Composable Semantic Models for Actor Theories

Carolyn L. Talcott
Stanford University
clt@sail.stanford.edu

Abstract

We define three semantic models for actor computation starting with a generalization to open systems of Clinger's event diagram model, and forming two abstractions: interaction diagrams and interaction paths. An algebra is defined on each semantic domain with operations for parallel composition, hiding of internal actors, and renaming. We use these models to provide semantics for descriptions of actor components based on actor theories and show that the semantics is a component algebra homomorphism.

1 Introduction

We are interested in developing a semantic foundation that supports specifying, composing, and reasoning about components of open distributed systems. Open distributed systems are highly dynamic: new components may be added; old components may be replaced or removed; interconnections change over time; and a component has little if any control over its environment. We take the actor model of computation [Hew77, Agh86, Agh90] as our starting point. Actors are independent computational agents that interact solely via asynchronous message passing. An actor can create other actors; send messages; and modify its own local state. An actor can only effect the local state of other actors by sending them messages, and it can only send messages to its acquaintances—names of actors it was given upon creation, it received in a message or actors it created. Actor semantics requires computations to be fair—any message sent will eventually be delivered. The actor model provides the basic elements for open distributed computation: encapsulation of state; independent concurrent units of computation; interaction; and dynamic creation and interconnection. System components are groups of actors provided with an interface for interaction with their environment. An interface, (ρ, χ), consists of a set of receptionists, ρ, that specifies which actors of the system are visible to the environment, and a set of externals, χ, that specifies names of actors in the environment known to actors of the system. Only receptionists can receive messages from outside the system and the system can only send messages out to externals. However both sets may grow as the system evolves and interacts with its environment.

Components provide for scalability and composability. In order to have scalable and composable semantics we need to have a wide range of composable models that

express different properties at different levels of abstraction. We consider an algebra of components with three operations in addition to be basic operation of forming a component from a collection of actors and messages. Parallel composition, $C_0 \parallel C_1$ is defined for components C_0, C_1 that satisfy appropriate composability criteria – such as having disjoint sets of actors. Restriction, $C \lceil \rho$, where ρ is a set of actor names, hides the receptionists of C that are not in ρ. Renaming, $\widehat{\alpha}(C)$, extends bijections, α, on actor names homomorphically to bijections on components. The component algebra satisfies the following.

Component Algebra Laws For composable C_0, C_1, C_2

(1) Parallel composition is associative and commutative with identity the empty component `mt`.

$$(C_0 \parallel C_1) \parallel C_2 = C_0 \parallel (C_1 \parallel C_2)$$

$$C_0 \parallel C_1 = C_1 \parallel C_0$$

$$C \parallel \mathtt{mt} = C$$

(2) Renaming commutes with parallel composition: $\widehat{\alpha}(C_0 \parallel C_1) = \widehat{\alpha}(C_0) \parallel \widehat{\alpha}(C_1)$.

(3) Renaming commutes with restriction: $\widehat{\alpha}(C \lceil \rho) = (\widehat{\alpha}(C)) \lceil \widehat{\alpha}(\rho)$.

Our semantic domains are subsets, S, of some set $\mathcal{S}$ of elements (computations, event diagrams, ...) representing possible interactions of the system at different levels of abstraction. Elements of these semantic sets have a common interface: $S \subset \mathcal{S}(\rho, \chi)$. We are interested in semantic domains that can be given an algebraic structure satisfying the component algebra laws. Composability of semantic sets is simply disjointness of receptionists. Also of interest are maps between semantic domains that preserve the algebraic structure. Such maps allow one to relate specifications based on different models and to transfer results obtained in one domain to another, as will be seen below. An actor component morphism from $\mathcal{S}$ to $\mathcal{S}'$ is a map φ sending subsets of $\mathcal{S}(\rho, \chi)$ to subsets of $\mathcal{S}'(\rho, \chi)$ such that (**Morphism[φ]**) holds.

Morphism[φ]

(1) if ρ_0 and ρ_1 are disjoint and $S_j \subseteq \mathcal{S}(\rho_j, \chi_j)$ for $j < 2$, then $\varphi(S_0 \parallel S_1) = \varphi(S_0) \parallel \varphi(S_1)$

(2) if $S \subseteq \mathcal{S}(\rho, \chi)$ and ρ' is a set of actor names, then $(S \lceil \rho') = (\varphi(S)) \lceil \rho'$

(3) if $S \subseteq \mathcal{S}(\rho, \chi)$ and α is a bijection on actor names, then $\varphi(\widehat{\alpha}(S)) = \widehat{\alpha}(\varphi(S))$

To describe actor system components at a syntactic level we use actor theories [Tal96b, Tal96a, MT97]. The notion of actor theory provides an axiomatic characterization of the essential features of actor languages. It reflects the view that an actor's behavior should be specified in terms of the messages sent and the actors created in response to receipt of a message. Associated to each actor theory is a set of configurations that describe actor system components. There is an algebra of configurations satisfying the component algebra laws, and associated to each configuration is a set of computations.

A semantics, $\mathcal{S}$, maps a configuration, K, with interface, (ρ, χ), to a subset of $\mathcal{S}(\rho, \chi)$. We say that $\mathcal{S}$ is compositional if (**Composable[$\mathcal{S}$]**) holds:

Composable[$\mathcal{S}$]

(1) if K_0 and K_1 are composable configurations, then
$\mathcal{S}(K_0) \parallel \mathcal{S}(K_1) = \mathcal{S}(K_0 \parallel K_1)$

(2) $\mathcal{S}(K \lceil \rho) = \mathcal{S}(K) \lceil \rho$ for any configuration K and set of actor names, ρ.

(3) $\mathcal{S}(\widehat{\alpha}(K)) = \widehat{\alpha}(\mathcal{S}(K))$ for any configuration K and any renaming α.

In [Tal96a] we studied a simple operational model and defined a compositional semantics based on sets of computations. In [Tal96b] we introduced the notion of interaction semantics, based on sets of (possibly infinite) sequences of interactions of a component with its environment that also yields a compositional semantics. In this paper we study two additional semantic models of actor computation and their relation to computation and interaction semantics: open event diagrams (OED); and interaction diagrams (ID).

Open event diagrams generalize the event diagrams introduced by Greif [Gre75] and formalized by Clinger [Cli81] to the open systems setting, making interactions with the environment explicit. Event diagrams capture the view that what is important about an actor is the messages sent and actors created in response to a message delivery. An event diagram specifies a set of realized events—messages that have been delivered, and a set of pending events—messages that have been sent, but not yet delivered. It specifies a partial order, the *activation order*, relating realized events to the delivery of messages sent in response. An event diagram also specifies the set of actors internal to the system, and for each internal actor a total order, the *arrival order*, on the events that represent deliveries to that actor. The transitive closure of the activation and arrival orders is called the *combined order* on events. The fairness requirement is expressed by the (**finite delay**) axiom which says that if the set of realized events is infinite, then the set of pending events must be empty. Event diagrams that have no pending events are said to be *complete*. Open event diagrams differ from those considered by Clinger in that messages may arrive from the environment and may be delivered to the environment. Event diagrams are abstractions of actor theory computations in two ways: they omit explicit representation of individual actor states; and they avoid sequentializing the events (picking a global time). On the other hand, one of the main results of Clinger [Cli81] is that each event diagram has at least one global time that respects the combined order on events and that the existence of a global time is equivalent to a version of the Baker-Hewitt [BH77] ordering laws. Global times are very convenient for defining composition of diagrams among other things.

Interaction diagrams are the result of hiding the internal events of complete event diagrams. What remains are: the input events—arrival of messages from the environment; the output events—delivery of messages to the environment; and the ordering relation on input and output events obtained by restricting the combined order. The interaction ordering contains information about output events that occur in response to input events, history dependence of output events on a group of input events, and dependencies involving newly created actors.

The interaction semantics of [Tal96b] is reformulated here in terms of *interaction paths* corresponding to the possible global times compatible with interaction diagrams. Interaction paths ignore ordering information that has no semantic consequences and is the most abstract of the semantics studied here.

Open event diagrams model both stages of system activity (when there are pending events) and complete computations (no pending events). Following Clinger, the notion of stages of a computation is formalized by defining a partial ordering, $\sqsubseteq$, on open event diagrams. Each event diagram is the limit (least upper bound) of a chain of finite event diagrams. Interaction diagrams and interaction paths are abstractions of complete computations. They each provide a basis for defining component equivalence and lead to rich and useful equational theories. Interaction diagrams retain more information about the ordering of internal events, thus allowing easier expression of history dependence. However, some of the ordering information is not relevant and interaction paths provide a way of forgetting this information.

In this paper, the open event diagram, interaction diagram, and interaction path models of actor computation are defined. A component algebra structure is defined on each of the semantic domains. Mappings from event diagrams to interaction diagrams and from interaction diagrams to interaction paths are defined and shown to preserve the algebraic structure. Semantic maps associating to actor theory configurations sets of event diagrams, complete event diagrams, interaction diagrams, and interaction paths are defined. These maps give compositional semantics. To establish this theorem a morphism from computations to event diagrams is defined, reducing the problem to the compositionality of the semantics base on sets of computations.

Plan

In §2 actor computation and the semantic models are introduced informally using a series of simple examples. In §3 notation relating to events that is used through out the remainder of the paper is developed. In §4 open event diagrams and the component algebra on event diagrams are defined. In §5 interaction diagrams and their component algebra are defined as well as a morphism from event diagrams to interaction diagrams. In §6 the interaction path semantics is defined and a morphism from interaction diagrams to interaction paths is defined. In §7 actor theories and their computational semantics is summarized. In §8 the semantic maps giving event diagram, interaction diagram, and interaction path semantics for configurations are defined and shown to be composable. §9 contains brief summary and discussion of future work.

Notation.

We conclude this section with a summary of our notation conventions. We use the usual notation for set membership and function application. $\mathbf{N}$ is the set of natural numbers. Let Y, Y_0, Y_1 be sets. We specify meta-variable conventions in the form: let y range over Y, which should be read as: the meta-variable y and decorated variants such as y', y_0, ..., range over the set Y. Y^* is the set of finite sequences of elements of Y, and $\bar{y} = [y_1, \ldots, y_n]$ is the sequence of length $\mathrm{Len}(\bar{y}) = n$ with ith element y_i. $u * v$ denotes the concatenation of the sequences u and v. $\mathcal{P}[Y]$ is the set of subsets of Y and

$\mathcal{P}_\omega[Y]$ is the set of finite subsets of Y. For $y \in Y$, $Y_0 \in \mathcal{P}[Y]$, we let y stand for the singleton $\{y\}$ when used in a context where a set is expected and write Y_0 , y or y , Y_0 for the addition of y to Y_0, i.e. $Y_0 \cup \{y\}$. $[Y_0 \to Y_1]$ is the set of total functions with domain Y_0 and range contained in Y_1. $[Y_0 \xrightarrow{\text{inj}} Y_1]$ is the subset of injective functions. $[Y_0 \overset{\circ}{\to} Y_1]$ is the set of partial functions, f, with domain contained in Y_0 and range contained in Y_1. We write $\mathrm{Dom}(f)$ for the domain of a function, f, and $\mathrm{Rng}(f)$ for its range. $f\lceil Y$ is the restriction of f to the set Y and f^{-1} is the inverse of f, with domain $\mathrm{Rng}(f)$ and range $\mathcal{P}[\mathrm{Dom}(f)]$. For I some index set, we write $\{y_i \mapsto z_i \mid i \in I\}$ for the function, f, with domain $\{y_i \mid i \in I\}$ such that $f(y_i) = z_i$ for $i \in I$. Viewing functions and relations as sets we obtain extensions by forming the union, where in the case of functions we require that the domains be disjoint. `mtmap` is the function with empty domain and range, and `mtrel` is the empty relation.

2 Actor Computation Informally

In this section we introduce actor computation and the semantic models of actor systems to be studied through a series of simple examples.

2.1 Ticker

A ticker is an actor the forever *ticks*, incrementing its notion of the current time. This is modeled by a `tick` message. When the ticker receives a `tick` message it sends itself another tick message. Thus time always passes. When it receives a request with a specified customer (reply address) it replies to that customer with its current time—the number of ticks it has received.

2.1.1 A Ticker Event Diagram

Figure 1 shows an event diagram (one of many possible) for a ticker actor named t, in which two messages of the form $t \triangleleft c$ arrive from outside the system. $t \triangleleft c$ is a message with target t and contents c, specifying the customer. In addition there is a message, $t \triangleleft \texttt{tick}$, that is initially present. The vertical line labeled t is the time-line for actor t. It gives the arrival order of messages for t. The delivery of a message to t is indicated by the point where the line from the message packet meets the time line. Lines going out from this point indicate messages sent in response to the message receipt. This relation gives the activation ordering. From the diagram, we see that the delivery of a tick message causes a new tick message to be sent, and that the delivery of a request message with customer c causes a message of the form $c \triangleleft n$ to be sent where n is the number of ticks that have arrived at t prior to the request. To distinguish different occurrences of the message packets with the same target and contents we use events. Events are an abstract data type with a function pkt mapping events to message packets. In the diagram this is represented by tagging packets by an event identifiers.

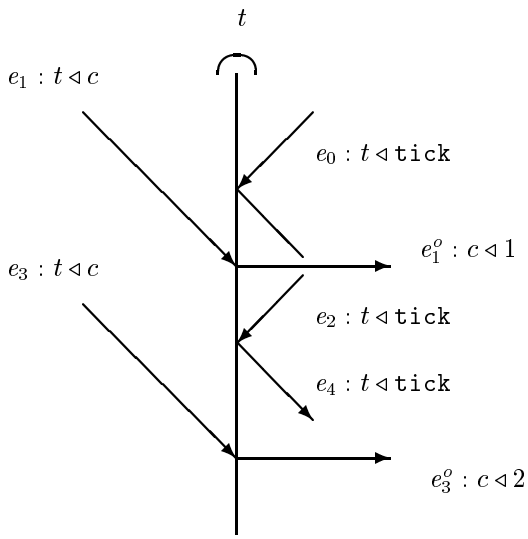

Figure 1: A Ticker Event Diagram

The following table summarizes the packet function for the ticker events.

$e:$	e_0	e_1	e_2	e_3	e_4	e_1^o	e_3^o
$pkt(e):$	$t \triangleleft \texttt{tick}$	$t \triangleleft c$	$t \triangleleft \texttt{tick}$	$t \triangleleft c$	$t \triangleleft \texttt{tick}$	$c \triangleleft 1$	$c \triangleleft 2$

The ticker event diagram represented in figure 1 is a structure

$$oed^{\mathrm{T}} = (t, \emptyset)(A^{\mathrm{T}}, E^{\mathrm{T}}, activator^{\mathrm{T}}, \xrightarrow{\mathrm{arr}}{}^{\mathrm{T}}, acqB^{\mathrm{T}}, Cr^{\mathrm{T}}, P^{\mathrm{T}})$$

where $(t, \emptyset)$ is the ticker interface with one receptionist, t, and no external references.
$A^{\mathrm{T}} = \{t\}$ is the set of internal actors – for the ticker, a single actor.
$E^{\mathrm{T}} = \{e_0, e_1, e_2, e_3, e_1^o, e_3^o\}$ is the set of realized events – messages that have be delivered locally or sent out to the environment.
$P^{\mathrm{T}} = \{e_4\}$ is the set of pending events — messages that have been sent by internal actors, or received from the environment but not yet delivered.
$activator^{\mathrm{T}}$ is the activator function given by the following table.

$e:$	e_0	e_1	e_2	e_3	e_4	e_1^o	e_3^o
$activator^{\mathrm{T}}(e):$	**!**	**?**	e_0	**?**	e_2	e_1	e_3

The activator function associates to each realized or pending event its activator which is either: **!** (an event present at the beginning); **?** (a message sent from the environment, the specific activator is *unknown*); or a realized event (the sending event). Thus, the initial tick message has activator **!**, meaning it was pending when the system was created. The events e_1 and e_3 have activator **?**, meaning they were generated by some unknown event in the environment. The activator of e_2 is e_0, meaning that e_2 corresponds to a message sent as a result of event e_0.

$\xrightarrow{\text{arr}}{}^{\mathrm{T}} = \{e_0 \to e_1 \to e_2 \to e_3\}$ the arrival order of events at t
$acqB^{\mathrm{T}} = \{e \mapsto \{t\} \mid e \in E^{\mathrm{T}}/t$ gives the acquaintances of an actor just before each event. E^{T}/t is the set of events with target t, and $acqB^{\mathrm{T}}(e) = \{t\}$ means that the ticker only knows its own name, in particular it remembers none of the actor names that appear as customers.
$Cr^{\mathrm{T}} = \{t \mapsto \mathbf{!}\}$ is the creation function, giving the creator of each actor in the system. $Cr^{\mathrm{T}}(t) = \mathbf{!}$ means that the ticker, t, was initially present.

2.1.2 A Complete Ticker Event Diagram

Notice that the event diagram, oed^{T}, is not complete, since there is a tick message that has not been delivered. A complete diagram for a ticker with only the inputs above is obtained by adding events e_i with $pkt(e_i) = t \triangleleft \texttt{tick}$ for $5 \leq i, i \in \mathbf{N}$ and delivering these (and the pending e_4).

$$oed^{\mathrm{cT}} = (t, \emptyset)(A^{\mathrm{cT}}, E^{\mathrm{cT}} activator^{\mathrm{cT}}, \xrightarrow{\text{arr}}{}^{\mathrm{cT}}, acqB^{\mathrm{cT}}, Cr^{\mathrm{cT}}, \emptyset)$$

where

$$A^{\mathrm{cT}} = A^{\mathrm{T}} = \{t\},$$
$$E^{\mathrm{cT}} = E^{\mathrm{T}} \cup \{e_i \mid 5 \leq i, i \in \mathbf{N}\}$$
$$activator^{\mathrm{cT}} = activator^{\mathrm{T}} \cup \{e_{i+1} \mapsto e_i \mid 4 \leq i\}$$
$$\xrightarrow{\text{arr}}{}^{\mathrm{cT}} = \xrightarrow{\text{arr}}{}^{\mathrm{T}} \circ [e_i \mid 4 \leq i]$$
$$acqB^{\mathrm{cT}} = \{e \mapsto \{t\} \mid e \in E^{\mathrm{cT}}/t\},$$
$$Cr^{\mathrm{cT}} = Cr^{\mathrm{T}} = \{t \mapsto \mathbf{!}\},$$

2.1.3 A Ticker Interaction Diagram

If we put a box around the ticker event diagram, hiding the internal events but maintaining the connection between inputs (messages sent by the environment) and outputs (messages set to the environment) we obtain the interaction diagram shown in figure 2. The connection between inputs and outputs results from forming chains of activation and arrival links, that is, it is the restriction of the combined order.
For each event with activator **?** there are two times of interest: the arrival of message from the environment and the delivery. In the event diagram we see the delivery time. In the interaction diagram we see the arrival. To distinguish these times we represent the arrival from the system by $\texttt{in}(e)$. Also, since in the global time we have no explicit information about which actors are internal, we write $\texttt{out}(e)$ if e is targeted to an external actor. We extend the activation ordering so that $\texttt{in}(e)$, the input, activates, e, the delivery. Formally the interaction diagram represented in figure 2 is a structure

$$id^{\mathrm{T}} = (\{t\}, \emptyset)(I^{\mathrm{T}}, O^{\mathrm{T}}, \prec^{\mathrm{T}})$$

where

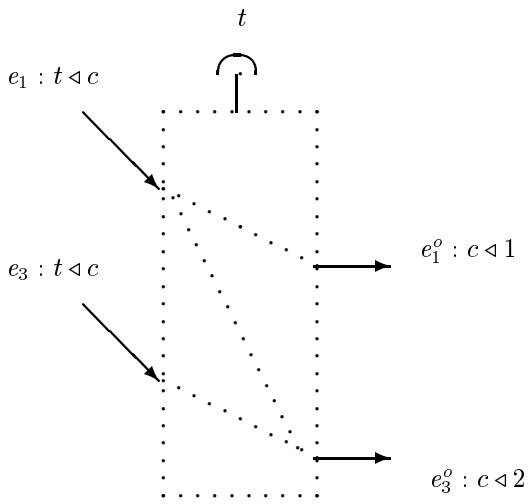

Figure 2: A Ticker Interaction Diagram

$I^{\mathrm{T}} = \{\mathtt{in}(e_1), \mathtt{in}(e_3)\}$ is the set of realized or pending events with activator ? (unknown) — observed at input time;
$O^{\mathrm{T}} = \{\mathtt{out}(e_1^o), \mathtt{out}(e_3^o)\}$ is the set of output events—realized events whose target is not one of the internal actors; and
$\prec^{\mathrm{T}} = \{(\mathtt{in}(e_1), \mathtt{out}(e_1^o)), (\mathtt{in}(e_3), \mathtt{out}(e_3^o))(\mathtt{in}(e_1), \mathtt{out}(e_3^o))\}$ is the combined order restricted to input and output events. We can think of the relations $\mathtt{in}(e_j) \prec \mathtt{out}(e_j^o)$ as representing the 'in response to' part of the combined order, while $\mathtt{in}(e_1) \prec \mathtt{out}(e_3^o)$ represents history information—the state of the actor when the response was made.

2.1.4 Global Times and Interaction Paths

As discussed in §1, Clinger [Cli81] showed that every event diagram has a (non-empty) set of global times – total orders on the set of events compatible with the activation and arrival ordering. Two possible global times for the ticker event diagram are:

$$g_1{}^{\mathrm{T}} = [\mathtt{in}(e_3), \mathtt{in}(e_1), e_0, e_1, e_2, e_3, \mathtt{out}(e_3^o), \mathtt{out}(e_1^o)]$$
$$g_2{}^{\mathrm{T}} = [e_0, \mathtt{in}(e_1), e_1, \mathtt{in}(e_3), e_2, \mathtt{out}(e_1^o), e_3, \mathtt{out}(e_3^o)]$$

Here we represent both the arrival and delivery time of messages coming from the environment. Note that the order of input events need not correspond to the order of their delivery. Similarly each interaction diagram has a set of global times which can be obtained by omitting the internal events from an event diagram global time. Pairing an interaction diagram global time with the interface gives an interaction path. Two possible interaction paths for the interaction diagram above are:

$$ip_1{}^{\mathrm{T}} = (t, \emptyset)[\mathtt{in}(e_3), \mathtt{in}(e_1), \mathtt{out}(e_3^o), \mathtt{out}(e_1^o)]$$

$$ip_2{}^{\mathrm{T}} = (t, \emptyset)[\mathtt{in}(e_1), \mathtt{in}(e_3), \mathtt{out}(e_1^o), \mathtt{out}(e_3^o)]$$

2.1.5 Ticker behavior and computation

To specify an actor that exhibits the behavior of a ticker we introduce a set of states $T(n)$ for $n \in \mathbf{N}$ and give reaction rules that determine the response of an actor in state $T(n)$ to each possible message. The rules for $T(n)$ are the following:

$$[\,T(n)\,]_t\ ,\ t \triangleleft c \qquad \Rightarrow \qquad [\,T(n)\,]_t\ ,\ c \triangleleft n \quad \text{if } c \in A$$

$$[\,T(n)\,]_t\ ,\ t \triangleleft \mathtt{tick} \quad \Rightarrow \quad [\,T(n+1)\,]_t\ ,\ t \triangleleft \mathtt{tick}$$

The first rule describes the response to request messages: the ticker state is unchanged and a message is sent. The second rule describes the response to tick messages: the ticker state changes by incrementing the parameter n, and a tick message is sent. Formally actor systems are described by configurations. A ticker system at start up time is described by the configuration

$$Ticker[t] = (t, \emptyset)([\,T(0)\,]_t\ ,\ e_0 \mid \emptyset)$$

It has interface $(t, \emptyset)$, a single actor, $[\,T(0)\,]_t$, with name, t, and state, $T(0)$, and a single pending event e_0—a tick message. We also keep in the configuration the set of past events, initially empty. Computations are sequences of labelled transitions where the labels correspond to input, delivery and output of messages. The computation, π^{T}, for the ticker system corresponding to the global time $g_2{}^{\mathrm{T}}$ follows.

$$\begin{aligned}
&(t, \emptyset)([\,T(0)\,]_t\ ,\ e_0 \mid \emptyset)\\
&\quad \xrightarrow{\mathtt{d}(e_0)} (t, \emptyset)([\,T(1)\,]_t\ ,\ e_2 \mid e_0)\\
&\quad \xrightarrow{\mathtt{in}(e_1)} (t, c)([\,T(1)\,]_t\ ,\ e_2\ ,\ e_1 \mid e_0)\\
&\quad \xrightarrow{\mathtt{d}(e_1)} (t, c)([\,T(1)\,]_t\ ,\ e_2\ ,\ e_1^o \mid e_0\ ,\ e_1)\\
&\quad \xrightarrow{\mathtt{in}(e_3)} (t, c)([\,T(1)\,]_t\ ,\ e_2\ ,\ e_3\ ,\ e_1^o \mid e_0\ ,\ e_1)\\
&\quad \xrightarrow{\mathtt{d}(e_2)} (t, c)([\,T(2)\,]_t\ ,\ e_4\ ,\ e_3\ ,\ e_1^o \mid e_0\ ,\ e_1\ ,\ e_2)\\
&\quad \xrightarrow{\mathtt{out}(e_1^o)} (t, c)([\,T(2)\,]_t\ ,\ e_4\ ,\ e_3 \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o)\\
&\quad \xrightarrow{\mathtt{d}(e_3)} (t, c)([\,T(2)\,]_t\ ,\ e_4\ ,\ e_3^o \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o\ ,\ e_3)\\
&\quad \xrightarrow{\mathtt{out}(e_3^o)} (t, c)([\,T(2)\,]_t\ ,\ e_4 \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o\ ,\ e_3\ ,\ e_3^o)
\end{aligned}$$

Notice that we can read off the interaction path from this computation by taking the initial interface and the subsequence of input/output transition labels. We can also read off an interaction diagram or and event diagram by keeping track of which delivery step generated which events. The sequence of labels will then be a global time for the diagram.

2.2 Ticker Factory

The ticker illustrates message passing, state change, and event ordering features of actor computation. The missing feature is actor creation. To illustrate this feature we introduce a ticker factory. A ticker factory is an actor system that accepts requests $\texttt{new}(c)$ for new tickers. For each such request, a new ticker is created and its name is sent to the requestor c. Figure 3 shows an event diagram for a ticker factory with receptionist, tf, in which two requests—$tf \triangleleft \texttt{new}(c_0)$ and $tf \triangleleft \texttt{new}(c_1)$—for new tickers arrive from outside and two time requests—$t_0 \triangleleft c_0$ and $t_0 \triangleleft c_3$—arrive for t_0, the first ticker created.

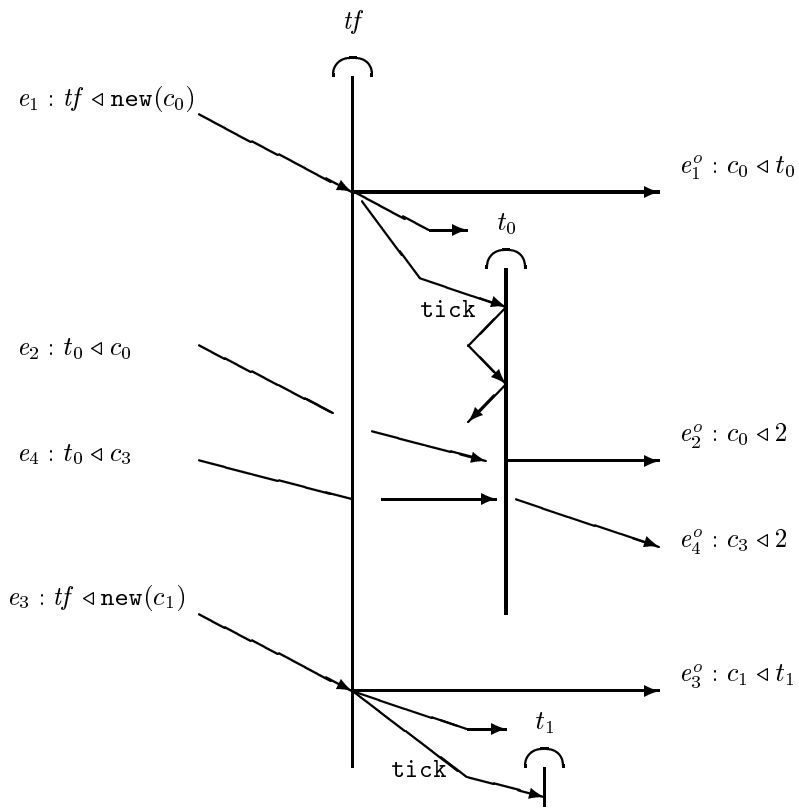

Figure 3: A Ticker Factory Event Diagram

The ticker factory events and packet function are given by the following table.

$e:$	e_1	e_2	e_3	e_4	e_1^o	e_2^o	e_3^o
$pkt(e):$	$tf \triangleleft \texttt{new}(c_0)$	$t_0 \triangleleft c_0$	$tf \triangleleft \texttt{new}(c_1)$	$t_0 \triangleleft c_3$	$c_0 \triangleleft t_0$	$c_0 \triangleleft 2$	$c_1 \triangleleft t_1$
$e:$	e_4^o	e_{01}	e_{02}	e_{03}	e_{11}		
$pkt(e):$	$c_3 \triangleleft 2$	$t_0 \triangleleft \texttt{tick}$	$t_0 \triangleleft \texttt{tick}$	$t_0 \triangleleft \texttt{tick}$	$t_1 \triangleleft \texttt{tick}$		

The mathematical structure corresponding to the ticker factory event diagram is the

following.

$$oed^{\mathrm{Tf}} = (tf, \emptyset)(A^{\mathrm{Tf}}, E^{\mathrm{Tf}}, activator^{\mathrm{Tf}}, \xrightarrow{\mathrm{arr}}{}^{\mathrm{Tf}}, acqB^{\mathrm{Tf}}, Cr^{\mathrm{Tf}}, P^{\mathrm{Tf}})$$

where

$$A^{\mathrm{Tf}} = \{tf, t_0, t_1\}$$
$$E^{\mathrm{Tf}} = \{e_1, e_2, e_3, e_4, e_1^o, e_2^o, e_3^o, e_4^o, e_{01}, e_{02}\},$$
$$P^{\mathrm{Tf}} = \{e_{03}, e_{11}\}$$
$$\xrightarrow{\mathrm{arr}}_{tf}{}^{\mathrm{Tf}} = [e_1 \to e_3],$$
$$\xrightarrow{\mathrm{arr}}_{t}{}^{\mathrm{Tf}} = [e_{01} \to e_{02} \to e_2 \to e_4]$$
$$acqB^{\mathrm{Tf}} = (\lambda e \in E^{\mathrm{Tf}}/t_0)\{t\} \cup (\lambda e \in E^{\mathrm{Tf}}/tf)\emptyset$$
$$Cr^{\mathrm{Tf}} = \{tf \mapsto \mathbf{!}, t_0 \mapsto e_1, t_1 \mapsto e_3\}$$

and the activator function is given by the following table

e :	e_1	e_2	e_3	e_4	e_1^o	e_2^o	e_3^o	e_4^o	e_{01}	e_{02}	e_{03}	e_{11}
$activator^{\mathrm{Tf}}(e)$:	**?**	**?**	**?**	**?**	e_1	e_2	e_3	e_4	e_1	e_{01}	e_{02}	e_3

$Cr^{\mathrm{Tf}}(t_0) = e_1$ means that actor t_0 was created in response to e_1. Notice that in the combined ordering e_2^o has two predecessor: e_1 and e_2. Hiding the internal details we obtain the interaction diagram shown in figure 4.
The mathematical structure corresponding to figure 4 is

$$id^{\mathrm{Tf}} = (tf, \emptyset)(I^{\mathrm{Tf}}, O^{\mathrm{Tf}}, \prec^{\mathrm{Tf}})$$

where

$$I^{\mathrm{Tf}} = \{\mathtt{in}(e_1), \mathtt{in}(e_2), \mathtt{in}(e_3), \mathtt{in}(e_4)\}$$
$$O^{\mathrm{Tf}} = \{\mathtt{out}(e_1^o), \mathtt{out}(e_2^o), \mathtt{out}(e_3^o), \mathtt{out}(e_4^o)\}$$
$$\prec^{\mathrm{Tf}} = \{\ (\mathtt{in}(e_1), \mathtt{out}(e_1^o)), (\{\mathtt{in}(e_1), \mathtt{in}(e_2)\}, \mathtt{out}(e_2^o)),$$
$$(\{\mathtt{in}(e_1), \mathtt{in}(e_3)\}, \mathtt{out}(e_3^o)), (\{\mathtt{in}(e_1), \mathtt{in}(e_2), \mathtt{in}(e_4)\}, \mathtt{out}(e_4^o))\}$$

As in the ticker example, the ordering, $\prec^{\mathrm{Tf}}$, includes the 'in response to' ordering and history information. It also includes the creator information $\mathtt{in}(e_1) \prec^{\mathrm{Tf}} \mathtt{out}(e_2^o)$. More generally it will be the case that $\mathtt{in}(e_1) \prec^{\mathrm{Tf}} \mathtt{out}(e)$ for any e representing a message sent by t_0. In addition to being compatible with the combined order, a global time for the ticker factory diagrams must reflect the fact that the actor name t_0 must be exported before it can receive messages from outside the system. Thus e_1^o must happen before e_2 and e_4. Two possible interaction paths for the ticker interaction diagram are

$$ip_1{}^{\mathrm{Tf}} = (tf, \emptyset)[\mathtt{in}(e_1), \mathtt{out}(e_1^o), \mathtt{in}(e_2), \mathtt{out}(e_2^o), \mathtt{in}(e_3), \mathtt{in}(e_4), \mathtt{out}(e_4^o), \mathtt{out}(e_3^o)]$$
$$ip_2{}^{\mathrm{Tf}} = (tf, \emptyset)[\mathtt{in}(e_1), \mathtt{in}(e_3), \mathtt{out}(e_1^o), \mathtt{in}(e_2), \mathtt{in}(e_4), \mathtt{out}(e_4^o), \mathtt{out}(e_3^o), \mathtt{out}(e_2^o)]$$

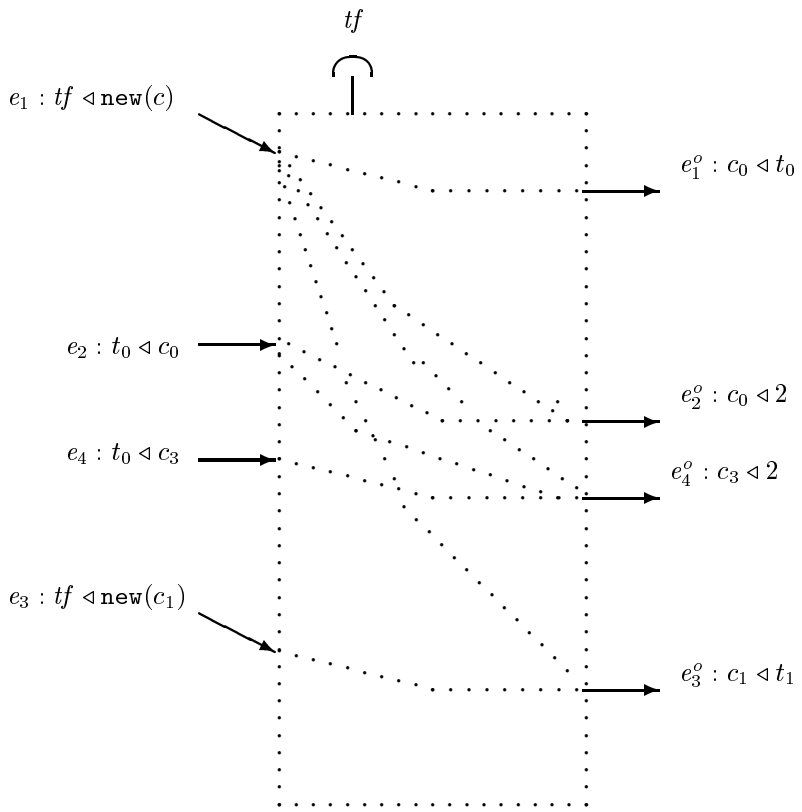

Figure 4: A Ticker Factory Interaction Diagram

2.3 Composing

In addition to modeling self-contained components we also want to compose systems out of simpler parts and to compose the semantic models in a corresponding manner.

As a simple example we build a task dispatcher from a switch and a ticker. For simplicity we consider a binary switch s which knows two workers (w_0, w_1) and forwards incoming tasks non-deterministically to these workers. It makes a choice by getting a number, n, from a ticker and picking worker w_0 if n is even and w_1 if n is odd. Figure 5 shows matching event diagrams for the two subsystems. The switch receives one task, sends a request to the ticker, gets a reply and forwards the task. The ticker gets one request with customer s and replies.

Figure 6 shows the event diagram for the composed system. It is obtained by identifying events with the same label (conveniently arranged). To form the dispatcher the receptionists are restricted to just the switch. The ticker is hidden, indicated by the brackets around its name. Although the precise rules for composition of event diagrams are a bit more subtle, this illustrates the basic idea – find diagrams from the two components such that each output from one component to a receptionists of the other has a

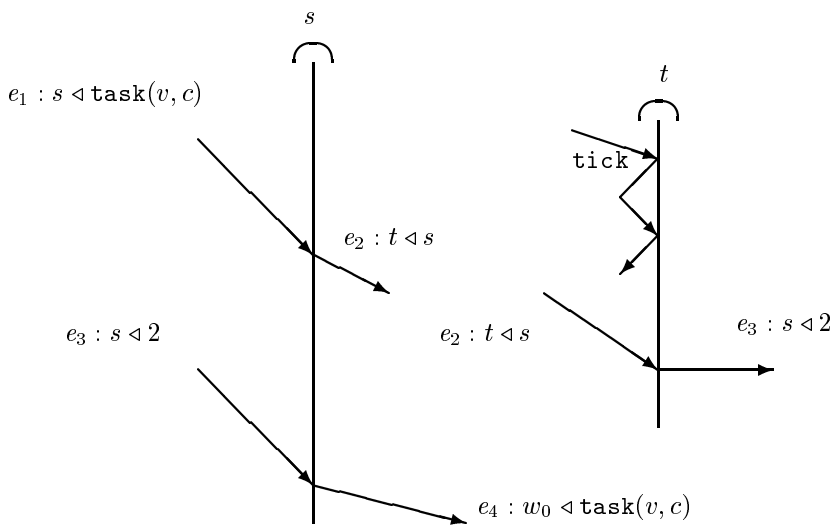

Figure 5: Event Diagrams for Dispatcher components

matching input and zip them up.

The formation of the dispatcher can be expressed as a term in the algebra of components. Let $[\![S(t, [w_0, w_1])]\!]_s$ be a switch actor with name s, and state $S(t, [w_0, w_1])$, It has worker acquaintances $[w_0, w_1]$, and ticker acquaintance t. The switch component

$$\mathit{Switch}[s](t, [w_0, w_1]) = (s, \{t, w_0, w_1\})([\![S(t, [w_0, w_1])]\!]_s \mid \emptyset)$$

has receptionists s, externals, $\{t, w_0, w_1\}$, and a single switch actor. There are no undelivered or previously delivered messages. A dispatcher that non-deterministically dispatches tasks to w_0 or w_1 is formed by parallel composition of the switch and ticker followed by restricting the receptionists to just the switch actor, thus hiding the ticker actor.

$$(\mathit{Switch}[s](t, [w_0, w_1]) \parallel \mathit{Ticker}[t](0))\lceil\{s\} =$$
$$(s, \{w_0, w_1\})([\![S(t, [w_0, w_1])]\!]_s \ , \ [\![T(t, 0)]\!]_t \ , \ t \triangleleft \texttt{tick} \mid \emptyset)$$

3 Messages and Events

This section contains the basic notation and definitions regarding messages and events that will be used throughout the remainder of the paper.

3.1 Actor Names and Messages

To model actor computations we need at least a set of actor names, $\mathbf{A}$, and a set, $\mathbf{M}$, representing the contents of messages that can be communicated between actors. We

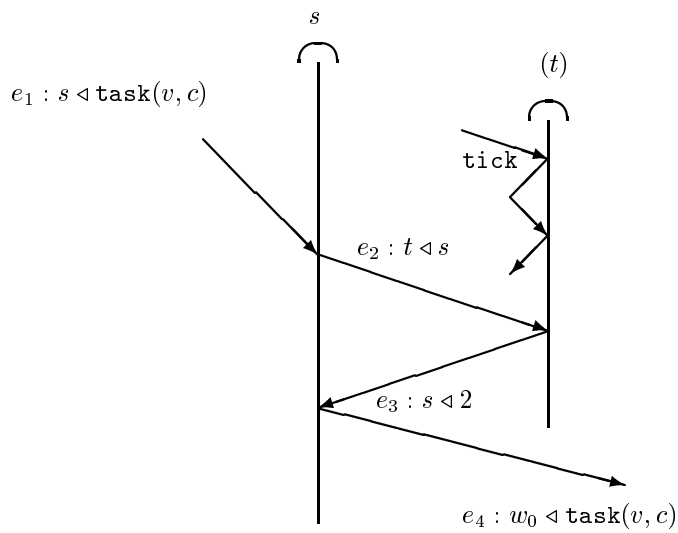

Figure 6: A Composed Event Diagram for Dispatcher

let a range over $\mathbf{A}$ and M range over $\mathbf{M}$. The set of message packets (briefly packets), $\mathbf{MP}$, is $\mathbf{A} \triangleleft \mathbf{M}$. We let mp range over $\mathbf{MP}$. Given a message packet $mp = a \triangleleft M$, the target, $tgt(mp)$ is a, and the message contents, $mc(mp)$ is M.

Actor names may occur in the contents of a message. Since the structure of message contents is not specified, we assume there is an acquaintance function $acq : \mathbf{M} \to \mathcal{P}_\omega[\mathbf{A}]$ that gives the (finite set of) actor names occurring in a message. The acquaintance function is extended to message packets by $acq(a \triangleleft M) = \{a\} \cup acq(M)$.

A renaming is a bijection on actor names. We let α range over renamings. Renamings are used to express the property that the choice of actor names can not matter. Different choices give the same results modulo a corresponding renaming. Renamings, α, are lifted to bijections $\widehat{\alpha}$ on message contents such that $acq(\widehat{\alpha}(M)) = \widehat{\alpha}(acq(M))$. A renaming that is the identity on the acquaintances of a object lifts to a function that is the identity on the object. Renaming is extended pointwise to pairs, sets, functions and relations (viewed as sets) and other structures. For example $\widehat{\alpha}(a \triangleleft M) = \alpha(a) \triangleleft \widehat{\alpha}(M)$.

In the examples of §2, $\mathbf{M}$ includes $\mathbf{A}$, $\texttt{tick}$, $\texttt{new}(c)$ for $c \in \mathbf{A}$, with $acq(\texttt{tick}) = \emptyset$, and $acq(c) = acq(\texttt{new}(c)) = \{c\}$ for $c \in \mathbf{A}$. Also, $\widehat{\alpha}(\texttt{tick}) = \texttt{tick}$, for any renaming, α, since $\texttt{tick}$ has no acquaintances it can't be changed by renaming, and $\widehat{\alpha}(\texttt{new}(c)) = \texttt{new}(c')$, if $\alpha(c) = c'$.

3.2 Events

To describe event diagrams we need a set, $\mathbf{E}$, of events over $\mathbf{A}$, $\mathbf{M}$. Events allow us to represent dynamic occurrences of message packets. Each event, e, has a message packet, $pkt(e) \in \mathbf{MP}$. We let e range over $\mathbf{E}$ and E range over $\mathcal{P}[\mathbf{E}]$. We assume renamings are lifted to bijections on events that commute with the packet function:

$\widehat{\alpha}(pkt(e)) = pkt(\widehat{\alpha}(e))$. We extend the message target and contents functions and the acquaintance function to events by composition with the packet function:

$$tgt(e) = tgt(pkt(e)) \qquad mc(e) = mc(pkt(e))$$
$$acq(e) = acq(pkt(e)) \qquad acq(E) = \bigcup_{e \in E} acq(e)$$

For $E \subseteq \mathbf{E}$ and $A \subseteq \mathbf{A}$, define E/A, the subset of E with target in A by:

$$E/A = \{e \in E \mid tgt(e) \in A\}$$

Events traditionally correspond to delivery of messages. In the case of open systems, messages may be delivered to internal actors (internal events), or delivered to the environment (output events) or they may come in from the environment (input events). We write $\mathtt{in}(e)$ to distinguish the arrival of a message from outside the system from its delivery e. We also write $\mathtt{out}(e)$ to distinguish delivery of a message to the environment from from the actual delivery to the target actor. We let $\widetilde{\mathbf{E}}$ be the set of events extended by these input/output events

$$\widetilde{\mathbf{E}} = \mathbf{E} \cup \mathtt{in}(\mathbf{E}) \cup \mathtt{out}(\mathbf{E})$$

and we let $\tilde{e}$ range over $\widetilde{\mathbf{E}}$ and $\widetilde{E}$ range over $\mathcal{P}[\widetilde{\mathbf{E}}]$.

3.3 Interfaces

An actor system interface is a pair (ρ, χ) of disjoint finite sets of actor names. ρ specifies the receptionists and χ specifies the external actors known to the system. We let $\mathbf{Iface}$ be the set of such interfaces. We define parallel composition of interfaces with disjoint receptionists by

$$(\rho_0, \chi_0) \parallel (\rho_1, \chi_1) = (\rho, \chi) \quad \text{where} \quad \rho = \rho_0 \cup \rho_1 \quad \text{and} \quad \chi = (\chi_0 \cup \chi_1) - \rho$$

Most of the structures we will be concerned with in the following have the form $\sigma = (\rho, \chi)(\ldots)$. That is they consist of an interface, (ρ, χ), paired with some additional structure, $(\ldots)$. In the ticker example of §2 there were event diagrams, interaction diagrams, interaction paths, and configurations for a ticker system, all with interface $(t, \emptyset)$. For σ of the form $(\rho, \chi)(\ldots)$ we define:

$recep(\sigma) = \rho$ the receptionists of σ

$extern(\sigma) = \chi$ the external actors of σ

$iface(\sigma) = (\rho, \chi)$ the interface of σ

In addition, for a set of interfaced structures S we define $S(\rho, \chi)$ to be the subset with interface (ρ, χ):

$$S(\rho, \chi) = \{\sigma \in S \mid iface(\sigma) = (\rho, \chi)\}$$

3.4 Global Times and Event Paths

A global time map, $g : \widetilde{E} \xrightarrow{\text{inj}} \mathbf{N}$, is simply an injective function from a set of extended events to the natural numbers. It defines a total ordering, $<_g$, on $\widetilde{E}$ by $\tilde{e} <_g \tilde{e}'$ just if $g(\tilde{e}) < g(\tilde{e}')$ for $\tilde{e}, \tilde{e}' \in \widetilde{E}$. An event path is a global time map plus an interface. We let p range over event paths. The functions $recep$ and $extern$ (are overloaded to) take an event path and a number and give the receptionists, respectively external actors, just prior to the ith event of the path. They are defined by induction on i as follows. Let $p = (\rho, \chi)g$. Then

$$recep(p, 0) = \rho \qquad extern(p, 0) = \chi$$

$$recep(p, i+1) = recep(p, i) \cup \begin{cases} acq(m(e)) - extern(p, i) & \text{if } g^{-1}(i) = \mathtt{out}(e) \\ \emptyset & \text{otherwise} \end{cases}$$

$$extern(p, i+1) = extern(p, i) \cup \begin{cases} acq(m(e)) - recep(p, i) & \text{if } g^{-1}(i) = \mathtt{in}(e) \\ \emptyset & \text{otherwise} \end{cases}$$

The functions $recep^\omega$ and $extern^\omega$ map event paths to the set of accumulated receptionists, respectively known external actors, along the path. Let $p = (\rho, \chi)g$. Then

$$recep^\omega(p) = \bigcup_{i \in \mathbf{N}} recep(p, i) \qquad extern^\omega(p) = \bigcup_{i \in \mathbf{N}} extern(p, i)$$

Returning to the ticker factory, recall the interaction path

$$ip_2{}^{\mathrm{Tf}} = (tf, \emptyset)g_2{}^{\mathrm{Tf}} \quad \text{where}$$
$$g_2{}^{\mathrm{Tf}} = [\mathtt{in}(e_1), \mathtt{in}(e_3), \mathtt{out}(e_1^o), \mathtt{in}(e_2), \mathtt{in}(e_4), \mathtt{out}(e_4^o), \mathtt{out}(e_3^o), \mathtt{out}(e_2^o)].$$

where the event packets are

$e:$	e_1	e_2	e_3	e_4
$pkt(e):$	$tf \triangleleft \mathtt{new}(c_0)$	$t_0 \triangleleft c_0$	$tf \triangleleft \mathtt{new}(c_1)$	$t_0 \triangleleft c_3$
$e:$	e_1^o	e_2^o	e_3^o	e_4^o
$pkt(e):$	$c_0 \triangleleft t_0$	$c_0 \triangleleft 2$	$c_1 \triangleleft t_1$	$c_3 \triangleleft 2$

$g_2{}^{\mathrm{Tf}}$ is a global time for the ticker factory interaction diagram, id^{Tf}, mapping the i-th element of the sequence, to i. For example $g_2{}^{\mathrm{Tf}}(\mathtt{in}(e_3)) = 2$. Thus $recep(ip_2{}^{\mathrm{Tf}}, i) = \{tf\}$ for $1 \leq i \leq 3$, $recep(ip_2{}^{\mathrm{Tf}}, i) = \{tf, t_0\}$ for $4 \leq i \leq 8$, and $extern(ip_2{}^{\mathrm{Tf}}, 1) = \emptyset$, $extern(ip_2{}^{\mathrm{Tf}}, 2) = \{c_0\}$, $extern(ip_2{}^{\mathrm{Tf}}, i) = \{c_0, c_1\}$, for $3 \leq i \leq 5$. It is easy to see that $recep$ and $extern$ are monotonic in the number argument and that the resulting sets of receptionists and externals are disjoint.

Lemma (event paths): For $i < i' \in \mathbf{N}$

(1) $recep((\rho, \chi)g, i) \subseteq recep((\rho, \chi)g, i') \subseteq recep^\omega((\rho, \chi)g)$

(2) $extern((\rho, \chi)g, i) \subseteq extern((\rho, \chi)g, i') \subseteq extern^\omega((\rho, \chi)g)$

(3) $recep((\rho,\chi)g,i) \cap extern((\rho,\chi)g,i) = \emptyset$

(4) $recep((\rho,\chi)g) \cap extern((\rho,\chi)g) = \emptyset$

The property $EPLaw$ expresses interface level locality laws [BH77, Cli81] as constraints on actors exported and imported via the path interface.

$$
\begin{aligned}
&EPLaw((\rho,\chi)g) \quad \text{iff} \\
&\qquad (\forall i \in \mathrm{Rng}(g), e \in \mathbf{E})(g^{-1}(i) = \mathtt{in}(e) \;\Rightarrow\; tgt(e) \in recep((\rho,\chi)g,i) \wedge \\
&\qquad\qquad\qquad\qquad g^{-1}(i) = \mathtt{out}(e) \;\Rightarrow\; tgt(e) \in extern((\rho,\chi)g,i) \\
&EPLaw((\rho,\chi)g, A) \quad \text{iff} \\
&\quad EPLaw((\rho,\chi)g) \wedge \\
&\quad (\forall i \in \mathrm{Rng}(g), e \in \mathbf{E})g^{-1}(i) = \mathtt{in}(e) \;\Rightarrow\; (acq(mc(e))) \cap A \subseteq recep((\rho,\chi)g,i)
\end{aligned}
$$

It is easy to see that $EPLaw({ip_2}^{\mathrm{Tf}})$ and $EPLaw({ip_2}^{\mathrm{Tf}}, \{tf, t_0, t_1\})$ hold. However, since a non-receptionist internal actor occurs in an input, $EPLaw({ip_2}^{\mathrm{Tf}}, \{tf, t_0, t_1, c_3\})$ fails. Also if we let ${g_3}^{\mathrm{Tf}}$ be the result of interchanging $\mathtt{in}(e_2)$ and $\mathtt{out}(e_1^o)$, then an input to t_0 arrives before it becomes a receptionist and $EPLaw((tf,\emptyset){g_3}^{\mathrm{Tf}})$ fails.

Composition of Event Paths

Now composability and composition are defined on event paths. Two extended events are *complements* if one is of the form $\mathtt{in}(e)$,and the other is of the form $\mathtt{out}(e)$.

Let $g_j : \widetilde{E}_j \xrightarrow{\text{inj}} \mathbf{N}$ and let (ρ_j,χ_j) be such that $EPLaw((\rho_j,\chi_j)g_j)$ holds for $j < 2$. The synchronization points of $((\rho_0,\chi_0)g_0, (\rho_1,\chi_1)g_1)$ are the times when one path outputs a message whose target is a receptionist of the other path. They are defined by

$$
\begin{aligned}
&synch((\rho_0,\chi_0)g_0, (\rho_1,\chi_1)g_1) = \\
&\qquad \{i \mid (\exists j < 2, e \in \mathbf{E})({g_j}^{-1}(i) = \mathtt{out}(e) \wedge tgt(e) \in recep^{\omega}((\rho_{1-j},\chi_{1-j})g_{1-j}))\}
\end{aligned}
$$

The composition of $(\rho_0,\chi_0)g_0$ and $(\rho_1,\chi_1)g_1$ is defined by

$$
\begin{aligned}
&(\rho_0,\chi_0)g_0 \parallel (\rho_1,\chi_1)g_1) = (\rho,\chi)g \quad \text{where} \\
&\quad (\rho,\chi) = (\rho_0,\chi_0) \parallel (\rho_1,\chi_1) \quad \text{and} \\
&\quad \mathrm{Dom}(g) = (\mathrm{Dom}(g_0) \cup \mathrm{Dom}(g_1)) - \\
&\qquad\qquad \textstyle\bigcup_{i \in synch((\rho_0,\chi_0)g_0,(\rho_1,\chi_1)g_1)} ({g_0}^{-1}(i) \cup {g_1}^{-1}(i)) \\
&\quad g(\tilde{e}) = g_j(\tilde{e}) \quad \text{if } \tilde{e} \in \mathrm{Dom}(g_j)
\end{aligned}
$$

Event path composability is defined by $(\rho_0,\chi_0)g_0 \bowtie (\rho_1,\chi_1)g_1$ just if

(1) $recep^{\omega}((\rho_0,\chi_0)g_0) \cap recep^{\omega}((\rho_1,\chi_1)g_1) = \emptyset$

(2) if $i \in (\mathrm{Rng}(g_0) \cap \mathrm{Rng}(g_1))$ then ${g_0}^{-1}(i)$ and ${g_1}^{-1}(i)$ are complements

(3) if $i \in synch((\rho_0, \chi_0)g_0, (\rho_1, \chi_1)g_1)$, then $i \in (\mathrm{Rng}(g_0) \cap \mathrm{Rng}(g_1))$

(4) $EPLaw((\rho_0, \chi_0)g_0 \parallel (\rho_1, \chi_1)g_1)$

Clearly if two event paths are composable then the composition is well-defined and results in an event path. As an example let p^s, p^t be event paths for the switch and ticker events in Figure 5

$$p^s = (s, \{t, w_0, w_1\})[\mathtt{in}(e_1), *, *, e_1, \mathtt{out}(e_2), *, *, \mathtt{in}(e_3), e_3, \mathtt{out}(e_4)]$$
$$p^t = (t, \emptyset)[*, e_0, e_0', *, \mathtt{in}(e_2), *, e_2, \mathtt{out}(e_3)]$$

where e_0, e_0' are $\mathtt{ticks}$, $pkt(e_1) = s \triangleleft \mathtt{task}(v, c)$, $pkt(e_2) = t \triangleleft s$, $pkt(e_3) = s \triangleleft 2$, $pkt(e_4) = w_0 \triangleleft \mathtt{task}(v, c)$,and $*$ is a place holder indicating the index of the place is not in the range of the global time map. p^s corresponds to a global time for the switch event diagram and p^t to a global time for the ticker event diagram. Then $p^s \bowtie p^t$ with synchronization points $5, 8$ and

$$p^s \parallel p^t = (\{s, t\}, \{w_0, w_1\})[\mathtt{in}(e_1), e_0, e_0', e_1, e_2, e_3, \mathtt{out}(e_4)]$$

4 Open Event Diagrams

In this section open event diagrams (OEDs) are defined as mathematical structures of a certain form that satisfy the Baker-Hewitt [BH77] ordering and locality laws for actor computation. A partial ordering on OEDs corresponding to stages of computation is defined and it is shown that any OED can be expressed at the limit of finite OEDs. The above is basically an adaption of parts of [Cli81] to account for component encapsulation in the form of explicit interfaces, and the interactions with the environment. Finally a component algebra structure is defined on OED with the complete OEDs forming a sub-algebra.

4.1 Structure and axioms

Open Event Diagram Structures are structures of the form

$$oed = (\rho, \chi)(A, E, activator, \xrightarrow{\mathrm{arr}}, acqB, Cr, P)$$

where

$(\rho, \chi) \in \mathbf{Iface}$ —initial receptionists and external actors

$A \subset \mathbf{A}$ —internal actors

$E \subset \mathbf{E}$ —the realized events

$P \subset \mathbf{E}$ —the pending events

$\rho \subseteq A \qquad \chi \cap A = \emptyset \qquad E \cap P = \emptyset$

$activator : E \cup P \to E \cup \{\mathbf{?}, \mathbf{!}\}$ —generates the activation order

$$\xrightarrow{\text{arr}} \;=\; \bigcup_{a \in A} \xrightarrow{\text{arr}}_a \qquad \text{where } \xrightarrow{\text{arr}}_a \text{ is a total order on } E/a$$

—arrival order on events with target $a \in A$

$acqB : E/A \to \mathcal{P}_\omega[\mathbf{A}]$ —acquaintances just before arrival time

$Cr : A \to E/A \cup \{\mathbf{!}\}$ —creation event, $Cr(a) = \mathbf{!}$ if initially present

Recall that $activator(e) = \mathbf{!}$ means that e was initially present, and $Cr(a) = \mathbf{!}$ means that a was initially present. $activator(e) = \mathbf{?}$ means that e arrived from environment.

To be an open event diagram, an OED structure must satisfy the ordering and locality laws (**Actor laws**) below. To state these laws, we introduce some further notation. Let $oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P)$ be an OED structure. The input events, $I(oed)$, output events, $O(oed)$, and internally delivered events, $Dlv(oed)$, and the extended event set, $\widetilde{E}(oed)$ of oed are defined by

$$In(oed) = \{\mathtt{in}(e) \mid e \in E \cup P \;\wedge\; activator(e) = \mathbf{?}\}$$

$$Out(oed) = \{\mathtt{out}(e) \mid e \in E \;\wedge\; tgt(e) \notin A\}$$

$$Dlv(oed) = E/A$$

$$\widetilde{E}(oed) = Dlv(oed) \cup In(oed) \cup Out(oed)$$

Note that $Dlv(oed)$ contains the input events (without the `in` wrapper), but not the output events (in any form). Let $e \in Dlv(oed)$ be an internal event. If $tgt(e) = a \in A$, then $rank(e)$ is the rank of e in E/a given by the total ordering $\xrightarrow{\text{arr}}_a$. For example $rank(e) = 0$ if e is the least element of $\xrightarrow{\text{arr}}_a$. The activation order on E, $\prec_{\text{ao}}$, is the least transitive relation such that $activator(e) \prec_{\text{ao}} e$ for $e \in E$. The combined order $\prec_{\text{co}}$, is the least transitive relation containing $\prec_{\text{ao}}$ and $\xrightarrow{\text{arr}}$. In the context of extended events we replace $activator(e) = \mathbf{?}$ by $activator(e) = \mathtt{in}(e)$ and we view $\prec_{\text{ao}}$ and $\xrightarrow{\text{arr}}$ as relations on extended events by also defining $activator(\mathtt{out}(e)) = activator(e)$.

For $e \in E$, the actors created by e, $crby(e)$, and the events sent by e, $sentby(e)$ are defined by

$$crby(e) = Cr^{-1}(a)$$

$$sentby(e) = \{e' \in E \cup P \mid activator(e') = e\}$$

The participants in an event, $participants(e)$, are the acquaintances of the target at delivery time together with the acquaintances delivered in the message. The associates, $associates(e)$, are the participants plus any actors created by the event.

$$participants(e) = acqB(e) \cup acq(mc(e))$$

$$associates(e) = participants(e) \cup crby(e)$$

Returning to the ticker factory example, consider the event e_1 representing the delivery of a message packet $tf \triangleleft \mathtt{new}(c_0)$. For this event $acqB(e_1) = \emptyset$, $participants(e_1) = \{c_0\}$, and $associates(e_1) = \{c_0, t_0\}$.

A global time for oed is a total ordering on the extended events of oed that is compatible with the (extended) combined ordering such that the event path law holds with respect to (ρ, χ). The set of global times, $GT(oed)$, for oed is defined as follows.

$$GT(oed) = \{g : \widetilde{E}(oed) \xrightarrow{\text{inj}} \mathbf{N} \mid g \text{ preserves } \prec_{\text{ao}} \text{ and } \xrightarrow{\text{arr}} \quad \text{and} \quad EPLaw((\rho, \chi)g, A)\}$$

Global time maps are extended to $\widetilde{E}(oed) \cup \{\mathbf{!}\}$, defining $g(\mathbf{!}) = -1$. $OED^{\dagger}$ is the set of OED global time pairs:

$$OED^{\dagger} = \{(oed, g) \mid oed \in OED \wedge g \in GT(oed)\}$$

This convention for denoting the set of structure-global time pairs will be used throughout for sets of structures for which the notion of global time is defined.

Actor laws

These laws are adaptations of the axioms given by Clinger [Cli81] for event diagrams. They fall into two groups: (**Ordering Laws**) having to do with the partial order on events; and (**Locality Laws**), having to do with actor creation and acquaintance relations. Let

$$oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P)$$

be an OED structure. Then oed is an open event diagram ($oed \in OED$) just if (**Ordering laws**) and (**Locality laws**) hold. (The labels are those used in [Cli81].)

Ordering laws

(the axiom of strong realizability): $GT(oed) \neq \emptyset$—there is a global time.

(finite delay): if E is infinite then $P = \emptyset$—fairness.

Locality laws

(law of original acquaintances): if $a \in A$, $e \in E/a$, $rank(e) = 0$, then

$$acqB(e) \subseteq A \cup \chi \quad \text{if } Cr(a) = \mathbf{!}$$
$$acqB(e) \subseteq associates(Cr(a)) \quad \text{if } Cr(a) \neq \mathbf{!}$$

(law of arrival precursor acquaintances): if $a \in A$, $tgt(e) = tgt(e') = a$, $rank(e) = rank(e') + 1$, then $acqB(e') \subseteq associates(e)$.

(law of activator acquaintances): if $e, e' \in Dlv(oed)$ and $activator(e) = e'$, then $tgt(e) \cup acq(mc(e)) \subseteq associates(e')$.

(finite activation): $associates(e)$ is finite.

$\square$

OED denotes the set of all open event diagrams and we let oed range over OED. It is easy to check that the OED structures oed^{T}, oed^{cT}, and oed^{Tf} satisfy the actor laws and thus are OEDs. The empty OED, `mtoed`, is the structure:

$$\texttt{mtoed} = (\emptyset,\emptyset)(\emptyset,\emptyset,\texttt{mtmap},\texttt{mtrel},\texttt{mtmap},\texttt{mtmap},\emptyset)$$

Recalling the convention regarding interfaces:

$$iface(oed) = (recep(oed), extern(oed)) = (\rho,\chi)$$
$$OED(\rho,\chi) = \{oed \in OED \mid iface(oed) = (\rho,\chi)\}$$

4.2 The OED domain

We define a partial order, $\sqsubseteq$, on OED. Intuitively, $oed_0 \sqsubseteq oed_1$ means that oed_1 could possibly be reached from oed_0 by internal computation (expanding local pending events) or interaction with the environment. Thus oed_0 and oed_1 have same initial receptionists and externals, the internal actors and the events of oed_1 contain those of oed_0, the activation and arrival orders and the acquaintances and creation functions of oed_0 are restrictions of those of oed_1. Any event targeted to an actor of oed_0 that is a newly realized event of oed_1 must be latter in the arrival order of that actor than any event realized in oed_0. Also, the pending events of oed_0 must be accounted for, either as pending or realized events of oed_1. Finally, new pending events must be due to input from the environment or newly realized events—any pending event of oed_1 with activator a realized event of oed_0 or **!** must be a pending event of oed_0. More precisely, let

$$oed_j = (\rho_j,\chi_j)(A_j, E_j, activator_j, \xrightarrow{\mathrm{arr}}_j, acqB_j, Cr_j, P_j)$$

for $j < 2$, then

$$\begin{aligned}
&oed_0 \sqsubseteq oed_1 \;\Leftrightarrow\\
&\quad (\rho_0,\chi_0) = (\rho_1,\chi_1) \wedge A_0 \subseteq A_1 \wedge E_0 \subseteq E_1 \wedge P_0 \subseteq E_1 \cup P_1 \wedge\\
&\quad activator_0 = (activator_1)\lceil(E_0 \cup P_0) \wedge\\
&\quad \xrightarrow{\mathrm{arr}}_0 \text{ is an inital segment of } \xrightarrow{\mathrm{arr}}_1 \wedge\\
&\quad (\forall e \in P_1)(activator_1(e) \in E_0 \cup \{\mathbf{!}\} \Rightarrow e \in P_0)
\end{aligned}$$

Returning to the ticker, it is easy to check that $oed^{\mathrm{T}} \sqsubseteq oed^{\mathrm{cT}}$.

Lemma (po): $\sqsubseteq$ is a partial order on OED

Representing OEDs as limits

Let Oed be a directed subset of OED (every finite subset has an upper bound in Oed). $\sqcup Oed$ is the $\sqsubseteq$-least upper bound of Oed, if defined.

$$oed = \sqcup Oed \;\Leftrightarrow$$

$$(\forall oed' \in Oed)\, oed' \sqsubseteq oed \wedge$$
$$(\forall oed'' \in OED)((\forall oed' \in Oed)(oed' \sqsubseteq oed'') \Rightarrow oed \sqsubseteq oed'')$$

Because of the finite delay axiom, not all directed sets, or even chains have least upper bounds in OED. Clinger gives an example of a chain that has infinitely many minimal but incomparable upper bounds [[Cli81], pp. 97–98]. As another example, consider a chain of ticker diagrams where there is one input request that is never delivered and infinitely many ticks. Then there is no upper bound in OED since adding the input event would mean in the arrival order it would have infinitely many predecessors, and the result would have no global time.

Lemma (oed-lub): (Clinger [[Cli81], IV.Theorem 1, pp. 98].) If Oed is a directed subset of OED and $oed = \sqcup Oed$, then for each realized event, e of oed there is some $oed' \in Oed$ such that e is a realized event of oed'.

An OED, $oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P) \in OED$, is *complete* if $P = \emptyset$. In this case we may omit mention of P in the structure. $cOED$ denotes the set of complete OEDs. Note that the ticker event diagram oed^{cT} of §2 is complete.

Any OED can be represented as a limit of finite OEDs. To see this let

$$(oed, g) \in OED^{\dagger}$$
$$oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P) \in OED$$

and define $[oed_i^g \mid i \in \mathbf{N}]$ as follows.

$$oed_i^g = (\rho, \chi)(A_i^g, E_i^g, activator_i^g, \xrightarrow{\text{arr}}{}_i^g, acqB_i^g, Cr_i^g, P_i^g)$$

where

$$A_i^g = \{a \in A \mid g(Cr(a)) < i\}$$
$$E_i^g = \{e \in E \mid g(e) < i\}$$

$activator_i^g, \xrightarrow{\text{arr}}{}_i^g, acqB_i^g$ are the restrictions of $activator, \xrightarrow{\text{arr}}, acqB$ to E_i^g

Cr_i^g is the restriction of Cr to A_i^g

$$P_i = \{e \in E \cup P \mid g(activator(e')) < i \leq g(e)\}$$

Lemma (oed.lim):

(1) $oed_i^g \sqsubseteq oed_{1+1}^g$ for $i \in \mathbf{N}$

(2) $\sqcup \{oed_i^g \mid i \in \mathbf{N}\} = oed$

Since clearly E_i^g is finite for $i \in \mathbf{N}$ we have the desired limit. It is easy to construct the approximation sequence for the complete ticker diagram oed^{cT} of §2. We leave this as an exercise.

4.3 A component algebra on OEDs

Parallel Composition

Intuitively parallel composition of event diagrams (whose internal actors are disjoint) is formed by finding global times that match, in order, all outputs from one diagram to receptionists of the other with corresponding inputs. Then the two diagrams are essentially unioned. Pairs of synchronizing events coalesce into a single event. Finally the activator function is modified to replace unknown activators of synchronization inputs by the activators of the corresponding outputs. Composition of sets of event diagrams is just the compositions of all composable pairs.

We begin by defining composability and composition on $OED^\dagger$. Let $(oed_j, g_j) \in OED^\dagger$ for $j < 2$ where

$$oed_j = (\rho_j, \chi_j)(A_j, E_j, activator_j, \xrightarrow{\text{arr}}_j, acqB_j, Cr_j, P_j)$$

Composability is defined by

$$(oed_0, g_0) \bowtie (oed_1, g_1) \quad \text{iff}$$

$$A_0 \cap A_1 = \emptyset = P_0 \cap P_1 \ \wedge \ (\rho_0, \chi_0)g_0 \bowtie (\rho_1, \chi_1)g_1$$

Note that event path composability implies $E_j / A_{1-j} \subset (E_{1-j} \cup P_{1-j})$ for $j < 2$.

If $(oed_0, g_0) \bowtie (oed_1, g_1)$, then parallel composition is defined by

$$(oed_0, g_0) \parallel (oed_1, g_1) = (oed, g)$$

where

$$(\rho, \chi)g = (\rho_0, \chi_0)g_0 \parallel (\rho_1, \chi_1)g_1$$

$$oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P)$$

$$A = A_0 \cup A_1, \qquad P = P_0 \cup P_1,$$

$$E = (E_0 - E_0/A_1) \cup (E_1 - E_1/A_0)$$

$$\xrightarrow{\text{arr}} \ = \ \xrightarrow{\text{arr}}_0 \cup \xrightarrow{\text{arr}}_1 \qquad acqB = acqB_0 \cup acqB_1 \qquad Cr = Cr_0 \cup Cr_1$$

$$activator(e) = \begin{cases} activator_j(e) & \text{if } e \in E_j \cup P_j \text{ and} \\ & activator_j(e) = ? \ \Rightarrow \ g_j(\texttt{in}(e)) \notin \text{Rng}(g_{1-j}) \\ activator_{1-j}(e) & \text{if } g_j(\texttt{in}(e)) = g_{1-j}(\texttt{out}(e)) \end{cases}$$

Note that in the first activator case $g_j(\texttt{in}(e)))$ is not a synchronization point while in the second case it is. Since composability at the interaction path level implies satisfaction of $EPLaw$, and the pending events of the composition is the union of the pending events of the parts, we have composition on composable event diagram global time pairs is an event diagram global time pair and composition preserves completeness.

Lemma (oed.pcmps): If $(oed_j, g_j) \in OED^\dagger$ and $(oed_0, g_0) \bowtie (oed_1, g_1)$, then

$$(oed_0, g_0) \parallel (oed_1, g_1) = (oed, g) \in OED^\dagger$$

and if $oed_0, oed_1 \in cOED$, then $oed \in cOED$.
Two sets $Oed_j \subseteq OED(\rho_j, \chi_j)$ are composable if $\rho_0 \cap \rho_1 = \emptyset$. For composable Oed_j, parallel composition is defined by

$$\begin{aligned} Oed_0 \parallel Oed_1 = \{oed \mid (\exists (oed_j, g_j) \in Oed_j{}^{\dagger} \quad \text{for} \quad j < 2)(\exists g) \\ ((oed_0, g_0) \bowtie (oed_1, g_1) \wedge \\ (oed, g) = (oed_0, g_0) \parallel (oed_1, g_1))\} \end{aligned}$$

Lemma (oeds.pcmps): If $Oed_j \subseteq OED(\rho_j, \chi_j)$, $\rho_0 \cap \rho_1 = \emptyset$, then

$$Oed_0 \parallel Oed_1 \in OED((\rho_0, \chi_0) \parallel (\rho_1, \chi_1))$$

and if $Oed_j \subseteq cOED$, then $Oed_0 \parallel Oed_1 \subseteq cOED$.

Since the semantic sets we are interested in are closed under packet preserving isomorphisms on **E** the requirement that input/output events are identical, not just events with the same packet is not a serious restriction.

Restriction

Restriction, $oed \lceil \rho'$, is defined on OED structures by

$$\begin{aligned} &(\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P) \lceil \rho' = \\ &\quad (\rho \cap \rho', \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P) \end{aligned}$$

As an example recall the construction of the dispatcher OED from the switch and ticker OED, where the ticker is hidden after composition. Restriction maps OED structures to OED structures, but may not map OEDs to OEDs since the result my violate the event path law by having inputs to hidden receptionists. In order to remain an OED it must be the case that no inputs to an actor in $\rho - \rho'$ occur before these names have been exported. As an example consider an OED, oed, with interface $(\{a_0, a_1\}, \emptyset)$, an empty set of realized events, and pending events $\{e_0, e_1\}$ where $pkt(e_j) = a_j \triangleleft M_j$ and $activator(e_j) = ?$. Then $oed \lceil \{a_0\}$ has no global time since the event path law can not be satisfied.

If $Oed \subseteq OED(\rho, \chi)$, and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then the restriction of Oed to ρ' is defined by

$$Oed \lceil \rho' = \{oed \lceil \rho' \mid oed \in Oed \wedge oed \lceil \rho' \in OED\}$$

Lemma (oeds.restr): If $Oed \subseteq OED(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then $Oed \lceil \rho' \subseteq OED(\rho \cap \rho', \chi)$. Furthermore, if $Oed \subseteq cOED$, then $Oed \lceil \rho' \subseteq cOED$.

Renaming

Renaming is applied pointwise to the pieces of an OED structure using the assumed renaming action on message contents and events.

$$\begin{aligned} &\widehat{\alpha}((\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P)) = \\ &\quad (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))(\widehat{\alpha}(A), \widehat{\alpha}(E), \widehat{\alpha}(activator), \widehat{\alpha}(\xrightarrow{\text{arr}}), \widehat{\alpha}(acqB), \widehat{\alpha}(Cr), \widehat{\alpha}(P)) \end{aligned}$$

where for example we have $\widehat{\alpha}(activator)(\widehat{\alpha}(e)) = \widehat{\alpha}(activator(e))$.
If $Oed \subseteq OED(\rho, \chi)$ and α is a renaming, then the renaming of Oed by α is defined by

$$\widehat{\alpha}(Oed) = \{\widehat{\alpha}(oed) \mid oed \in Oed\}$$

Lemma (oeds.ren): If $Oed \subseteq OED(\rho, \chi)$ and α is a renaming, then

$$\widehat{\alpha}(Oed) \subseteq OED(\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))$$

and if $Oed \subseteq cOED$, then $\widehat{\alpha}(Oed) \subseteq cOED$.

Lemma (alg.oed): The OED component algebra of operations acting on sets of OEDs with a common interface satisfies the component algebra laws.

Corollary (alg.coed): The restriction of the parallel composition, restriction, and renaming operations to $cOED$ is a sub-algebra also satisfying the component algebra laws.

5 Interaction Diagrams

In this section the set, ID, of interaction diagrams is defined along with a component algebra structure on this set. A component algebra morphism from $cOED$ to ID is also defined.

5.1 Structure

Interaction diagrams (IDs) are abstractions of complete OEDs that hide the internal events. Interaction diagrams are structures of the form

$$id = (\rho, \chi)(I, O, \prec)$$

such that

$(\rho, \chi) \in \mathbf{Iface}$ —the interface

$I \subset \mathtt{in}(\mathbf{E})$ —input events

$O \subset \mathtt{out}(\mathbf{E})$ —output events

$\prec \ \subset I \times O$ —the visible combined ordering

and $(\exists g : I \cup O \xrightarrow{\text{inj}} \mathbf{N})$ such that g preserves $\prec$ and $EPLaw((\rho, \chi)g)$.
□

The empty interaction diagram is $\mathtt{mtid} = (\rho, \chi)(\emptyset, \emptyset, \mathtt{mtrel})$.

For $id = (\rho, \chi)(I, O, \prec)$, define the set of global times for id by

$$GT(id) = \{g : I \cup O \xrightarrow{\text{inj}} \mathbf{N} \mid g \text{ preserves } \prec \wedge EPLaw((\rho, \chi)g)\}$$

5.2 A Component Algebra on IDs

Parallel Composition

As for OEDs, composability and composition is first defined on $ID^\dagger$. Let $(id_j, g_j) \in ID^\dagger$ where $id_j = (\rho_j, \chi_j)(I_j, O_j, \prec_j)$ for $j < 2$. Then composability is defined by

$$(id_0, g_0) \bowtie (id_1, g_1) \quad \text{iff} \quad (\rho_0, \chi_0)g_0 \bowtie (\rho_1, \chi_1)g_1 .$$

If $(id_0, g_0) \bowtie (id_1, g_1)$, then parallel composition is defined by

$$(id_0, g_0) \parallel (id_1, g_1) = ((\rho, \chi)(I, O, \prec), g)$$

where

$$(\rho, \chi)g = (\rho_0, \chi_0)g_0 \parallel (\rho_1, \chi_1)g_1$$

$$I = \mathrm{Dom}(g) \cap (I_0 \cup I_1)$$

$$O = \mathrm{Dom}(g) \cap (O_0 \cup O_1)$$

$$\begin{aligned} \prec \; &= (\prec_0 \cup \prec_1)\lceil(I \times O) \cup \\ &\quad \{(\mathtt{in}(e), \mathtt{out}(e')) \mid (\exists e'', j < 2)(\mathtt{in}(e) \prec_j \mathtt{out}(e''), \\ &\qquad\qquad \mathtt{in}(e'') \prec_{1-j} \mathtt{out}(e'), \quad \text{and} \\ &\qquad\qquad g_j(\mathtt{out}(e'')) = g_{1-j}(\mathtt{in}(e''))\} \end{aligned}$$

The second component of $\prec$ corresponds to tracing links through the synchronization points. By definition of composability on event paths, g is a global time for id.

Lemma (id.pcmps): If $(id_j, g_j) \in ID^\dagger$ and $(id_0, g_0) \bowtie (id_1, g_1)$, then

$$(id_0, g_0) \parallel (id_1, g_1) \in ID^\dagger$$

Two semantic sets $Id_j \subseteq ID(\rho_j, \chi_j)$ are composable if $\rho_0 \cap \rho_1 = \emptyset$. For composable Id_j parallel composition is defined by

$$\begin{aligned} Id_0 \parallel Id_1 = \{id \mid (\exists (id_j, g_j \in {Id_j}^\dagger, \quad \text{for} \quad j < 2)(\exists g) \\ ((id_0, g_0) \bowtie (id_1, g_1) \;\wedge\; (id, g) = (id_0, g_0) \parallel (id_1, g_1))\} \end{aligned}$$

Lemma (ids.pcmps): If $Id_j \subseteq ID(\rho_j, \chi_j)$, $\rho_0 \cap \rho_1 = \emptyset$, then

$$Id_0 \parallel Id_1 \in ID((\rho_0, \chi_0) \parallel (\rho_1, \chi_1))$$

.

Restriction

If $id = (\rho, \chi)(I, O, \prec)$, then $id\lceil\rho' = (\rho \cap \rho', \chi)(I, O, \prec)$. In general the restriction of an interaction diagram may not be an interaction diagram, for the same reason as in the OED case.

If $Id \subseteq ID(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then the restriction of Id by ρ' is defined by

$$Id\lceil\rho' = \{id\lceil\rho' \mid id \in Id \ \wedge \ id\lceil\rho' \in ID\}$$

Lemma (ids.restr): If $Id \subseteq ID(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then $Id\lceil\rho' \subseteq ID(\rho \cap \rho', \chi)$.

Renaming

Renaming of interaction diagrams is pointwise as for event diagrams.

$$\widehat{\alpha}((\rho, \chi)(I, O, \prec)) = (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))(\widehat{\alpha}(I), \widehat{\alpha}(O), \widehat{\alpha}(\prec))$$

If $Id \subseteq ID(\rho, \chi)$, then the renaming of Id by α is defined by

$$\widehat{\alpha}(Id) = \{\widehat{\alpha}(id) \mid id \in Id\}$$

Lemma (ids.ren): If $Id \subseteq ID(\rho, \chi)$ and α is a renaming, then

$$\widehat{\alpha}(Id) \subseteq ID(\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))$$

Lemma (alg.id): The parallel composition, restriction and renaming operations on IDs satisfy the component algebra laws.

5.3 A Component Algebra Morphism from OED to ID

We define a map $oed2id$ from complete OEDs to IDs and show that this map is a component algebra morphism. The function $oed2id$ restricts attention to input and output events and the combined ordering on events and is defined for $oed \in cOED$ by

$$oed2id(oed) = (\rho, \chi)(I(oed), O(oed), \prec))$$

where $\prec$ the restriction of $\prec_{\mathrm{co}}(oed)$ to $I(oed) \times O(oed)$.

As an example, observe that the ticker interaction diagram, id^{T}, (§2) is the result of applying this mapping to the event diagram, oed^{T}: $id^{\mathrm{T}} = oed2id(oed^{\mathrm{T}})$.

$oed2id$ is lifted pointwise to sets of event diagrams. For $Oed \subset cOED(\rho, \chi)$ define

$$oed2id(Oed) = \{oed2id(oed) \mid oed \in Oed\}$$

Lemma (oed2id):

(0) $oed2id : cOED \to ID$

(1) $oed2id : \mathcal{P}[cOED(\rho, \chi)] \to \mathcal{P}[ID(\rho, \chi)]$ for $(\rho, \chi) \in \mathbf{Iface}$

The restriction of any global time of an OED, oed, is a global time of its $oed2id$ abstraction id. Furthermore every global time of id is the restriction of a global time of

oed, since there are only finitely many events in any combined order interval and thus delivery events can be inserted between i/o events to form the desired global time.

Lemma (oed2id.gt): If $oed \in cOED$ and $id = oed2id(oed)$, then
(1) $g \in GT(oed)$ implies $g\lceil(In(oed) \cup Out(oed)) \in GT(id)$
(2) $g \in GT(id)$ implies $g = g'\lceil(In(oed) \cup Out(oed))$ for some $g' \in GT(oed)$.

Thus we extend $oed2id$ to OED, global time pairs, defining

$$oed2id(oed, g) = (oed2id(oed), g\lceil(In(oed) \cup Out(oed)))$$

To establish that $oed2id$ is component algebra morphism, we first prove a lemma showing that $oed2id$ commutes with parallel composition on $cOED^\dagger$.

Lemma (oed2id.0): If $(oed_j, g_j) \in cOED^\dagger$ and $(oed_0, g_0) \bowtie (oed_1, g_1)$, then

(1) $oed2id(oed_0, g_0) \bowtie oed2id(oed_1, g_1)$

(2) $oed2id(oed_0, g_0) \parallel oed2id(oed_1, g_1) = oed2id((oed_0, g_0) \parallel (oed_1, g_1))$

Proof (oed2id.0): (1) is direct from the definitions. For (2) let

$$oed2id(oed_0, g_0) \parallel oed2id(oed_1, g_1) = (\rho, \chi)(I_l, O_l, \prec_l)$$

$$oed2id((oed_0, g_0) \parallel (oed_1, g_1)) = (\rho, \chi)(I_r, O_r, \prec_l)$$

$$S = \bigcup_{i \in synch((\rho_0,\chi_0)g_0,(\rho_1,\chi_1)g_1)} {g_0}^{-1}(i) \cup {g_1}^{-1}(i)$$

then from the defintions we have $I_l = I_r = In(oed_0) \cup In(oed_1) - S$ and $O_l = O_r = Out(oed_0) \cup Out(oed_1) - S$. Also

$$\begin{aligned} \prec_l \;=\; & (\prec_{\mathrm{co}0} \cup \prec_{\mathrm{co}1})\lceil(I_l \times O_l) \cup \\ & \{(\mathtt{in}(e), \mathtt{out}(e')) \mid (\exists e'', j < 2)\mathtt{in}(e) \prec_j \mathtt{out}(e''), \mathtt{in}(e'') \prec_{1-j} \mathtt{out}(e'), \\ & \qquad g_j(\mathtt{out}(e'')) = g_{1-j}(\mathtt{in}(e''))\} \end{aligned}$$

and $\prec_r$ is the transitive closure of $\xrightarrow{\mathrm{arr}}_0 \cup \xrightarrow{\mathrm{arr}}_1 \cup \prec_{\mathrm{ao}01}$ restricted to $I_l \times O_l$ where $\prec_{\mathrm{ao}01}$ is the least relation such that for $j < 2$

$$activator_j(e) \prec_{\mathrm{ao}01} e \quad \text{if } e \in (E_j - E_j/A_{1-j})$$

$$activator_{1-j}(e) \prec_{\mathrm{ao}01} e \quad \text{if } e \in E_j/A_{1-j}$$

Thus we see $\prec_l \;=\; \prec_r$. □

Theorem (oed2id.morph): The map $oed2id$ is a component algebra morphism. That is:

(1) If $\rho_0 \cap \rho_1 = \emptyset$ and $Oed_j \subseteq cOED(\rho_j, \chi_j)$, then $oed2id(Oed_0 \parallel Oed_1) = oed2id(Oed_0) \parallel oed2id(Oed_1)$.

(2) If $Oed \subseteq cOED(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then $oed2id(Oed\lceil\rho') = oed2id(Oed)\lceil\rho'$.

(3) If $Oed \subseteq cOED(\rho, \chi)$ and α is a renaming, then $oed2id(\widehat{\alpha}(Oed)) = \widehat{\alpha}(oed2id(Oed))$.

Proof (oed2id.morph): We prove (1), the remainder are easy. We let $Id_l = oed2id(Oed_0 \parallel Oed_1)$ and $Id_r = oed2id(Oed_0) \parallel oed2id(Oed_1)$, and show (1.1) $Id_l \subseteq Id_r$ and (1.2) $Id_r \subseteq Id_l$.

For 1.1 assume $id \in Id_l$.
Pick $oed \in (Oed_0 \parallel Oed_1)$ such that $oed2id(oed) = id$.
Pick $(oed_j, g_j) \in cOED^\dagger$ such that $oed_0, g_0 \bowtie oed_1, g_1$ and $(oed, g) = (oed_0, g_0) \parallel (oed_1, g_1)$.
Using **(o2i.0)** we have $oed2id(oed, g) = oed2id((oed_0, g_0) \parallel (oed_1, g_1)) = oed2id(oed_0, g_0) \parallel oed2id(oed_1, g_1)$ and hence $id \in Id_r$.
For 1.2 assume $id \in Id_r$
Pick $(id_0, g_0') \bowtie (id_1, g_1')$ such that $id_j \in oed2id(Oed_j)$ and $(id, g') = (id_0, g_0') \parallel (id_1, g_1')$.
Pick (oed_j, g_j) such that $oed_j \in Oed_j$ and $(id_j, g_j') = oed2id(oed_j, g_j)$ (using **(oed2id.gt)**)
Using **(o2i.0)** we have $(id, g') = oed2id(oed_0, g_0) \parallel oed2id(oed_1, g_1)) = oed2id((oed_0, g_0) \parallel (oed_1, g_1))$ and hence $id \in Id_l$.
□

6 Interaction paths

6.1 Structure

Interaction paths are event paths (see §3) in which the domain of the global time map contains no internal events. Thus interaction paths are structures of the form

$$(\rho, \chi)g$$

where $(\rho, \chi) \in \mathbf{Iface}$, $\mathrm{Dom}(g)) \subset (\mathtt{in}(\mathbf{E}) \cup \mathtt{out}(\mathbf{E}))$ and $EPLaw((\rho, \chi)g)$ holds. Interaction paths model possible global times and again abstract completed computations. The global time g is just the representation of a total, linear ordering on its domain. Thus we say ip and ip' are the same up to choice of order representation if $ip = (\rho, \chi)g$, $ip' = (\rho, \chi)g'$, where $\mathrm{Dom}(g) = \mathrm{Dom}(g')$, and the induced orderings, $<_g$ and $<_{g'}$, are the same.[1]

6.2 A component algebra on IPs

Parallel composition

Composability and composition on IP is just that defined for event paths. If $Ip_j \subseteq IP(\rho_j, \chi_j)$ for $j < 2$ and $\rho_0 \cap \rho_1 = \emptyset$ then parallel composition is defined by

$$Ip_0 \parallel Ip_1 = \{ip \mid (\exists ip_o \in Ip_0, ip_1 \in Ip_1)(ip_0 \bowtie ip_1 \wedge ip = ip_0 \parallel ip_1)\}$$

[1] Interaction paths are just the paths of interaction semantics [Tal96b] presented using injective functions from input/output events to numbers rather that functions from numbers to input/output events and silent events (corresponding to numbers not in the range). The two forms of presentation are equivalent.

Lemma (ips.pcmps): If $Ip_j \subseteq IP(\rho_j, \chi_j)$, $\rho_0 \cap \rho_1 = \emptyset$, then

$$Ip_0 \parallel Ip_1 \in IP((\rho_0, \chi_0) \parallel (\rho_1, \chi_1))$$

.

Restriction

If $ip = (\rho, \chi)(I, O, g)$, then $ip\lceil\rho' = (\rho \cap \rho', \chi)(I, O, g)$. As before the restriction of an interaction path may not be an interaction path, because of inputs to unexported receptionists.

If $Ip \subseteq IP(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then the restriction of Ip to ρ' is defined by

$$Ip\lceil\rho' = \{ip\lceil\rho' \mid ip \in Ip \wedge ip\lceil\rho' \in IP\}$$

Lemma (ips.restr): If $Ip \subseteq IP(\rho, \chi)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then $Ip\lceil\rho' \subseteq IP(\rho \cap \rho', \chi)$.

Renaming

Renaming of interaction paths is pointwise as for event diagrams.

$$\widehat{\alpha}((\rho, \chi)g) = (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))\widehat{\alpha}(g)$$

If $Ip \subseteq IP(\rho, \chi)$ and α is a renaming, then the renaming of Ip by α is defined by

$$\widehat{\alpha}(Ip) = \{\widehat{\alpha}(ip) \mid ip \in Ip\}$$

Lemma (ips.ren): If $Ip \subseteq IP(\rho, \chi)$ and α is a renaming, then

$$\widehat{\alpha}(Ip) \subseteq IP(\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))$$

Lemma (alg.ip): The parallel composition, restriction and renaming operations on IPs satisfy the component algebra laws.

6.3 A Component Algebra Morphism from ID to IP

We define a map $id2ip$ from IDs to IPs and show that this map is a component algebra morphism. This map produces for each interaction diagram the (non empty) set of interaction paths corresponding to its set of global times. For $(id, g) \in ID^\dagger$ define

$$id2ip(id, g) = (\rho, \chi)g \quad \text{where} \quad (\rho, \chi) = iface(id)$$

For $Id \subset ID(\rho, \chi)$ define the image under $id2ip$ by

$$id2ip(Id) = \{id2ip(id, g) \mid (id, g) \in Id^\dagger\}$$

Lemma (id2ip):

(0) $id2ip : ID^\dagger \to IP$
(1) $id2ip : \mathcal{P}[ID(\rho,\chi)] \to \mathcal{P}[IP(\rho,\chi)]$

Returning to the examples of §2 we see that the example ticker interaction paths are the result of mapping the interaction diagram using the example global times: $ip_j^{\mathrm{T}} = id2ip(id^{\mathrm{T}}, {g_j}^{\mathrm{T}})$ for $j < 2$.

As in the case of the $oed2id$ morphism, to establish that $id2ip$ is component algebra morphism, we first note that $id2ip$ commutes with parallel composition on $ID^\dagger$.

Lemma (id2ip.0.): If $(id_j, g_j) \in ID^\dagger$ and $(id_0, g_0) \bowtie (id_1, g_1)$, then

(0) $id2ip(id_0, g_0) \bowtie id2ip(id_1, g_1)$

(1) $id2ip((id_0, g_0) \parallel (id_1, g_1)) = id2ip(id_0, g_0) \parallel id2ip(id_1, g_1)$

Furthermore, for any $g' \in GT(id)$ we can find $g'_j \in GT(id_j)$ such that

$$id2ip(id_0, g'_0) \parallel id2ip(id_1, g'_1) = id2ip(id, g')$$

(up to choice of order representation).

Theorem (id2ip.morph): The map $id2ip$ is a component algebra morphism.

The proof is analogous to the proof of (**oed2id.morph**) and we omit it.

Corollary (oed2ip.morph): The map, $oed2ip$, from $cOED$ to IP defined by $oed2ip = id2ip \circ oed2id$ is a component algebra morphism.

7 Actor Theories and Their Models

Actor theories are a generalization of the notion of actor structures presented in [Tal96b, Tal96a]. Here we focus on a special form of actor theory we call HBC-actor theories. Details about the more general case may be found in [MT97]. HBC-actor theories correspond to the actor systems studied by Baker and Hewitt [BH77] and by Clinger[Cli81]. They have the property that an actor is always enabled to receive a message, and the processing of a message can be treated as an atomic step producing a finite number of new actors and messages.[2]

7.1 The Basics

An HBC actor theory (briefly an actor theory) is given by specifying the sets

A – of actor names

M – of message contents

E – of events

[2] We study HBC-actor theories here for simplicity. Actor theories such as those studied in [Tal96b, MT97] can be transformed into HBC-theories. Alternately the definitions given here can be generalized to accommodate a wider class. This is a topic for future work.

$\mathbf{S}$ – of actor states

and functions

$pkt : \mathbf{E} \to \mathbf{MP}$ – the packet function

$acq : \mathbf{M} \cup \mathbf{S} \to \mathcal{P}_\omega[\mathbf{A}]$ – the acquaintances function

$\widehat{_}$ – $\widehat{\alpha}$ lifts the renaming α to a bijection on $\mathbf{M} \cup \mathbf{S} \cup \mathbf{E}$, preserving sort

$\#\mathtt{a} : \mathbf{A} \times \mathbf{S} \times \mathbf{M} \to \mathbf{N}$

$deliver : \mathbf{A} \times \mathbf{S} \times \mathbf{M} \to \mathbf{A}^* \overset{\circ}{\to} \mathbf{S} \times \mathbf{S}^* \times \mathbf{MP}^*$

satisfying the renaming and acquaintance axioms (**HBC-ax**) below. In the following we take $\mathbf{A}$, $\mathbf{M}$, $\mathbf{E}$, to be the sets introduced in §3 and assume that pkt, acq, and $\widehat{\alpha}$ are the functions introduced in §3 extended to act on $\mathbf{S}$. Actor states, $\mathbf{S}$, are the new entities. They are intended to carry information traditionally contained in the script (methods) and acquaintances (values of instance variables), as well as the local message queue and the current processing state. We let s range over $\mathbf{S}$. We also let $\vec{a}$ range over sequences of distinct actor names and $\vec{e}$ range over sequences of distinct events. (Note that there may be multiple events with the same packet in a such an event sequence.)

$deliver(a, s, M)(\vec{a})$ specifies the response of an actor, a, in state, s, upon receiving a message with contents, M and using elements of $\vec{a}$ to name newly created actors. $\#\mathtt{a}(a, s, M)$ is the number of actors that will be created. $deliver(a, s, M)(\vec{a})$ is defined just if $\mathrm{Len}(\vec{a}) = \#\mathtt{a}(a, s, M)$. If

$$deliver(a, s, M)(\vec{a}) = (s', [s_i \mid 1 \leq i \leq k], [mp_i \mid 1 \leq i \leq n])$$

then $k = \#\mathtt{a}(a, s, M)$, s' is the new state of a, s_i is initial state of the i-th new actor, and $[mp_i \mid 1 \leq i \leq n]$ is the list of message packets sent.

An actor theory describing the ticker and ticker factory can be given by defining the delivery function as follows. Let Tf be the ticker factory state. Then

$$\#\mathtt{a}(t, T(n), \mathtt{tick}) = \#\mathtt{a}(t, T(n), c) = 0$$

$$deliver(t, T(n), \mathtt{tick})[\,] = (T(n+1), [\,], [t \triangleleft \mathtt{tick}])$$

$$deliver(t, T(n), c)[\,] = (T(n), [\,], [c \triangleleft n]) \quad \text{if } c \in \mathbf{A}$$

$$\#\mathtt{a}(t, Tf, \mathtt{tick}) = 1$$

$$deliver(tf, Tf, \mathtt{new}(c))[t] = (Tf, [T(0)], [c \triangleleft t])$$

To state the axioms we introduce two additional sets: $\mathbf{AE}$—actor entities (briefly actors); and $\mathbf{I}$—configuration interiors (briefly interiors).

$$\mathbf{AE} = [\mathbf{S}]_{\mathbf{A}}$$

$$\mathbf{I} \subset \mathcal{P}_\omega[\mathbf{AE} \cup \mathbf{E}]$$

$[\,s\,]_a$ is an *actor* with *name*, a, in *state*, s. A *configuration interior*, I, is a set of actors and events (message occurrences) in which no two actor entities have the same name. We let I range over $\mathbf{I}$. We define functions $InAct$, $ExtAct$, $Undel$, $aState$ on $\mathbf{I}$. $InAct(I)$ is the set of names of actors that occur in I, and $ExtAct(I)$ is the set of names of external actors referred to in I. $Undel(I)$ is the set of undelivered messages of I and $aState(I, a)$ is the state of a in I if $a \in InAct(I)$.

$$InAct(I) = \{a \in \mathbf{A} \mid (\exists s \in \mathbf{S})([\,s\,]_a \in I)\}$$

$$ExtAct(I) = acq(I) - InAct(I)$$

$$Undel(I) = I \cap \mathbf{E}$$

$$aState(I, a) = s \quad \text{if } [\,s\,]_a \in I$$

Renaming and Acquaintance Axioms (HBCax)

(i) $acq(\widehat{\alpha}(v)) = \widehat{\alpha}(acq(v)) \quad \text{for} \quad v \in \mathbf{S} \cup \mathbf{M} \cup \mathbf{E}$

(ii) $(\forall a \in acq(v))(\alpha(a) = a) \Rightarrow \widehat{\alpha}(v) = v \quad \text{for} \quad v \in \mathbf{S} \cup \mathbf{M} \cup \mathbf{E}$

(iii) $\widehat{\alpha_0 \circ \alpha_1} = \widehat{\alpha_0} \circ \widehat{\alpha_1}$

(iv) $deliver(\alpha(a), \widehat{\alpha}(state), \widehat{\alpha}(M))(\alpha(\vec{a})) = \widehat{\alpha}(deliver(a, s, M)(\vec{a}))$

(v) $(acq(s') \cup \bigcup_{1 \le i \le k} acq(s_i) \cup \bigcup_{1 \le i \le n} acq(mp_i)) \subseteq (\{a\} \cup acq(s) \cup acq(M) \cup \vec{a})$

$$\text{if } deliver(a, s, M)(\vec{a}) = (s', [s_i \mid 1 \le i \le k], [mp_i \mid 1 \le i \le n])$$

(i) states that renaming commutes with the acquaintance function. (ii) states that if a renaming fixes the acquaintances of an object then applying it does not change the object. (iii) states that the renaming mechanism commutes with function composition. (ii-iii) imply that two renamings that agree on the acquaintances of an object have the same result when applied to the object, and that $\widehat{\alpha}$ is a bijection on $\mathbf{S} \cup \mathbf{M} \cup \mathbf{E}$. (iv) states that the $deliver$ function commutes with renaming. (v) is the key acquaintance axiom, which ensures that the locality laws hold for computations of actor theory configurations.

7.2 Configurations and Computations

An actor system configuration, $(\rho, \chi)(I \mid E)$, consists of an interface, (ρ, χ), a configuration interior, I, and a set of previously delivered messages E. We require that the receptionists be a subset of the internal actors and that the external acquaintances of I and E be a subset of the externals. Thus the set of configurations is defined by

$$\mathbf{K} = \{(\rho, \chi)(I \mid E) \mid \rho \subseteq InAct(I) \wedge ExtAct(I) \subseteq \chi \wedge acq(E) \subseteq InAct(I) \cup \chi\}$$

We let K range over $\mathbf{K}$. We extend $InAct$, $ExtAct$, $Undel$, and $aState$ to configurations in the obvious manner. The set E of previously delivered messages is included in order to facilitate the mapping between event diagrams and computations by allowing us to locally ensure that each event input or sent by an internal actor is uniquely

identifiable throughout the computation. A *starting* configuration is one in which E is empty.

Algebra of configurations

The component algebra operations on configurations are defined as follows. For two configurations to compose they must have disjoint sets of internal actors and external references of one configuration to internal actors of the other must be in the receptionists set. Furthermore no undelivered events of one configuration may appear as events (delivered or undelivered) in the other, and any past event of one configuration with target a receptionist of the other must appear as an event in the other. Let $K_j = (\rho_j, \chi_j)(I_j \mid E_j)$. Then

$$
\begin{aligned}
& K_0 \bowtie K_1 \;\Leftrightarrow \\
& \quad \mathit{InAct}(I_0) \cap \mathit{InAct}(I_1) = \emptyset \;\wedge\; \textstyle\bigwedge_{j<2} \chi_j \cap \mathit{InAct}(I_{1-j}) \subseteq \rho_{1-j} \\
& \quad \textstyle\bigwedge_{j<2} (\mathit{Undel}(I_{1-j}) \cap (E_j \cup \mathit{Undel}(I_j)) = \emptyset \;\wedge\; E_{1-j}/\rho_j \subseteq (E_j \cup \mathit{Undel}(I_j))) \\
& (\rho_0, \chi_0)(I_0 \mid E_0) \parallel (\rho_1, \chi_1)(I_1 \mid E_1) = (\rho, \chi)(I_0 \cup I_1 \mid E_0 \cup E_1) \\
& \quad \text{where} \quad (\rho, \chi) = (\rho_0, \chi_0) \parallel (\rho_1, \chi_1) \\
& \quad \text{if } (\rho_0, \chi_0)(I_0 \mid E_0) \bowtie (\rho_1, \chi_1)(I_1 \mid E_1) \\
& (\rho, \chi)(I \mid E) \lceil \rho' = (\rho \cap \rho', \chi)(I \mid E) \\
& \widehat{\alpha}((\rho, \chi)(I \mid E)) = (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))(\widehat{\alpha}(I) \mid \widehat{\alpha}(E))
\end{aligned}
$$

Computations

The computations of a configuration are given by a labelled transition relation with elements of the form $K \xrightarrow{l} K'$. K is the *source* of the transition and K' is the *target* and l is the *label*. Transition labels are either delivery labels, input/output labels, or a special label, `idle`. A delivery label, $\mathtt{d}(e, \vec{a}, \vec{e})$, indicates delivery of an event, e, whose target is an internal actor. An input label, $\mathtt{in}(e)$, indicates arrival of e from the environment. An output label, $\mathtt{out}(e)$, indicates transmission of e to the environment. The idle label, `idle` indicates no action. The transition rules are the following.

(d) $$
\begin{aligned}
& (\rho, \chi)(I \;,\; [\,s\,]_a \;,\; e \mid E) \xrightarrow{\mathtt{d}(e, \vec{a}, \vec{e})} (\rho, \chi)(I, I' \mid E \;,\; e) \\
& \quad \text{if} \\
& \quad \vec{a} \cap (\mathit{InAct}(I \;,\; [\,s\,]_a \;,\; e) \cup \chi) = \emptyset = \vec{e} \cap (\mathit{Undel}(I \;,\; [\,s\,]_a \;,\; e) \cup E) \\
& \quad I' = [\,s'\,]_a \;,\; \{[\,s_i\,]_{a_i} \mid 1 \le i \le \mathrm{Len}(\vec{a})\} \;,\; \vec{e} \\
& \quad \mathit{deliver}(a, s, mc(e))(\vec{a}) = \\
& \qquad (s', [s_i \mid 1 \le i \le \mathrm{Len}(\vec{a})], [\mathit{pkt}(e_i) \mid 1 \le i \le \mathrm{Len}(\vec{e})])
\end{aligned}
$$

(i) $$(\rho, \chi)(I \mid E) \xrightarrow{\mathtt{in}(e)} (\rho, \chi')(I \;,\; e \mid E) \quad \text{where} \quad \chi' = \chi \cup (\mathit{acq}(mc(e)) - \rho)$$

$$\text{if } e \notin E \cup Undel(I) \ \wedge \ acq(mc(e)) \cap InAct(I) \subseteq \rho \ \wedge \ tgt(e) \in \rho$$

(o) $\quad (\rho, \chi)(I \ , \ e \mid E) \xrightarrow{\texttt{out}(e)} (\rho', \chi)(I \mid E \ , \ e)$

$$\text{where} \quad \rho' = \rho \cup (acq(mc(e)) \cap InAct(I)) \quad \text{if } tgt(e) \in \chi$$

(τ) $\quad K \xrightarrow{\texttt{idle}} K$

Notice that the transition is relation is deterministic in the sense that if $K \xrightarrow{l} K_j$ for $j < 2$, then $K_0 = K_1$.

Using the deliver function defined above for the ticker we see that the reaction rules given in §2 are simply delivery transitions for the case I is empty. Such rules provide an alternative form of presenting an actor theory.

A *computation* is a sequence of transitions such that the target of the i-th element is the source of the $i+1$-st element. We require that the initial configuration of a computation have no delivered messages, i.e. that it be a starting configuration. Using the idle transition we may without loss consider only infinite computations. Thus a computation, π, has the form

$$\pi = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}]$$

where K_0 is a starting configuration. Since the transition relation is deterministic, π is determined by K_0 and the sequence of labels, $[l_i \mid i \in \mathbf{N}]$. A finite computation is a sequence that is idle from some point on. $Tree(K)$ is the set of finite computations with initial configuration K. A fair computation is one in which every message eventually gets delivered. Any number of input transitions may happen (including none or infinitely many). Only when a message is in the system is it required to be delivered. $Fair(K)$ is the set of fair computations with initial configuration K. $Path(K)$ is the set of finite or fair computations for K: $Path(K) = Tree(K) \cup Fair(K)$.

The sequence of transitions, π^{T}, given in §2 is a finite computation for the ticker starting configuration (ignoring the idle transitions). To be a fair computation it must be extended with an infinite sequence of ticks.

For a computation, $\pi = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}]$, we define $\widetilde{E}(\pi)$ to be the set of extended events corresponding to non-idle labels, and g^π to be the corresponding global time on this set.

$$\widetilde{E}(\pi) = \{\tilde{e} \in \widetilde{\mathbf{E}} \mid (\exists i)(l_i = \tilde{e} \ \vee \ (\exists \vec{a}, \vec{e}) l_i = \mathtt{d}(\tilde{e}, \vec{a}, \vec{e})\}$$

$$g^\pi : \widetilde{E}(\pi) \xrightarrow{\text{inj}} \mathbf{N} \quad \text{such that} \quad g^\pi(\tilde{e}) = i \quad \text{if } l_i = \tilde{e} \ \vee \ (\exists \vec{a}, \vec{e}) l_i = \mathtt{d}(\tilde{e}, \vec{a}, \vec{e})$$

g^π is well defined since each event appears at most once as the event of a label in a computation.

7.3 Component algebra on computations

The definitions of composability and parallel composition for computations is an adaption of the definitions given in [Tal96a] to account for the addition of the delivered mes-

sages component of configurations. Let $\pi_j = [K_i^j \xrightarrow{l_i^j} K_{i+1}^j \mid i \in \mathbf{N}] \in Path(K_j)$ for $j < 2$ with $K_i^j = (\rho_i^j, \chi_i^j)(I_i^j \mid E_i^j)$. Two computation are composable just if corresponding configurations are composable and the associated event paths are composable.

$$\pi_0 \bowtie \pi_1 \;\Leftrightarrow\; K_i^0 \bowtie K_i^1 \quad \text{for} \quad i \in \mathbf{N}, \quad \text{and} \quad (\rho_0^0, \chi_0^0) g^{\pi_0} \bowtie (\rho_0^1, \chi_0^1) g^{\pi_1}$$

For composable paths, parallel composition is defined by

$$\pi_0 \parallel \pi_1 = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}]$$

where

$$K_0 = K_0^0 \parallel K_0^1 \quad \text{and} \quad l_i = \begin{cases} l_i^j & \text{if } l_i^{1-j} = \texttt{idle} \\ \texttt{idle} & \text{if } l_i^0 \text{ and } l_i^1 \text{ are complements} \end{cases}$$

and K_{i+1} is defined by induction using l_i and the fact that the transition system is deterministic. Note that K_i is the restriction of $K_i^0 \parallel K_i^1$ to $\rho_i \subseteq \rho_i^0 \cup \rho_i^1$. The restriction is non-trivial if some receptionist exported by one of the computations is not exported in the composition.

If $K_0 \bowtie K_1$ and $P_j \subseteq Path(K_j)$ for $j < 2$ then parallel composition is defined by

$$P_0 \parallel P_1 = \{\pi_0 \parallel \pi_1 \mid \bigwedge_{j<2} (\pi_j \in P_j) \wedge \pi_0 \bowtie \pi_1\}$$

Restriction and renaming are defined in the obvious manner and the details are omitted.

7.4 Mapping Computations to OEDs

Now we define a mapping $c2oed$ from $Path(K)$ to $OED(iface(K))^\dagger$, which preserves finiteness and fairness. The event diagram is obtained by mapping finite initial stages of π and then taking the limit. The global time is just g^π. In §8 we show that the image of $Path(K)$ is $OED(K)$. Let

$$\pi = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}] \in Path(K)$$

where $K_i = (\rho_i, \chi_i)(I_i \mid E_i)$ (recall $E_0 = \emptyset$). Then $c2oed(\pi)$ is defined by

$$c2oed(\pi) = (oed, g^\pi) \quad \text{where} \quad oed = \sqcup_{i \in \mathbf{N}} c2oed(\pi, i)$$

and

$$c2oed(\pi, i) = (\rho, \chi)(A_i, E_i, activator_i, \xrightarrow{\text{arr}}_i, acqB_i, Cr_i, P_i)$$

where

$$(\rho, \chi) = (\rho_0, \chi_0)$$

$$A_i = InAct(I_i)$$

$$acqB_i(e) = acq(aState(I_k, tgt(e))) \quad \text{if } l_k = \texttt{d}(e, \ldots) \qquad \text{with } k < i$$

$$Cr_i(a) = e_k \quad \text{if } a \in A_{k+1} - A_k$$

and E_i, P_i, $activator_i$, and $\xrightarrow{\text{arr}}_i$ are defined by induction on i and cases according to l_i as follows.

$E_0 = \emptyset \qquad P_0 = Undel(I_0) \qquad activator_0(e) = \mathbf{!} \quad \text{if } e \in P_0 \qquad \xrightarrow{\text{arr}}_0 = \texttt{mtrel}$

If $l_i = \texttt{d}(e, \vec{a}, \vec{e})$, then

$E_{i+1} = E_i \cup \{e\} \qquad P_{i+1} = P_i - \{e\} \cup \vec{e}$

$activator_{i+1} = activator_i \cup \{e_k \mapsto e \mid 1 \leq k \leq \text{Len}(\vec{e})\}$

$\xrightarrow{\text{arr}}_{i+1} / a \;=\; \xrightarrow{\text{arr}}_i / a * [e]$ making e the latest in the arrival ordering of a

If $l_i = \texttt{in}(e)$, then

$E_{i+1} = E_i \qquad P_{i+1} = P_i \cup \{e\}$

$activator_{i+1} = activator_i \cup \{e \mapsto \mathbf{?}\} \qquad \xrightarrow{\text{arr}}_{i+1} \;=\; \xrightarrow{\text{arr}}_i$

If $l_i = \texttt{out}(e)$, then

$E_{i+1} = E_i \cup \{e\} \qquad P_{i+1} = P_i - \{e\}$

$activator_{i+1} = activator_i \qquad \xrightarrow{\text{arr}}_{i+1} \;=\; \xrightarrow{\text{arr}}_i$

If $l_i = \texttt{idle}$, then

$E_{i+1} = E_i \qquad P_{i+1} = P_i \qquad activator_{i+1} = activator_i \qquad \xrightarrow{\text{arr}}_{i+1} = \xrightarrow{\text{arr}}_i$

□

Lemma (c2oed): For $\pi \in Path(K)$ and $(oed, g) = c2oed(\pi)$

(0) $\widetilde{E}(oed) = \widetilde{E}(\pi)$

(1) $(oed, g) \in OED(iface(K))^{\dagger}$

(2) $oed \in cOED$ just if $\pi \in Fair(K)$

Continuing the example of §2, we see that $oed^{\mathrm{T}} = c2oed(\pi^{\mathrm{T}})$.

As usual, to show that $c2oed$ is a component algebra morphism the main work is to establish that it commutes with parallel composition of paths.

Lemma (c2oed.pcmps): If $K_0 \bowtie K_1$ and $\pi_j \in Path(K_j)$, then
(a) $\pi_0 \bowtie \pi_1$ iff $c2oed(\pi_0) \bowtie c2oed(\pi_1)$
(b) if $\pi_0 \bowtie \pi_1$, then $c2oed(\pi_0) \parallel c2oed(\pi_1) = c2oed(\pi_0 \parallel \pi_1)$

Theorem (c2oed.morph): The map $c2oed$ is a configuration algebra morphism. That is for starting configurations, K, K_0, K_1

(1) If $K_0 \bowtie K_1$ and $P_j \subseteq Path(K_j)$, then $c2oed(P_0 \parallel P_1) = c2oed(P_0) \parallel c2oed(P_1)$.

(2) If $P \subseteq Path(K)$ and $\rho' \in \mathcal{P}_\omega[\mathbf{A}]$, then $c2oed(P \lceil \rho') = c2oed(P) \lceil \rho'$.

(3) If $P \subseteq Path(K)$ and α is a renaming, then $c2oed(\widehat{\alpha}(P)) = \widehat{\alpha}(c2oed(P))$.

8 Semantics for Configurations

In this section semantic maps $\mathcal{S}$ from configurations to the semantic domains over OED, $cOED$, ID, and IP are defined and shown to be component algebra morphisms. The main work is defining $OED(K)$ and showing $OED(K) = c2oed(Path(K))$. The rest follows from the various morphism results.

8.1 The OED semantics of configurations

Intuitively an OED, oed, is an event diagram for a configuration K if it has the same interface, initial actors and messages and we can associate to each actor of oed an initial state (in $\mathbf{S}$), and to each internal event a post event state for the target and an enumeration of the actors created and events activated in a manner consistent with the delivery function $deliver$. Clearly, the initial state of actors present at start time is given by their state in K.

For K a starting configuration, the set, $OED(K) \subset OED(recep(K), extern(K))$, of event diagrams for K is defined as follows.

$$
\begin{aligned}
&oed = (\rho, \chi)(A, E, \prec_{\mathrm{ao}}, \xrightarrow{\mathrm{arr}}, acqB, Cr, P) \in OED(K) \;\Leftrightarrow \\
&\quad oed \in OED(iface(K)) \\
&\quad Cr^{-1}(!) = InAct(K) \\
&\quad activator^{-1}(!) = Undel(K) \\
&\quad (\exists ae : Dlv(oed) \to \mathbf{A}^*, me : Dlv(oed) \to \mathbf{E}^*, iState : A \to \mathbf{S}, sAfter : Dlv(oed) \to \mathbf{S}) \\
&\qquad (\forall e \in Dlv(oed))(ae(e) \text{ enumerates } crby(e)), \\
&\qquad (\forall e \in Dlv(oed))(me(e) \text{ enumerates } sentby(e)), \quad \text{and} \\
&\qquad oed \in OED(K, ae, me, iState, sAfter)
\end{aligned}
$$

where $oed \in OED(K, ae, me, iState, sAfter)$ holds iff

(1) $iState(a) = aState(K, a)$ if $Cr(a) = !$

(2) if $e \in Dlv(oed)$, $tgt(e) = a$, $ae(e) = \vec{a}$, $me(e) = \vec{e}$,

$$
s = \begin{cases} iState(a) & \text{if } rank(e)=0 \\ sAfter(e') & \text{if } tgt(e') = a \text{ and } rank(e') = rank(e) - 1 \end{cases}\text{'}
$$

$$
\begin{aligned}
&deliver(a, s, mc(e))(\vec{a}) = (s', \vec{s}, \vec{mp}) \text{ where} \\
&\quad sAfter(e) = s' \\
&\quad \vec{s} = [iState(a_i) \mid 1 \le i \le \mathrm{Len}(\vec{a})] \\
&\quad \vec{mp} = [pkt(e_i) \mid 1 \le i \le \mathrm{Len}(\vec{e})]
\end{aligned}
$$

Note that if $oed \in OED(K, ae, me, iState, sAfter)$ then $iState, sAfter$ are uniquely determined.

For K a starting configuration define $cOED(K) = OED(K) \cap cOED$

Example: Ticker OEDs

As an example of OED semantics we describe the set of event diagrams for the Ticker system of §2. Recall the starting ticker configuration $Ticker[t] = (t, \emptyset)(\llbracket T(0) \rrbracket_t \,,\, t \triangleleft \texttt{tick} \,|\, \emptyset)$. Then

$$OED(Ticker[t]) =$$

$$\{(t, \emptyset)(\{t\}, E, activator, \xrightarrow{\text{arr}}, \lambda e \in E/t.\{t\}, \lambda a \in A.\mathbf{!}, P) \in OED \mid \Phi\}$$

where Φ holds just if

(1) there is some $e \in E \cup P$ such that $activator^{-1}(\mathbf{!}) = \{e\}$ and $pkt(e) = t \triangleleft \texttt{tick}$.

(2) if $e \in E/t$, then

(2.1) if $m(e) = \texttt{tick}$, then there is some $e' \in E \cup P$ such that $sentby(e) = \{e'\}$ and $pkt(e') = t \triangleleft \texttt{tick}$

(2.2) if $m(e) = c$ with $c \in \mathbf{A}$, then there is some $e' \in E \cup P$ such that $sentby(e) = \{e'\}$, $pkt(e') = c \triangleleft n + k$ where $k = |(\{e'' \in E/t \mid mc(e'') = \texttt{tick} \wedge rank(e'') < rank(e)\})|$

(2.3) ow $sentby(e) = \emptyset$

Lemma (Ticker OED properties): $OED(Ticker[t])$ is closed under packet preserving isomorphisms on $\mathbf{E}$, and if $oed \in OED(Ticker[t])$ as defined above, then

(1) either E is infinite or there is at least one pending event with packet $t \triangleleft \texttt{tick}$,

(2) if $e \in (E \cup P)/t$ and $m(e) \neq \texttt{tick}$, then $activator(e) = \mathbf{?}$ or e has packet $t \triangleleft n'$ and $m(activator(e)) = t$

8.2 An Alternative Characterization of OED semantics

Now we show that the image of $c2oed$ on the computations of a configuration is the set of OEDs for that configuration.

Lemma (c2oed):
(1) $OED(K) = \bigcup_{\pi \in Path(K)} c2oed(\pi)$
(2) $cOED(K) = \bigcup_{\pi \in Fair(K)} c2oed(\pi)$

Proof : (2) follows from (1) and the fact that $c2oed$ preserves fairness. For (1) first consider containment of the right-hand side in the left-hand side. Assume $\pi \in Path(K)$ and let $oed = c2oed(\pi)$. Clearly the initial element conditions hold. Furthermore the $iState$ and $sAfter$ functions are given by the configuration following the event and the enumeration functions are given by the labels $\texttt{d}(e, \vec{a}, \vec{e})$ associated with delivery events. For containment in the other direction we define a function, $oed2c$, taking an element

of $OED(K)$ together with existential witnesses and a global time and producing a computation π of K such that $c2oed(\pi) = oed$. Let

$$(oed, g) \in OED^{\dagger} \quad \text{where}$$

$$oed = (\rho, \chi)(A, E, \prec_{\mathrm{ao}}, \xrightarrow{\mathrm{arr}}, acqB, Cr, P) \in OED(K)$$

and pick $ae, me, iState, sAfter$ such that $oed \in OED(K, ae, me, iState, sAfter)$.

$$oed2c(oed, K, ae, me, g) = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}]$$

where

$$K_i = (\rho_i, \chi_i)(I_i \mid E_i) \qquad \rho_i = recep((\rho, \chi)g, i) \qquad \chi_i = extern((\rho, \chi)g, i)$$

$$l_i = \begin{cases} \mathtt{in}(e) & \text{if } g(\mathtt{in}(e)) = i \\ \mathtt{out}(e) & \text{if } g(\mathtt{out}(e)) = i \\ \mathtt{d}(e, ae(e), me(e)) & \text{if } g(e) = i \\ \mathtt{idle} & \text{if } i \notin \mathrm{Rng}(g) \end{cases}$$

$$InAct(I_i) = \{a \in A \mid g(Cr(a)) < i\}$$

$$aState(I_i, a) = \begin{cases} sAfter(g^{-1}(i')) & \text{if } J \neq \emptyset, i' = max(J) \\ iState(a) & \text{if } J = \emptyset \end{cases}$$

where $\quad J = \{e \in Dlv(oed)/a \mid g(e) < i\}$

$$Undel(I_i) = \{e \mid e \in Dlv(oed) \cup Out(oed) \cup P \wedge g(activator(e)) < i \leq g(e)\}$$

$$E_i = \{e \mid e \in Dlv(oed) \cup Out(oed) \wedge g(e) < i\}$$

Then letting $\pi = oed2c(oed, K, ae, me, iState, sAfter, g)$ we claim that

(1) $\quad \pi \in Path(K)$

(2) $\quad \pi \in Fair(K) \quad$ if $oed \in cOED$

(3) $\quad c2oed(\pi) = oed \wedge g = g^{\pi}$

To prove (1) we must show that $K_i \xrightarrow{l_i} K_{i+1}$ is a transition for each i. This is trivial if $i \notin \mathrm{Rng}(g)$. Otherwise the argument is by cases on the extended event type. We omit the details. (3) is a straightforward calculation.

8.3 ID and IP semantics

The semantics $ID(K)$ and $IP(K)$ are defined by

$$ID(K) = oed2id(cOED(K)) = \{oed2id(oed) \mid oed \in cOED(K)\}$$

$$IP(K) = id2ip(ID(K)) = \{id2ip(id) \mid id \in ID(K)\}$$

As for OEDs we can directly define maps from computations to interaction diagrams and paths and arrive at equivalent definitions for the semantic maps.

Ticker ID semantics

As an example, the interaction diagram semantics of the ticker system is given by the following.

$$ID(Ticker[t]) = \{id = (t, \emptyset)(I, O \prec) \in ID \mid \Psi\}$$

where Ψ holds just if $m(e) \in \mathbf{N}$ for $\mathtt{out}(e) \in O$ and letting $AE = \{\mathtt{in}(e) \in I \mid m(e) \in \mathbf{A} - \{t\}\}$, there exists a total order $\xrightarrow{\text{arr}}_t$ on AE, and a bijection B from AE to O such that for $\mathtt{in}(e) \in AE$

(1) $\mathtt{in}(e) \prec B(\mathtt{in}(e)$ and $m(e) = tgt(B(\mathtt{in}(e))$

(2) $\mathtt{in}(e) \prec \mathtt{out}(e')$ implies $\mathtt{out}(e') = B(\mathtt{in}(e))$ or $\mathtt{in}(e) \xrightarrow{\text{arr}}_t B^{-1}(\mathtt{out}(e'))$

(3) $\mathtt{in}(e) \xrightarrow{\text{arr}}_t \mathtt{in}(e')$ implies $m(B(\mathtt{in}(e))) \leq m(B(\mathtt{in}(e')))$

(4) if AE is infinite, then for all $n \in \mathbf{N}$ there is some $\mathtt{out}(e) \in O$ and $n' \geq n$ such that $m(e) = n'$

The bijection B and the total ordering $\xrightarrow{\text{arr}}_t$ above correspond to the 'in response to' relation and the visible part of the arrival ordering of the underlying event diagram. Again the set $ID(Ticker[t])$ is closed under packet preserving isomorphisms on $\mathbf{E}$.

8.4 Distinctions

We give some examples to show that the different semantic models give rise to different notions of equivalence of configurations.

Path vs OED: Consider two actors with different states that give rise to computations that differ only in a the representation of states. A simple example would be actors that compute factorial using different sequential algorithms. The semantics of these actors (considered as single actor systems) will have different $Path$ and $Fair$ semantics but the same OED semantics.

OED vs IP and ID: Consider actors that compute factorial using different internal patterns of message passing. For example one may use a sequential algorithm, another may create a new actor for each request and that actor implements tail recursion by sending messages to itself. These actors will have different OED semantic sets, but the same ID or IP semantics.

IP vs ID: Let $[S(c)]_b$ be an actor with name b that sends a message to c containing some constant in response to the first message received and ignores all other messages. Let $[N]_a$ be an actor that ignores all messages. Define

$$K_0 = (a, c)([N]_a, [S(c)]_b, b \triangleleft b \mid \emptyset)$$

$$K_1 = (a, c)([S(b)]_a, [S(c)]_b, b \triangleleft b \mid \emptyset)$$

then $IP(K_0) = IP(K_1)$ is the set of interaction paths containing any number of inputs to a and exactly one output to c. The interaction diagrams of K_0 and K_1 also have any

number of inputs to a and exactly one output to c. However the ordering $\prec_0$ in the case of K_0 is always the empty relation while for K_1 there will be orderings with a non empty set of inputs $\prec_1$ the single output when the message from a to b caused by the input arrives before the initially present message.

8.5 Compositionality

We have defined configuration semantics $\mathcal{S}(K)$ for a variety of semantics $\mathcal{S}$. We have also defined a component algebra structure on the associated semantic domains and morphisms relating these algebras. Finally we establish composability for these semantics. For $\mathcal{S}$ one of $Tree$, $Path$, $Fair$, OED, $cOED$, ID, IP:

Theorem (Composable[$\mathcal{S}$]):

(1) $\mathcal{S}(K_0) \parallel \mathcal{S}(K_1) = \mathcal{S}(K_0 \parallel K_1)$ if $K_0 \bowtie K_1$

(2) $\mathcal{S}(K \lceil \rho) = \mathcal{S}(K) \lceil \rho$ for $\rho \in \mathcal{P}_\omega[\mathbf{A}]$

(3) $\mathcal{S}(\widehat{\alpha}(K)) = \widehat{\alpha}(\mathcal{S}(K))$ for any renaming α

(**Composable[$\mathcal{S}$]**) for $\mathcal{S}$ one of $Path$, $Tree$, or $Fair$ was shown in [Tal96a]. To extend these results to $\mathcal{S}$ chosen from OED, $cOED$, ID, and IP it suffices to show (**Composable[OED]**). The rest follow from the morphism results for mappings between OEDs, IDs and IPs. We show the argument for (1), the rest are easy.

Proof (Composable[$\mathcal{S}$].1): Assume $K_0 \bowtie K_1$ and let $Oed_l = OED(K_0) \parallel OED(K_1)$ and $Oed_r = OED(K_0 \parallel K_1)$. We show (lr) $Oed_l \subseteq Oed_r$ and (rl) $Oed_r \subseteq Oed_l$.

For (lr) assume $oed \in Oed_l$
Pick $(oed_j, g_j) \in OED(K_j)^\dagger$ and $g \in GT(oed)$ such that $(oed_0, g_0) \bowtie (oed_1, g_1)$ and $(oed, g) = (oed_0, g_0) \parallel (oed_1, g_1)$.
Pick $\pi_j \in Path(K_j)$ such that $c2oed(\pi_j) = (oed_j, g_j)$ by (**c2oed**).
$(oed, g) = c2oed(\pi_0 \parallel \pi_1)$ by (**c2oed.pcmps**), and $c2oed(\pi_0 \parallel \pi_1) \in Oed_r{}^\dagger$ by (**c2oed**) and (**Composable[$Path$]**).
For (rl) assume $oed \in Oed_r$.
Pick $(\pi) \in Path(K_0 \parallel K_1)$ such that $(oed, g) = c2oed(\pi)$ by (**c2oed**).
Pick $(\pi_j) \in Path(K_j)$ such that $\pi_0 \bowtie \pi_1$ and $\pi = \pi_0 \parallel \pi_1$ by (**Composable[$Path$]**).
Using (**c2oed.pcmps**) $(oed, g) = c2oed(\pi_0) \parallel c2oed(\pi_1) \in Oed_l{}^\dagger$.

9 Summary and Future Work

The actor model of computation is expressive and scalable. It provides the basic units of encapsulation of state and threads of control needed for open distribributed systems. Semantic models of actor based systems are needed to provide a basis for specifying and reasoning about components of open distributed systems in a modular way. Purely operational semantics based on transition systems, such as those given in [Agh86, AMST97], are simple and intuitive, but too concrete for many purposes. Operational models based on rewriting logic [Tal96a] provide a first level of abstraction, defining

an equivalence relation on computations that hides the ordering of independent transitions. Event diagrams [Gre75, Cli81] provide the next level of abstraction, avoiding irrelevant interleaving details and hiding the representation of actor states. Interaction diagrams abstract further by hiding internal events but keeping the induced ordering relation that contains information about output events that occur in response to input events, history dependence, and creation dependence. Interaction paths ignore ordering information that has no semantic consequences and is the most abstract of the semantics studied here. All of these semantic models satisfy the composability property and thus semantics of a system can be defined in terms of the semantics of its parts. This spectrum of semantic models yields a spectrum of equivalence relations that can be used for specification and for reasoning about replacing subcomponents of a system. Interaction path semantics was first introduced in [Tal96b] (called simply interaction semantics). In studying this semantics it became clear that in order to give a mathematical characterization of the possible interaction paths of configuration one needed to postulate the existence of arrival orderings and input/output orderings satisfying additional application specific properties. The motivation for studying the event diagram semantics of actor based components of open systems was to give such specifications a clean foundation.

The work presented in this paper builds directly on the work of Clinger and extends it by studying two abstractions of event diagrams and by developing a component algebra structure and showing composability of these semantics.

There are a number of directions for future work. One is to develop specification languages based on event diagrams or interaction diagrams as an alternative to using traditional temporal languages which are based on global times.

A second direction is extensions of the basic actor model. Many computational abstractions that are important for dynamic adaptive distributed open systems are not directly supported. Examples are synchronization primitives and more generally abstractions that support transactions, security policies, reliable communication, dependability, and real time computation. There are a number of experiments in designing extensions to actor languages that support these features [Agh97]. A next step is to develop semantic models that support these abstractions. An important new aspect that arises here is understanding dynamic composability as well as static composability. One example is understanding in a modular way, the semantics of imposing new protocols, security measures, or dependability policies as the system is running. Another example is the dynamic introduction of new components, or replacement of existing components.

Acknowledgements

The author would like to thank Carl Hewitt and Richard Weyhrauch for helping me understand the concepts underlying the actor model. She thanks her her collaborators in actor research, Gul Agha, Ian Mason, and Scott Smith for many fruitful discussions about actor semantics. Special thanks to Takayasu Ito, Ian Mason, and Scott Smith for helpful comments on earlier versions.

This research was partially supported by ARPA grant ONR grant N00014-94-1-0857, NSF grant CCR-9312580, ARPA/SRI subcontract C-Q0483, NSF grant CRR-9633419, and ARPA/Rome Labs grant AF F30602-96-1-0300.

References

[Agh86] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.

[Agh90] G. Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–141, September 1990.

[Agh97] G. Agha. Abstracting interaction patterns: A programming paradigm for open distribute systems. In E. Najm and J-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 135–153. Chapmn & Hall, 1997.

[AMST97] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.

[BH77] Henry G. Baker and Carl Hewitt. Laws for communicating parallel processes. In *IFIP Congress*, pages 987–992. IFIP, August 1977.

[Cli81] W. D. Clinger. *Foundations of Actor Semantics*. PhD thesis, MIT, 1981. MIT Artificial Intelligence Laboratory AI-TR-633.

[Gre75] I. Greif. Semantics of communicating parallel processes. Technical Report 154, MIT, Project MAC, 1975.

[Hew77] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8(3):323–364, 1977.

[MT97] I. A. Mason and C. L. Talcott. A semantically sound actor translation. In *ICALP'97*, 1997. to appear.

[Tal96a] C. L. Talcott. An actor rewriting theory. In *Workshop on Rewriting Logic*, number 4 in Electronic Notes in Theoretical Computer Science, 1996.

[Tal96b] C. L. Talcott. Interaction semantics for components of distributed systems. In E. Najm and J-B. Stefani, editors, *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'96*, 1996. proceedings published in 1997 by Chapman & Hall.

Contents

1 Introduction **1**

2 Actor Computation Informally **5**
2.1 Ticker . . . 5
2.1.1 A Ticker Event Diagram . . . 5
2.1.2 A Complete Ticker Event Diagram . . . 7
2.1.3 A Ticker Interaction Diagram . . . 7
2.1.4 Global Times and Interaction Paths . . . 8
2.1.5 Ticker behavior and computation . . . 9
2.2 Ticker Factory . . . 10
2.3 Composing . . . 12

3 Messages and Events **13**
3.1 Actor Names and Messages . . . 13
3.2 Events . . . 14
3.3 Interfaces . . . 15
3.4 Global Times and Event Paths . . . 16

4 Open Event Diagrams **18**
4.1 Structure and axioms . . . 18
4.2 The OED domain . . . 21
4.3 A component algebra on OEDs . . . 23

5 Interaction Diagrams **25**
5.1 Structure . . . 25
5.2 A Component Algebra on IDs . . . 26
5.3 A Component Algebra Morphism from OED to ID . . . 27

6 Interaction paths **29**
6.1 Structure . . . 29
6.2 A component algebra on IPs . . . 29
6.3 A Component Algebra Morphism from ID to IP . . . 30

7 Actor Theories and Their Models **31**
7.1 The Basics . . . 31
7.2 Configurations and Computations . . . 33
7.3 Component algebra on computations . . . 35
7.4 Mapping Computations to OEDs . . . 36

8 Semantics for Configurations **38**
8.1 The OED semantics of configurations . . . 38
8.2 An Alternative Characterization of OED semantics . . . 39
8.3 ID and IP semantics . . . 40
8.4 Distinctions . . . 41
8.5 Compositionality . . . 42

9 Summary and Future Work **42**