A process-calculus-based abstraction for coordinating multi-agent groups

Manibrata Mukherji\textsuperscript{a,}* , Dennis Kafura\textsuperscript{b}

\textsuperscript{a}Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA
\textsuperscript{b}Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, USA

Abstract

Coordination, the act of imposing a desired behavior on a group of autonomous, independently conceived agents,\textsuperscript{2} has been an important issue in the design and development of software systems, both process-based and object-based. In this paper, the Calculus of Coordinating Environments (CCE) is proposed to study coordination as the behavioral union of coordinated and coordinating agents. In CCE, the behavior of coordinated components is expressed as agents in the Calculus of Communicating Systems (CCS) and the behavior of coordinators is expressed as agents (called CE agents) of an extension of CCS. Two composition rules that capture the interaction among CE agents and CCS agents are provided. The applicability of the new formalism is shown by specifying two simple coordination problems in CCE.

Keywords. CCS; Coordination; Object composition

1. Introduction

In this paper, an abstract, formal approach, based on the Calculus of Communicating Systems (CCS) [17], for describing and reasoning about the coordination of a group of independently conceived, concurrent objects is developed. Such concurrent objects are the cornerstone of the concurrent object-oriented programming (COOP) paradigm. The formalism developed seeks to separate the coordination specification from the specifications of the objects being coordinated (the components). The advantages of realizing such a separation are ease of specifying general-purpose objects, increasing the reuse potential of the specifications of both the components and the coordinating agents, and gaining the ability to specify software systems by composing component specifications.

* Corresponding author. E-mail: mukherji@cis.udel.edu.

\textsuperscript{1} An earlier version of this paper was presented at COORDINATION '96 and appeared in the proceedings of that conference.

\textsuperscript{2} Note that the term agent is used in the sense used by Milner in [17] and not in the sense used in the multi-agent systems literature.
Studying coordination independent of the programming paradigms used to realize components is relatively new. Several proposals [5, 8, 13, 6, 16, 21, 14, 3] have considered coordination and communication among software processes using high-level process-abstractions. Also, the issue of coordination among objects in object-oriented programming languages (OOPLs), both sequential and concurrent, has received significant attention recently [7, 2, 11, 25, 4, 22, 9, 23, 1]. But, no endeavor has been made to propose formalisms that can capture the coordination among concurrent objects in a direct and simple way. Such a formalism would enable a semantic comparison of coordination constructs and a comparative study of related coordination schemes in concurrent object-oriented programming languages (COOPLs).

The different ways in which CCS (and its variants [18, 20]) may be used to specify coordination have two weaknesses. First, coordination is modeled at a very low level by making agents engage in explicit communications. Such low-level specifications are very poor candidates for specifying designs of software components that must satisfy software engineering criteria like separation of concerns and reusability. Also, when the coordinated behavior of the agents is determined using the Expansion Law of CCS, many terms are generated that represent incorrect coordination sequences among the agents. Second, the simplicity achieved by the CCS specification of an object group is not reflected by its corresponding object-based realization. That happens due to the lack of both a suitable formalism to capture multi-object coordination and a corresponding object-based implementation strategy.

Motivated by the above observations, the Calculus of Coordinating Environments (CCE) is proposed to study coordination as the behavioral union of coordinated and coordinating agents. CCE views a coordinating agent as a “container” agent that establishes a transparent boundary around the coordinated agents (through which the agents are visible to the environment) and elicits correct behavior from the group by observing actions of the coordinated agents and taking coordinating actions on them. The modeling of the observe-coordinate property of coordinating agents in CCE was inspired by the Theory of Contexts developed in [15]. Note that CCE is not claimed to be a general-purpose calculus. Instead, the goal is to augment the COOP paradigm by laying the foundation for a special-purpose calculus that enables the modeling of communication and coordination among concurrent objects.

The paper is organized as follows. Section 2 motivates the need for designing a high-level formal abstraction in CCS for specifying coordination. Section 3 introduces CCE. Section 4 presents solutions to two example coordination problems, a panel of two buttons and a vending machine, in CCE. Section 5 addresses the issue of hierarchical composition of coordinating agents. Section 6 concludes the paper.

2. Modeling object-group behavior in CCS

Autonomous agents can be coordinated in one of three ways. First, agents may bear the full responsibility of coordinating themselves and engage in explicit communication.
tion with each other for the purpose of coordination. Second, the responsibility of coordinating the agents may be delegated to a central group coordinator agent that hides the coordinated agents from the external environment. Third, the responsibility of coordination may be shared among the agents and a central group coordinator that does not hide the coordinated agents from the external environment. Each of these three ways of modeling coordination will be illustrated in this section using a simple coordination problem specified in CCS. The shortcomings of each of the CCS specifications will be discussed and the need for a better coordination abstraction will be shown.

The coordination problem considered is as follows: Two button agents are assumed to be working as a group to realize a panel of two buttons. Each button may be either in the depressed state or in the undepressed state. The environment in which the panel exists is assumed to provide the stimuli for changing the states of the buttons. The constraint that gives rise to the need for coordination is that at any point in time only one button may remain in the depressed state. Thus, when a button is depressed, the other button, if already depressed, must be undepressed.

2.1. Coordination using explicit communication

One way of modeling the coordination among the two button agents is using explicit communication. In this scheme, a button, when depressed, queries the state of the other button and undepresses it, if it is already depressed. Although direct and straightforward, the explicit inclusion of the coordination functionality complicates the design of a button agent and prevents it from participating in groups where the implemented constraint is absent.

Coordination using explicit communication leads to the following specification of the panel of buttons in CCS:

\[(B1 \mid B2), \text{ where} \]
\[B1 \overset{\text{def}}{=} \text{depress1}.(\text{undepress2}.B1' + B1'); B1' \overset{\text{def}}{=} \text{undepress1}.B1 \]
\[B2 \overset{\text{def}}{=} \text{depress2}.(\text{undepress1}.B2' + B2'); B2' \overset{\text{def}}{=} \text{undepress2}.B2 \]

The CCS agents modeling the buttons, their ports, the environment in which they exist, and the relationship of their environment with the external environment is captured in Fig. 1. The agents exist in an environment that will be referred to as the composition environment that results due to the parallel composition of the two button agents. Unless explicitly hidden, the ports of the button agents are visible to the agents in the external environment. The agents are stimulated by actions originating in the external environment and the sequence of events resulting from such stimuli are controlled by the properties of the composition environment.

The rule of CCS that defines the properties of a composition environment is the Expansion Law. The latter rule defines how the behavior of agents, composed using the composition combinator, evolve. Using the Expansion Law, one may generate
a derivation tree that depicts all possible sequences of actions that the agents in the composition environment may engage in. A partial derivation tree for the composition of the two button agents is shown in Fig. 2. The behavior of the composition as
depicted by the leftmost path in the derivation tree in Fig. 2 is as follows:

\[
(B1 | B2) = \text{depress1.depress2.}((\text{undepress2.B1'} + B1') | \\
(\text{undepress1}.B2' + \text{undepress2}.B2)) + \cdots
\]

\[
= \text{depress1.depress2.} \tau. (B1' | B2) + \cdots
\]

The special action symbol \( \tau \) captures the communication among complementary ports and is called an internal action. In the above case, the internal action is produced due to the handshake between the ports \text{undepress2} and \text{undepress2}.

The above execution path represents an incorrect coordination sequence: on pressing button two after button one, instead of button one being undepressed, button two is undepressed. Note that the above inconsistent behavior of the group of buttons is not induced by the the external environment. Instead, it is the ability of composed agents to engage in unrestricted internal communications that leads to the inconsistent behavior. The inconsistent behavior would not have arisen if there was a way of specifying that after the input action at port depress2, there must be an internal communication between port undepress1 of button agent B2 and port undepress1 of button agent B1.

Note that the above example must not be interpreted as showing the inability of CCS to model the communication among a pair of buttons. CCS is an extremely expressive calculus and it can model the correct interactions among the buttons. But, such modeling is possible only at the cost of simplicity and conciseness. The buttons must introduce hidden ports and engage in an elaborate sequence of internal communications to model the correct, coordinated behavior. Such a specification would not be a direct and simple representation of a conceptually simple problem.

2.2. Centralized group coordinators

If a centralized group coordinator is used to coordinate the buttons, then the group can be specified as follows:

\[
\text{Panel1} \overset{\text{def}}{=} (B1 | B2 | GC) \backslash \{\text{depress1, undepress1, depress2, undepress2}\}, \text{ where}
\]

\[
B1 \overset{\text{def}}{=} \text{depress1.B1'}; \quad B1' \overset{\text{def}}{=} \text{undepress1}.B1
\]

\[
B2 \overset{\text{def}}{=} \text{depress2.B2'}; \quad B2' \overset{\text{def}}{=} \text{undepress2}.B2
\]

\[
GC \overset{\text{def}}{=} \text{depressButton1.depress1.GC} + \text{depressButton2.depress2.GC''}
\]

\[
GC' \overset{\text{def}}{=} \text{depressButton1.undepress1.depress2.GC''}
\]

\[
+ \text{depressButton1.undepress1.GC}
\]

\[
GC'' \overset{\text{def}}{=} \text{depressButton1.undepress2.depress1.GC'}
\]

\[
+ \text{depressButton2.undepress2.GC}
\]

Fig. 3 shows the group coordinator agent, the button agents, and the hidden and visible ports. The group coordinator hides the two button agents completely from the
view of the external environment. That is done neither to hide any complicated internal communication details from the clients of the group nor to hide the number of components in the group (in which case a centralized group coordinator is the most desirable abstraction). Instead, the components are hidden so that a strict centralized control can be enforced over all communications inside the group thereby ensuring that the Expansion Law does not yield any terms that display uncoordinated behavior of the group.

In an object-based realization of the above group, however, the centralized group coordinator agent will be implemented as an object that has an interface consisting of the union of the interfaces of all the button objects. This replication of the interfaces could have been avoided if clients could communicate directly with the button objects and the group coordinator could observe the interactions instead of engaging in them. The replication of interfaces also has the side effects of altering the signatures of some operations of the components (for example, to disambiguate operations in different components that have the same name) and the inclusion of operations in the interface of the group coordinator that do not play any role in the coordination of the group (in order to ensure that a client does not have to send some messages to the group coordinator and some to the components). Thus, a different formalism is required to model group coordinators that do not hide communication complexity or the composition of a group.

Another issue that must be addressed is that of extending centralized group coordinator specifications. Consider the addition of one more button agent to the existing panel of two button agents. The coordination constraint is assumed to remain the same
in the new panel. One way to specify the new group is as follows:

\[ \text{Panel2} \overset{\text{def}}{=} (B1 \parallel B2 \parallel B3 \parallel GC1) \\backslash \{\text{depress1, undepress1, depress2, undepress2, depress3, undepress3}\}, \text{ where} \]

\[ B3 \overset{\text{def}}{=} \text{depress3.B3'; B3'} \overset{\text{def}}{=} \text{undepress3.B3} \]

\[ GC1 \overset{\text{def}}{=} \text{depressButton1.depress1.GC1' + depressButton2.depress2.GC1''} + \text{depressButton3.depress3.GC1'''}, \]

\[ GC1' \overset{\text{def}}{=} \text{depressButton2.undepress1.depress2.GC1''} + \text{depressButton3.undepress1.depress3.GC1'''}, \]

\[ + \text{undepressButton1.undepress1.GC1} \]

\[ GC1'' \overset{\text{def}}{=} \text{depressButton1.undepress2.depress1.GC1'}, \]

\[ + \text{depressButton2.undepress2.depress3.GC1''}, \]

\[ + \text{undepressButton2.undepress2.GC1} \]

\[ GC1''' \overset{\text{def}}{=} \text{depressButton1.undepress3.depress1.GC1'}, \]

\[ + \text{depressButton2.undepress3.depress2.GC1''}, \]

\[ + \text{undepressButton3.undepress3.GC1} \]

Like the two-button group, the above three-button group does not display any uncoordinated behavior when the Expansion Law is applied. But note that in the specification of Panel2, the specification of Panel1 cannot be reused as a composable sub-specification. Reusing the specification of Panel1 would call for the following composition:

\[ \text{Panel3} \overset{\text{def}}{=} (B3 \parallel GC2 \parallel \text{Panel1}) \\backslash \{\text{depress3, undepress3, depressButton1, undepressButton1, depressButton2, undepressButton2}\}, \text{ where} \]

\[ GC2 \overset{\text{def}}{=} \text{depressButton1.p3.depressButton1.GC2',} \]

\[ + \text{depressButton2.p3.depressButton2.GC2''}, \]

\[ + \text{depressButton3.p3.depress3.GC2'''} \]

\[ GC2' \overset{\text{def}}{=} \text{depressButton2.p3.undepressButton1.depressButton2.GC2''} + \text{depressButton3.p3.undepressButton1.depress3.GC2'''} + \text{undepressButton1.p3.undepressButton1.GC2} \]

\[ GC2'' \overset{\text{def}}{=} \text{depressButton1.p3.undepressButton2.depressButton1.GC2'} + \text{depressButton3.p3.undepressButton2.depress3.GC2'''} + \text{undepressButton2.p3.undepressButton2.GC2} \]

\[ GC2''' \overset{\text{def}}{=} \text{depressButton1.p3.undepress3.depressButton1.GC2'} \]
In the above specification, GC2 must display an interface (that is, a set of ports) that is completely different from that of Panel1 (otherwise same ports will be exposed twice to the external environment leading to incorrect behavior sequences). In an object-based implementation, that leads to the replication of the interface of the inner centralized group coordinator by the outer one.

A more serious problem arises when the Expansion Law is applied to Panel3. The first few steps of the expansion is shown below. The behavior of Panel1 is substituted in the following and the restricted ports are not shown after the initial step.

\[
\text{Panel3} \overset{\text{def}}{=} (B3 \mid GC2 \mid \text{Panel1}) \setminus \{\text{depress3, undepress3, depressButton1, undepressButton1, depressButton2, undepressButton2}\}
\]

\[
= \text{depressButton1}_{p3} \cdot (B3 \mid \text{depressButton1}.GC2'')
\]

\[
+ \text{depressButton2}_{p3} \cdot \text{undepressButton1}.\text{depressButton2}.GC2''
\]

\[
+ \text{depressButton3}_{p3} \cdot \text{undepressButton1}.\text{depress3}.GC2''
\]

\[
+ \text{undepressButton}_{p3} \cdot \text{undepressButton1}.\text{GC2}
\]

\[
\mid \tau.\text{depressButton2}.\tau.\tau.(B1 \mid B2' \mid GC'')
\]

\[
+ \text{depressButton1}_{p3} \cdot \tau.(B3
\]

\[
\mid (\text{depressButton2}_{p3} \cdot \text{undepressButton1}.\text{depressButton2}.GC2''
\]

\[
+ \text{depressButton3}_{p3} \cdot \text{undepressButton1}.\text{depress3}.GC2''
\]

\[
+ \text{undepressButton}_{p3} \cdot \text{undepressButton1}.\text{GC2}
\]

\[
\mid \tau.\text{undepressButton1}.\tau.(B1 \mid B2 \mid GC)) + \cdots
\]

\[
= \text{depressButton1}_{p3} \cdot \tau.(B3
\]

\[
\mid \text{undepressButton1}.\text{depressButton2}_{p3}.(B3
\]

\[
\mid \tau.\text{depressButton2}.\tau.\tau.(B1 \mid B2' \mid GC'')
\]

\[
+ \text{depressButton1}_{p3} \cdot \tau.\text{depressButton2}_{p3}.(B3
\]

\[
\mid \text{undepressButton1}.\text{depress3}.GC2''. \mid \tau.\text{depressButton2}.\tau.\tau.(B1 \mid B2' \mid GC'')
\]

\[
+ \text{depressButton1}_{p3} \cdot \tau.\text{undepressButton1}_{p3}.(B3
\]

\[
\mid \text{undepressButton1}.\text{GC2} \cdot \tau.\text{depressButton2}.\tau.\tau.(B1 \mid B2' \mid GC'') + \cdots
\]
The three terms of the last step above cannot be expanded any further thereby denoting deadlock states. The deadlock arises because the outer group coordinator (that is, GC2) is ready to undepress button one but Panel1 is not ready to accept undepress actions. The reason behind this behavior is a combination of (i) the presence of non-determinism in the behavior of Panel1 and (ii) the inability of blocking the evolution of terms that give rise to uncoordinated behavior. Two terms in Panel1 start with action depressButton1. As a result when GC2 takes the depressButton1 action, it interacts with both of these terms. Interaction with \((\text{depressButton1} \cdot \tau \cdot \text{depressButton2} \cdot \tau \cdot \tau \cdot (B1 | B2' | GC''))\) generates the three deadlock terms above and interaction with \((\text{depressButton1} \cdot \tau \cdot \text{undepressButton1} \cdot \tau \cdot (B1 | B2 | GC))\) generates the terms displaying correct coordinated behavior (which have not been shown above). Note that the former interaction is not due to an incorrect design of the group coordinator. Instead it is due to the inability of the Expansion Law to restrict behavioral evolution based on the coordination needs of autonomous agents.

2.3. Hybrid group coordinators

Unlike a centralized group coordinator described above, a group coordinator that does not hide coordinated components and that allows components to share part of the coordination responsibility is called a hybrid group coordinator. Such group coordinators allow component agents to retain their autonomy by allowing them to interact with external agents and to make decisions about the availability of their own operations. In such a coordination scheme, the panel of buttons can be specified as follows:

\[
(C | B1 | B2) \backslash \{\text{button1Depressed}, \text{button1Undepressed}, \text{button2Depressed}, \text{button2Undepressed}\}, \text{ where}
\]

\[
B1 \overset{\text{def}}{=} \text{depress1} \cdot \text{button1Depressed} \cdot B1'
\]

\[
B1' \overset{\text{def}}{=} \text{undepress1} \cdot \text{button1Undepressed} \cdot B1
\]

\[
B2 \overset{\text{def}}{=} \text{depress2} \cdot \text{button2Depressed} \cdot B2'
\]

\[
B2' \overset{\text{def}}{=} \text{undepress2} \cdot \text{button2Undepressed} \cdot B2
\]

\[
C \overset{\text{def}}{=} \text{button1Depressed} \cdot C' \cdot \text{button2Depressed} \cdot C''
\]

\[
C' \overset{\text{def}}{=} \text{button2Depressed} \cdot \text{undepress1} \cdot C'' \cdot \text{button1Undepressed} \cdot C
\]

\[
C'' \overset{\text{def}}{=} \text{button1Depressed} \cdot \text{undepress2} \cdot C' \cdot \text{button2Undepressed} \cdot C
\]

Fig. 4 shows the hybrid group coordinator agent, the button agents, and the hidden and visible ports. Since the ports of the button agents are not hidden, they communicate directly with external agents. For example, button one may be depressed by communicating with it at port depress1. By not hiding the ports of the button agents, the group coordinator allows buttons to share part of its coordination responsibility, namely, operation scheduling: Each button decides when to schedule its own operations and controls the availability of its operations by making transitions between its states.
Although the button agents do not communicate among themselves, they engage in internal communications with the group coordinator agent in order to inform it about every communication they participate in. Such interactions are necessary in order to determine whether a communication with an external agent is consistent with the constraints of the group. For example, after being depressed, button agent $B1$ explicitly engages in a communication with the group coordinator agent $C$ at the output port $\text{button1Depressed}$ before progressing with its computations. This explicit communication with the central coordinator leads to complications when designing components since components must be designed with regard to their possible use in groups. If a component is not used in a group, then provisions must be made to capture its standalone behavior. An ideal approach would be to provide a formal abstraction that could transparently observe the operation scheduling decisions made by components thereby relieving them from engaging in such explicit internal communications.

The internal communications cause much more serious problems when the Expansion Law is used to expand the composition of the buttons and the coordinator by generating the following terms:

$$\cdots = \text{depress1.depress2.}((\text{button1Depressed}.C' + \text{button2Depressed}.C'')) | $$
$$\text{button1Depressed}.B1' | \text{button2Depressed}.B2') + \cdots$$
$$= \text{depress1.depress2.}.(C'' | \overline{\text{button1Depressed}.B1'} | B2') + \cdots$$
that represent the same incorrect coordination sequence that occurred in the case of explicit communication: when button two is depressed after button one, button two is undepressed instead of button one. The latter problem occurs because the depressing of button one is informed to the coordinator after the depressing of button two is informed. Terms corresponding to the correct sequence (informing the depressing of button one before informing the depressing of button two) are also generated along with the above incorrect sequence. The incorrect sequence is produced due to the generation of every possible action of an agent in every possible order by the Expansion Law. Some of the alternatives must be prevented from occurring since they represent incorrect coordination of the components.

Even if only those terms are considered in which the occurrence of events at the button agents are informed to the group coordinator in the correct order, the continuous visibility of the \( \text{depress1} \), \( \text{depress2} \), \( \text{undepress1} \), and \( \text{undepress2} \) ports to the external environment causes two problems. First, the following terms are generated:

\[
\cdots = \text{depress1}. \tau. \text{depress2}. \tau. (\text{undepress1}. C'' | \text{B1}' ) \\
\text{undepress1}. \text{button1 Undepressed } B1 | B2' ) + \cdots \\
= \text{depress1}. \tau. \text{depress2}. \tau. \text{undepress1}. (\text{undepress1}. C'' | \text{B1}' | \text{button2 Undepressed } B2 ) + \cdots
\]

that represent an incorrect coordination sequence. On depressing button two after button one, the coordinator's action of undepressing button one must be accepted by button one. Instead, in the above expression, button one decides not to interact with the coordinator and prepares to interact with the external environment thereby deadlocking the coordinator.

The second problem is that the following terms are generated:

\[
\cdots = \text{depress1}. \tau. \text{depress2}. \tau. (\text{undepress1}. C'' | \\
\text{undepress1}. \text{button1 Undepressed } B1 | B2' ) + \cdots \\
= \text{depress1}. \tau. \text{depress2}. \tau. \text{undepress1}. (C'' | \\
\text{undepress1}. \text{button1 Undepressed } B1 | B2' ) + \cdots
\]

that represent another incorrect coordination sequence in which the coordinator, instead of taking the coordinating action to undepress button one, decides to take it on the external environment thereby yielding an inconsistent state of the group. This happens due to the visibility of the \( \text{undepress1} \) port of the group coordinator to the external environment, as shown in Fig. 4.
2.4. Non-intrusive hybrid group coordinators

The uncoordinated behavior associated with the hybrid group coordinator specified in the last section arose due to (i) the inability to control the order of occurrence of internal actions and (ii) the inability to control the visibility of those ports that are both participating in a dialogue with the environment and are being observed by the coordinator agent. What is required is a formalism that allows the specification of non-intrusive, hybrid group coordinator agents.

A hybrid group coordinator will be termed non-intrusive if it: (i) transparently observes events at specific ports of the coordinated agents, (ii) transparently initiates coordinating actions at specific ports of the coordinated agents, (iii) allows a *dynamic control* over the visibility of ports of the coordinated agents, and (iv) composes with coordinated agents to yield a pruned state-space. Event observation will be transparent if (i) a coordinator does not have to intercept requests for service from clients and (ii) a coordinated component does not have to engage in explicit communications to inform the occurrence of observed events to the group coordinator. Coordinating actions will be transparent if a coordinated component does not have to engage in interactions to explicitly trigger, accept, or respond to such actions. As a result, a non-intrusive hybrid group coordinator will be invisible to both the clients of the group and the components of the group. But note that such a coordinator is not expected to be any less complicated either in its specification or in its coordinating actions than a centralized group coordinator. The advantages of using non-intrusive, hybrid group coordinators over centralized ones are the ability to provide a distributed specification of the coordination among autonomous agents (an inherently distributed problem) and the simpler object-based implementations of such group coordinators.

The group of the button agents coordinated by a non-intrusive, hybrid group coordinator may be captured as shown in Fig. 5. Note that the button agents have no explicit ports to communicate with the group coordinator, the ports using which the group coordinator takes coordinating actions are hidden from the external environment, and the coordinator does not hide or replicate the interfaces of the buttons thereby allowing clients to communicate with components directly.

A new formalism is required to capture non-intrusive, hybrid group coordinators because the explicit-communication mode of interaction among agents in process-calculi like CCS is inadequate to capture the implicit, transparent mode of interaction required by these coordinators. Such a formalism is proposed in the following section.

3. A calculus of coordinating environments

In this section, the Calculus of Coordinating Environments (CCE) is proposed. CCE enables a simple and direct specification of non-intrusive, hybrid group coordinator agents, called *Coordinating Environment agents* (CE agents). CE agents coordinate compositions of CCS agents. The coordinated CCS agents have the two following
Fig. 5. The non-intrusive, hybrid group coordinator agent, the button agents, and the hidden and visible ports.

properties: (i) they do not engage in any internal communications and (ii) they do not hide any of their ports explicitly.

The first property promotes the use of independently conceived agents that are unaware of the other participants in a group. The property also relieves a CE agent from the burden of dealing with $\tau$ actions that, as demonstrated in the last section, cause problems in coordinating groups. Instead, if two agents ever have to communicate, the CE agent implements that communication. The second property enables a CE agent to control dynamically the visibility of the ports of the CCS agents instead of permanently hiding them (as done by the restriction combinator of CCS).

Fig. 6 captures the purpose of a coordinating environment and a CE agent at a very abstract level. The behavior of the coordinating environment is embodied in the CE agent that is not visible to both the external environment and the coordinated agents. A CE agent controls the composition environment so that the external environment may not take actions on the agents in the composition environment that may result in inconsistent states of the group. Instead, the CE agent, depending on the collective state of the agents in the composition environment, selectively exposes ports of the CCS agents to the external environment. Such selective exposure of ports is achieved through observation: A port is exposed to the external environment only if the CE agent observes that port. In Fig. 6, only ports $a$ and $f$ are exposed to the external
environment in the current state of the CE agent (that reflects the current state of the group). The set of exposed ports may change in a subsequent state of the group.

A CE agent may take a coordinating action by communicating with any of the agents at any of their ports. These coordinating actions are internal communications that generateτ actions but the τ actions do not escape the boundary of the coordinating environment. The coordinating actions serve two important purposes: (i) they enable a CE agent to force component agents to change their states and (ii) they enable the replacement of direct communication among component agents by communications among the CE agent and the components.

3.1. The operational semantics of CCE

The operational semantics of CE agents is described by a labeled transition system (LTS). Let 𝒜 be a set of names (:hidden, ... range over 𝒜) and 𝒜 be the corresponding set of co-names (:hidden, ... range over 𝒜), as in CCS. Let 𝒟 = 𝒜 ∪ 𝒜 be the set of labels and Act_{CCS} = 𝒟 ∪ {τ} be the set of actions, as in CCS. Let 𝒪 be the set containing all CE expressions (C, D, ... range over 𝒪), 𝒩_{巧合} be the set of CE variables (X, Y, ... range over 𝒩), 𝒩_{巧合} be the set of CE constants (C_0, C_1, ... range over 𝒩_{巧合}), let ObsE_{CE} = 𝒟, and let CoordAct_{CE} = ObsE_{CE} ∪ {Ø}, where Ø ∉ Act_{CCS} is a distinguished no-action symbol. Then, the operational semantics of
CE agents is defined by \((\mathcal{E}, \text{ObsEv}_{\text{CE}} \times \text{CoordAct}_{\text{CE}}^*, \mapsto)\) where the transition relation \(\mapsto \subseteq (\mathcal{E}, \text{ObsEv}_{\text{CE}} \times \text{CoordAct}_{\text{CE}}^*, \mathcal{E})\). For \((C, (x_0, \bigcirc), C') \in \mapsto\), we write, \(C \xrightarrow{(x_0)} C'\) and for \((C, (x_0, x_1 \ldots x_n), C') \in \mapsto\), we write, \(C \xrightarrow{(x_0, x_1 \ldots x_n)} C'\).

The operational semantics described above highlights several important properties of both CE agents and the coordinated CCS agents. First, the two-tuple \((x_0, (x_1 \ldots x_n))\) partitions an action of a CE agent into two distinct steps: observation followed by a coordinating action sequence. In \((x_0, (x_1 \ldots x_n))\), the CE agent observes an action at port \(x_0\) of a component CCS agent and takes actions at its ports \(x_1\) through \(x_n\), in that sequence, that are directed to one or more component agents. Note that these actions must be taken in a specific sequence because different sequences may result in different final states of the group. When \(n = 0\), the coordinating action sequence is empty and is represented by the \(\bigcirc\) symbol. Thus, a CE agent must observe an event to progress with its computation but it may not take a coordinating action on making an observation.

The second important property is that there is a tight synchronization of the computation steps of a CE agent and those of the coordinated CCS agents. The synchronization ensures that when a CE agent takes a sequence of coordinating actions, the component agents are ready to engage in such communications with the CE agent. In the absence of such a synchronization, the CE agent would not be able to progress thereby deadlocking the group.

The third important property is that a CE agent does not observe a \(\tau\) action. Since the communication among any pair of complementary ports is represented by a \(\tau\), a CE agent cannot distinguish between different \(\tau\) actions. As a result, a CE agent coordinates only those CCS agents that do not communicate among themselves.

### 3.1.1. CE agent expressions

CE agent expressions are formed using the following grammar:

\[
C ::= \emptyset | X \mid (x_0, p) \triangleright C \mid C + C \mid C\{\Phi_1, \Phi_2\} \mid \text{fix}(X = C)
\]

where \(\emptyset\) is the no-action CE agent, \(p \in \text{CoordAct}_{\text{CE}}^*\), \(\triangleright\) is like the "\(\cdot\)" (prefix) combinator of CCS, "\(+\)" is the non-deterministic choice combinator of CCS, \(\Phi_1 : \text{ObsEv}_{\text{CE}} \rightarrow \text{ObsEv}_{\text{CE}}\) and \(\Phi_2 : \text{CoordAct}_{\text{CE}} \rightarrow \text{CoordAct}_{\text{CE}}\) are two renaming functions, "\{ \}\" is the renaming combinator like "\([\ ]\)" in CCS, and the recursion expression (denoted by \(\text{fix}\) which must be read as "the CE agent \(X\) such that \(X \overset{\text{def}}{=} C\)\") allows recursive definition of CE agents. Thus, CE agent expressions are obtained using the familiar CCS combinators. The only difference is that a CE agent is not a CCS agent and hence cannot execute actions until it is composed, in a special way, with CCS agents.

### 3.1.2. The transition rules

The transition rules for the CCE combinators are given below. The names \(\text{Act}, \text{Choice}, \text{Rel},\) and \(\text{Rec}\) imply that the rules are associated with \(\triangleright, +, \{\}\), and \(\text{fix}\),
respectively. Also, $p \in \text{CoordAct}_{CE}^*$.

<table>
<thead>
<tr>
<th>Act</th>
<th>$$(x_0, p) \xrightarrow{\phi_0} C$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td>$$\sum_{i \in I} C_i \xrightarrow{(x_0, p)} C'_i$$</td>
</tr>
<tr>
<td>Rel</td>
<td>$$C{\Phi_1, \Phi_2} \xrightarrow{(\Phi_1(x_0), \Phi_2(p))} C'{\Phi_1, \Phi_2}$$</td>
</tr>
<tr>
<td>Rec</td>
<td>$$C{\text{fix}(X = C)/X} \xrightarrow{(x_0, p)} C'$$</td>
</tr>
</tbody>
</table>

Note that $\Phi_2(x_1 \ldots x_n) = \Phi_2(x_1) \ldots \Phi_2(x_n)$. Also note that the relabeling operator enables the separate relabeling of both the ports at which observations are made and the ports at which coordinating actions are taken. This allows a CE agent to adjust its observations and actions to match any relabeling done to the ports of component agents.

### 3.1.3. The new composition combinator

The interaction between CE agents and CCS agents is captured by introducing a new composition combinator in CCS that augments CCS agent expressions as follows:

$$P ::= 0 \mid X \mid x_0.P \mid \overline{x_0}.P \mid \tau.P \mid \tau.P \mid P \mid P \mid P \mid P \mid P \mid \emptyset.P \mid \emptyset.P \mid \emptyset.P \mid f \mid \text{fix}(X = E) \mid C[P_1]$$

$$P_1 ::= 0 \mid X \mid x_0.P_1 \mid \overline{x_0}.P_1 \mid P_1 + P_1 \mid P_1 \mid P_1 \mid f \mid \text{fix}(X = E) \mid C[P_1]$$

Where $f : \text{Act}_{CCS} \rightarrow \text{Act}_{CCS}$ is a renaming function ($f(\tau) = \tau$), $C$ is a CE agent expression and $[ ]$ is the new composition combinator (note that the usual renaming combinator, $[ ]$, of CCS has been replaced by $\{ \}$) that composes a CE-agent expression with a CCS-agent expression to yield a CCS-agent expression.

Note that the CCS agents coordinated by a CE agent may not have the form $\tau.P$ and may not use the restriction combinator. The hiding of ports of coordinated agents from the external environment is achieved by the coordinating CE agent by not observing events at certain ports of the coordinated CCS agents. Unlike the permanent form of restriction achieved by the restriction combinator of CCS, CE agents allow a dynamic form of port restriction based on selective observation.
The two transition rules for \([\cdot]\) that capture the interaction among CE agents and CCS agents are given below.

\[
\text{Comp}_1 \quad \frac{P \xrightarrow{\pi_0} P_1 \xrightarrow{C \langle \pi_0, \gamma \rangle} D}{C[P] \xrightarrow{\pi_0} D[P_1]}
\]

\[
\text{Comp}_2 \quad \frac{P \xrightarrow{\pi_n} P_1 \xrightarrow{\pi_1} \ldots \xrightarrow{\pi_n} P_{n+1} \xrightarrow{C \langle \pi_0, \pi_1, \ldots, \pi_n \rangle} D}{C[P] \xrightarrow{\pi_0} D[P_{n+1}]}
\]

Rules \text{Comp}_1 and \text{Comp}_2 capture a step of computation by the composition of a CE agent and a CCS process-agent. In \text{Comp}_1, the CE agent \(C\) does not take any coordinating action on the CCS agent \(P\). Instead, the CE agent allows \(\pi_0\) to escape its boundary and be available for interaction with the external environment. As a result of this computation step, \(P\) changes its state to \(P_1\) and \(C\) changes its state to \(D\).

Rule \text{Comp}_2 captures the \(n\)-step coordinating action property of CE agents. In an atomic step, a CE agent may observe an action at port \(\pi_0\) and interact with the CCS agents \(P_1 - P_n\) at ports \(\pi_1\) through \(\pi_n\). The interactions with the CCS agents \(P_1 - P_n\) take place at complementary ports (for example, \(\pi_1\) and \(\bar{\pi}_1\)) but the resulting \(\tau\) actions are consumed by the CE agent and are not allowed to escape its boundary. In the CCS agent resulting from the \(n\)-step coordinating action, \(D[P_{n+1}]\), the CE agent has progressed by one step whereas the CCS agent \(P\) has progressed by \((n + 1)\) steps. The CE agent deliberately suppresses \(n\) states, \(P_1 - P_n\), of the coordinated CCS agent. The latter suppression of states prunes the actions at the nodes \(P_1 - P_n\) in the derivation tree of \(P\) and selects only the path from \(P\) to \(P_{n+1}\) that is relevant for the correct coordination of the group.

### 4. Example coordination problems

In this section, two example coordination problems are solved using CCE. The first is the panel of buttons problem that was introduced in Section 2. The second is a vending machine that is composed of a few autonomous components. Both examples illustrate the capability of CCE in specifying coordination problems and how the new transition rules may be applied to derive proper coordinated behavior of multi-agent groups.

#### 4.1. The panel of two buttons in CCE

Consider the two button agents defined below.

\[
B_1 \overset{\text{def}}{=} \text{depress1}.B_1'; B_1' \overset{\text{def}}{=} \text{undepress1}.B_1
\]

\[
B_2 \overset{\text{def}}{=} \text{depress2}.B_2'; B_2' \overset{\text{def}}{=} \text{undepress2}.B_2
\]
The button agents do not communicate either among themselves or with any group coordinator agent. Thus, they never engage in any internal communication in the composition environment. The same set of buttons were coordinated by a group coordinator in Section 2.2. This section shows how a non-intrusive, hybrid group coordinator coordinates these two button agents.

Fig. 7 shows a partial derivation tree for the composition of the two button agents defined above. In the tree, the leaf nodes are not expanded further because they already appear at other (expanded) internal nodes of the tree. Note that the state $B_1'|B_2'$ represents an incorrect state of the group in which both the buttons are depressed. Thus, the task of a group coordinator agent will be to suppress this state from being perceived by the external environment. Another important task of the group coordinator agent will be to block the transitions marked A and B. The path from the root of the tree to the leaf node marked Transition A represents a behavior of the group in which button two, being depressed after button one, is undepressed. Similarly, the path from the root of the tree to the leaf node marked Transition B represents a behavior of the group in which button one, being depressed after button two, is undepressed. According to the behavioral constraint imposed on the group, these two paths lead to inconsistent states of the group and hence, must be blocked.

Consider the CE agent defined below for coordinating the group of two button agents.

\[
\text{noneDepressed} \overset{\text{def}}{=} (\text{depress1}, \bigcirc) \triangleright \text{button1Depressed} \\
+ (\text{depress2}, \bigcirc) \triangleright \text{button2Depressed} \\
\text{button1Depressed} \overset{\text{def}}{=} (\text{depress2, undepress1}) \triangleright \text{button2Depressed}
\]
The states of the CE agent model the different consistent states of the button agents it coordinates. The CE agent starts in the state `noneDepressed` in which none of the buttons are depressed. From that state, it has the option of observing an action either at the `depress1` port or at the `depress2` port. This observation amounts to the selective exposing of those two ports so that the external environment may communicate with them. No coordinating actions are associated with the latter observations. The state `button1Depressed` captures the relative configuration of the buttons in which button one is depressed and button two is undepressed. From this state, the CE agent either observes an action at the `undepress1` port or observes an action at the `depress2` port. In the former case, the CE agent makes a transition to the `noneDepressed` state without taking any coordinating action and in the latter case, it makes a transition to the `button2Depressed` state after taking an output action at the port `undepress1`. The behavior from the state `button2Depressed` is analogous to the behavior of the CE agent from the state `button1Depressed`.

The behavior of the composition of the buttons with the CE agent is shown below.

\[
\begin{align*}
\text{noneDepressed} & \lor \text{button2Depressed} \\
& = \text{noneDepressed} \lor (\text{depress1, undepress2}) \lor \text{button1Depressed} \\
& + (\text{undepress2, O}) \lor \text{noneDepressed}
\end{align*}
\]

Note that unlike the previous problem of incorrect coordination, in the above expansion, when button two is depressed after button one, button one is undepressed, button two remains depressed, and the CE agent is in a state in which it may either observe the undepressing of button two or the depressing of button one.

The composed behavior of the CE agent and the two button agents may be better understood by superposing their derivation trees, as shown in Fig.8. The state \((B1 | B2)\)
of the buttons corresponds to the state noneDepressed of the CE agent. From the latter state, the composition may evolve by either an input action at port depress1 or an input action at port depress2. The state \((B1' | B2)\) of the buttons corresponds to the state button1Depressed of the CE agent. From the latter state, the composition may evolve by either an input action at port undepress1 or an input action at port depress2 followed by an internal action (that results due to a communication between the complementary ports undepress1 and undepress1). The latter, one-step coordinating action takes the group directly to the state \((B1 | B2')\) that corresponds to the state button2Depressed of the CE agent. Three important observations must be made: (i) the one-step coordinating action following the action at port depress2 is not perceived by the external environment, (ii) the state \((B1' | B2')\) of the buttons is rendered a transient, internal state that is not perceived by the external environment, and (iii) the inconsistent transition from the state \((B1' | B2')\) through an input action at port undepress2 is blocked by the CE agent. A similar explanation applies to the transitions in the right subtree of the root node in Fig. 8.

### 4.2. Specifying a vending machine in CCE

Consider the two slot agents, \(S1\) and \(S2\), and a coin acceptor agent, \(CA\), defined below.

\[
S1 \overset{def}{=} open1 . amount1 . (\overline{\text{dispense}1 . S1 + \overline{\text{fail}1 . S1})}
\]

\[
S2 \overset{def}{=} open2 . amount2 . (\overline{\text{dispense}2 . S2 + \overline{\text{fail}2 . S2})}
\]
CA \overset{\text{def}}{=} \text{insert}.CA' \\
CA' \overset{\text{def}}{=} \text{insert}.CA' + \text{refund}.CA + \text{insertedAmount}.CA' + \text{refundExcess}.CA

Slot S1 starts with an input action at port open1. Next, it engages in another input action at port amount1 (which models the transfer of an amount value to the slot). After the latter communication, the slot may perform an output action at port disable1 (which models the successful extraction of an item) or may perform an output action at port fail1 (which models the failure of a slot to dispense an item). Slot S2 behaves analogously. The coin acceptor agent CA starts with an input action at port insert and makes a transition to state CA'. In the latter state, it may engage in one of four actions. First, it may engage in an input action at port insert and remain in state CA' (which models the insertion of multiple coins). Second, it may engage in an input action at port refund and transit to state CA (which models the extraction of all inserted coins). Third, it may engage in an output action at port insertedAmount and remain in state CA' (which models reading the inserted amount from the coin acceptor). Fourth, it may engage in an input action at port refundExcess and transit to state CA (which models the refunding of the excess amount inserted).

The role of a non-intrusive, hybrid group coordinator in coordinating the composition of the two slot agents and the coin acceptor agent is to elicit from them the behavior of a vending machine. A quiescent vending machine is triggered by inserting one or more coins after which either a request to refund all the coins may be made or a request to open one of the slots may be made. If a request to open a slot is made, then the group coordinator must obtain the inserted amount from the coin acceptor and transfer it to that slot. This transfer illustrates how the group coordinator relieves components from engaging in explicit internal communications. Then, if the coordinator observes the dispensing of an item by the slot, it refunds the excess amount inserted and returns to its initial state. Otherwise, if it observes the failure to dispense an item by the slot, it returns to a state in which it allows requests for either further coin insertions, refunding the inserted amount, or opening a slot.

The vending machine CE agent is defined below. Note, to increase readability, a semicolon separates two consecutive actions in a sequence of coordinating actions of the CE agent.

\[
\begin{align*}
\text{waitForCoins} & \overset{\text{def}}{=} (\text{insert}, \emptyset) \triangleright \text{processRequests} \\
\text{processRequests} & \overset{\text{def}}{=} (\text{insert}, \emptyset) \triangleright \text{processRequests} \\
& + (\text{refund}, \emptyset) \triangleright \text{waitForCoins} \\
& + (\text{open1}, \text{insertedAmount}; \text{amount1}) \triangleright \text{slot1Request} \\
& + (\text{open2}, \text{insertedAmount}; \text{amount2}) \triangleright \text{slot2Request} \\
\text{slot1Request} & \overset{\text{def}}{=} (\text{dispense1}, \text{refundExcess}) \triangleright \text{waitForCoins}
\end{align*}
\]
$+(\text{fail1}, \circ) \triangleright \text{processRequests}$

$\text{slot2Request} \overset{\text{def}}{=} (\text{dispense2}, \text{refundExcess}) \triangleright \text{waitForCoins}$

$+(\text{fail2}, \circ) \triangleright \text{processRequests}$

The behavior of the composition of the slot and coin acceptor agents with the CE agent is shown below.

\[
\text{waitForCoins}[S_1 | S_2 | CA]
= \text{insert . processRequests}[S_1 | S_2 | CA']
\]

\[
= \text{insert . insert . processRequests}[S_1 | S_2 | CA']
= \text{insert . insert . processRequests}[S_1 | S_2 | CA']
+ \text{insert . refund . waitForCoins}[S_1 | S_2 | CA]
+ \text{insert . open1 . slot1Request}[(\text{dispense1}.S_1 + \text{fail1}.S_1)| S_2 | CA']
+ \text{insert . open2 . slot2Request}[S_1 | (\text{dispense2}.S_2 + \text{fail2}.S_2) | CA']
\]

\[
= \text{insert . insert . processRequests}[S_1 | S_2 | CA']
+ \text{insert . refund . waitForCoins}[S_1 | S_2 | CA]
+ \text{insert . open1 . dispense1 . waitForCoins}[S_1 | S_2 | CA]
+ \text{insert . open1 . fail1 . processRequests}[S_1 | S_2 | CA']
+ \text{insert . open2 . dispense2 . waitForCoins}[S_1 | S_2 | CA]
+ \text{insert . open2 . fail2 . processRequests}[S_1 | S_2 | CA']
\]

The state $(S_1 | S_2 | CA)$ corresponds to the state $\text{waitForCoins}$ of the CE agent. From the latter state, the composition may evolve by only an input action at port $\text{insert}$. The state $(S_1 | S_2 | CA')$ corresponds to the state $\text{processRequests}$ of the CE agent. From the latter state, the composition may evolve by any one of four input actions at the ports $\text{insert}$, $\text{open1}$, $\text{open2}$, and $\text{refund}$. The actions at ports $\text{insert}$ and $\text{refund}$ takes the composition to the already expanded states $(S_1 | S_2 | CA')$ and $(S_1 | S_2 | CA)$, respectively. The actions at ports $\text{open1}$ and $\text{open2}$ result in a sequence of two atomic, unobservable, internal coordinating actions before resulting in the states $((\text{dispense1}.S_1 + \text{fail1}.S_1)| S_2 | CA')$ and $(S_1 | (\text{dispense2}.S_2 + \text{fail2}.S_2) | CA')$, respectively. The former state corresponds to the state $\text{slot1Request}$ of the CE agent and the latter state corresponds to the state $\text{slot2Request}$ of the CE agent. From $((\text{dispense1}.S_1 + \text{fail1}.S_1)| S_2 | CA')$, the composition may evolve by either an output action at port $\text{dispense1}$ or an output action at port $\text{fail1}$. The former event causes an unobservable, internal coordinating action before resulting in the already expanded state $(S_1 | S_2 | CA)$ and the latter event results in the already expanded state $(S_1 | S_2 | CA')$. 
The behavior from \((S1 | (\text{dispense2}.S2 + \text{fail2}.S2) | CA')\) is analogous to that from \(((\text{dispense1}.S1 + \text{fail1}.S1) | S2 | CA')\).

5. Hierarchical composition of CE agents

The transition rules \(\text{Comp1}\) and \(\text{Comp2}\) for the new composition combinator of CCE did not reveal the structure of the agent \(P\) in the composition \(C[P]\). The agent \(P\) in \(C[P]\) can have multiple CE agents as its components thereby giving rise to hierarchically composed CE agents. In the composition \(C_1[C_2[P]]\), for example, CE agents \(C_1\) and \(C_2\) have been hierarchically composed. In that expression, \(P\) and \(C_2[P]\) are CCS process agents, \(C_2\) is the inner CE agent, and \(C_1\) is the outer CE agent. Computation in such a composition progresses inside-out. Actions at ports of \(P\) must be observed by \(C_2\) first so that they are exposed to be observed by \(C_1\) next. Any actions taken by \(C_2\) on \(P\) occurs first and independent of any intervention of \(C_1\). Only after \(C_2\)'s actions on \(P\) are over, \(C_1\) may take actions on the composition \(C_2[P]\).

An example of the hierarchical composition of CE agents is shown below using a three-button panel. Unlike the centralized group coordinator that could not reuse the specification of the two-button centralized group coordinator, the CE agent for the three-button panel will reuse the CE agent for the two-button panel entirely.

Consider the addition of a third button agent to the two-button panel group of Section 4.1 to form a three-button panel in which the coordination constraint remains the same. The CE agent that coordinates the composition of the CE agent for the two-button panel and the third button agent is specified as follows:

\[
\text{noneDepressed}_{P_1} \overset{\text{def}}{=} (\text{depress1}, \bigodot) \triangleright \text{button1Depressed}_{P_1}
\]
\[
+ \ (\text{depress2}, \bigodot) \triangleright \text{button2Depressed}_{P_1}
\]
\[
+ \ (\text{depress3}, \bigodot) \triangleright \text{button3Depressed}_{P_1}
\]

\[
\text{button1Depressed}_{P_1} \overset{\text{def}}{=} (\text{depress2}, \bigodot) \triangleright \text{button2Depressed}_{P_1}
\]
\[
+ \ (\text{depress3}, \text{undepress1}) \triangleright \text{button3Depressed}_{P_1}
\]
\[
+ \ (\text{undepress1}, \bigodot) \triangleright \text{noneDepressed}_{P_1}
\]

\[
\text{button2Depressed}_{P_1} \overset{\text{def}}{=} (\text{depress1}, \bigodot) \triangleright \text{button1Depressed}_{P_1}
\]
\[
+ \ (\text{depress3}, \text{undepress2}) \triangleright \text{button3Depressed}_{P_1}
\]
\[
+ \ (\text{undepress2}, \bigodot) \triangleright \text{noneDepressed}_{P_1}
\]

\[
\text{button3Depressed}_{P_1} \overset{\text{def}}{=} (\text{depress1}, \text{undepress3}) \triangleright \text{button1Depressed}_{P_1}
\]
\[
+ \ (\text{depress2}, \text{undepress3}) \triangleright \text{button2Depressed}_{P_1}
\]
\[
+ \ (\text{undepress3}, \bigodot) \triangleright \text{noneDepressed}_{P_1}
\]

Note that in the above, the states of the new CE agent are subscripted by \(P_1\) (for Panel1, assuming the two-button panel to be Panel0). Also note that on observing the
sequence \textit{depress1.depress2}, the above CE agent does not have to take the coordinating action \textit{undepress1} because that action is taken by the inner CE agent. Similarly, on observing the sequence \textit{depress2.depress1}, the CE agent does not have to take the coordinating action \textit{undepress2} because that action is taken by the inner CE agent as well. As a result, the outer CE agent has reduced coordination responsibility due to reusing the two-button panel as a component of the three-button panel.

Consider the behavior of the three-button panel in which the third button is composed with the asynchronously evolved behavior of the two-button panel, as shown below.

\[
\text{noneDepressed}_p_1[B3 | \text{noneDepressed}_p_1[B1 | B2]] = \\
\text{noneDepressed}_p_1[B3 | (\text{depress1.undepress1.noneDepressed}_p_1[B1 | B2] + \text{depress1.depress2.button2Depressed}_p_1[B1 | B2'] + \text{depress2.undepress2.noneDepressed}_p_1[B1 | B2] + \text{depress2.depress1.button1Depressed}_p_1[B1' | B2])] = \\
\text{depress1.button1Depressed}_p_1[B3 | \text{undepress1.noneDepressed}_p_1[B1 | B2]] (\text{term 1}) + \\
\text{depress1.button1Depressed}_p_1[B3 | \text{depress2.button2Depressed}_p_1[B1 | B2']]] (\text{term 2}) + \\
\text{depress2.button2Depressed}_p_1[B3 | \text{undepress2.noneDepressed}_p_1[B1 | B2]] (\text{term 3}) + \\
\text{depress2.button2Depressed}_p_1[B3 | \text{depress1.button1Depressed}_p_1[B1' | B2]] (\text{term 4}) + \\
\text{depress3.button3Depressed}_p_1[\text{undepress3.B3}] = \\
\]

The terms in the above expression have been marked for the purpose of reference. In \text{term 1}, \text{button1Depressed}_p_1 can observe an action at port \textit{depress2} but no such port is available for observation. As a result, a term beginning with the sequence \textit{depress1.depress2} is not generated by \text{term 1}. Similarly, in \text{term 2}, \text{button1Depressed}_p_1 can observe an action at port \textit{undepress1} but no such port is available for observation and so a term beginning with the sequence \textit{depress1.undepress1} is not generated by \text{term 2}. But the most important observation about \text{term 2} is that although \text{button1Depressed}_p_1 can observe an action at port \textit{depress3} and that port is available for observation, a term beginning with the sequence \textit{depress1.depress3} is not generated by \text{term 2}. That is because of the atomic nature of the "observation action" interaction.
between CE agents and coordinated components. Although \( \text{button1Depressed}_{p1} \) can observe an action at port \( \text{depress3} \) and that port is available for observation, it cannot take the \( \text{undepress1} \) coordinating action thereby preventing the \((\text{depress3, undepress1})\) observation–action step from occurring. Terms 3 and 4 also prevent some uncoordinated behavior sequences from manifesting. Thus, the atomic "observation–action" interaction property along with the fact that unavailable ports are not observed enable CE agents to prune the state space so that uncoordinated behavior sequences are not generated.

Continuing with the expansion of the behavior of the three-button panel, we get the following:

\[
= \text{depress1.undepress1.noneDepressed}_{p1}[B3|\text{noneDepressed}[B1|B2]] \\
+ \text{depress1.depress3.button3Depressed}_{p1} \\
\hspace{1cm} [\text{undepress3.B3|noneDepressed[B1|B2]]] \\
+ \text{depress1.depress2.button2Depressed}_{p1}[B3|\text{button2Depressed}[B1|B2')] \\
+ \text{depress2.undepress2.noneDepressed}_{p1}[B3|\text{noneDepressed}[B1|B2)] \\
+ \text{depress2.depress3.button3Depressed}_{p1} \\
\hspace{1cm} [\text{undepress3.B3|noneDepressed[B1|B2]]] \\
+ \text{depress2.depress1.button1Depressed}_{p1}[B3|\text{button1Depressed}[B1'|B2]] \\
+ \text{depress3.undepress3.noneDepressed}_{p1}[B3] \\
\hspace{1cm} [(\text{depress1.undepress1.noneDepressed}[B1|B2)] \\
+ \text{depress1.depress2.button2Depressed}[B1|B2'] \\
+ \text{depress2.undepress2.noneDepressed}[B1|B2] \\
+ \text{depress2.depress1.button1Depressed}[B1'|B2') \\
+ \text{depress3.depress1.button1Depressed}_{p1} \\
\hspace{1cm} [B3|\text{undepress1.noneDepressed}[B1|B2)] \\
+ \text{depress3.depress1.button1Depressed}_{p1} \\
\hspace{1cm} [B3|\text{depress2.button2Depressed}[B1|B2')] \\
+ \text{depress3.depress2.button2Depressed}_{p1} \\
\hspace{1cm} [B3|\text{undepress2.noneDepressed}[B1|B2)] \\
+ \text{depress3.depress2.button2Depressed}_{p1} \\
\hspace{1cm} [B3|\text{depress1.button1Depressed}[B1'|B2])
\]

Two terms in the above need further expansion. The first one is shown below.

\[
\text{depress1.depress2.button2Depressed}_{p1}[B3|\text{button2Depressed}[B1|B2']) \\
= \text{depress1.depress2.button2Depressed}_{p1}[\text{depress3 B3'}
\]
\[(\text{depress1.button1Depressed}[B1' \mid B2] + \text{undepress2.noneDepressed}[B1 \mid B2])\]
\[= \text{depress1.depress2.depress3.button3Depressed}_p[B3' \mid \text{noneDepressed}[B1 \mid B2]]\]
\[+ \text{depress1.depress2.depress1.button1Depressed}_p[\text{depress3.B3'}\]
\[\text{button1Depressed}[B1' \mid B2]\]
\[+ \text{depress1.depress2.undepress2.noneDepressed}_p[\text{depress3.B3'}\]
\[\text{noneDepressed}[B1 \mid B2]\]

The second expansion is shown below.

\[\text{depress2.depress1.button1Depressed}_p[\text{depress3.B3'}\]
\[\text{button1Depressed}[B1' \mid B2]\]
\[= \text{depress2.depress1.button1Depressed}_p[\text{depress3.B3'}\]
\[\text{depress2.button1Depressed}[B1 \mid B2' + \text{undepress1.noneDepressed}[B1 \mid B2]]\]
\[= \text{depress2.depress1.depress3.button3Depressed}_p[B3' \mid \text{noneDepressed}[B1 \mid B2]]\]
\[+ \text{depress2.depress1.depress2.button2Depressed}_p[\text{depress3.B3'}\]
\[\text{button2Depressed}[B1 \mid B2']\]
\[+ \text{depress2.depress1.undepress1.noneDepressed}_p[\text{depress3.B3'}\]
\[\text{noneDepressed}[B1 \mid B2]\]

6. Conclusions and future work

This paper addresses the issue of designing high-level formal abstractions for coordinating agents specified in CCS. The Calculus of Coordinating Environments (CCE) is proposed that extends CCS by providing a new composition combinator and a special agent called a Coordinating Environment (CE) agent. A CE agent coordinates the composition of multiple CCS agents by observing actions at specific ports of the coordinated agents and taking coordinating actions on them. A CE agent has a simpler object-based realization compared to a centralized group coordinator.

In its current form, CCE cannot deal with the \(\tau\) action of CCS in an elegant manner. The basic premise of the observation–action computing scheme of a CE agent is that it observes an action at a named port of a coordinated agent. Since the label \(\tau\) has no port names associated with it (any two complementary ports always produce the unique name \(\tau\)), a CE agent cannot guarantee that it will prevent uncoordinated action sequences from being produced if \(\tau\) were added to its observable events list. As a result of this, the standard notions of bisimilarity of CCS will not be applicable to CCE. For example, in CCS, \(x.P \approx x.\tau.P\), but \(C[x.P] \approx C[x.\tau.P]\) is false in CCE for a CE agent \(C\). Thus, the notion of bisimilarity of CCS must be extended to handle the equivalence of CCS-CCE agent compositions.
A working model for coordinating concurrent objects that is based on the formal coordination abstraction proposed in CCE [19] has been proposed. Among ongoing investigations are the formal properties of CCE and the applicability of formal specification schemes based on Communicating Sequential Processes [10] in modeling non-intrusive, hybrid group coordinators.

CCE is considered to be the first step towards a much more expressive calculus that will provide an integrated approach for specifying concurrency, communication, and coordination. An approach of future work would be to consider the \textit{pi-calculus} [18] for modeling coordination among agents whose interconnection topology is dynamic. Another interesting avenue to pursue would be to utilize the Asynchronous CCS [12] to model coordination among agents that engage in asynchronous communication using messages. Such a calculus would capture more faithfully, the coordination among concurrently executing objects that are the cornerstone of the COOP paradigm.

\textbf{Acknowledgements}

The comments of two anonymous referees and the participants of COORDINATION ’96 are gratefully acknowledged.

\textbf{References}