $synch$(hop.MOD1) = w1, $comm$(put.MOD1) = 1 $\rightarrow$ mi1, $comm$(get.MOD1) = ~mo1, $Synch$(MOD1) = {w1}, $Comm$(MOD1) = {mi1, ~mo1}. The splitter receives a value along in, passes this value along ~mi1, ~mi2, ~mi3, and waits for an acknowledgment along ack. The voter receives data along mo1, mo2, mo3, and sends the majority value along ~out; then it sends an acknowledgment along ~ack. The effective actions are made by the operations behind these synchronization and communication elements.

3 CCS

The Calculus of Communicating Systems (CCS) was originally developed in the '80s by Milner [12]. CCS provides a minimal formal framework to describe and study synchronized and communicating concurrent processes and various behavioral equivalences. Interaction among processes is established by a non-

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\frac{\{\overrightarrow{b}/\overrightarrow{a}\}P_A \xrightarrow{\alpha} P'}{A\langle \overrightarrow{b} \rangle \xrightarrow{\alpha} P'}\ A(\overrightarrow{a}) \overset{\text{def}}{=} P_A \qquad \frac{P \xrightarrow{\alpha} P'}{\texttt{new}\ L\ P \xrightarrow{\alpha} \texttt{new}\ L\ P'}\ \alpha \notin L \cup \overline{L}$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \frac{P \overset{\tau}{\Longrightarrow}{}^{*} P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \overset{\tau}{\Longrightarrow}{}^{*} Q}{P \overset{\alpha}{\Longrightarrow} Q}$$

Fig. 2. CCS operational semantics

deterministic matching between complementary ends of some synchronization and communication channels. When there are many pairs which can satisfy the matching condition, only a single pair $(a, \overline{a})$ is selected.

We assume a set $\mathcal{A}$ of *names*; the elements of the set $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ are called *co-names*, and the elements of the set $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ are *labels* naming ordinary actions. The standard definition of CCS includes only one special action called *silent action* and denoted by τ, intended to represent an internal action of the system.

The *processes* are defined over the set A of names by the following syntactical rules:

$$P ::= 0 \mid \alpha.P \mid P+Q \mid P|Q \mid \texttt{new}\ L\ P \mid A\langle a_1, \ldots, a_n\rangle$$

where P and Q range over processes, α over actions, a_i over names, L over sets of names, and A over process identifiers.

A structural congruence relation is defined over the set of processes. The relation $\equiv$ over the set of processes is called *structural congruence*, and is defined as the smallest congruence which satisfies:

$P \equiv Q$ if Q can be obtained from P by α-conversion,
$P + 0 \equiv P$, $P + Q \equiv Q + P$, $(P + Q) + R \equiv P + (Q + R)$,
$P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$.

The structural operational semantics is shown in Figure 2, where we have already assumed that the summation and parallel composition are associative and commutative. If $\overrightarrow{a} = (a_1, \ldots, a_n)$ and $\overrightarrow{b} = (b_1, \ldots, b_n)$, then $\{\overrightarrow{b}/\overrightarrow{a}\}P$ denotes the simultaneous substitution $P[b_1/a_1, \ldots, b_n/a_n]$. We also assume that every process identifier A has a defining equation of the form $A(\overrightarrow{a}) \overset{\text{def}}{=} P_A$ where P_A is a summation of processes, and $\overrightarrow{a} = (a_1, \ldots, a_n)$ includes all the free names in P_A.

Strong bisimulation, written $\sim$, is defined over the processes as the largest symmetrical relation such that: if $P \sim Q$ and $P \xrightarrow{\alpha} P'$, then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$. *Weak bisimulation*, written $\approx$, is defined over the processes as the largest symmetrical relation such that: if $P \approx Q$ and $P \xrightarrow{\alpha} P'$, then there exists Q' such that $Q \overset{\alpha}{\Longrightarrow} Q'$ and $P' \approx Q'$.

In our formalism, we provide a computational structure to the CCS actions, and it is enough to consider pure CCS to model the synchronization and communication between concurrent objects. According to the computational structure behind each action, it is possible to decide the type of an interaction. The synchronization elements are provided by pairs $(a, \overline{a})$ with each component associated with a method; in this way we have a method-method interaction. The communication elements are also provided by pairs $(a, \overline{a})$; however, now $\overline{a}$ is associated with an attribute, and a is associated with a method.

Hennessy-Milner Modal Logic (HML) is a simple modal logic of actions used for describing local capabilities of CCS processes. HML formulas are as follows:

$$\varphi ::= \mathtt{tt} \mid \mathtt{ff} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \mid [\![\alpha]\!]\varphi \mid \langle\!\langle\alpha\rangle\!\rangle\varphi$$

If P is a CCS process and φ a HML formula, the satisfaction relation $P \models \varphi$ is inductively defined as:

$$\begin{array}{l}
P \models \mathtt{tt} \\
P \models \varphi_1 \wedge \varphi_2 \ \text{ iff } \ P \models \varphi_1 \text{ and } P \models \varphi_2 \\
P \models \varphi_1 \vee \varphi_2 \ \text{ iff } \ P \models \varphi_1 \text{ or } P \models \varphi_2 \\
P \models [\alpha]\varphi \ \text{ iff } \ (\forall P' \in \{P'' \mid P \xrightarrow{\alpha} P''\})\ P' \models \varphi \\
P \models \langle\alpha\rangle\varphi \ \text{ iff } \ (\exists P' \in \{P'' \mid P \xrightarrow{\alpha} P''\})\ P' \models \varphi \\
P \models [\![\alpha]\!]\varphi \ \text{ iff } \ (\forall P' \in \{P'' \mid P \overset{\alpha}{\Longrightarrow} P''\})\ P' \models \varphi \\
P \models \langle\!\langle\alpha\rangle\!\rangle\varphi \ \text{ iff } \ (\exists P' \in \{P'' \mid P \overset{\alpha}{\Longrightarrow} P''\})\ P' \models \varphi
\end{array}$$

4 HiddenCCS Specifications

The integration of CCS and object specification in hidden algebra is given by the elements of synchronization and communication. A CCS process over the elements of synchronization and communication works as a coordinating module that manages the interaction between objects. A *hiddenCCS specification* is a triple $(\mathcal{B}, \mathcal{P}, \mathcal{IC})$ consisting of objects specifications $\mathcal{B}$ given in hidden algebra, a CCS description $\mathcal{P}$ of the coordinating module, and a set $\mathcal{IC}$ of integration consistency requirements. The semantics of hiddenCCS specifications is given by a labeled transition system defined over configurations (hidden state, CCS process) as follows:

(i) If $P \xrightarrow{\alpha} P'$ and α is open, then $(st, P) \xrightarrow{\alpha} (st', P')$, where st' is obtained from st by applying the method associated to α.

(ii) If $P \xrightarrow{\alpha} P'$, $Q \xrightarrow{\overline{\alpha}} Q'$, and α is closed, then $(st, P \mid Q) \xrightarrow{\tau} (st', P' \mid Q')$, where st' is obtained from st by synchronously applying the methods associated to α and $\overline{\alpha}$ whenever the integration consistency requirements are satisfied.

(iii) If $P \xrightarrow{a} P'$, $Q \xrightarrow{\overline{a}} Q'$ where $k{\rightarrow}a = \mathit{comm}(g)$ and $\overline{a} \in \mathit{comm}(q)$, then $(st, P \mid Q) \xrightarrow{\tau} (st', P' \mid Q')$ such that st' is obtained from st applying the method g having the kth argument provided the attribute q, whenever the integration consistency requirements are satisfied. Moreover, the sort of the kth argument of g should be equal to the sort of q.

This definition is sound if each (co-)name is uniquely associated to a method or attribute. Whenever the same name a is related to more than one method, e.g., to the methods $g_1, \ldots, g_n$, then we consider n distinct copies $a_1, \ldots, a_n$ of the name a, each of them for the corresponding method, and we define a relation $\_\mathtt{eq}\_$ given by $a_i \;\mathtt{eq}\; a$ for $i = 1, \ldots, n$. The operational semantics of CCS is modified as follows. The "synchronization" rule is replaced with:

$$\frac{P \xrightarrow{\alpha_i} P' \quad Q \xrightarrow{\alpha_j} Q'}{P|Q \xrightarrow{\tau(\alpha_i,\alpha_j)} P'|Q'} \quad \alpha_i \;\mathtt{eq}\; \alpha \text{ and } \alpha_j \;\mathtt{eq}\; \overline{\alpha}$$

For the silent action τ we use a more exact notation $\tau(\alpha_i, \alpha_j)$ indicating the names involved in such an internal action. This notation is necessary to integrate CCS semantics with the behavioral semantics of hidden algebra. Since τ is used in the definition of the CCS bisimulation, the following rules restore it from $\tau(\alpha_i, \alpha_j)$, where $\alpha^* = \mathtt{if}\ (\alpha = \tau(\alpha_i, \alpha_j))\ \mathtt{then}\ \tau\ \mathtt{else}\ \alpha$:

$$\frac{P \xrightarrow{\tau(\alpha_i,\alpha_j)} Q}{P \overset{\tau}{\Longrightarrow} Q} \qquad \frac{P \overset{\tau}{\Longrightarrow}{}^{*} P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \overset{\tau}{\Longrightarrow}{}^{*} Q}{P \overset{\alpha^*}{\Longrightarrow} Q}$$

A *CCS*($\mathcal{B}$)*-process* is a CCS process built over the set $\mathit{Synch}(\mathcal{B}) \cup \mathit{Comm}(\mathcal{B})$. An *integration consistency requirement* expresses the availability of an interaction resource and consists of a finite set of equations $q(\_, d_1, \ldots, d_n) = d$, where q is an attribute in Γ, and $d, d_1, \ldots, d_n \in D(\mathcal{B})$. Let $ic(a, \overline{a})$ denote the integration consistency requirement corresponding to the interaction pair $(a, \overline{a})$. A state st of a model M *satisfies* $ic(a, \overline{a})$ whenever $[\![q]\!]_M(st, d_1, \ldots, d_n) = d$ for each $q(\_, d_1, \ldots, d_n) = d$ in $ic(a, \overline{a})$; we write $st \models ic(a, \overline{a})$. Our TMR example does not use the integration consistency requirements. An example of integration consistency requirement is offered by a critical resource where it is expressed by an equation of the form $\mathtt{isAvailable(\_)} = \mathtt{true}$ meaning that the critical resource may interact only if it is available in the current state. Such an example is presented in [2].

$$\frac{g : h_i\overrightarrow{v} \to h_i,\ synch(g) = \alpha,\ st' = [\![g.\mathcal{B}_i]\!]_M(st, \overrightarrow{d}),\quad P \xrightarrow{\alpha} P'}{(st, P) \xrightarrow{\alpha} (st', P')}$$

$$\frac{g : h_i\overrightarrow{v} \to h_i,\ comm(g) = k{\to}a,\ st' = [\![g.\mathcal{B}_i]\!]_M(st, \overrightarrow{d}),\quad P \xrightarrow{a} P'}{(st, P) \xrightarrow{a} (st', P')}$$

$$\frac{q : h_i\overrightarrow{v} \to h_i,\ \overline{a} \in comm(q),\quad P \xrightarrow{\overline{a}} P'}{(st, P) \xrightarrow{\overline{a}} (st, P')}$$

$$\frac{g : h_i\overrightarrow{v} \to h_i \in \Gamma \text{ noninteracting},\ st' = [\![g.\mathcal{B}_i]\!]_M(st, \overrightarrow{d})}{(st, P) \xrightarrow{idle} (st', P)}$$

$$\frac{\begin{array}{c} g : h_i\overrightarrow{v} \to h_i,\ synch(g) = \alpha_i,\ g' : h_j\overrightarrow{v}' \to h_j, synch(g') = \alpha_j, \\ st \models ic(\alpha_i, \alpha_j),\ st' = [\![g.\mathcal{B}_i \| g'.\mathcal{B}_j]\!]_M(st, \overrightarrow{d}, \overrightarrow{d}'),\quad P \xrightarrow{\tau(\alpha_i,\alpha_j)} P' \end{array}}{(st, P) \xrightarrow{\tau} (st', P')}$$

$$\frac{\begin{array}{l} g : h_i\overrightarrow{v} \to h_i,\ comm(g) = k{\to}a_i,\ q : h_j\overrightarrow{v}' \to v', \overline{a}_j \in comm(q) \\ st \models ic(a_i, \overline{a}_j),\ st' = [\![g.\mathcal{B}_i]\!]_M(st, \overrightarrow{d}) \text{ where } d_k = [\![q.\mathcal{B}_j]\!]_M(st, \overrightarrow{d}'),\quad P \xrightarrow{\tau(a_i,\overline{a}_j)} P' \end{array}}{(st, P) \xrightarrow{\tau} (st', P')}$$

$$\frac{(st_1, P)\,(\xrightarrow{\tau} \cup \xrightarrow{idle})^*\,(st_1', P')\quad (st_1', P') \xrightarrow{\alpha} (st_2', Q')\quad (st_2', Q')\,(\xrightarrow{\tau} \cup \xrightarrow{idle})^*\,(st_2, Q)}{(st_1, P) \overset{\alpha}{\Longrightarrow} (st_2, Q)}$$

Fig. 3. HiddenCCS operational semantics

Given an object specification $\mathcal{B}$ and a hidden $\mathcal{B}$-model M, we denote by $LTS_{\Gamma,CCS}(M)$ the labeled transition system defined by the rules in Figure 3, where P, P' and Q are CCS($\mathcal{B}$)-processes, and $\overrightarrow{d}$ and $\overrightarrow{d}'$ are sequences of data values from $D(\mathcal{B})$. The transitions corresponding to noninteracting behavioral methods are labeled by *idle*. If st is a state in M and P is a CCS process, then $LTS_{\Gamma,CCS}(M, st, P)$ denotes the subsystem induced by the subset of the configurations which are reachable from (st, P).

Definition 4.1 Let Σ be an object signature, Γ a subsignature of Σ, and let M and M' be two Σ-models. The relation $\simeq^{\Sigma,\Gamma}_{M,M'} \subseteq M_h \times M'_h$ is defined by:

$st \simeq^{\Sigma,\Gamma}_{M,M'} st'$ iff $[\![q]\!]_M(st, d_1 \ldots, d_n) = [\![q]\!]_{M'}(st', d_1 \ldots, d_n)$ for any attribute $q \in \Gamma$, and data values $d_1, \ldots, d_n \in D(\mathcal{B})$.

Definition 4.2 Given an object specification $\mathcal{B} = (\Sigma, \Gamma, E)$, and two $\mathcal{B}$-models M and M', then the *behavioral CCS-based strong* Γ*-bisimulation* between M and M' is the largest relation $\sim_{M,M'}$ such that $(st_1, P) \sim_{M,M'} (st'_1, P')$ implies

(i) $st_1 \simeq^{\Sigma,\Gamma}_{M,M'} st'_1$,

(ii) if $(st_1, P) \xrightarrow{\alpha} (st_2, Q)$ then there is (st'_2, Q') such that $(st'_1, P') \xrightarrow{\alpha} (st'_2, Q')$ and $(st_2, Q) \sim_{M,M'} (st'_2, Q')$, and

(iii) if $(st'_1, P') \xrightarrow{\alpha} (st'_2, Q')$ then there is (st_2, Q) such that $(st_1, P) \xrightarrow{\alpha} (st_2, Q)$ and $(st_2, Q) \sim_{M,M'} (st'_2, Q')$.

We say that a state st is *consistent* with a process P iff for each configuration (st', P') reachable from (st, P), st' satisfies the integration consistencies required by P'. A ground term t of state sort is *consistent* with a process P iff $[\![t]\!]_M$ is consistent with a P for each $\mathcal{B}$-model M.

Proposition 4.3 *Given an object specification $\mathcal{B} = (\Sigma, \Gamma, E)$, two $CCS(\mathcal{B})$-processes P and P', and two $\mathcal{B}$-models M and M', then $P \sim P'$ whenever there are st in M and st' in M' such that st is consistent with P, st' is consistent with P', and $(st, P) \sim_{M,M'} (st', P')$.*

Proof. Let us consider $(st, P) \sim_{M,M'} (st', P')$ such that st is consistent with P and st' is consistent with P'. We have to show that $P \sim P'$. Let R be the relation defined by $Q\, R\, Q'$ iff there are st_1 and st'_1 such that $(st_1, Q) \in LTS_{\Gamma,CCS}(M, st, P)$, $(st'_1, Q') \in LTS_{\Gamma,CCS}(M', st', P')$, and $(st_1, Q) \sim_{M,M'} (st'_1, Q')$. We show that if $Q_1\, R\, Q'_1$ and $Q_1 \xrightarrow{\alpha} Q_2$, then there is Q'_2 such that $Q_2 \xrightarrow{\alpha} Q'_2$ and $Q_2 R Q'_2$. Let Q_1, Q_2, Q'_1 be such that $Q_1 R Q'_1$ and $Q_1 \xrightarrow{\alpha} Q_2$. By definition of R there are st_1 and st'_1 such that $(st_1, Q_1) \in LTS_{\Gamma,CCS}(M, st, P)$, $(st'_1, Q'_1) \in LTS_{\Gamma,CCS}(M', st', P')$, and $(st_1, Q_1) \sim_{M,M'} (st'_1, Q'_1)$. We distinguish the following cases:

(i) $\alpha = a$. Let g be the method in $\mathcal{B}_i$ associated to a. We have $(st_1, Q_1) \xrightarrow{a} (st_2, Q_2)$, where $st_2 = [\![g.\mathcal{B}_i]\!]_M(st_1, \overrightarrow{d})$ for certain $\overrightarrow{d}$. There is (st'_2, Q'_2) such that $(st'_1, Q'_1) \xrightarrow{\alpha} (st'_2, Q'_2)$ and $(st_2, Q_2) \sim_{M,M'} (st'_2, Q'_2)$ by the definition of $\sim_{M,M'}$. Since a is uniquely associated to g, it follows that $st'_2 = [\![g.\mathcal{B}_i]\!]_{M'}(st'_1, \overrightarrow{d})$ and $Q'_1 \xrightarrow{a} Q'_2$. Hence $Q_2\, R\, Q'_2$.
The case $\alpha = \overline{a}$ is similar.

(ii) $\alpha = \tau$. We suppose that τ is restored from $\tau(\alpha_i, \alpha_j)$. Since st is consistent with P and (st_1, Q_1) is reachable from (st, P), it follows that $st_1 \models ic(\alpha_i, \alpha_j)$ which implies the existence in $LTS_{\Gamma,CCS}(M)$ of a transition $(st_1, Q_1) \xrightarrow{a} (st_2, Q_2)$. Since $(st_1, Q) \sim_{M,M'} (st'_1, Q')$, it follows that there is (st'_2, Q'_2) such that $(st'_1, Q'_1) \xrightarrow{a} (st'_2, Q'_2)$ and $(st_2, Q_2) \sim_{M,M'} (st'_2, Q'_2)$.

By the definition of our transition system, there exists (α_i', α_j') such that $Q_1' \xrightarrow{\tau(\alpha_i',\alpha_j')} Q_2'$. It follows that $Q_1' \xrightarrow{\tau} Q_2'$ by the restoring rule, and $Q_2 \, R \, Q_2'$ by definition of R.

Therefore we obtained that there is Q_2' such that $Q_2 \xrightarrow{\alpha} Q_2'$ and $Q_2 \, R \, Q_2'$ in all cases. It follows that $R \subseteq \sim$. $\square$

Proposition 4.4 *Given an object specification $\mathcal{B} = (\Sigma, \Gamma, E)$, and a $\mathcal{B}$-model M, then $(st, P) \sim_{M,M} (st', P')$ whenever $st \equiv_\Sigma^\Gamma st'$ and $P \sim P'$.*

Proof. We define the relation R by $(st, P)\, R \,(st', P')$ iff $st \equiv_\Sigma^\Gamma st'$ and $P \sim P'$. If $(st, P)\, R\, (st', P')$ then $st \simeq_{M,M}^{\Sigma,\Gamma} st'$ because $\simeq_{M,M}^{\Sigma,\Gamma}$ is included in the behavioral equivalence. We suppose that $(st_1, P_1)\, R\, (st_1', P_1')$ and $(st_1, P_1) \xrightarrow{\alpha} (st_2, P_2)$. We have the following cases:

(i) $\alpha = a$. Let g be the method in $\mathcal{B}_i$ associated to a. Then $st_2 = [\![g.\mathcal{B}_i]\!]_M(st_1, \overrightarrow{d})$ for certain $\overrightarrow{d}$, and $P_1 \xrightarrow{a} P_2$ by the definition of the transition system. Since $P_1 \sim P_1'$, there is P_2' such that $P_1' \xrightarrow{a} P_2'$ and $P_2 \sim P_2'$. We consider $st_2' = [\![g.\mathcal{B}_i]\!]_M(st_1', \overrightarrow{d})$. We have $st_2 \equiv_\Sigma^\Gamma st_2'$ because $st_1 \equiv_\Sigma^\Gamma st_1'$ and g is behavioral. It follows $(st_1', P_1') \xrightarrow{a} (st_2', P_2')$ and $(st_2, P_2)\, R\, (st_2', P_2')$. The case $\alpha = \overline{a}$ is similar.

(ii) $\alpha = \tau$. By the definition of the transition system, there exists (α_i, α_j) such that $P_1 \xrightarrow{\tau(\alpha_i,\alpha_j)} P_2$. Then there is P_2' such that $P_1' \xrightarrow{\tau(\alpha_i,\alpha_j)} P_2'$ because $P_1 \sim P_1'$. Since $st_1 \equiv_\Sigma^\Gamma st_1'$, it follows that $st_1' \models ic(\alpha_i, \alpha_j)$. A reasoning similar to that of the previous case implies that there are st_2' and P_2' such that $(st_1', P_1') \xrightarrow{\tau} (st_2', P_2')$ and $(st_2, P_2)\, R\, (st_2', P_2')$.

(iii) $\alpha = idle$. We have $P_1 = P_2$. Then there is a noninteracting method g in $\mathcal{B}_i$, for certain i, and $\overrightarrow{d}$ such that $st_2 = [\![g.\mathcal{B}_i]\!]_M(st_1, \overrightarrow{d})$. We take $P_2' = P_1'$ and $st_2' = [\![g.\mathcal{B}_i]\!]_M(st_1', \overrightarrow{d})$. We have $(st_2, P_2)\, R\, (st_2', P_2')$ because g is behavioral.

We obtained in all the cases that there are st_2' and P_2' such that $(st_1', P_1') \xrightarrow{\alpha} (st_2', P_2')$ and $(st_2, P_2)\, R\, (st_2', P_2')$. Therefore $R \subseteq \sim$. $\square$

Remark 4.5 The converse of Proposition 4.4 is not generally true [9]. This is due to the fact that we may have $\sim_{M,M'} \neq \equiv_\Sigma^\Gamma$.

5 Describing the LTS in Rewriting Logic

Rewriting logic [4] is a logic able to deal with the concurrent changes of states and with concurrent computations. It has good properties, and provides a general semantic framework for executable implementations of a wide range of languages and models of concurrency. In particular, it supports the implementations of CCS and HML [16].

Maude [4] is a system extending OBJ3 with support for membership equational logic and rewriting logic. A distinguished feature of Maude is the use of the reflection property of the rewriting logic for creating executable environments for different logics, theorem provers, and models of computations. However, the current version of Maude does not support yet the hidden logic used by BOBJ for behavioral specification and verification.

We denote by $\mathcal{R}(\mathcal{P}) = (\Sigma(\mathcal{P}), E(\mathcal{P}), R(\mathcal{P}))$ the rewriting specification associated with the CCS specification $\mathcal{P}$ as in [16], and by $\models_{HML}$ the deduction relation for HML. The sort of the CCS processes in $R(\mathcal{P})$ is denoted by `Process`. `succ`(P, α) is the function which returns the set $\{Q \mid P \xrightarrow{\alpha} Q\}$. Let $SP = (\mathcal{B}, \mathcal{P}, \mathcal{IC})$ be a hiddenCCS specification where $\mathcal{B} = (\Sigma, \Gamma, E)$ and the state sort of $\mathcal{B}$ is h. We associate with SP a rewriting specification $\mathcal{R}(SP) = (\Sigma', E', R)$, where Σ' is $\Sigma \cup \Sigma(\mathcal{P}) \cup \Sigma(HML)$ together with a new sort `St` and a new operation $(\_,\_) : h\,\texttt{Process} \to \texttt{St}$, and $E' = E \cup E(\mathcal{P}) \cup E(HML)$. R includes the following rules:

(i) if $g : h_i v_1 \ldots v_n \to h_i$ is a method in $\mathcal{B}_i$ such that $synch(g) = \alpha$ or $comm(g) = k{\to}\alpha$ and α is open, then we add rewriting rules of the form:

$$\begin{aligned}\alpha : (st, P) \to (g.\mathcal{B}_i(st, d_1, \ldots, d_n), Q)\\ \text{if } P \models_{HML} \langle\alpha\rangle\texttt{tt} \wedge Q \in succ(P, \alpha)\end{aligned}$$

for appropriate $d_1, \ldots, d_n \in D(\mathcal{B})$;

(ii) if $q : h_i v_1 \ldots v_n \to v$ is an attribute in $\mathcal{B}_i$ such that $\overline{a} \in comm(q)$ and $\overline{a}$ is open, then we add rewriting rules of the form:

$$\overline{a} : (st, P) \to (st, Q) \quad \text{if } P \models_{HML} \langle\overline{a}\rangle\texttt{tt} \wedge Q \in succ(P, \overline{a})$$

(iii) if $g : h_i v_1 \ldots v_n \to h_i$ is a noninteracting method in Γ, then we add a rewriting rule of the form:

$$idle : (st, P) \to (g.\mathcal{B}_i(st, d_1, \ldots, d_n), P)$$

for appropriate $d_1, \ldots, d_n \in D(\mathcal{B})$.

(iv) if $g : h_i v_1 \ldots v_n \to h_i$ is a method in $\mathcal{B}_i$ with $synch(g) = \alpha_i$, $g' : h_j v'_1 \ldots v'_{n'} \to h_j$ is a method in $\mathcal{B}_j$ with $synch(g') = \overline{\alpha}_j$, and there is α such that α_i `eq` α and α_j `eq` $\overline{\alpha}$, then we add rewriting rules of the form:

$$\begin{aligned}\tau : (st, P) &\rightarrow (g.\mathcal{B}_i \| g'.\mathcal{B}_j(st, d_1, \ldots, d_n, d'_1, \ldots, d'_{n'}), Q)\\ &\text{if } P \models_{HML} \langle\tau(\alpha_i, \alpha_j)\rangle\mathtt{tt} \wedge Q = succ(P, \tau(\alpha_i, \alpha_j)) \wedge ic(\alpha_i, \alpha_j)\end{aligned}$$

for appropriate $d_1, \ldots, d_n, d'_1, \ldots, d'_{n'} \in D(\mathcal{B})$;

(v) if $g : h_i v_1 \ldots v_n \rightarrow h_i$ is a communicating method with $comm(g) = k \rightarrow a_i$, q in $\mathcal{B}_j$, $\overline{a}_j \in comm(q)$, and there is a such that a_i `eq` a and $\overline{a}_j$ `eq` $\overline{a}$, then we add rewriting rules of the form:

$$\begin{aligned}\tau : (st, P) &\rightarrow (g.\mathcal{B}_i(st, d_1, \ldots, d_n), Q)\\ &\text{if } P \models_{HML} \langle\tau(a_i, \overline{a}_j)\rangle\ \mathtt{tt}\ \wedge\ Q \in \mathtt{succ}(P, \tau(a_i, \overline{a}_j))\ \wedge\ ic(a_i, \overline{a}_j)\end{aligned}$$

where $d_\ell \in D(\mathcal{B})_{v_\ell}$ if $\ell \neq k$, and $d_k = q.\mathcal{B}_j(st, d'_1, \ldots, d'_m)$ for appropriate $d'_1, \ldots, d'_m \in D(\mathcal{B})$;

Remark 5.1 1. If $D(\mathcal{B})$ is infinite, then $\mathcal{R}(SP)$ could include an infinite number of rewriting rules. A particular case when the number of rules is finite is that when all the communicating methods in Γ have all the visible arguments bound to communication channels, the communicating attributes in Γ are unary – i.e., the only argument is the current state, the communication elements are closed, and Γ does not include noninteracting methods.
2. We note that $\mathcal{R}(SP)$ forgets the behavioral operations. This has some drastic semantic consequences; e.g., not all $\mathcal{R}(SP)$-models produce SP-models. Therefore we restrict the semantics of $\mathcal{R}(SP)$ to those models where the states behaviorally satisfy the equations E. This restriction together with Proposition 4.4 make sound the following deduction rule:

$$\frac{st'_1 \equiv st_1 \quad \alpha : (st_1, P) \rightarrow (st_2, Q) \quad st_2 \equiv st'_2}{\alpha : (st'_1, P) \rightarrow (st'_2, Q)}$$

The use of this rule can reduce the state space within a model checking algorithm over the rewriting specifications according to the approach presented in [10].

We use the TMR example to exhibit how the Maude system can be used to build $\mathcal{R}(SP)$. We first modify the Maude module implementing the operational semantics of CCS by replacing the rule for synchronization and communication with the following two rules:

```
crl P | Q => {tau(L)}(P' | Q') if P => {L}P' /\ Q => {~ M}Q' /\ L eq M .
crl P | Q => {tau(M)}(P' | Q') if P => {L}P' /\ Q => {~ M}Q' /\ L eq M .
```

where `L eq M` represents the implementation of the `_eq_` relation. Then we change the definition of τ, representing it by a function having as argument one of the actions involved in synchronization. This is not a restriction, because $P \mid Q \xrightarrow{\tau(\alpha_i,\alpha_j)} P' \mid Q'$ is the same as $P \mid Q \xrightarrow{\tau(\alpha_i)} P' \mid Q'$ and $P \mid Q \xrightarrow{\tau(\alpha_j)} P' \mid Q'$. Moreover, we have $\tau(\overline{\alpha}) = \tau(\alpha)$.

The Maude description of the CCS expressions for TMR is as follows:

```
mod TMR-CCSPROC is
  inc CCS .
  ops in out mi1 mi2 mi3 mo1 mo2 mo3 w1 w2 w3 ack : -> Label .
  ops  S S1 S2 M1 M2 M3 V TMR : -> ProcessId .
  eq context = ( S =def in . S1 ) &
     ( S1 =def ~ mi1 . ~ mi2 . ~ mi3 . S2 + ~ mi1 . ~ mi3 . ~ mi2 . S2 +
               ~ mi2 . ~ mi1 . ~ mi3 . S2 + ~ mi2 . ~ mi3 . ~ mi1 . S2 +
               ~ mi3 . ~ mi1 . ~ mi2 . S2 + ~ mi3 . ~ mi2 . ~ mi1 . S2 ) &
     ( S2 =def ack . S ) & ( M1 =def mi1 . w1 . ~ mo1 . M1 ) &
     ( M2 =def mi2 . w2 . ~ mo2 . M2 ) &
     ( M3 =def mi3 . w3 . ~ mo3 . M3 ) &
     ( V =def mo1 . mo2 . mo3 . out . ~ ack . V ) &
     ( TMR =def ( S | M1 | M2 | M3 | V )
                \ mi1 \ mi2 \ mi3 \ mo1 \ mo2 \ mo3 \ ack ) .
endm
```

Note that the operator `new` L P is represented by the restriction $P \setminus L$. The Maude description of the rewriting specification associated with the TMR hiddenCCS specification is :

```
mod TMR-CCS is
  pr TMR-TEST .  pr  MODAL-LOGIC .
  sort CcsTmr .
  op <_‘,_> : Tmr Term -> CcsTmr .
  op 1*_ : CcsTmr -> Tmr .  op 2*_ : CcsTmr -> Term .
  var C : CcsTmr .  vars A A’ : Tmr . vars P P’ Q Q’ : Term .
  eq < 1* C, 2* C > = C .
  eq 1* < A, P > = A .  eq 2* < A, P > = P .
  crl [in] : < A, P > =>
           < < rec(1* A, 0), 2* A, 3* A, 4* A, 5* A >,succ(P,’in.Label) >
           if P  |= < ’in.Label > tt .
  *** missing one ’in’ rule
  crl [out] : < A, P > => < A, succ(P,’out.Label) >
            if P  |= < ’out.Label > tt .
  crl [w1] : < A, P > =>
           < < 1* A, hop(2* A), 3* A, 4* A, 5* A >,succ(P,’idle1.Label) >
           if P  |= < ’idle1.Label > tt .
  *** missing two ’idle’ rules
  crl [tau] : < A, P > =>
            < < 1* A, put(2* A, split(1* A)), 3* A, 4* A, 5* A >,
              succ(P,’tau[’mi1.Label]) >
            if P  |= < ’tau[’mi1.Label] > tt .
    *** missing five ’tau’ rules
    crl [tau] : < A, P > =>
              < < rack(1* A), 2* A, 3* A, 4* A, sack(5* A) >,
                succ(P,’tau[’ack.Label]) >
            if P  |= < ’tau[’ack.Label] > tt .
endm
```

The Maude module `MODAL-LOGIC` describes HML and "`|=`" is the Maude implementation of the relation $\models_{HML}$. Since HML is implemented at the metalevel, the names used in the CCS terms are quoted. In this example the successor is unique. We have to include a rule for each successor when we have more successors.

Maude has some useful commands to analyze the dynamics of the rewriting specifications. Here we use the `search` command to generate possible evolutions for the TMR system:

```
search [50]  < init, 'TMR.ProcessId > =>+ ST:CcsTmr .
```

Maude provided 50 solutions possible:

```
...
Solution 50 (state 50)
states: 51  rewrites: 4079783942 in 8365430ms cpu (8438250ms real)
    (487695 rewrites/second)
ST:CcsTmr --> < < rec(rack(rec(initS, 1)), 0),put(hop(put(initM1, 1)), 0),
  put(hop(put(initM2, 1)), 0),hop(put(initM3, 1)),sack(rec(initV,
  get(hop(put(initM1, (1).Data))), get(hop(put(initM2, (1).Data))),
  get(hop(put(initM3, (1).Data))))) >,
  '_\_['_\_['_\_['_\_['_\_['_\_['_\_['_|_['M3.ProcessId,
  'V.ProcessId,'_._['w1.Label,'_._['~_['mo1.Label],'M1.ProcessId]],
  '_._['w2.Label,'_._['~_['mo2.Label],'M2.ProcessId]],'_._['~_['mi3.Label],
  'S2.ProcessId]],'mi1.Label],'mi2.Label],'mi3.Label],'mo1.Label],
  'mo2.Label],'mo3.Label],'ack.Label] >
```

The execution performance of Maude is not encouraging. A reason for this weak performance is that CCS and LTL model checker are implemented at the metalevel. Built-in CCS module and LTL model checker could improve the Maude execution time.

A Linear Temporal Logic (LTL) model checker is implemented in Maude [6], and it can be used to verify temporal properties of the rewriting specifications. The atomic temporal propositions for the state sort h are equations having one of the forms:

$q(\_, d_1, \ldots, d_n) = d$ with q an attribute in Γ and $d, d_1, \ldots, d_n \in D(\mathcal{B})$, or

$\langle - \rangle = g(\_, d_1, \ldots, d_n)$ with g a method in Γ and $d_1, \ldots, d_n \in D(\mathcal{B})$.

The intuitive meaning of an atomic Γ-proposition $q(\_, d_1, \ldots, d_n) = d$ is that we obtain d whenever we execute a query $q(\_, d_1, \ldots, d_n)$ over the current state. The intuitive meaning of $\langle - \rangle = g(\_, d_1, \ldots, d_n)$ is that there is a state which can be obtained from the current state by applying the method g over the current state with the arguments $d_1, \ldots, d_n$.

The linear temporal logic model checker works only if the set of reachable states is finite. The state space generated by the hiddenCCS specification `TMR-CCS` is infinite. However it can be abstracted and reduced into a finite state space by considering the behavioral equivalences. For instance, we have such behavioral equivalences for the splitter: `rack(rec(initS, 1))` $\equiv$ `initS` and `rack(rec(initS, 0))` $\equiv$ `initS`. Similar equivalences are given for the other components.

We verify now the following property: if two modules are working properly, then the voter will select the right value. In order to have a module working properly (without failures), we add the equation `get(hop(M))` = `vop(get(M))` to its specification. We use the linear temporal logic formula

```
[] (in-is-eq-0 -> <> (in-is-eq-0 U out-is-eq-0))
```

expressing that the splitter value (here 0) remains unchanged until the voter outputs it. The state for which we verify the temporal formula is defined as follows:

```
op initS : -> Splitter .
op initM1 : -> Mod1 .
op initM2 : -> Mod2 .
op initM3 : -> Mod3 .
op initV : -> Votter .
op init : -> Tmr .
eq init = < initS, initM1, initM2, initM3, initV > .
```

Then we add to `TMR` the equations expressing the behavioral equivalences:

```
vars D1 D2 D3 : Data .

eq rack(rec(initS, 0)) = initS .
eq rack(rec(initS, 1)) = initS .

eq put(op(put(initM1, D1)), D2) = put(initM1, D2) .
eq put(op(put(initM2, D1)), D2) = put(initM2, D2) .
eq put(op(put(initM3, D1)), D2) = put(initM3, D2) .

eq sack(rec(initV, D1, D2, D3)) = initV .
```

The following two Maude specifications define the predicates and load the model checker.

```
mod TMR-PREDS is
  protecting TMR-CCS .
  including SATISFACTION .
  subsort CcsTmr < State .
  ops out-is-eq-0 in-is-eq-0   : -> Prop .
  ops out-is-eq-1 in-is-eq-1   : -> Prop .
  var CT : CcsTmr .
  cq CT |= in-is-eq-0 = true if (split(1* 1* CT) == 0) .
  cq CT |= in-is-eq-1 = true if (split(1* 1* CT) == 1) .
  cq CT |= out-is-eq-0 = true if (val(5* 1* CT) == 0) .
  cq CT |= out-is-eq-1 = true if (val(5* 1* CT) == 1) .
endm
mod TMR-CHECK is
  including TMR-PREDS .
  including MODEL-CHECKER .
  including LTL-SIMPLIFIER .
```

```
    ops initconf : -> CcsTmr .
    eq initconf = < init, 'TMR.ProcessId > .
  endm
```

Maude provides the following output which express that the property holds:

```
red modelCheck(initconf, [](in-is-eq-0 -> <> (in-is-eq-0 U out-is-eq-0))) .

Maude> red modelCheck(initconf, [](in-is-eq-0 ->
                              <>(in-is-eq-0 U out-is-eq-0))) .
reduce in TMR-CHECK : modelCheck(initconf, [](in-is-eq-0 ->
    <> (in-is-eq-0 U out-is-eq-0))) .
rewrites: 5752427265
result Bool: true
Maude>
```

On the contrary, if we consider only one module working properly, then the property does not hold:

```
Maude>
red modelCheck(initconf, [](in-is-eq-0 -> <>(in-is-eq-0 U out-is-eq-0))) .
reduce in TMR-CHECK : modelCheck(initconf, [](in-is-eq-0 ->
     <>(in-is-eq-0 U out-is-eq-0))) .
rewrites: 900617437
result ModelCheckResult: counterexample({< <initS,initM1,initM2,initM3,...
```

6 Conclusion

The complexity and dynamic interaction of software components provide challenging research issues in large system design and verification. In this paper we investigate the integration between hidden algebra and CCS, using a new specification technique called hiddenCCS. The way we combine process algebra CCS used for interaction aspects with hidden algebra used for objects descriptions allows to take advantage of both approaches: high abstraction level, expressiveness, and verification tools. To summarize, in hiddenCCS we use hidden algebra to specify concurrent objects, and CCS to coordinate them by their synchronization and communication elements. A hiddenCCS specification is a triple $(\mathcal{B}, \mathcal{P}, \mathcal{IC})$ consisting of an object specification $\mathcal{B}$ given in hidden algebra, a CCS description $\mathcal{P}$ of the coordinating module, and a set $\mathcal{IC}$ of integration consistency requirements. The semantics of hiddenCCS specifications is given by a labeled transition system defined over configurations of form (hidden state, CCS process). We investigate how Maude system is able to describe and verify useful properties of the synchronized and communicating concurrent objects. Maude is used to represent the CCS-based semantics of the synchronized concurrent objects. Since CCS, its Hennessy-Milner logic, and linear temporal logic are implemented in Maude, this system is used to represent the hiddenCCS specifications and their semantics, as well as to verify some properties expressed in linear temporal logics. Rewriting logic is

considered as a logical and semantic framework for object-oriented systems and process calculi of communicating concurrent systems.
A further step is to develop a software tool which automatically builds this rewriting logic specification using the reflection property of the rewriting logic.

The object-oriented features of hidden algebra add a new dimension that is missing in other approaches including LOTOS [1], a formalism combining a dynamic part based on CCS and CSP with the algebraic specification language ACT ONE. Another formalism with a high level of abstraction is presented in [15]; it combines CCS with the Common Algebraic Specification Language (CASL). In hiddenCCS we have a clear concern separation that allows a better reuse of the coordination and objects than in [15].

References

[1] T. Bolognesi, and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P.H.J.van Eijk, C.A.Vissers, and M.Diaz (Eds.), *The Formal Description Technique LOTOS*, pp. 23-73, Elsevier, 1989.

[2] G. Ciobanu, and D. Lucanu. Specification and Verification of Synchronizing Concurrent Objects. In E.Boiten, J.Derrick, and G.Smith (Eds.) *IFM 2004*, Lecture Notes in Computer Science vol.2999, pp. 307-327, Springer, 2004.

[3] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science*, vol.285(2), pp. 187-243, Elsevier, 2002.

[5] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems. In *ACM Transactions on Programming Languages and Systems*, vol.15(1), ACM Press, pp. 36-72, 1993.

[6] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. In *4th WRLA*, *Electronic Notes in Theoretical Computer Science*, vol.71, Elsevier, 2002.

[7] J. Goguen, K. Lin, and G. Roşu. Circular Coinductive Rewriting. In *Proc. Automated Software Engineering*, IEEE Press, pp.123-131, 2000.

[8] J. Goguen, and G. Malcolm. A hidden agenda. *Theoretical Computer Science* vol.245(1), pp.55-101, 2000.

[9] D. Lucanu, and G. Ciobanu. Model Checking for Object Specifications in Hidden Algebra. In B.Steffen, G.Levi (Eds.) *Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science vol.2937, Springer, pp.97-109, 2004.

[10] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational Abstractions. `http://maude.cs.uiuc.edu/papers`, submitted for publication, 2003.

[11] J. Meseguer, and G. Roşu. Behavioral Membership Equational Logic. *Electronic Notes in Theoretical Computer Science*, vol.65, Elsevier, 2002.

[12] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[13] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[14] G. Roşu. *Hidden Logic*. PhD Thesis, Univ. California at San Diego, 2000.

[15] G. Salaün, M. Allemand, and C. Attiogbé. A Formalism Combining CCS and CASL. Research Report 00.14, IRIN, Univ. Nantes, 2001.

[16] A. Verdejo, and N. Martí-Oliet. Implementing CCS in Maude 2. In *4th WRLA*, *Electronic Notes in Theoretical Computer Science*, vol.71, Elsevier, 2002.