, , 1–66 ()

Composable Semantic Models for Actor Theories

CAROLYN L. TALCOTT clt@cs.stanford.edu
Stanford University

Editor:

Abstract. We are interested in developing a semantic foundation that supports specifying, composing, and reasoning about components of open distributed systems. 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. In order to provide for composability, and for reasoning about properties at many different levels of abstraction we introduce the notions of actor component, actor component algebra, and actor component algebra morphism. Morphisms from syntactic to semantic algebras give composable semantics.

We first illustrate features of the actor model and a variety of component algebras and morphisms through a series of simple examples. We then 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, and we show that component algebra laws hold. 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.

Keywords: actors, actor component algebra, morphism, event diagram, composable semantics

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 [9, 1, 2] as our starting point. Actors are independent computational agents that interact solely via asynchronous message passing. 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. We model system components as 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.

In order to provide for composability, and for reasoning about properties at many different levels of abstraction we introduce the notions of *actor component*, *actor component algebra*, and *actor component algebra morphism*. An actor component models an actual actor system at some level of abstraction. An actor component has an interface, and could be simply that. Modeling an actor system by its interface corresponds to forgetting almost everything but how to interact with it. An actor component could also be a detailed but static description of an actor system, such as an initial state description, or a program or module. This level allows us to perform static analyses and to talk about program transformations. Of course for these operations to make sense we need semantic models. The set of possible computations of an actor system, expressed for example as sequences of labelled state transitions is an actor component giving a detailed semantic model. A mapping from programs to such components gives an operational semantics and a basis for implementation and for reasoning about program analyses. More abstract semantic models arise by forgetting computational details. For example we can forget the representation of states and just look at the traces of actions or partial orders of events. An even more abstract model is obtained by forgetting all details of the computation except the interactions of a component with its environment. Composing these abstractions with the operational semantics gives useful notions of program equivalence and of correctness of program transformations. It also allows us to express conditions under which one component can be replaced by another.

Actor component algebras provide the basis for building up components (models) from smaller parts and for analyzing components into simpler subcomponents. The carrier of a component algebra is a set of components (modeling system components at some given level of abstraction) which includes the empty component, called `mt`. We require that there is a function, *iface*, mapping each component to its interface. A component algebra has three operations: parallel composition (a partial operation); restriction; and renaming. Two components, C_0 and C_1 are *composable* if their interfaces have disjoint receptionist sets. If C_0 and C_1 are composable, then their parallel composition, $C_0 \parallel C_1$, is defined. 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 operations of a component algebra must satisfy 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: $\widehat{\alpha}(C\lceil\rho) = (\widehat{\alpha}(C))\lceil\widehat{\alpha}(\rho)$.

A component algebra of special interest is, **Iface**, the algebra of interfaces. In this algebra, the empty component is the empty interface, $(\emptyset, \emptyset)$, which has no receptionists and no externals.

Component algebra morphisms, or simply *morphisms* are maps between component algebras that preserve the algebraic structure. Let $\mathcal{C}, \mathcal{C}'$ be component algebras and let $\mathcal{C}(\rho, \chi)$ be the set of components of $\mathcal{C}$ with interface (ρ, χ). A component algebra morphism from $\mathcal{C}$ to $\mathcal{C}'$ is a map φ sending components of $\mathcal{C}$ to components of $\mathcal{C}'$ such that the following holds.

Component Algebra Morphism:

(0) φ preserves the empty component and component interface

- $\varphi(\mathtt{mt}_{\mathcal{C}}) = \mathtt{mt}_{\mathcal{C}'}$, and
- $\varphi : \mathcal{C}(\rho, \chi) \to \mathcal{C}'(\rho, \chi)$.

(1) If ρ_0 and ρ_1 are disjoint and $C_j \in \mathcal{C}(\rho_j, \chi_j)$ for $j < 2$, then $\varphi(C_0 \parallel C_1) = \varphi(C_0) \parallel \varphi(C_1)$

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

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

Finally, for each component algebra, $\mathcal{C}$, we require that its interface function be a component algebra morphism from $\mathcal{C}$ to **Iface**. In practice this is a constraint on the operations of $\mathcal{C}$. Note that this requirement implies that interface of an empty component is the empty interface.

For static descriptions of actor components we use actor theories [15, 14, 12]. The notion of an 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 the receipt of a message. Associated to each actor theory is a set of configurations that describe actor system components. There is a natural algebraic structure on the configurations of an actor theory that satisfies the component algebra laws,

An actor theory semantics, $\mathcal{S}$, maps configurations to components of a semantic algebra, preserving interface. $\mathcal{S}$ is *compositional* just if it is a component algebra morphism. Morphisms between semantic algebras allow one to relate specifications based on different models and to transfer results obtained in one domain to another, as will be seen below.

In [14] a simple operational semantics for actor theory configurations was given, and a compositional semantics was defined taking sets of computations as components. In [15] 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 [8], used by Hewitt in [9] and formalized by Clinger [7] to the open systems setting, making interactions with the environment explicit. Event diagrams express the view that what is important about an actor computation are the messages sent and actors created in response to a message delivery, and the *precedes* ordering among these events.

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 precedes relation. 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 [15] is reformulated here in terms of *interaction paths* corresponding to the possible global times compatible with event or interaction diagrams. Interaction paths ignore ordering information which can note be observed semantically, and is the most abstract of the semantics studied here.

Open event diagrams can model both stages of system activity and complete computations in which all messages have been delivered. 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. The notion of actor theory is recalled and semantic maps associating to actor theory configurations sets of event diagrams, complete event diagrams, interaction diagrams, and interaction paths are defined. The main result is that these maps give compositional semantics. To establish this result a morphism from computations to event diagrams is defined, reducing the problem to the compositionality of the semantics based on sets of computations.

Plan

In §3 actor computation and the semantic models are introduced informally using a series of simple examples. In §4 notation relating to events that is used throughout the remainder of the paper is developed. In §5 open event diagrams and the component algebra on event diagrams are defined. In §6 interaction diagrams and their component algebra are defined as well as a morphism from event diagrams to interaction diagrams. In §7 the interaction path semantics is defined and a morphism from interaction diagrams to interaction paths is defined. In §8 actor theories and their computational semantics is summarized. In §9 the semantic maps giving event diagram, interaction diagram, and interaction path semantics for configurations are defined and shown to be composable. §10 contains a 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, $\ldots$, 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. $[Y_0 \to Y_1]$ is the set of total functions with domain Y_0 and range contained in Y_1. $[Y_0 \xrightarrow{\mathrm{inj}} Y_1]$ 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. About Actors and Actor Computation

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. In this section we review some of the motivations and basic precepts of the actor model and the laws postulated for actor computation in the early papers of Hewitt and his students [9, 6, 7].

The foundational objective for the actor model was a "mathematical theory of computation which can represent any kind of discrete behavior that can be physically realized." [9]. One of the applications for the actor model of computation was a basis for modeling intelligence in terms of a society of communicating knowledge-based problem-solving experts. A part of this effort was to analyze control structures in terms of message passing patterns. The actor model was also an attempt to remedy the difficulties of using semantic theories based on interleaving and incremental changes of global state to specify and reason about communicating parallel processes. It addresses the problem of distributed system design by offering an abstract conceptual and formal model for thinking about and proving theorems about distributed systems.

2.1. Events and ordering relations on events

Actors are a local model of computation. There is no such thing as "action at a distance", for example no action of one actor can directly affect the state of another actor. Neither is there any "global state" of all actors in the universe. The action in an actor computation consists solely of actors sending messages to other actors. It is sufficient to model only message receipt since the actor model requires that every message sent will be received. An event e is the receipt of a message $message(e)$ by the actor $target(e)$. As a result of an event e, the target actor may send new messages or create new actors. The events corresponding to the receipt of these new messages are said to be *activated* by the event e and the *activation order*, $\prec_{\mathrm{ao}}$, is the partial order generated by $e \prec_{\mathrm{ao}} e'$ if e' is activated by e. Activation is the actor notion of causality. If actor a is created as a result of e we write $Cr(a) = e$. The creation relation also provides ordering information: no event with target a can happen before the creation of a. Processing of one message by an actor should not be interrupted by processing of another message to that actor. Thus it is postulated that the set of events with the same target is totally ordered. The total order on events, $\xrightarrow{\mathrm{arr}}_a$, at a given actor, a, is called its *arrival* order and it represents that actor's *local time*. Individual actors operate independently and the arrival orders for different actors are only related by activation links. In [6] an actor computation is modeled as a set of events together with a partial order on these events called the *precedes* relation that contains all possible dependencies among events. The precedes relation is the least relation containing the activation order, the arrival order for each actor, and satisfying the additional constraint that the event creating an actor must precede the first event in the arrival ordering of that actor. Using the partially ordered event model, specifications for an actor system and correctness assertions for a computation can be given very naturally in terms of events and causal relations among events.

Here are two simple examples of actor computations. The first shows a computation with a simple procedure call-like subcomputation.

An event e_1 with target a_1 causes a request, e_2, to be sent to a_2. This in turn causes a reply e_3 to be sent to a_1 which causes an event e_4, a reply to the customer, a, specified in e_1. Thus we have the activation and arrival orderings:

$$e_1 \prec_{\mathrm{ao}} e_2 \prec_{\mathrm{ao}} e_3 \prec_{\mathrm{ao}} e_4 \quad \text{and} \quad e_1 \xrightarrow{\mathrm{arr}}_{a_1} e_3$$

with the event pattern $e_2 \prec_{\mathrm{ao}} e_3$ being a subcomputation of the overall cause and effect, $e_1 \prec_{\mathrm{ao}} e_4$. Thus e_1 *precedes* e_4.

The second shows parallel subcomputations using a join continuation actor to combine the results. Here an event e_1 with target a_1 causes creation of a new actor a_j, and requests e_2 to a_2 and e_3 to a_3 specifying that replies be sent to a_j. These requests cause, respectively, replies e_4, e_5 with target a_j. Upon the arrival of the second reply to a_j a reply e_6 is sent to a. The activation and arrival orderings for this example are:

$$e_1 \prec_{\mathrm{ao}} e_2 \quad \text{and} \quad e_1 \prec_{\mathrm{ao}} e_3 \quad \text{and} \quad Cr(a_j) = e_1$$

$$e_2 \prec_{\text{ao}} e_4 \quad \text{and} \quad e_3 \prec_{\text{ao}} e_5 \quad \text{and} \quad e_4 \xrightarrow{\text{arr}}_{a_j} e_5 \prec_{\text{ao}} e_6$$

Here we have e_1 *precedes* e_4 for two reasons: there is an activation chain from e_1 to e_4; also e_1 is the creation event for the target of e_4. e_2 *precedes* e_5 using the activation link to e_4 followed by the arrival order link to e_5.

Baker and Hewitt [6] postulate some laws that must be satisfied by computations involving communicating parallel processes. The laws take the form of stating plausible restrictions on the histories of parallel computations to make them physically realizable. The laws are justified by appeal to physical intuition and are to be regarded as falsifiable assertions about the kinds of computations that occur in nature rather than as proven theorems in mathematics. These laws are stated in terms of the actor model as properties that must hold of the precedes relation on the events of an actor computation. The precedes relation must be a *strict* partial order. Thus no event can precede itself. For simplicity an initial event is postulated. In an actor computation, the number of events between any two events must be finite. This property is called *discreteness*. It implies that we can talk meaningfully about *immediate predecessors* and *immediate successors* of an event. Finally, the immediate successors of an event must be finite.

The finiteness conditions do not rule out non-terminating computations, but they do rule out the possibility of "Zeno Machines". Zeno machines are paradoxical machines that can do infinitely many things in a finite amount of time. A Zeno machine if it existed could do many useful things such as providing decision procedures for first-order predicate calculus. Since they do not exist, such behavior must be ruled out.

An actor computation in which the activation and arrival sub-orderings are made explicit is called an *event diagram* [9, 8]. Clinger [7] gave a precise mathematical formulation of event diagrams. He first considered the *combined order*, $\prec_{\text{co}}$, which is obtained from the activation and arrival ordering ignoring the effects of actor creation. He relaxed the finite successors condition and reduced the laws for this relation to the following three:

(Law of Strict Causality–LSC) For no e does $e \prec_{\text{co}} e$.

(Law of Countability–LC) There are at most countably many events.

(Law of Finite Predecession–LFP) For all events e, the set of predecessors, $\{e' \mid e' \prec_{\text{co}} e\}$ is finite.

Baker and Hewitt point out that the laws for the precedes relation imply that every actor computation has a monotonic injection into the set of natural numbers. Clinger showed ([7] **Theorem 1.**, p32) that the conjunction of the above three laws is *equivalent* to what he calls the (strong) axiom of realizability:

(Axiom of (Strong) Realizability–SR) There exists a one-one mapping, $g : \mathbf{E} \to \mathbf{N}$, from the set of events, $\mathbf{E}$, to the set of natural numbers, $\mathbf{N}$ giving a total order on E that is compatible with the activation and arrival orderings (and hence compatible with with the combined ordering).

This result gives a very simple, easy to use, axiomatization of the ordering laws and shows that actor computation is consistent with the interleaving (all shuffles) approach. While properties of actor computations are generally more easily expressed in terms of the partial order on events, the existence of a global time turns out to be a useful technical device and in addition simple operational semantics lead to interleaving semantics more directly than to partial order semantics.

2.2. Locality

So far we have only discussed the ordering of events in an actor computation. The point of an actor computation is the exchange of information among participating actors with new information emerging as the result. The fundamental unit of information in an actor system is one actor knowing about the existence of another actor. The only means of transmitting information in an actor computation is by sending messages. An actor a may send a message to another actor a' only if it "knows about" a', i.e. knows the name of a'. The actors that an actor knows about are called *acquaintances* of that actor. How can an actor come to know another actors name? Only if it was either created with that knowledge, received it in a message, or created the actor. Thus an actor can gain acquaintances as its local time evolves. It may also forget some of its acquaintances. To be more precise about how an actors acquaintances evolve over time, we define the notion of *participants*. A participant in an event, e, is either the target of e, an acquaintance of the target of e at the local time of the event, an actor occurring in the message of e, or an actor created by e. The locality laws for actor computation may now be stated as follows:

- The participants of an event must be finite.
- The acquaintances an actor is given initially must be a subset of the participants of the actor's creation event.
- The acquaintances of an actor just after receiving a message must be a subset of the participants of the receipt event.
- The target and the actors mentioned in the message of an event must be participants of the event that activated that event.

The locality laws rule out the possibility of "broadcasting" protocols in which a message is sent to every actor in the system. This is reasonable, since in a model with no global state that allows creation of new actors, the phrase "all actors" is not well-defined. Of course it is easy to express a protocol that broadcasts messages to any given group of known actors.

The locality laws are independent of the ordering laws and further constrain the possible computations. The fact that the event creating an actor precedes the first event in that actors arrival order is a consequence of the ordering and locality laws and need not be stated as an additional constraint as Hewitt and Baker originally did. Clinger ([7] pp. 130–131) gives another example of added expressive power of locality axioms. He describes a computation in which there are two actors that never communicate with each other (directly or indirectly), but each sends a message to each member of an infinite set of other actors. The ordering on events satisfies the laws for the combined order. However, the computation entails infinite shared information without communication and is ruled out by the locality laws.

2.3. Incomplete computations

Baker-Hewitt considered only completed computations. They stated that complete computations are to be generated inductively by adding events to some initial base in finite increments, but they provided no mechanism for doing this.

Clinger's formalization treats incomplete event diagrams that represent stages in the generation of an actor computation. This is accomplished by including in event diagrams a set of pending events [5] representing messages that have been sent, but not yet delivered. A complete diagram is one in which the set of pending events is empty. To construct a complete, possibly infinite, computation from finite initial stages, Clinger defines a partial ordering, $\sqsubseteq$, on event diagrams. Each complete event diagram is the limit (least upper bound) of a chain of finite event diagrams. Not all chains have a limit that is a complete event diagram since the limit may have residual pending events. Such diagrams are ruled out by the finite delay axiom:

> (Finite Delay) If the set, E, of events is infinite then the set of pending events is empty.

This formalizes the fairness requirement on the incremental generation of actor computations. An infinite chain of incomplete event diagrams whose limit is incomplete corresponds to an unfair computation.

3. Actor Computation Informally

In this section we introduce informally the mathematical notions and structures we will use to model actor computation — actor theories, computation paths, event and interaction diagrams, interaction paths — through a series of simple examples. Following Clinger [7] we take event diagrams as the basic model. Thus we make the activation and arrival orderings and the creation and acquaintances information explicit. Because we are modeling computations of components which interact with their environment, our event diagrams and other component structures also have an interface specifying the receptionists and external actors known to the system. The examples present event diagrams both graphically and as formal mathematical structures. In the remainder of the paper we focus on the latter.

3.1. Ticker

A ticker is an actor that 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.

3.1.1. Ticker behavior and computation We can specify an actor that exhibits the behavior of a ticker using a set of states $T(n)$ for $n \in \mathbf{N}$. A ticker actor with name t and current time n is written $[T(n)]_t$. A request to t from actor c for the current time has the form $t \triangleleft c$ and a tick message for t has the form $t \triangleleft \mathtt{tick}$. $t \triangleleft c$ and $t \triangleleft \mathtt{tick}$ are examples of message packets — structures of the form $t \triangleleft v$ with target t and message v. The behavior of a ticker is given by reaction rules that determine its response to these messages. The rules specify the new state of the ticker as well as messages sent.

$$\text{(t1)} \quad [T(n)]_t \,,\, t \triangleleft c \quad \Rightarrow \quad [T(n)]_t \,,\, c \triangleleft n$$

$$\text{(t2)} \quad [T(n)]_t \,,\, t \triangleleft \mathtt{tick} \quad \Rightarrow \quad [T(n+1)]_t \,,\, t \triangleleft \mathtt{tick}$$

The first rule describes the response to a request to t from c: the ticker state is unchanged and a reply $c \triangleleft n$ containing n is sent to the customer c. The second rule describes the response to a tick message: the ticker state changes by incrementing the parameter n by 1 and sends a tick message to itself.

A collection of states, messages, and reaction rules such as given above for a ticker is an actor theory. It determines the possible computations of actor systems built from such actors. Formally actor systems are described by configurations which have an interface and an interior. The interface is a pair of sets of actor names, the first specifying the names of system actors visible to the environment (receptionists) and the second specifying the names of actors the system expects to find in the environment (externals). The interior is a set of actors and pending events — message packets to be delivered. We also include in a configuration a history consisting of the set of past events. We treat events as an abstract data type with a function *pkt* mapping events to message packets. Concretely we represent an event as a message packet tagged with a distinct identifier that can be used to refer to that event. Once the event is introduced we often use only the tag to refer to it, omitting the packet. Note that multiple distinct events may have the same packet.

A ticker system at start up time is described by the configuration

$$\mathit{Ticker}[t] = (t, \emptyset)([T(0)]_t \,,\, e_0 : t \triangleleft \mathtt{tick} \mid \emptyset)$$

It has interface $(t, \emptyset)$, with a single receptionist, t, and no externals. The interior has a single actor, $[T(0)]_t$, with name t and state $T(0)$, and a single pending event e_0 containing a $\mathtt{tick}$ message for t. The history is initially empty.

Computations are sequences of labelled transitions where the labels correspond to input, delivery and output of messages. Delivery transitions are internal transitions determined by the reaction rules. Input and output transitions are the means of interaction with the environment. Messages to receptionists may be input, and messages to external actors may be output. A computation, π^{T}, for the ticker system is shown in figure 1.

$$(t,\emptyset)(\texttt{[}T(0)\texttt{]}_t\ ,\ e_0 : t \triangleleft \texttt{tick} \mid \emptyset)$$

$$\xrightarrow{\texttt{d}(e_0)} (t,\emptyset)(\texttt{[}T(1)\texttt{]}_t\ ,\ e_2 : t \triangleleft \texttt{tick} \mid e_0)$$

the pending `tick` is delivered to t using rule (t2)

$$\xrightarrow{\texttt{in}(e_1 : t \triangleleft c)} (t,c)(\texttt{[}T(1)\texttt{]}_t\ ,\ e_2\ ,\ e_1 \mid e_0)$$

a request from c arrives and is added to the pending events

$$\xrightarrow{\texttt{d}(e_1)} (t,c)(\texttt{[}T(1)\texttt{]}_t\ ,\ e_2\ ,\ e_1^o : c \triangleleft 1 \mid e_0\ ,\ e_1)$$

the pending request from c is delivered to t using rule (t1)

$$\xrightarrow{\texttt{in}(e_3 : t \triangleleft c)} (t,c)(\texttt{[}T(1)\texttt{]}_t\ ,\ e_2\ ,\ e_3\ ,\ e_1^o \mid e_0\ ,\ e_1)$$

$$\xrightarrow{\texttt{d}(e_2)} (t,c)(\texttt{[}T(2)\texttt{]}_t\ ,\ e_4 : t \triangleleft \texttt{tick}\ ,\ e_3\ ,\ e_1^o \mid e_0\ ,\ e_1\ ,\ e_2)$$

$$\xrightarrow{\texttt{out}(e_1^o)} (t,c)(\texttt{[}T(2)\texttt{]}_t\ ,\ e_4\ ,\ e_3 \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o)$$

the message to c is ouput to the environment

$$\xrightarrow{\texttt{d}(e_3)} (t,c)(\texttt{[}T(2)\texttt{]}_t\ ,\ e_4\ ,\ e_3^o \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o\ ,\ e_3)$$

$$\xrightarrow{\texttt{out}(e_3^o)} (t,c)(\texttt{[}T(2)\texttt{]}_t\ ,\ e_4 \mid e_0\ ,\ e_1\ ,\ e_2\ ,\ e_1^o\ ,\ e_3\ ,\ e_3^o)$$

Figure 1. A Ticker Computation π^{T}, in which two requests arrive from the environment and replies are sent out.

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
$pkt(e)$:	$t \triangleleft \texttt{tick}$	$t \triangleleft c$	$t \triangleleft \texttt{tick}$	$t \triangleleft c$	$t \triangleleft \texttt{tick}$	$c \triangleleft 1$	$c \triangleleft 2$

If we ignore states of a computation and keep just the interface and the sequence of events we get an event path. The events are just the transition labels with the `d` wrapping removed from internal events. The event path resulting from π^{T} is

$$ep^{\mathrm{T}} = (t,\emptyset)[e_0, \texttt{in}(e_1), e_1, \texttt{in}(e_3), e_2, \texttt{out}(e_1^o), e_3, \texttt{out}(e_3^o)]$$

3.1.2. A Ticker Event Diagram Now let us see what the ticker computation shown above looks like when represented as an event diagram. This is shown in figure 2. 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.

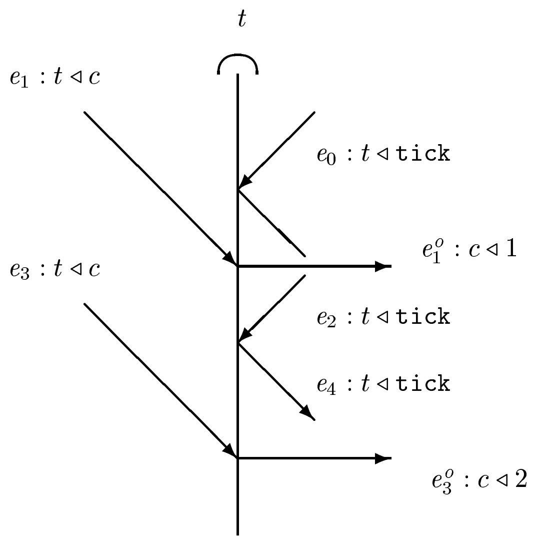

Figure 2. A Ticker Event Diagram. The vertical line labeled t is the local time-line for actor t. It gives the arrival order of message delivery events for t. A delivery to t is indicated by the point where the line labelled by the event meets the time line. Lines going out from this point indicate events activated by the delivery, corresponding messages sent in response to the message receipt.

The ticker event diagram represented in figure 2 is a structure

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

where

- $(t, \emptyset)$ is the ticker interface with one receptionist, t, and no externals.
- $A^{\mathrm{T}} = \{t\}$ 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 been 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.

- $\xrightarrow{\text{arr}}{}^{\mathrm{T}} = \{e_0 \to e_1 \to e_2 \to e_3\}$ is the arrival order of events at t
- $acqB^{\mathrm{T}} = \{e \mapsto \{t\} \mid e \in E^{\mathrm{T}}/t\}$ gives the acquaintances of each internal actor just before each of its local events. 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.
- $Cr^{\mathrm{T}} = \{t \mapsto !\}$ is the creation function, giving the creator of each actor in the system. $Cr^{\mathrm{T}}(t) = !$ means that the ticker, t, was initially present.

3.1.3. 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 $pkt(e_i) = t \triangleleft \mathtt{tick}$ 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

$$A^{\mathrm{cT}} = A^{\mathrm{T}} = \{t\},$$
$$E^{\mathrm{cT}} = E^{\mathrm{T}} \cup \{e_i \mid 5 \leq i, i \in \mathbf{N}\}$$
$$activator^{\mathrm{cT}} = activator^{\mathrm{T}} \cup \{e_{i+1} \mapsto e_i \mid 4 \leq i\}$$
$$\xrightarrow{\text{arr}}{}^{\mathrm{cT}} = \xrightarrow{\text{arr}}{}^{\mathrm{T}} \cup \{e_i \to e_{i+1} \mid 3 \leq i\}$$
$$acqB^{\mathrm{cT}} = \{e \mapsto \{t\} \mid e \in E^{\mathrm{cT}}/t\},$$
$$Cr^{\mathrm{cT}} = Cr^{\mathrm{T}} = \{t \mapsto !\},$$

3.1.4. 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 sent to the environment) we obtain the interaction diagram shown in figure 3. The connection between inputs and outputs, represented by the interior dotted lines, results from forming chains of activation and arrival links, that is, it is the restriction of the combined orderings. Each event with activator **?** can be seen from two points of view: from outside the system we see the input of the message from the environment; and from inside the system we see its delivery to an internal actor. In the event diagram we see the delivery time. In the interaction diagram we see the input. To distinguish these

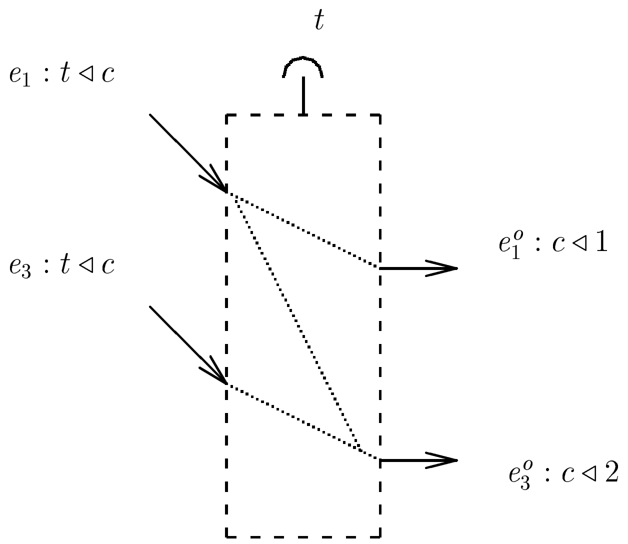

Figure 3. A Ticker Interaction Diagram

times we represent the input to the component by $\mathtt{in}(e)$. Also, since we have no explicit information about which actors are internal, we write $\mathtt{out}(e)$ if e is targeted to an external actor. We extend the activation ordering so that $\mathtt{in}(e)$, the input, activates e, the delivery. Formally the interaction diagram represented in figure 3 is a structure

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

where

- $I^{\mathrm{T}} = \{\mathtt{in}(e_1), \mathtt{in}(e_3)\}$ is the set of events with activator **?** (unknown) — observed at the time they are input from the environment;
- $O^{\mathrm{T}} = \{\mathtt{out}(e_1^o), \mathtt{out}(e_3^o)\}$ 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.
- A^{T} is the set of reserved actor names. These are names reserved for naming actors created and possibly exported by the component. They abstract the notion of internal actors of event diagrams.

Notice that although each output is the response to a unique request, in the combined ordering e_3^o has two predecessors: e_1 and e_3, since e_1 precedes e_3 in the arrival order of t. 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.

3.1.5. Global Times and Interaction Paths As discussed in §2, Clinger [7] 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. In an event diagram, the input of a message from the environment is not represented explicitly. It is implicit in that the corresponding delivery event has activator $\mathtt{?}$. In the global time we explicitly represent input of an event e with activator $\mathtt{?}$ by $\mathtt{in}(e)$, and extend the activation ordering so that $\mathtt{in}(e) \prec_{\mathrm{ao}} e$. We also distinguish events corresponding to output of a message to the environment by writing $\mathtt{out}(e)$ rather than simply e. Two possible global times for the ticker event diagram are:

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

Note that the order of input events need not correspond to the order of their delivery. The global time g^{T} is just the event sequence of the event path ep^{T} derived from the computation, π^{T}, for the ticker system. 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. Encapsulating an interaction diagram global time with the interface, and pairing this with the reserved actor names gives an interaction path. Two possible interaction paths for the ticker event and interaction diagrams above are:

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

Notice that $[\mathtt{in}(e_3), \mathtt{out}(e_3^o), \mathtt{in}(e_1), \mathtt{out}(e_1^o)]$ is not a possible global time for id^{T} since according to t's local time delivery of input with response 1 must precede the input with response 2 thus forcing $\mathtt{in}(e_1) \prec \mathtt{out}(e_3^o)$.

3.2. Ticker Factory

The ticker illustrates message passing, state change, non-determinism, and event ordering features of actor computation. The missing feature is actor creation. To illustrate this feature we introduce the notion of a ticker factory. A ticker factory is an actor system that accepts requests, $\mathtt{new}(c)$, for new tickers. For each such request, a new ticker is created and sent an initial `tick` message, and its name is sent to the requester c. Figure 4 shows an event diagram for a ticker factory.

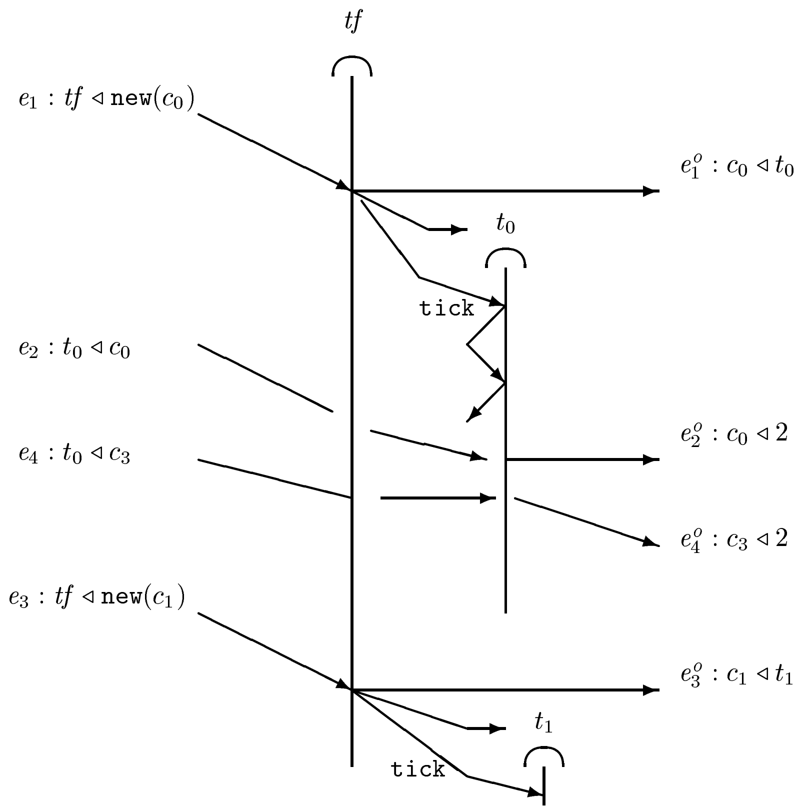

Figure 4. An Event Diagram for a Ticker Factory with receptionist, tf, in which two requests—$tf \triangleleft \texttt{new}(c_0)$ and $tf \triangleleft \texttt{new}(c_1)$—for new tickers arrive from outside and two time requests—$t_0 \triangleleft c_0$ and $t_0 \triangleleft c_3$—arrive for t_0, the first ticker created.

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

$e:$	e_1	e_2	e_3	e_4	e^o_1	e^o_2	e^o_3
$pkt(e):$	$tf \triangleleft \texttt{new}(c_0)$	$t_0 \triangleleft c_0$	$tf \triangleleft \texttt{new}(c_1)$	$t_0 \triangleleft c_3$	$c_0 \triangleleft t_0$	$c_0 \triangleleft 2$	$c_1 \triangleleft t_1$
$e:$	e^o_4	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 events e_{01}, e_{02}, and e_{03}, are the `tick` events at t_0 (with e_{03} pending) and e_{11} is the `tick` event at t_1. The labels are omitted in the picture to avoid clutter.

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

$$A^{\mathrm{Tf}} = \{tf, t_0, t_1\}$$
$$E^{\mathrm{Tf}} = \{e_1, e_2, e_3, e_4, e_1^o, e_2^o, e_3^o, e_4^o, e_{01}, e_{02}\},$$
$$P^{\mathrm{Tf}} = \{e_{03}, e_{11}\}$$
$$\xrightarrow{\mathrm{arr}}_{tf}{}^{\mathrm{Tf}} = [e_1 \rightarrow e_3],$$
$$\xrightarrow{\mathrm{arr}}_{t_0}{}^{\mathrm{Tf}} = [e_{01} \rightarrow e_{02} \rightarrow e_2 \rightarrow e_4]$$
$$\xrightarrow{\mathrm{arr}}_{t_1}{}^{\mathrm{Tf}} = [\,]$$
$$acqB^{\mathrm{Tf}} = \{x \mapsto \emptyset \mid x \in E^{\mathrm{Tf}}/tf\} \cup \{x \mapsto \{t_0\} \mid x \in E^{\mathrm{Tf}}/t_0\}$$
$$Cr^{\mathrm{Tf}} = \{tf \mapsto !, t_0 \mapsto e_1, t_1 \mapsto e_3\}$$

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. In this event diagram, tf has no acquaintances when a message arrives, in contrast to tickers, which must know their own name in order to send themselves `tick` messages. It would be equally valid to define the acquaintances of tf to be $\{tf\}$ however it is unnecessary.
Hiding the internal details we obtain the interaction diagram shown in Figure 5. The mathematical structure corresponding to this picture is

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

where

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

As in the ticker example, the ordering, $\prec^{\mathrm{Tf}}$, includes the 'in response to' ordering and history information. It also includes the information that $\mathtt{in}(e_1)$ causes the initial tick event at t_0 which happens to precede all other delivery events at t_0. Thus we have $\mathtt{in}(e_1) \prec^{\mathrm{Tf}} \mathtt{out}(e_2^o)$, and similarly for all other messages 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 (occur in a message output to an external actor) before it can receive messages from outside the system. In particular e_1^o must happen before e_2 and e_4. Two possible interaction paths for the ticker interaction diagram are

$$ip_1{}^{\mathrm{Tf}} = ((tf, \emptyset)g_1{}^{\mathrm{Tf}}, A^{\mathrm{Tf}})$$

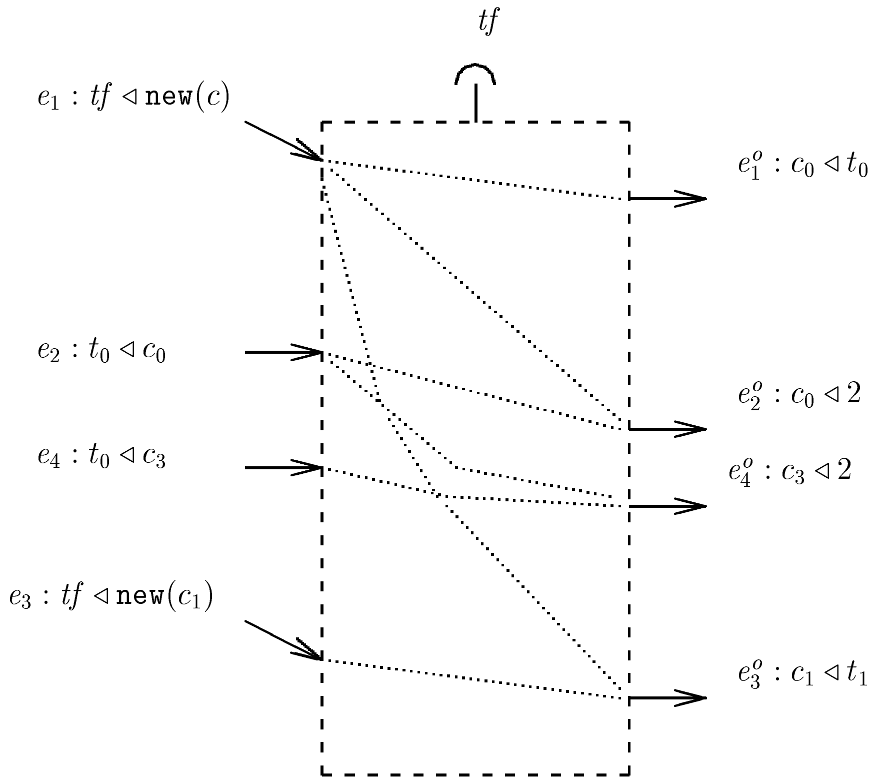

Figure 5. A Ticker Factory Interaction Diagram

$$
\begin{aligned}
&{g_1}^{\mathrm{Tf}} = [\mathtt{in}(e_1), \mathtt{out}(e_1^o), \mathtt{in}(e_2), \mathtt{out}(e_2^o), \mathtt{in}(e_3), \mathtt{in}(e_4), \mathtt{out}(e_4^o), \mathtt{out}(e_3^o)] \\
&{ip_2}^{\mathrm{Tf}} = ((tf, \emptyset){g_2}^{\mathrm{Tf}}, A^{\mathrm{Tf}}) \\
&{g_2}^{\mathrm{Tf}} = [\mathtt{in}(e_1), \mathtt{in}(e_3), \mathtt{out}(e_1^o), \mathtt{in}(e_2), \mathtt{in}(e_4), \mathtt{out}(e_4^o), \mathtt{out}(e_3^o), \mathtt{out}(e_2^o)]
\end{aligned}
$$

To complete the picture for the ticker factory we extend the ticker actor theory to describe ticker factory actors as well. For this we extend the set of states to include Tf, the state of a ticker factory actor. There is one reaction rule for ticker factory actors:

$$
\text{(tf1)} \quad \mathtt{[}Tf\mathtt{]}_{tf} \ , \ tf \triangleleft \mathtt{new}(c) \qquad \Rightarrow \qquad \mathtt{[}Tf\mathtt{]}_{tf} \ , \ \mathtt{[}T(0)\mathtt{]}_{t} \ , \ c \triangleleft t
$$

The interpretation of the rules ensures that t a new name.

3.3. 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 about two workers (w_0, w_1) and forwards incoming tasks non-deterministically to one of these workers. It makes a choice by getting a number, n, from a ticker, t, and picking worker w_0 if n is even and w_1 if n is odd. We further extend the ticker actor theory to describe the behavior of switch actors. We extend the set of states to include $S(t, [w_0, w_1])$ and $Sx(t, [w_0, w_1])$ for actor names t, w_0, w_1. $S(t, [w_0, w_1])$ is the state of a switch actor waiting for a task to dispatch. It has as acquaintances a ticker t and two workers $[w_0, w_1]$. $Sx(t, [w_0, w_1], v, c)$ is the state of a switch actor waiting for a reply from the ticker. It has additional parameters v and c specifying the task parameter and customer.

(s) $\quad [S(t, [w_0, w_1])]_s \,,\, s \lhd \mathtt{task}(v, c) \quad \Rightarrow \quad [Sx(t, [w_0, w_1], v, c)]_s \,,\, t \lhd s$

(sx) $\quad [Sx(t, [w_0, w_1], v, c)]_s \,,\, s \lhd n$

$$\Rightarrow \quad [S(t, [w_0, w_1])]_s \,,\, \begin{cases} w_0 \lhd \mathtt{task}(v, c) & \text{if } n \text{ is even} \\ w_1 \lhd \mathtt{task}(v, c) & \text{if } n \text{ is odd} \end{cases}$$

Figure 6 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.

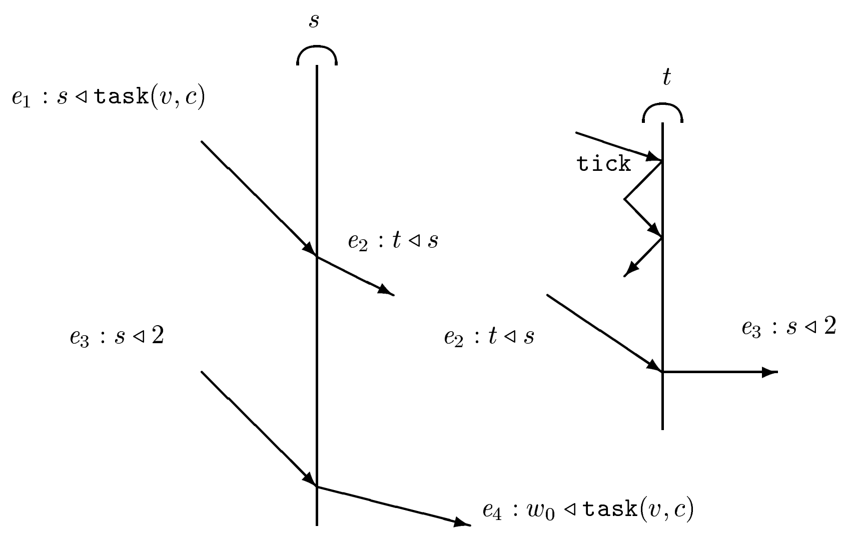

Figure 6. Event Diagrams for Dispatcher components

Figure 7 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; and matching input/output events are connected. Although in general, the rules for composition of event diagrams are a bit more

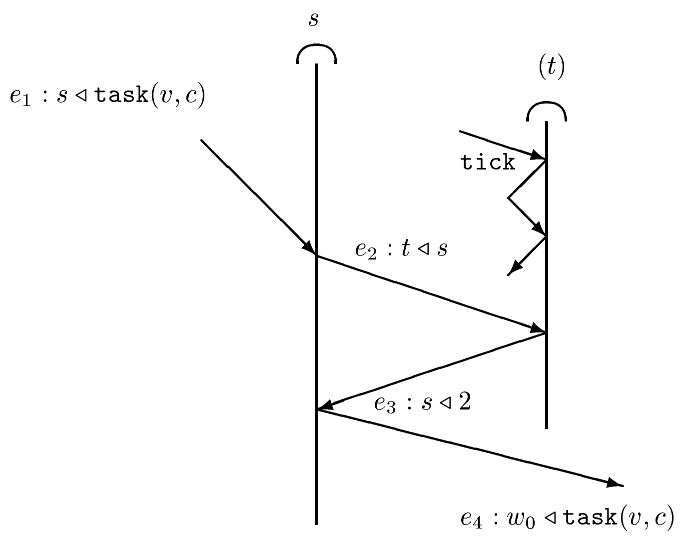

Figure 7. A Composed Event Diagram for Dispatcher

complex, the composition in figure 7 illustrates the basic idea – find diagrams for the two components such that each output from one component to a receptionist of the other has a matching input and zip them up. Here we considered only individual diagrams and paths and their composition. Later we will see that the every event diagram of the dispatcher can be obtained by zipping an event diagram of the switch and a matching diagram of the ticker.

To conclude our example we show how the formation of the dispatcher can be expressed as a term in the algebra of configurations. The starting configuration of a switch actor component

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

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.

$$(Switch[s](t, [w_0, w_1]) \parallel Ticker[t])\lceil\{s\} =$$
$$(s, \{w_0, w_1\})(\,[\![S(t, [w_0, w_1])]\!]_s\,,\ [\![T(t, 0)]\!]_t\,,\ e_0 : t \triangleleft \mathtt{tick} \mid \emptyset)$$

We leave to the reader the exercise of generating a computation for the composed configuration, corresponding to above event diagram.

4. Messages and Events

This section contains the mathematical notation and definitions regarding messages, events and event paths that will be used throughout the remainder of the paper. First we discuss the basic operations on actor names, messages, and events. Then we define the interface algebra, **Iface**. Finally we introduce the notions of global time map and event path (global time map with interface) and state the event path law *EPLaw* which adapts the actor locality laws to what can be observed about interactions of a system with its environment. We then define composability for event paths and prove that parallel composition is commutative and associative. This result is the starting point in proving, that the actor component algebra laws hold for each of the semantic domains studied.

4.1. 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 messages that can be communicated between actors. We let a range over $\mathbf{A}$ and M range over $\mathbf{M}$. Message packets (briefly packets) are pairs consisting of an actor name, the *target*, and a message. We let $\mathbf{MP}$ be the set of message packets and let mp range over $\mathbf{MP}$. The functions $target : \mathbf{MP} \to \mathbf{A}$ and $message : \mathbf{MP} \to \mathbf{M}$ select the first and second components of a message packet respectively. We write $a \triangleleft M$ for the packet with target a $(target(a \triangleleft M) = a)$ and message M $(message(a \triangleleft M) = M)$.

Actor names may be communicated in messages. Since the structure of messages 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 cannot matter. Different choices give the same results modulo a corresponding renaming. Renamings, α, are lifted to bijections, $\widehat{\alpha}$, on messages such that $acq(\widehat{\alpha}(M)) = \widehat{\alpha}(acq(M))$. A renaming that is the identity on the acquaintances of an object lifts to a function that maps the object to itself. Also the extension of the (functional) composition of two renamings is the composition of the extensions: $\widehat{\alpha \circ \alpha'} = \widehat{\alpha} \circ \widehat{\alpha'}$. 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 §3, $\mathbf{M}$ includes $\mathbf{A}$, $\mathtt{tick}$, $\mathtt{new}(c)$ and $\mathtt{task}(v, c)$ for $c \in \mathbf{A}$ and a set of task parameters, $v \in V$. We have $acq(\mathtt{tick}) = \emptyset$, $acq(c) = acq(\mathtt{new}(c)) = \{c\}$, and $acq(\mathtt{task}(v, c)) = \{c\} \cup acq(v)$ for $c \in \mathbf{A}$ and $v \in V$. Also, $\widehat{\alpha}(\mathtt{tick}) = \mathtt{tick}$ for any renaming α since $\mathtt{tick}$ has no acquaintances. Similarly, $\widehat{\alpha}(\mathtt{new}(c)) = \mathtt{new}(c')$, and $\widehat{\alpha}(\mathtt{task}(v, c)) = \mathtt{task}(v', c')$, if $\alpha(c) = c'$ and $\widehat{\alpha}(v) = v'$.

4.2. Events

To describe event diagrams we need a set, $\mathbf{E}$, of events over $\mathbf{A}$ and $\mathbf{M}$. Formally, events represent dynamic occurrences of message packets. We let e range over $\mathbf{E}$ and E range over $\mathcal{P}[\mathbf{E}]$. Each event, e, has a message packet, $pkt(e) \in \mathbf{MP}$. Also for each packet, mp, we assume that there are countably many events with packet mp. Which event represents an occurrence of a message packet is not important, what matters is the number of occurrences and that we can identify a particular occurrence when we need to. A *packet preserving* bijection on events is a bijection γ on events such that $pkt(\gamma(e)) = pkt(e)$. Such bijections will be lifted homomorphically to all structures containing events.

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:

$$target(e) = target(pkt(e)) \qquad message(e) = message(pkt(e))$$
$$acq(e) = acq(pkt(e)) \qquad acq(E) = \bigcup_{e \in E} acq(e)$$

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

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

In the actor world, 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 input of a message packet to the system from outside from its delivery e to the target actor. We also write $\mathtt{out}(e)$ in place of e to distinguish delivery of a message to the environment from the actual delivery to the (external) target actor (which we can not observe). 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 $\mathcal{P}[\widetilde{\mathbf{E}}]$. Packet preserving bijections on $\mathbf{E}$ extend naturally to $\widetilde{\mathbf{E}}$.

4.3. 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 **Iface** be the set of such interfaces. The empty interface is $(\emptyset, \emptyset)$. The interface component algebra is obtained by defining parallel composition, restriction, and renaming as follows. As mentioned in §1 parallel composition is restricted to components whose interfaces have disjoint receptionist sets. Renaming is just the pointwise lifting of the bijection to the components.

$$(\rho_0, \chi_0) \parallel (\rho_1, \chi_1) = (\rho_0 \cup \rho_1, (\chi_0 \cup \chi_1) - (\rho_0 \cup \rho_1)) \quad \text{if } \rho_0 \cap \rho_1 = \emptyset$$
$$(\rho, \chi)\lceil \rho_0 = (\rho \cap \rho_0, \chi)$$
$$\widehat{\alpha}(\rho, \chi) = (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))$$

Lemma (Interface algebra): The interface algebra satisfies the component algebra laws.

Proof : Basic properties of sets and pairs.□

Notice that although renaming commutes with parallel composition and restriction, it is not the case that restriction commutes with parallel composition, even in the case of interface components. That is, there are components C_0 and C_1, and a finite set of actor names, ρ, such that $(C_0 \parallel C_1)\lceil\rho \neq (C_0\lceil\rho \parallel C_1\lceil\rho))$. To see this let $C_0 = (\{a, b\}, c)$, $C_1 = (\{c, d\}, a)$, and $\rho = \{b, d\}$. Then

$$(C_0 \parallel C_1)\lceil\rho = (\{a, b, c, d\}, \{a, c\} - \{a, b, c, d\})\lceil\{b, d\} = (\{b, d\}, \emptyset)$$
$$(C_0\lceil\rho \parallel C_1\lceil\rho)) = (\{b, d\}, \{a, c\} - \{b, d\}) = (\{b, d\}, \{a, c\})$$

Since the interface algebra is a part of all of the other component algebras (every component has an interface), we see that distribution of restriction over parallel compositions fails in all the component algebras.

4.3.1. Interfaced structures Many of the structures we will be concerned with have the form $\sigma = (\rho, \chi)(\ldots)$. That is they consist of an interface, (ρ, χ), paired with some additional structure, $(\ldots)$. We call such structures *interfaced structures*. In the ticker example of §3 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:

$$\mathit{recep}(\sigma) = \rho \qquad \text{the receptionists of } \sigma$$
$$\mathit{extern}(\sigma) = \chi \qquad \text{the external actors of } \sigma$$
$$\mathit{iface}(\sigma) = (\rho, \chi) \qquad \text{the interface of } \sigma$$

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

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

4.4. 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 *ep* range over event paths. We use global time maps rather than simply the total ordering as they are a convenient means of defining compositions, as we will see below. The empty event path, `mtep`, has an empty interface and empty global time map: $\texttt{mtep} = (\emptyset, \emptyset)\texttt{mtmap}$. If $ep = (\rho, \chi)g$, we write $ep(\tilde{e})$ meaning $g(\tilde{e})$ and $\mathrm{Dom}(ep)$ meaning $\mathrm{Dom}(g)$.

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 $ep = (\rho, \chi)g$. Then

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

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

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

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

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

Note that receptionists and externals of an event path do not depend on internal events, only on input and output events.

Returning to the ticker factory of § 3, recall the interaction path

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

is one of the global times for the ticker factory interaction diagram, id^{Tf} (figure 5). This sequence of events represents a function with domain the elements of the sequence, mapping the i-th element of the sequence, to i. For example $g_2{}^{\mathrm{Tf}}(\texttt{in}(e_3)) = 1$. Recall that the event packets for the ticker factory example are:

$e:$	e_1	e_2	e_3	e_4
$pkt(e):$	$tf \triangleleft \texttt{new}(c_0)$	$t_0 \triangleleft c_0$	$tf \triangleleft \texttt{new}(c_1)$	$t_0 \triangleleft c_3$

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

Thus, letting $\rho_i = recep({ip_2}^{\mathrm{Tf}}, i)$ and $\chi_i = extern({ip_2}^{\mathrm{Tf}}, i)$ we have

$$\begin{array}{lllllll} i: & 0 & 1 & 2 & 3 & 4 & \ldots \\ \rho_i: & tf & tf & tf & tf, t_0 & tf, t_0 & \ldots \\ \chi_i: & \emptyset & c_0 & c_0, c_1 & c_0, c_1 & c_0, c_1 & \ldots \end{array}$$

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) $recep^{\omega}((\rho, \chi)g) \cap extern^{\omega}((\rho, \chi)g) = \emptyset$

4.5. The Event Path Law and Scoped Event Paths

The *event path law*, $EPLaw(ep, A)$, for an event path ep and a set of actor names A (intended to be names reserved for internal use by ep) expresses the actor locality laws [6, 7] as constraints on actors exported and imported via the path interface. It requires that each input event of an interaction path must be a message to a current receptionist, that a name in A can only appear as an acquaintance of an input message if it is currently a receptionist, and that an output event of an interaction path must be to a currently known external actor.

Definition ($EPLaw(ep, A)$): The event path law $EPLaw(ep, A)$ holds for an event path ep and reserved names A just if conditions (ep.in) and (ep.out) hold:

(ep.in) $ep(\mathtt{in}(e)) = i \Rightarrow (target(e) \in recep(ep, i) \wedge (acq(message(e)) \cap A) \subseteq recep(ep, i))$

(ep.out) $ep(\mathtt{out}(e)) = i \Rightarrow target(e) \in extern(ep, i)$

It is easy to see that $EPLaw({ip_2}^{\mathrm{Tf}}, A^{\mathrm{Tf}})$ holds, but 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 $EPLaw((tf, \emptyset){g_3}^{\mathrm{Tf}}, A^{\mathrm{Tf}})$ fails.

Event paths exist in the context of a set of actor names that are reserved for use by the component being modeled, i.e. names that may be exported by the component as new receptionists, but may not be imported as new external actors (receptionists of some other component). In order to restrict composition of event paths (merging of global time views of two components) sufficiently to ensure associativity, we make these reserved actors explicit. We do this by defining a *scoped event path* to be a pair (ep, A) such that $recep^{\omega}(ep) \subseteq A$ and $extern^{\omega}(ep) \cap A = \emptyset$, and $target(e) \in A$ for $e \in \mathrm{Dom}(ep)$. Thus every receptionist name is a reserved name, every internal event is targeted to an actor with a reserved name, and no external name is a reserved name.

For more compact definitions and proofs, we adopt the following notational conventions. For event paths ep_x we let (ρ_x, χ_x) be the interface and g_x be the global time. Thus $ep_x = (\rho_x, \chi_x)g_x$. (Here x stands for some subscript.) Finally we define

$$\rho_x^i = recep(ep_x, i) \qquad \chi_x^i = extern(ep_x, i)$$
$$\rho_x^\omega = recep^\omega(ep_x) \qquad \chi_x^\omega = extern^\omega(ep_x)$$

4.6. Composing Scoped Event Paths

Parallel composition of scoped event paths is the cornerstone for defining composition in all the semantic models.

Two extended events are *complements* if one is of the form $\mathtt{in}(e)$ and the other is of the form $\mathtt{out}(e)$. The parallel composition of $(ep_0, A_0), (ep_1, A_1)$, when defined, has as interface the composition of (ρ_0, χ_0) and (ρ_1, χ_1). Its global time map is the *zip*ping of g_0 with g_1 — the union of the two maps, omitting complementary event pairs with the same global time. The reserved actors of the composition is the union of A_0 and A_1.

Composability is defined in two stages: the first stage simply guarantees that the parallel composition of two event paths given by zipping them together is well-defined. The second requires that the result of zipping satisfies the event path law in order for composition to be meaningful.

Definition (Zips With): If (ep_0, A_0) and (ep_1, A_1) are scoped event paths, then (ep_0, A_0) *zips* (ep_1, A_1) (read (ep_0, A_0) zips with (ep_1, A_1)) just if conditions (z.1 - z.4) below hold.

(z.1) $A_0 \cap A_1 = \emptyset$

(z.2) $\mathrm{Dom}(g_0) \cap \mathrm{Dom}(g_1) = \emptyset$

(z.3) $ep_j(\mathtt{out}(e)) = i \wedge target(e) \in A_{1-j} \Rightarrow ep_{1-j}(\mathtt{in}(e)) = i \quad \text{for} \quad i \in \mathbf{N}, j < 2$

(z.4) $ep_0(\tilde{e}_0) = ep_1(\tilde{e}_1)$ implies $\tilde{e}_0$ and $\tilde{e}_1$ are complements

Note that (z.4) implies that $ep_0(e_0) \neq ep_1(e_)$ if $e_j \in \mathrm{Dom}(ep_j) \cap \mathbf{E}$ for $j < 2$ since events in $\mathbf{E}$ do not have compliments. It is easy to see from the definition that if (ep_0, A_0) *zips* (ep_1, A_1) and $recep^\omega(ep_j) \subseteq A'_j \subseteq A_j$, then (ep_0, A'_0) *zips* (ep_1, A'_1).

Definition (Zipping): If (ep_0, A_0) and (ep_1, A_1) are scoped event paths, and (ep_0, A_0) *zips* (ep_1, A_1), then the result of *zipping* them together, $(ep_0, A_0) \parallel (ep_1, A_1)$ is defined by

$$(ep_0, A_0) \parallel (ep_1, A_1) = ((ep_0 \parallel ep_1), (A_0 \cup A_1))$$

where (recalling our convention that $ep_0 = (\rho_0, \chi_0)g_0$, etc.)

$$((\rho_0, \chi_0)g_0 \parallel (\rho_1, \chi_1)g_1) = (\rho, \chi)g \quad \text{where}$$

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

$$\mathrm{Dom}(g) = (\mathrm{Dom}(g_0) \cup \mathrm{Dom}(g_1)) - S$$

$$S = \bigcup_{\{e \mid (\exists j<2)(g_j(\mathtt{in}(e)) = g_{1-j}(\mathtt{out}(e)))\}} \{\mathtt{in}(e), \mathtt{out}(e)\} \qquad \text{the complementary events}$$

$$g(\tilde{e}) = g_j(\tilde{e}) \quad \text{if } \tilde{e} \in \mathrm{Dom}(g_j)$$

We extend our notation conventions, defining $ep_{x,y} = ep_x \parallel ep_y$ if ep_x *zips* ep_y (omitting the ',' if x, y are single characters). Thus $(ep_{01}, A_{01}) = (ep_0, A_0) \parallel (ep_1, A_1) = (ep_0 \parallel ep_1, A_0 \cup A_1)$.

Lemma (zip.0): If ep_0 *zips* ep_1 then for $i \in \mathbf{N}$

(i) $\rho^i_{01} \subseteq (\rho^i_0 \cup \rho^i_1)$

(ii) $(\chi^i_0 \cup \chi^i_1) - (\rho^i_0 \cup \rho^i_1) \subseteq \chi^i_{01}$

Proof : By induction on i. For $i = 0$ it is a trivial calculation. Assume $i > 0$. To prove (i) note that if $\rho^i_{01} = \rho^{i-1}_{01}$ then we are done by the induction hypothesis and monotonicity. Otherwise, assume $a \in \rho^i_{01} - \rho^{i-1}_{01}$. Then, by definition of receptionists, there is some e such that $ep_{01}(\mathtt{out}(e)) = i - 1$ and $a \in acq(message(e)) - (\chi^{i-1}_{01} \cup \rho^{i-1}_{01})$. Without loss, suppose $ep_0(\mathtt{out}(e)) = i - 1$, then either $a \in \rho^i_0$ or $a \in (\chi^{i-1}_0 - \rho^{i-1}_0)$ and by the induction hypothesis for (ii) $a \in \rho^{i-1}_1$.

To prove (ii) note that if $(\chi^i_0 \cup \chi^i_1) = (\chi^{i-1}_0 \cup \chi^{i-1}_1)$ then again we are done by the induction hypothesis and monotonicity. Otherwise, assume that a and e are such that $a \in \chi^i_0 - (\chi^{i-1}_0 \cup \chi^{i-1}_1 \cup \rho^i_0 \cup \rho^i_1)$, $ep_0(\mathtt{in}(e)) = i - 1$, and $a \in acq(message(e)) - (\chi^{i-1}_0 \cup \rho^{i-1}_0)$. We must show $a \in \chi^i_{01}$. If $ep_{01}(\mathtt{in}(e)) = i - 1$, then $a \in (\rho^{i-1}_{01} \cup \chi^i_{01})$ and we are done since $a \in \rho^{i-1}_{01}$ is impossible by (i-IH). Otherwise $ep_1(\mathtt{out}(e)) = i - 1$ and $a \in (\rho^i_1 \cup \chi^{i-1}_1)$ which contradicts our assumption and we are done.

□

Definition (scoped composability): If (ep_0, A_0) and (ep_1, A_1) are scoped event paths satisfying *EPLaw*, then $(ep_0, A_0) \bowtie (ep_1, A_1)$ (read (ep_0, A_0) and (ep_1, A_1) are composable) just if (ep_0, A_0) *zips* (ep_1, A_1), $EPLaw(ep_{01}, A_{01})$ and condition (z.5) holds:

(z.5) $\chi^i_j \cap A_{1-j} \subseteq \rho^i_{1-j}$ for $i \in \mathbf{N}, j < 2$

(z.5) says that any name reserved for one component that is known to another component at stage i must be a receptionist of the reserving component at stage i.

Lemma (scoped composition): If (ep_0, A_0) and (ep_1, A_1) are scoped event paths satisfying *EPLaw* and $(ep_0, A_0) \bowtie (ep_1, A_1)$, then $(ep_0, A_0) \parallel (ep_1, A_1)$, is a scoped event path satisfying *EPLaw*.

Proof : By definition of $\bowtie$ and (**zip.0**) the only thing to check is that $\chi^{\omega}_{01} \cap (A_{01}) = \emptyset$. This follows from (**ep.in**). □

As an example let p^s, p^t be event paths for the switch and ticker events in Figure 6

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

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 complementary events at 4 and 7 and

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

As an example of event paths that are zippable but not composable consider

$$ep_0 = (tf, \emptyset)[\mathtt{in}(e_1), \mathtt{out}(e_1^o), \mathtt{in}(e_4)]$$
$$ep_1 = (c_0, tf)[\mathtt{out}(e_1), \mathtt{in}(e_1^o)]$$

where $pkt(e_1) = tf \triangleleft \mathtt{new}(c_0)$, $pkt(e_1^o) = c_0 \triangleleft t_0$, and $pkt(e_4) = t_0 \triangleleft c_3$. ep_0 is a possible interaction of the ticker factory and ep_1 is a possible environment component interaction. It is easy to see that the two event paths are zippable and the result of zipping is

$$ep_{01} = (\{tf, c_0\}, \emptyset)[\mathtt{in}(e_4)]$$

But the event path law fails for ep_{01} because the target of e_4 is not a receptionist of the zipping.

Requiring the events and not just the packets to be identical in complementary input/output events may seem unnecessarily restrictive. However, since the semantic sets we are interested in are closed under packet preserving bijections on $\mathbf{E}$ this requirement is in fact not a serious restriction.

The following lemmas are the key properties of scoped event path composition needed to establish the algebraic laws for composition of components.

Lemma (ep.id.commute): Let (ep, A) and (ep_j, A_j) for $j \leq 2$ be scoped event paths satisfying the event path law. Then we have the following:

(0) $(ep, A) \bowtie (\mathtt{mtep}, \emptyset)$ and $(ep, A) \parallel (\mathtt{mtep}, \emptyset) = (ep, A)$

(1) $(ep_0, A_0) \bowtie (ep_1, A_1) \Rightarrow (ep_1, A_1) \bowtie (ep_0, A_0) \wedge$
$(ep_0, A_0) \parallel (ep_1, A_1) = (ep_1, A_1) \parallel (ep_0, A_0)$

Ahyp is a convenient set of conditions for which we can prove that composition of scoped event paths is associative. Zipping is not really a problem. The reason for the extra fuss is to assure that the result satisfies *EPLaw*.

Definition (Ahyp): *Ahyp* says (ep_j, A_j) for $j < 3$ are scoped event paths such that (a.0–4) below hold.

(a.0) $\quad EPLaw(ep_j, A_j) \quad \text{for} \quad j < 3$

(a.1) $\quad (ep_0, A_0) \bowtie (ep_1, A_1)$

(a.2) $\quad (ep_{01}, A_{01}) \bowtie (ep_2, A_2)$

(a.3) $\quad (\mathrm{Rng}(g_0) \cup \mathrm{Rng}(g_1)) \cap \mathrm{Rng}(g_2) \subseteq \mathrm{Rng}(g_{01})$

Note that since we will generally work with sets closed under re-representation, *Ahyp* condition (a.3) is not a serious constraint. Thus *Ahyp* expresses just what we can readily assume or directly infer when $((ep_0, A_0) \parallel (ep_1, A_1)) \parallel (ep_2, A_2)$.

Lemma (ahyp.cor): *Ahyp* implies

$$(\mathrm{Dom}(g_0) \cup \mathrm{Dom}(g_1)) \cap \mathrm{Dom}(g_2) = \emptyset$$

Proof : Assume *Ahyp* and without loss assume $\tilde{e} \in \mathrm{Dom}(g_0) \cap \mathrm{Dom}(g_2)$. By *EPLaw* and disjointness of reserved names, A_j, we have $\tilde{e} = \mathtt{out}(e)$ for some e with $target(e) \in \chi_0^\omega \cap \chi_2^\omega$. Furthermore, by (a.2) $\tilde{e} \notin \mathrm{Dom}(g_{01})$, so $g_1(\mathtt{in}(e)) = g_0(\mathtt{out}(e))$. Thus $target(e) \in A_1$. But by composability of ep_{01} with ep_2, $\chi_2^\omega \cap \rho_1^\omega \subseteq \rho_{01}^\omega$ which means that $\mathtt{in}(e) \in \mathrm{Dom}(g_{01})$. This is a contradiction. □

Lemma (ep.assoc): *Ahyp* implies

(assoc.1) $\quad (ep_j, A_j) \bowtie (ep_2, A_2) \quad \text{for} \quad j < 2$

(assoc.2) $\quad (ep_j, A_j) \bowtie (ep_{1-j,2}, A_{1-j,2}) \quad \text{for} \quad j < 2$

Corollary (ep.assoc): *Ahyp* implies all ways of composing the three are defined and equal:

$$(ep_{01}, A_{01}) \parallel (ep_2, A_2) = (ep_0, A_0) \parallel (ep_{12}, A_{12}) = (ep_1, A_1) \parallel (ep_{02}, A_{02})$$

We split the proof of (**ep.assoc**) into a series of lemmas. We extend the indexing notation as follows. (z.1[x,y]) is condition (z.1) applied to $(ep_x, A_x), (ep_y, A_y)$. Similarly for other conditions.

Lemma (zips.j.2): *Ahyp* implies (ep_j, A_j) *zips* (ep_2, A_2) for $j < 2$.

Proof : We take $j = 1$, the case $j = 0$ follows by symmetry. Conditions (z.1) and (z.2) follow directly from *Ahyp*, (**ahyp.cor**) and the definition of $\parallel$.

For (z.3) first assume $ep_2(\mathtt{out}(e)) = i$ and $a = target(e) \in A_1$ and show $ep_1(\mathtt{in}(e)) = i$. By event path properties, $a \in \chi_2^i \cap A_{01}$, so by (z.5[01,2]) $a \in \rho_{01}^i$, and by (z.3[01,2]) $ep_{01}(\mathtt{in}(e)) = i$. Thus by disjointness of A_0 and A_1 and *EPLaw* we have $ep_1(\mathtt{in}(e)) = i$.

Next assume $ep_1(\mathtt{out}(e)) = i$ and $a = target(e) \in A_2$ and show $ep_2(\mathtt{in}(e)) = i$. By disjointness of reserved actors i is not a global time of a pair of complementary events for ep_0, ep_1, thus $ep_{01}(\mathtt{out}(e)) = i$ and we are done by (z.3[01,2]).

For (z.4) assume $ep_1(\tilde{e}_1) = ep_2(\tilde{e}_2) = i$ and show $\tilde{e}_1$ and $\tilde{e}_2$ are complements. By composability of ep_0 and ep_1 and (a.3) $i \notin \mathrm{Rng}(ep_0)$, thus $ep_{01}(\tilde{e}_1) = i$ and by (z.4[01,2]) we are done.

□

Lemma (zips.j2.1-j): *Ahyp* implies (ep_j, A_j) *zips* $(ep_{1-j,2}, A_{1-j,2})$ for $j < 2$.

Proof : We take $j = 0$, the case $j = 1$ follows by symmetry. Again conditions (z.1) and (z.2) follow directly from *Ahyp*, (**ahyp.cor**), and the definition of $\|$.

For (z.3) first assume $ep_0(\mathtt{out}(e)) = i$ and $a = \mathit{target}(e) \in A_{12}$ and show $ep_{12}(\mathtt{in}(e)) = i$. If $a \in A_1$ then $ep_1(\mathtt{in}(e)) = i$ by (z.3[0,1]). If $a \in A_2$ then $ep_2(\mathtt{in}(e)) = i$ by (z.3[01,2]). Thus by disjointness conditions on the domains $\mathtt{out}(e) \notin \mathrm{Dom}(ep_1) \cup \mathrm{Dom}(ep_2)$, and $ep_{12}(\mathtt{in}(e)) = i$.

Next assume $ep_{12}(\mathtt{out}(e)) = i$ and $a = \mathit{target}(e) \in A_0$ and show $ep_0(\mathtt{in}(e)) = i$. If $ep_1(\mathtt{out}(e)) = i$, then we are done by (z.3[0,1]), and if $ep_2(\mathtt{out}(e)) = i$, then we are done by (z.3[0,2]).

For (z.4) assume $ep_0(\tilde{e}_1) = ep_{12}(\tilde{e}_{12}) = i$ and show $\tilde{e}_1$ and $\tilde{e}_2$ are complements. There are two cases. If $ep_1(\tilde{e}_{12}) = i$ then we are done by (z.4[0,1]). If $ep_2(\tilde{e}_{12}) = i$ then we are done by (z.4[0,2]).

□

By (**zips.j2.1-j**) (assuming *Ahyp*) we see that $ep_{1-j} \| ep_{j2}$ is well-defined and equal to $ep_{01} \| ep_2$ for $j < 2$. We will denote this composition by ep_{012} in the following, and write ρ^i_{012} for $\mathit{recep}(ep_{012}, i)$, etc.

Lemma (z.5[j.2]): *Ahyp* implies (z.5[j,2]) for $j < 2$.

Proof : $\chi^i_2 \cap A_1 \subseteq \rho^i_1$ follows from *Ahyp* condition (z.5[01,2]) and (**zip.0**). To show that $(\chi^i_0 \cup \chi^i_1) \cap A_2 \subseteq \rho^i_2$ we note that

$$
\begin{aligned}
&(\chi^i_0 \cup \chi^i_1) \cap A_2 \\
&\quad = ((\chi^i_0 \cup \chi^i_1) - (\rho^i_0 \cup \rho^i_1)) \cap A_2 \qquad \text{by disjointness of reserved sets} \\
&\quad \subseteq \chi^i_{01} \cap A_2 \qquad \text{by } (\textbf{zip.0}) \\
&\quad \subseteq \rho^i_2 \qquad \text{by composability of } ep_{01} \text{ and } ep_2
\end{aligned}
$$

□

Lemma (ep[j.2]): *Ahyp* implies (ep.in[j,2]) and (ep.out[j,2]) for $j < 2$.

Proof : We show $ep_{12}(\mathtt{in}(e)) = i$ implies $(\{\mathit{target}(e)\} \cup \mathit{acq}(\mathit{message}(e)) \cap A_{12}) \subseteq \rho^i_{12}$ and $ep_{12}(\mathtt{out}(e)) = i$ implies $\mathit{target}(e) \in \chi^i_{12}$ by induction on i.

For $i = 0$ if $ep_{12}(\mathtt{in}(e)) = 0$ then $ep_1(\mathtt{in}(e)) = 0$ or $ep_2(\mathtt{in}(e)) = 0$. Thus by scoped event path properties $(\{\mathit{target}(e)\} \cup (\mathit{acq}(\mathit{message}(e)) \cap A_{12})) \subseteq \rho^0_1 \cup \rho^0_2 = \rho^0_{12}$.

If $ep_{12}(\mathtt{out}(e)) = 0$ then $ep_1(\mathtt{out}(e)) = 0$ or $ep_2(\mathtt{out}(e)) = 0$ and thus $\mathit{target}(e) \in (\chi^0_1 \cup \chi^0_2 - \rho^0_1 \cup \rho^0_2) = \chi^0_{12}$ by event path and zips with properties.

Now assume (ep.in-out[1,2]) for $i' \leq i$ (call this assumption IH).

Assume $ep_{12}(\mathtt{in}(e)) = i + 1$ and $a \in \{\mathit{target}(e)\} \cup \mathit{acq}(\mathit{message}(e)) \cap (A_{12})$. We must show $a \in \rho^{i+1}_{12}$. Note that $a \notin A_0$ by disjointness of A_0 and A_{12}. There are two cases. If $ep_{012}(\mathtt{in}(e)) = i+1$, then $a \in \rho_{012}^{i+1} \cap A_{12} \subseteq (\rho_0^{i+1} \cup \rho_{12}^{i+1}) \cap A_{12} = \rho_{12}^{i+1}$ by (**zip.0**). Otherwise $ep_0(\mathtt{out}(e)) = i+1$, $a \in \chi_0^{i+1} \cap A_{12}$. By (z.5[0,j]) $a \in \rho_1^{i+1} \cup \rho_2^{i+1}$. We consider the least $i' \leq i$ such that $a \in \chi_0^{i'}$. If $i' = 0$ we are done since $\rho_{12}^0 = \rho_1^0 \cup \rho_2^0$. Otherwise there is some e' such that $ep_0(\mathtt{in}(e')) = i' - 1$ and $a \in (acq(message(e')) - A_0) \cap A_{12}$. There are two cases. If $ep_{012}(\mathtt{in}(e')) = i'$ then by *Ahyp* and (**zip.0**) $a \in \rho_{012}^{i'} \subseteq \rho_{12}^{i'}$. Otherwise $ep_{12}(\mathtt{out}(e')) = i'$ and by IH $a \in \rho_{12}^{i'}$.

Assume $ep_{12}(\mathtt{out}(e)) = i+1$ and $a = target(e)$. We must show $a \in \chi_{12}^{i+1}$. Suppose $ep_1(\mathtt{out}(e)) = i+1$. Then $a \in \chi_1^{i+1}$ also $a \notin (\rho_1^{i+1} \cup \rho_2^{i+1})$ by zips[1,2]. Thus by (**zip.0**) we are done. Similarly for $ep_2(\mathtt{out}(e)) = i+1$.

□

Lemma (cmps[1-j,j2]): *Ahyp* implies $(ep_{1-j}, A_{1-j}) \bowtie (ep_{j2}, A_{j2})$ for $j < 2$.

Proof : (ep.in,ep.out) follow from the observation that $ep_{1-j,j2} = ep_{01,2}$. For (z.5) we show $\chi_0^i \cap A_{12} \subseteq \rho_{12}^i$ by induction on i. The other case follows by symmetry for each i.

If $i = 0$ we have $\chi_0^0 \cap A_{12} \subseteq \rho_1^0 \cup \rho_2^0 = \rho_{12}^0$ by (z.5[0,1]) and (z.5[0,2]). Now, assume $a \in \chi_0^{i+1} \cap A_{12}$ and show $a \in \rho_{12}^{i+1}$. If $a \in \chi_0^i$, then we are done by the induction hypothesis and monotonicity of receptionists and externals. Thus we may assume there is some e such that $ep_0(\mathtt{in}(e)) = i$ and $a \in (acq(message(e)) - \chi_0^i - \rho_0^i) \cap A_{12}$. There are two cases according to the source of the event. If $ep_{12}(\mathtt{out}(e)) = i$, then $a \in \rho_{12}^{i+1}$. If $ep_{012}(\mathtt{in}(e)) = i$, then $a \in \rho_{012}^i \cap A_{12} \subseteq \rho_{12}^i$ by *Ahyp* and (**zip.0**).

□

Proof (Assoc): We assume (ep_j, A_j) for $j < 3$ are scoped event paths such that *Ahyp* holds. (assoc.1) follows from lemmas (**zips.j.2,z.5[j.2],ep[j.2]**) (assoc.2) follows from lemmas (**zips.j2.1-j,cmps[1-j,j2]**)

□

5. Open Event Diagrams

In this section open event diagrams (OEDs) are defined as mathematical structures of a certain form that satisfy the Baker-Hewitt [6] ordering and locality laws for actor computation. Also a component algebra structure is defined on OED with the complete OEDs forming a sub-algebra. The definition of OEDs is basically an adaption of parts of [7] to account for component encapsulation in the form of explicit interfaces, and for the interactions with the environment, particularly the expanding set of receptionists.

5.1. *Structure and axioms*

Open Event Diagrams are structures of the form

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

such that

$$
\begin{array}{ll}
(\rho, \chi) \in \mathbf{Iface} & \text{—initial receptionists and external actors} \\
A \subseteq \mathbf{A} & \text{—internal actors} \\
E \subseteq \mathbf{E} & \text{—the realized events} \\
P \subseteq \mathbf{E} & \text{—the pending events} \\
\rho \subseteq A \qquad \chi \cap A = \emptyset \qquad E \cap P = \emptyset & \\
\mathit{activator} : E \cup P \to E \cup \{\mathbf{?}, !\} & \text{—generates the activation order} \\
\xrightarrow{\text{arr}} & \text{—arrival order on events with target } a \in A \\
\quad = \bigcup_{a \in A} \xrightarrow{\text{arr}}_a & \text{where } \xrightarrow{\text{arr}}_a \text{ is a total order on } E/a \\
\mathit{acqB} : E/A \to \mathcal{P}_\omega[\mathbf{A}] & \text{—acquaintances of the target actor just before arrival time} \\
\mathit{Cr} : A \to E/A \cup \{!\} & \text{—creation event, } \mathit{Cr}(a) = ! \text{ if initially present}
\end{array}
$$

Recall that $E/A = \{e \in E \mid \mathit{target}(e) \in A\}$. Also, recall that $\mathit{activator}(e) = !$ means that e was initially present, and $\mathit{Cr}(a) = !$ means that a was initially present. $\mathit{activator}(e) = \mathbf{?}$ means that e was activated by an input from the environment.

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

$$
\begin{aligned}
&\mathit{In}(\mathit{oed}) = \{\mathtt{in}(e) \mid e \in E \cup P \wedge \mathit{activator}(e) = \mathbf{?}\} \\
&\mathit{Out}(\mathit{oed}) = \{\mathtt{out}(e) \mid e \in E \wedge \mathit{target}(e) \notin A\} \\
&\mathit{Dlv}(\mathit{oed}) = E/A \\
&\widetilde{E}(\mathit{oed}) = \mathit{Dlv}(\mathit{oed}) \cup \mathit{In}(\mathit{oed}) \cup \mathit{Out}(\mathit{oed})
\end{aligned}
$$

Note that if $\mathtt{in}(e) \in \widetilde{E}(\mathit{oed})$ then $e \in E \cup P$, thus e may appear both with and without the input wrapper in $\widetilde{E}(\mathit{oed})$. However, if $e \in E$, then either $\mathit{target}(e) \in A$ and $e \in \widetilde{E}(\mathit{oed})$ or $\mathit{target}(e) \notin A$ and $\mathtt{out}(e) \in \widetilde{E}(\mathit{oed})$. But never are e and $\mathtt{out}(e)$ both elements of $\widetilde{E}(\mathit{oed})$.

Let $e \in \mathit{Dlv}(\mathit{oed})$ be an internal event with $\mathit{target}(e) = a \in A$. Then $\mathit{rank}(e)$ is the rank of e in E/a given by the total ordering $\xrightarrow{\text{arr}}_a$. For example $\mathit{rank}(e) = 0$ if e is the least element of $\xrightarrow{\text{arr}}_a$.

The *activation order* on E, $\prec_{\text{ao}}$, is the least transitive relation with $\mathit{activator}(e) \prec_{\text{ao}} e$ for $e \in E$. The *combined order*, $\prec_{\text{co}}$, is the least transitive relation containing

$\prec_{\rm ao}$ and $\xrightarrow{\rm arr}$. As examples consider the ticker factory event diagram of figure 4.

$$e_1 : \mathit{tf} \triangleleft \mathtt{new}(c_0) \prec_{\rm ao} e_{01} : t_0 \triangleleft \mathtt{tick}$$

$$e_1 \prec_{\rm ao} e_1^o : c_0 \triangleleft t_0$$

$$e_1 \prec_{\rm co} e_3 : \mathit{tf} \triangleleft \mathtt{new}(c_1)$$

and the dispatcher event diagram of figure 7

$$e_1 : s \triangleleft \mathtt{task}(v, c) \prec_{\rm ao} e_4 : w_0 \triangleleft \mathtt{task}(v, c)$$

since

$$e_1 : s \triangleleft \mathtt{task}(v, c) \prec_{\rm ao} e_2 : t \triangleleft s \prec_{\rm ao} e_3 : s \triangleleft 2 \prec_{\rm ao} e_4 : w_0 \triangleleft \mathtt{task}(v, c)$$

We view $\prec_{\rm ao}$, $\xrightarrow{\rm arr}$, and $\prec_{\rm co}$ as relations on extended events by requiring $\mathtt{in}(e) \prec_{\rm ao} e$ if $\mathit{activator}(e) = \mathbf{?}$ and defining $\mathit{activator}(\mathtt{out}(e)) = \mathit{activator}(e)$ (recall that in this case $\mathtt{out}(e)$ has replaced e). Returning to the ticker factory example, notice that $\mathtt{in}(e_1) \prec_{\rm co} e_3$ but $\neg(\mathtt{in}(e_1) \prec_{\rm co} \mathtt{in}(e_3))$. Also, in the dispatcher example we have $\mathtt{in}(e_1) \prec_{\rm co} \mathtt{out}(e_4)$.

Notice that the combined order relates input events to deliveries, deliveries to deliveries, and deliveries to outputs, but never outputs to inputs. In particular for all $\mathtt{in}(e)$, $\mathtt{out}(e')$ in $\widetilde{E}(\mathit{oed})$ we have $\neg(\mathtt{out}(e') \prec_{\rm co} \mathtt{in}(e))$. This observation will be used when we define interaction diagrams.

The participants in an event, $\mathit{participants}(e)$, are the acquaintances of the target at delivery time together with the acquaintances delivered in the message and the actors created by that event.[1]

$$\mathit{participants}(e) = \mathit{acqB}(e) \cup \mathit{acq}(e) \cup \mathit{Cr}^{\,-1}(e)$$

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

A global time for *oed* is a total ordering on the extended events of *oed*, given by a global time mapping g, that is compatible with the (extended) combined ordering. We further require that the event path law holds for the event path $(\rho, \chi)g$ with reserved names, A. The set of global times, $GT(\mathit{oed})$, for *oed* is defined as follows.

$$\begin{aligned} GT(\mathit{oed}) = \{g : \widetilde{E}(\mathit{oed}) \xrightarrow{\rm inj} \mathbf{N} \mid \\ & (\forall \tilde{e}, \tilde{e}' \in \widetilde{E}(\mathit{oed}))(\tilde{e} \prec_{\rm co} \tilde{e}' \Rightarrow g(\tilde{e}) < g(\tilde{e}')) \\ \text{and} \\ & \mathit{EPLaw}((\rho, \chi)g, A)\} \end{aligned}$$

$\mathit{OED}^{\dagger}$ is the set of OED global time pairs:

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

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 [7] 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 [7].)

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

(in the beginning)

$$\rho \subseteq Cr^{\,-1}(!)$$
if $e \in E \cup P$ and $activator(e) = !$, then $acq(e) \subseteq Cr^{\,-1}(!) \cup \chi$.

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

$$acqB(e) \subseteq Cr^{\,-1}(!) \cup \chi \quad \text{if } Cr(a) = !$$
$$acqB(e) \subseteq participants(Cr(a)) \quad \text{if } Cr(a) \neq !$$

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

(law of activator acquaintances): if $e \in E \cup P$, $e' \in Dlv(oed)$ and $activator(e) = e'$, then $acq(e) \subseteq participants(e')$.

(finite activation): $participants(e)$ is finite for $e \in Dlv(oed)$.

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} of § 3 satisfy the actor laws and thus are OEDs. The empty OED, `mtoed`, is the structure:

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

An OED, $oed = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P) \in OED$, is *complete* if $P = \emptyset$. *cOED* denotes the set of complete OEDs. Note that the ticker event diagram oed^{cT} of §3 is complete.

Recalling the convention regarding interfaces we have:

$$\mathit{iface}(\mathit{oed}) = (\mathit{recep}(\mathit{oed}), \mathit{extern}(\mathit{oed})) = (\rho, \chi) \qquad \text{for } \mathit{oed} \text{ as above}$$

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

The event ordering in an event diagram has no direct representation of the causal link between the event creating an actor event and the events targeted to that actor. The actor laws however entail an implicit ordering in the sense that in any global time for that event diagram the creation event will happen before the first message arrival.

Lemma (creation): If $(\mathit{oed}, g) \in \mathit{OED}^\dagger$, with notation as above for the parts of oed, $e \in \mathit{Dlv}(\mathit{oed})$, $a \in (\mathit{acq}(e) \cup \mathit{acqB}(e)) \cap A$, and $\mathit{Cr}(a) \neq\, !$, then

$$g(\mathit{Cr}(a)) < g(e)$$

Proof :
With notation as in the hypothesis, assume $a \in (\mathit{acq}(e) \cup \mathit{acqB}(e)) \cap A$, and $\mathit{Cr}(a) \neq\, !$ and show $g(\mathit{Cr}(a)) < g(e)$ by induction on $g(e)$. There are two cases to consider: (1) $a \in \mathit{acq}(e)$; and (2) $a \in \mathit{acqB}(e)$. For case (1) there are three sub-cases according to the activator of e: !; **?**; or an element of E. $\mathit{activator}(e) = !$ is ruled out by (initial acquaintances) and the second assumption. If $\mathit{activator}(e) = \mathbf{?}$, then by EPLaw, $a \in \mathit{recep}((\rho, \chi)g, g(e))$. $a \in \rho$ implies $\mathit{Cr}(a) = !$ and thus there is some previous output event, $\mathtt{out}(e')$ that exports a: $g(\mathtt{out}(e')) < g(e)$ and $a \in \mathit{acq}(e')$. By induction $g(\mathit{Cr}(a)) < g(\mathtt{out}(e'))$. Finally, if $\mathit{activator}(e) = e' \in E$, then $g(e') < g(e)$ and by (activator acquaintances) either $\mathit{Cr}(a) = e'$ and we are done, or $a \in \mathit{acq}(e') \cup \mathit{acqB}(e')$ and by induction $g(\mathit{Cr}(a)) < g(e')$. For case (2) let $a' = \mathit{target}(e)$. There are two cases according to whether or not e is least in the arrival ordering of a'. Suppose e is least. $\mathit{Cr}(a') = !$ is ruled out since by (initial acquaintances) it implies $\mathit{Cr}(a) = !$. So let $e' = \mathit{Cr}(a') \in E$ and $e'' = \mathit{activator}(e) \in \mathrm{Dom}(g)$. Then by (activator acquaintances) $e' = e''$ and hence $g(e') < g(e)$, or $a' \in \mathit{acq}(e'') \cup \mathit{acqB}(e'')$ and by induction, $g(e') < g(e)$. By (initial acquaintances) $a \in \mathit{participants}(e')$ and by induction $g(\mathit{Cr}(a)) < g(e)$. Finally, if e is not the least in the arrival ordering of a', let e' be the immediate predecessor. By (arrival precursor) $a \in \mathit{participants}(e')$ and by induction case (2) is complete.
□

Lemma (global time): Let oed be an event diagram with interface (ρ, χ), internal actors A, and events E. Let g be a global time for oed. Then

1. $A \cap \mathit{extern}^\omega((\rho, \chi)g) = \emptyset$
2. $\mathit{recep}^\omega((\rho, \chi)g) \subseteq A$
3. $(\forall e \in E)(\mathit{acq}(e) \subseteq A \cup \mathit{extern}((\rho, \chi)g, g(e))$

$$\wedge$$

$$\mathit{target}(e) \in A \Rightarrow \mathit{acqB}(e) \subseteq A \cup \mathit{extern}((\rho, \chi)g, g(e)))$$

Corollary (global time): If $(oed, g) \in OED^\dagger$, $iface(oed) = (\rho, \chi)$, and A is the internal actors of oed, then $((\rho, \chi)g, A)$ is a scoped event path, satisfying $EPLaw$.

Proof : (1) follows from the second part of (ep.in) in the definition of $EPLaw$ using $\chi \cap A = \emptyset$. We prove (2) and (3) by (simultaneous) induction on the stage for (2) and $g(e)$ for (3).

For (2) $recep((\rho, \chi)g, 0) \subseteq \rho \subseteq A$ by definition. If $g^{-1}(i) = \mathtt{out}(e)$, then $recep((\rho, \chi)g, i+1) = recep((\rho, \chi)g, i) \cup (acq(e) - extern((\rho, \chi)g, i))$. By induction, $recep((\rho, \chi)g, i) \subseteq A$ and by part (3) of the induction hypothesis $acq(e) \subseteq A \cup extern((\rho, \chi)g, i)$ and we are done. Otherwise $recep((\rho, \chi)g, i) = recep((\rho, \chi)g, i+1)$ and we are done by induction.

For (3) first assume $e \in E$ and show $a \in A \cup extern((\rho, \chi)g, g(e))$. If $g(e) = 0$ then $activator(e) = \,!$ and we are done by (**in the beginning**). Assume $g(e) = i+1$ and $a \in acq(e)$. There are three cases according to the activator of e. If $activator(e) = \,!$ we are done as above. If $activator(e) = \mathbf{?}$, then $g(\mathtt{in}(e)) \leq i$ and we are done by $EPLaw$ and the induction hypothesis part (2). If $activator(e) = e' \in E$, then $acq(e) \subseteq acq(e') \cup acqB(e') \cup Cr^{-1}(e')$ and by induction and monotonicity we are done (recalling that $Cr^{-1}(e') \subseteq A$)).

Finally assume $e \in E$ and $target(e) \in A$ and show $acqB(e) \subseteq recep((\rho, \chi)g, g(e)) \cup extern((\rho, \chi)g, g(e))$. Again if $g(e) = 0$ we are done by (**in the beginning**). Assume $g(e) = i+1$ and consider cases on the rank of e. If e is the first, let e' be the creating event of the target. By lemma (**creation**) $g(e') < g(e)$ and by induction and the locality laws we are done. If e has an immediate predecessor, e', then again by induction and the locality laws we are done. □

5.2. A component algebra on OEDs

The OED component algebra has as components sets of event diagrams with a common interface: $Oed \subseteq OED(\rho, \chi)$. The interface of a component is the common interface of the element OEDs. In practice we will be interested in sets of event diagrams closed under renamings that fix the interface and closed under packet preserving bijections on events. However for the present we can consider arbitrary sets. The empty component is $\{\mathtt{mtoed}\} \subseteq OED(\emptyset, \emptyset)$. To define the operations of the algebra, we first define composition, restriction, and renaming as partial operations on individual event diagrams. These are then lifted pointwise to (total) operations on OED components.

5.3. 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 complementary events coalesce into a single internal event. Finally the activator function is modified to replace unknown activators of complemented inputs by the activators of the complementing outputs.

Composition of sets of event diagrams is just the compositions of all composable pairs.

Parallel Composition on $\boldsymbol{OED}^\dagger$

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

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

Let $ep_j = (\rho_j, \chi_j)g_j$ for $j < 2$. Then *composability* is defined by

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

iff the following two conditions hold:

(1) $(ep_0, A_0) \bowtie (ep_1, A_1)$

(2) $P_j \cap (E_{1-j} \cup P_{1-j}) = \emptyset$ for $j < 2$, and if $P_0 \cup P_1 \neq \emptyset$, then $E_0 \cup E_1$ is finite.

Note that (1) implies for $j < 2$ that $\chi_j \cap A_{1-j} \subset \rho_{1-j}$ and if $ep_j(\texttt{out}(e)) = i$, then $acq(e) \cap A_{1-j} \subseteq \rho^i_{1-j}$; and $E_j/A_{1-j} \subset (E_{1-j} \cup P_{1-j})$ 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

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

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

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

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

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

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

where $\quad e \in E_j \cup P_j$

Note that in the first activator case there is no corresponding complementary event for $\texttt{in}(e)$ while in the second case there is.

Before proceeding further, we show that composition of composable event diagram, global time pairs gives event diagram, global time pairs. Also completeness of event diagrams is preserved.

Lemma (oedg.pcmps.def): 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 $oed \in cOED$.

Proof : Locality laws for oed follow directly from the locality laws for oed_o, oed_1, the composability disjointness requirements and the definition of composition. Finite delay follows from composability. For strong realizability we need only show that the g ordering is compatible with the arrival and activation ordering of oed, since by composability $((\rho, \chi)g, A)$ satisfies the second condition for a global time. Compatibility with the arrival ordering follows from compatibility of g_j with the arrival ordering for oed_j for $j < 2$. What remains is to show that $g(activator(e)) < g(e)$ for $e \in E \cup P$. Suppose without loss that $e \in E_0 \cup P_0$. If $activator_0(e) \neq$? or $g_0(\mathtt{in}(e)) \notin \mathrm{Dom}(g_1)$, then $activator(e) = activator_0(e)$ and we are done by compatibility for g_0. Otherwise $g_0(\mathtt{in}(e)) = g_1(\mathtt{out}(e))$ and $activator(e) = activator_1(e)$. Thus $g(activator(e)) = g_1(activator_1(e)) < g_1(\mathtt{out}(e)) = g_0(\mathtt{in}(e)) < g_0(e) = g(e)$. □

The next lemma shows that the event diagram part of the composition of composable event diagram – global time pairs does not depend on the choice of global times.

Lemma (oedg.gtime): If $(oed_0, g_0) \bowtie (oed_1, g_1)$, $(oed_0, g_0') \bowtie (oed_1, g_1')$, $(oed, g) = (oed_0, g_0) \parallel (oed_1, g_1)$, and $(oed', g') = (oed_0, g_0') \parallel (oed_1, g_1')$, then $oed = oed'$.

Proof : The only part of the composite event diagram that depends on the global time is the activation order. By composability, $g_j(\mathtt{in}(e)) \in \mathrm{Rng}(g_{1-j})$ iff $g_j'(\mathtt{in}(e)) \in \mathrm{Rng}(g_{1-j}')$. Thus the two activation functions must agree as well. □

Lemma (oedg.pcmps.law): If $(oed_j, g_j) \in OED^\dagger(\rho_j, \chi_j)$ for $j < 3$. then

(1) $(oed_0, g_0) \bowtie (\mathtt{mtoed}, \mathtt{mtmap}) \wedge (oed_0, g_0) \parallel (\mathtt{mtoed}, \mathtt{mtmap}) = (oed_0, g_0)$

(2) $$(oed_0, g_0) \bowtie (oed_1, g_1) \Rightarrow (oed_1, g_1) \bowtie (oed_0, g_0) \wedge (oed_0, g_0) \parallel (oed_1, g_1) = (oed_1, g_1) \parallel (oed_0, g_0)$$

(3) $$\begin{aligned}&(oed_0, g_0) \bowtie (oed_1, g_1) \wedge \\ &(oed_{01}, g_{01}) = ((oed_0, g_0) \parallel (oed_1, g_1)) \bowtie (oed_2, g_2) \wedge \\ &\mathrm{Rng}(g_0 \cup g_1) \cap \mathrm{Rng}(g_2) \subseteq \mathrm{Rng}(g_{01}) \\ &\qquad \Rightarrow \\ &(oed_1, g_1) \bowtie (oed_2, g_2) \wedge (oed_0, g_0) \bowtie ((oed_1, g_1) \parallel (oed_2, g_2)) \wedge \\ &((oed_0, g_0) \parallel (oed_1, g_1)) \parallel (oed_2, g_2) = (oed_0, g_0) \parallel ((oed_1, g_1) \parallel (oed_2, g_2))\end{aligned}$$

Proof : (1) and (2) follow from (**ep.id.commute**). (3) follows from (**ep.assoc**) (it is easy to check that *Ahyp* holds for the relevant scoped event paths) and the fact that operations composing the event diagram parts are associative. The only possible complication is the activation function, but this works because of disjointness of the pairs of complementary events. □

Parallel Composition on OED components

For OED components $Oed_j \subseteq OED(\rho_j, \chi_j)$ for $j < 2$ such that $\rho_0 \cap \rho_1 = \emptyset$ parallel composition is defined as the set of event diagrams occurring in the pointwise composition of composable pairs from ${Oed_0}^\dagger \times {Oed_1}^\dagger$.

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

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 $Oed_0 \parallel Oed_1 \subseteq cOED$.

Lemma (Oeds.pcmps.law): If $Oed_j \subseteq OED(\rho_j, \chi_j)$ with $\rho_j \cap \rho_k = \emptyset$, for $j, k < 3$ and $j \neq k$, then

(id) $\quad Oed_0 \parallel \{\texttt{mtoed}\} = Oed_0$

(commute) $\quad (Oed_0 \parallel Oed_1) = (Oed_1 \parallel Oed_0)$

(assoc) $\quad (Oed_0 \parallel Oed_1) \parallel Oed_2 = Oed_0 \parallel (Oed_1 \parallel Oed_2)$

Proof : (id) and (commute) follow from (**oedg.pcmps.law.1-2**) For (assoc) we show $(Oed_0 \parallel Oed_1) \parallel Oed_2 \subseteq Oed_0 \parallel (Oed_1 \parallel Oed_2)$. The other direction then follows by commutativity. Assume $oed \in (Oed_0 \parallel Oed_1) \parallel Oed_2$, then we can find $(oed_j, g_j) \in {Oed_j}^\dagger$ for $j < 3$ such that the hypotheses of (**oedg.pcmps.law.3**) hold and such that $(oed, g) = ((oed_0, g_0) \parallel (oed_1, g_1)) \parallel (oed_2, g_2)$. Thus by (**oedg.pcmps.law.3**) $(oed_1, g_1) \bowtie (oed_2, g_2)$, $(oed_0, g_0) \bowtie ((oed_1, g_1) \parallel (oed_2, g_2))$, and $(oed, g) = (oed_0, g_0) \parallel ((oed_1, g_1) \parallel (oed_2, g_2))$, and hence $oed \in Oed_0 \parallel (Oed_1 \parallel Oed_2)$. □

5.4. Restriction

Restriction on OEDs

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

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

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 OEDs to structures of the same form, but the result may not be an OED since it may 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 the composition of the switch and ticker before the restriction hiding the ticker, t. Figure 8 shows the event diagram obtained from the dispatcher diagram of figure 7 by omitting the hiding of t and by adding events $e_5 : t \triangleleft c$ and $e_6 : c \triangleleft 2$. Here e_5 follows e_2 in the arrival order at t and e_5 is the activator of e_6. This is a valid event diagram. However restriction of this diagram to hide t is not, because t is never exported in a message and hence never becomes a receptionist. Thus there is no global time for this structure, in particular there is no ordering of events that satisfies the event path law, *EPLaw*.

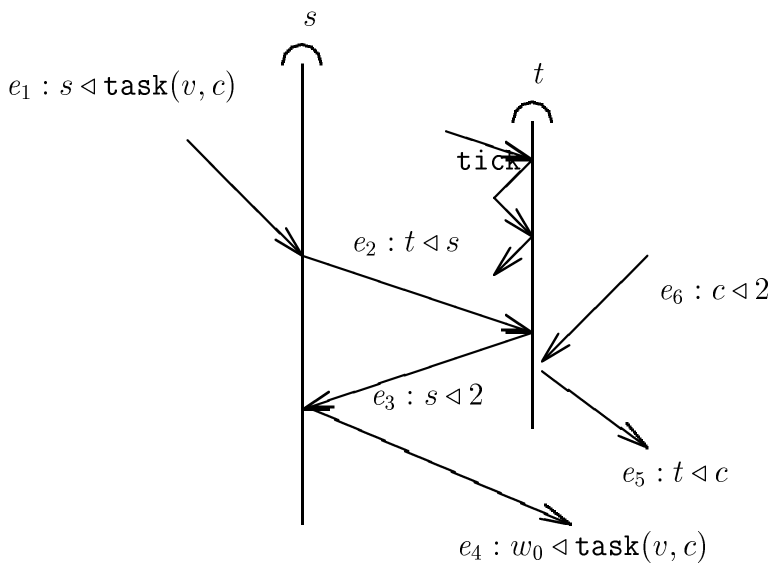

Figure 8. Extended Unrestricted Dispatcher Event Diagram

Restriction on OED components

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 $Oed\lceil\rho' \subseteq cOED$.

Note that although restriction is partial as an operation on OEDs, it is total on OED components.

5.5. *Renaming*

Renaming on OEDs

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

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

where for example we have $\widehat{\alpha}(activator)(\widehat{\alpha}(e)) = \widehat{\alpha}(activator(e))$.

Renaming on OED components

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 $\widehat{\alpha}(Oed) \subseteq cOED$.

Now we can verify that we have indeed defined a an actor component algebra.

Theorem (alg.oed): The OED component algebra with operations as defined above satisfies the component algebra laws.

Proof: The properties of parallel composition are just (**Oed.pcmps.law**). That renaming commutes with parallel composition and restriction is easy to see from the definitions. □

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.

6. Interaction Diagrams

In this section the set, ID, of interaction diagrams is defined along with a component algebra structure on sets of interaction diagrams with a common interface. A mapping from $cOED$ to ID is defined and shown to be a component algebra morphism.

6.1. *Structure*

Interaction diagrams (IDs) are abstractions of complete OEDs that hide the internal events and restrict the combined order to the input/output events. Recall from the discussion of the combined order in § 5 that the combined order so restricted is a subset of $\texttt{in}(\mathbf{E}) \times \texttt{out}(\mathbf{E})$. Interaction diagrams are structures of the form

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

such that

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

$I \subseteq \texttt{in}(\mathbf{E})$ —input events

$O \subseteq \texttt{out}(\mathbf{E})$ —output events

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

$A \subseteq \mathbf{A}$ —the reserved actor names

and $(\exists g : I \cup O \xrightarrow{\text{inj}} \mathbf{N})$ such that g preserves $\prec$, $\mathit{EPLaw}((\rho, \chi)g, A)$, and $((\rho, \chi)g, A)$ is a scoped event path.

□

The empty interaction diagram is the structure $\texttt{mtid} = (\emptyset, \emptyset)(\emptyset, \emptyset, \texttt{mtrel}, \emptyset)$.

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

$$\begin{aligned} GT(id) = \{ g : I \cup O \xrightarrow{\text{inj}} \mathbf{N} \mid \\ g \text{ preserves } \prec \wedge \mathit{EPLaw}((\rho, \chi)g, A) \wedge ((\rho, \chi)g, A) \text{ is a scoped event path} \} \end{aligned}$$

And for $Id \subset ID$ define

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

6.2. A Component Algebra on IDs

Similarly to the OED component algebra, the ID component algebra has as components sets of interaction diagrams with a common interface: $Id \subseteq ID(\rho, \chi)$ and the empty component is $\{\texttt{mtid}\}$. Also, to define the operations of the algebra, we first define composition, restriction, and renaming as partial operations on individual interaction diagrams and lift pointwise to (total) operations on ID components.

Parallel Composition

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

$$(id_0, g_0) \bowtie (id_1, g_1) \quad \text{iff} \quad ((\rho_0, \chi_0)g_0, A_0) \bowtie ((\rho_1, \chi_1)g_1, A_1).$$

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

$$
\begin{aligned}
&(id_0, g_0) \parallel (id_1, g_1) = ((\rho, \chi)(I, O, \prec, A), g) \\
&\quad \text{where} \\
&(\rho, \chi)g = (\rho_0, \chi_0)g_0 \parallel (\rho_1, \chi_1)g_1 \\
&I = \mathrm{Dom}(g) \cap (I_0 \cup I_1) \\
&O = \mathrm{Dom}(g) \cap (O_0 \cup O_1) \\
&\prec \;\; = (\prec_0 \cup \prec_1)\lceil(I \times O) \cup \\
&\qquad \{(\mathtt{in}(e), \mathtt{out}(e')) \bigm| (\exists e'', j < 2)(\mathtt{in}(e) \prec_j \mathtt{out}(e''), \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{in}(e'') \prec_{1-j} \mathtt{out}(e'), \quad \text{and} \\
&\qquad\qquad\qquad\qquad\qquad\qquad g_j(\mathtt{out}(e'')) = g_{1-j}(\mathtt{in}(e''))\} \\
&A = A_0 \cup A_1
\end{aligned}
$$

The second component of $\prec$ corresponds to tracing links through the complementary pairs of events mapped to the same time. 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

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

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

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

.

Lemma (idg.pcmps.law): If $(id_j, g_j) \in ID^\dagger(\rho_j, \chi_j)$ for $j < 3$. then

(1) $(id_0, g_0) \bowtie (\mathtt{mtid}, \mathtt{mtmap}) \wedge$

$(id_0, g_0) \parallel (\mathtt{mtid}, \mathtt{mtmap}) = (id_0, g_0)$

(2) $(id_0, g_0) \bowtie (id_1, g_1) \Rightarrow (id_1, g_1) \bowtie (id_0, g_0) \wedge$

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

(3) $(id_0, g_0) \bowtie (id_1, g_1) \wedge$
$((id_0, g_0) \parallel (id_1, g_1)) \bowtie (id_2, g_2) \wedge$
$\mathrm{Rng}(g_0 \cup g_1) \cap \mathrm{Rng}(g_2) \subseteq \mathrm{Rng}(g_{01})$
$\Rightarrow$
$(id_1, g_1) \bowtie (id_2, g_2) \ \wedge\ (id_0, g_0) \bowtie ((id_1, g_1) \parallel (id_2, g_2)) \wedge$
$((id_0, g_0) \parallel (id_1, g_1)) \parallel (id_2, g_2) = (id_0, g_0) \parallel ((id_1, g_1) \parallel (id_2, g_2))$

Proof : The same argument as for (**oedg.pcmps.law**) □

Restriction

If $id = (\rho, \chi)(I, O, \prec, A)$, then $id\lceil\rho' = (\rho\cap\rho', \chi)(I, O, \prec, A)$. In general the restriction of an interaction diagram may not be an interaction diagram. The interaction diagram obtained from the event diagram of figure 8 is such an example. Again the problem is that t never becomes a receptionist and hence there is no ordering of the input/output events that satisfies *EPLaw*.

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.

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

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

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

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

Proof : The parallel composition laws follow easily from (**idg.pcmps.law**). That renaming commutes with parallel composition and restriction is easy to see from the definitions. □

6.3. 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

$$oed2id(oed) = (\rho, \chi)(In(oed), Out(oed), \prec, A))$$

where (ρ, χ) is the interface of *oed* and $\prec$ the restriction of the combined order $\prec_{\mathrm{co}}$ of *oed* to $In(oed) \times Out(oed)$ (the input and output events of *oed* respectively) and A is the set of internal actors of *oed*. (Recall that these functions and relations were defined in §5.1.)

As an example, observe that the ticker interaction diagram, id^{T}, (of §3) 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 by defining, for $Oed \subset cOED(\rho, \chi)$,

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

The following lemma shows that *oed2id* maps cOED components to ID components.

Lemma (oed2id):

$$oed2id : cOED(\rho, \chi) \to ID(\rho, \chi)$$

The restriction of any global time, g, of an OED, *oed*, to input/output events, $In(oed) \cup Out(oed)$, is a global time of the interaction diagram $oed2id(oed)$. Furthermore every global time of $oed2id(id)$ is the restriction of some global time of *oed*. The latter is because 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. This is made precise in the following lemma.

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 $cOED^{\dagger}$, 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 $(\rho, \chi) = (\rho_0, \chi_0) \parallel (\rho_1, \chi_1)$, $A = A_0 \cup A_1)$, and

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

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

$$S = \bigcup_{\{e \,\big|\, (\exists j<2)(g_j(\mathtt{in}(e)) = g_{1-j}(\mathtt{out}(e)))\}} \{\mathtt{in}(e), \mathtt{out}(e)\} \qquad \text{the complementary events}$$

then from the definitions 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

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

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 $j < 2$

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

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

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

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

(0) $oed2id(\{\mathtt{mtoed}\}) = \{\mathtt{mtid}\}$,

(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): (2) relies on (**oed2id.gt**). 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 $id \in Id_l$.

$\square$

7. Interaction paths

7.1. *Structure*

Interaction paths are scoped event paths (see §4) in which the domain of the global time map contains no internal events. Thus interaction paths are structures of the form

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

where $(\rho, \chi) \in \mathbf{Iface}$, $\mathrm{Dom}(g) \subseteq (\mathtt{in}(\mathbf{E}) \cup \mathtt{out}(\mathbf{E}))$ and $EPLaw((\rho, \chi)g, A)$ holds and $((\rho, \chi)g, A)$ is a scoped event path. 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 $ip = ((\rho, \chi)g, A)$, $ip' = ((\rho, \chi)g', A)$, where $\mathrm{Dom}(g) = \mathrm{Dom}(g')$, and the induced orderings, $<_g$ and $<_{g'}$, are the same.[2]

7.2. *A component algebra on IP sets*

The IP component algebra has as components sets of interaction paths with a common interface: $Ip \subseteq IP(\rho, \chi)$ and the empty component is $\{(\mathtt{mtep}, \emptyset)\}$. We formalize the convention that two interaction paths with the same interface whose global time maps induce the same total ordering are considered the same, buy requiring that IP components be closed under change of order representation. Thus if Ip is an IP component, $ip = ((\rho, \chi)g, A) \in Ip$ and g' is such that $\mathrm{Dom}(g) = \mathrm{Dom}(g')$ and $<_g$ and $<_{g'}$ are the same total ordering, then $((\rho, \chi)g', A) \in Ip$

As for OEDs and IDs, to define the operations of the algebra, we first define composition, restriction, and renaming as partial operations on individual interaction paths and lift pointwise to (total) operations on IP components.

Parallel composition

Composability and composition on IP is just that defined for scoped 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

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

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

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

.

Restriction

If $ip = ((\rho, \chi)g, A)$, then $ip\lceil\rho' = ((\rho \cap \rho', \chi)g, A)$. As before the restriction of an interaction path may not be an interaction path, because of inputs to unexported receptionists. For example, let $ip = ((\{s,t\}, \{w_0, w_1\})g, \{s,t\})$ where g is a global time for the interaction diagram obtained from figure 8. Then ip is an interaction path but $ip\lceil s$ is not an interaction path, due to the input to a non-receptionist t.

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.

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

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

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

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

Proof : The parallel composition laws follow easily from (**ep.id.commute**) and (**ep.assoc**) using the implicit closure under re-representation of the global time ordering. That renaming commutes with parallel composition and restriction is easy to see from the definitions. □

7.3. 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

$$id2ip(id, g) = ((\rho, \chi)g, A)$$

where $(\rho, \chi) = iface(id)$ and A is the reserved names of 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):

$$id2ip : ID^{\dagger} \to IP(\rho, \chi)$$

That is $id2ip$ maps ID components to IP components preserving interface.

Returning to the examples of §3 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 it is easy to see 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, letting $(id, g) = (id_0, g_0) \parallel (id_1, g_1)$, 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).

Proof : (1) and (2) follow easily from the definitions. For the rest, we construct g_j' as follows. $g_j'(\tilde{e}) = g'(\tilde{e})$ if $g_j(\tilde{e})$ is not a synchronization point. Suppose $g_j(\mathtt{out}(event)) = g_{1-j}(\mathtt{in}(e))$. Then we claim that $\mathtt{out}(e)$ can be inserted into the g_j ordering just after the id_j predecessors of $\mathtt{out}(e)$ and $\mathtt{in}(e)$ can be inserted into the g_{1-j} ordering just before the id_{1-j} predecessors of $\mathtt{in}(e)$, since the former must all precede the latter in g and in g'. □

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.

8. Actor Theories and Their Models

Actor theories are a generalization of the notion of actor structures presented in [15, 14]. They are an abstract framework for executable specification of the behavior of actor system components inspired by rewriting logic [10, 11]. 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 [12]. HBC-actor theories reflect the semantics of the actor systems studied by Baker and Hewitt [6] and by Clinger[7]. They have the property that an actor is always enabled to receive a message, and that the processing of a message can be treated as an atomic step producing a finite number of new actors and messages.[3]

8.1. The Basics

We call the mathematical structure

$$\langle \mathbf{A}, \mathbf{M}, \mathbf{E}, acq, pkt, \widehat{\_}\rangle$$

given in the first part of § 4 an *event system.* Recall that $\mathbf{A}$ is a set of actor names, $\mathbf{M}$ a set of messages, $\mathbf{E}$ is a set of events, acq is the acquaintances function, pkt maps an event to its message packet (target and contents), and $\widehat{\alpha}$ applies the renaming α to messages and events.

An HBC actor theory (briefly an actor theory) over an event system as above is given by specifying an additional set, $\mathbf{S}$, of actor states, extending acq and $\widehat{\_}$ to act on $\mathbf{S}$, and introducing two new functions:

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

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

such that the renaming and acquaintance axioms (**HBC-ax**) below are satisfied.

In the following we fix the event system to be the one introduced in § 4. Actor states, $\mathbf{S}$, 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 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, using elements of $\vec{a}$ to name newly created actors. $\#\mathsf{a}(a, s, M)$ is the number of actors that will be created. $deliver(a, s, M)(\vec{a})$ is defined just if $\mathrm{Len}(\vec{a}) = \#\mathsf{a}(a, s, M)$. 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 $k = \#\mathsf{a}(a, s, M)$, 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.

As stated above, an HBC actor theory must satisfy (**HBCax**).

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

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

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

(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. Note that (i–iii) for $\mathbf{M}$ and $\mathbf{E}$ hold by requirements for an event system.

An HBC 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

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

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

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

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

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

As we will see below, this specifies the same computations as the rewrite rules given for the ticker and ticker factory in § 3.

8.2. Configurations and Computations

Intuitively, configurations are collections of actors and messages together with an interface. The collection of actors and messages is called a configuration interior. To facilitate establishing the correspondence between computations and event diagrams we use events to represent messages occurring in a configuration. We also augment configurations to carry along the set of past events.

We begin by defining **AE**—actor entities (briefly actors); and **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 packet 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 $\mathbf{I}$. $\mathit{InAct}(I)$ is the set of names of actors that occur in I, and $\mathit{ExtAct}(I)$ is the set of names of external actors referred to in I. $\mathit{Undel}(I)$ is the set of undelivered messages of I and $\mathit{aState}(I, a)$ is the state of a in I if $a \in \mathit{InAct}(I)$.

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

$$\mathit{ExtAct}(I) = \mathit{acq}(I) - \mathit{InAct}(I)$$

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

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

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. We also require that the sets of undelivered events and past events are disjoint. Thus the set of configurations is defined by

$$\begin{aligned}\mathbf{K} = \{(\rho, \chi)(I \mid E) \mid\ & \rho \subseteq \mathit{InAct}(I) \ \wedge\ \mathit{ExtAct}(I) \subseteq \chi \wedge \\ & (\mathit{acq}(I) \cup \mathit{acq}(E)) \subseteq \mathit{InAct}(I) \cup \chi \wedge \\ & \mathit{Undel}(I) \cap E = \emptyset\}\end{aligned}$$

We let K range over $\mathbf{K}$. The set E of previously delivered messages is included in order to facilitate the mapping between event diagrams and computations. It allows us to ensure locally 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.

Definition (Composability of configurations): Let $K_j = (\rho_j, \chi_j)(I_j \mid E_j)$. Then

$$K_0 \bowtie K_1 \;\Leftrightarrow$$
$$\mathit{InAct}(I_0) \cap \mathit{InAct}(I_1) = \emptyset \;\wedge\; \bigwedge_{j<2}(\chi_j \cap \mathit{InAct}(I_{1-j})) \subseteq \rho_{1-j}$$
$$\bigwedge_{j<2}(\mathit{Undel}(I_{1-j}) \cap (E_j \cup \mathit{Undel}(I_j)) = \emptyset \;\wedge\; E_{1-j}/\rho_j \subseteq (E_j \cup \mathit{Undel}(I_j)))$$

The composition of composable configurations is simple to describe. Its interface is the composition of the interfaces of the composees, its interior is the union of the interiors of the composees, and its set of past events is the union of the past events of the composees. Restriction is defined as usual by restricting the interface. Note that in the case of configurations, the result is always a configuration. Renaming is applied to each of the parts of a configuration.

(cmps) $K_0 \parallel K_1 = (\rho_0, \chi_0)(I_0 \mid E_0) \parallel (\rho_1, \chi_1)(I_1 \mid E_1) = (\rho, \chi)(I_0 \cup I_1 \mid E_0 \cup E_1)$

where $(\rho, \chi) = (\rho_0, \chi_0) \parallel (\rho_1, \chi_1)$

if $(\rho_0, \chi_0)(I_0 \mid E_0) \bowtie (\rho_1, \chi_1)(I_1 \mid E_1)$

(restr) $(\rho, \chi)(I \mid E)\lceil\rho' = (\rho \cap \rho', \chi)(I \mid E)$

(ren) $\widehat{\alpha}((\rho, \chi)(I \mid E)) = (\widehat{\alpha}(\rho), \widehat{\alpha}(\chi))(\widehat{\alpha}(I) \mid \widehat{\alpha}(E))$

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. $\vec{a}$ is a list of names to be used for newly created actors, and $\vec{e}$ is a list of events to be used for the newly sent messages. $\vec{a}$ and $\vec{e}$ are included in the label to make the transition relation deterministic. 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. Here we use the notational convention that $\_,\_$ is the (associative) operation of adding an element to a set. Thus for a set X we have $X, x = X \cup \{x\}$, and x, y used where a set is expected is $\emptyset, x, y$.

(d) $(\rho, \chi)(I\,,\,\mathtt{[}s\mathtt{]}_a\,,\,e \mid E) \xrightarrow{\mathtt{d}(e,\vec{a},\vec{e})} (\rho, \chi)(I, I' \mid E\,,\,e)$

if

$$\vec{a} \cap (\mathit{InAct}(I\,,\,\mathtt{[}s\mathtt{]}_a\,,\,e) \cup \chi) = \emptyset = \vec{e} \cap (\mathit{Undel}(I\,,\,\mathtt{[}s\mathtt{]}_a\,,\,e) \cup E)$$
$$I' = \mathtt{[}s'\mathtt{]}_a\,,\,\{\mathtt{[}s_i\mathtt{]}_{a_i} \bigm| 1 \le i \le \mathrm{Len}(\vec{a})\}\,,\,\vec{e}$$
$$\mathit{deliver}(a, s, \mathit{message}(e))(\vec{a}) =$$
$$(s', [s_i \bigm| 1 \le i \le \mathrm{Len}(\vec{a})], [\mathit{pkt}(e_i) \bigm| 1 \le i \le \mathrm{Len}(\vec{e})])$$

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

$\qquad \text{if } e \notin E \cup \mathit{Undel}(I) \ \wedge\ acq(message(e)) \cap \mathit{InAct}(I) \subseteq \rho \ \wedge\ target(e) \in \rho$

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

$\qquad \text{where} \quad \rho' = \rho \cup (acq(message(e)) \cap \mathit{InAct}(I)) \quad \text{if } target(e) \in \chi$

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

Notice that the transition 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 §3 are simply delivery transitions for the case in which I is empty. Such rules provide an alternative form of presenting an actor theory (see [14]).

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. $\mathit{Tree}(K)$ is the set of finite computations with initial configuration K. A fair computation is one in which every message eventually gets delivered. That is, if e is an undelivered message of K_i, then there is some $j \geq i$, and some $\vec{a}, \vec{e}$ such that $l_j = \mathtt{d}(e, \vec{a}, \vec{e})$. 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. $\mathit{Fair}(K)$ is the set of fair computations with initial configuration K. $\mathit{Path}(K)$ is the set of finite or fair computations for K: $\mathit{Path}(K) = \mathit{Tree}(K) \cup \mathit{Fair}(K)$.

The sequence of transitions, π^{T}, given in §3 figure 1 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.

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

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

g^π is well defined since each event appears at most once as the event of a label in a computation. We extend *inActs* to computations by defining

$$inActs(\pi) = \bigcup_{i \in \mathbf{N}} (inActs(I_i))$$

where I_i is the interior of K_i.

Lemma (path.ep): Let $\pi \in Path(K)$, $(\rho, \chi) = iface(K)$, and $A = inActs(\pi)$. Then $EPLaw((\rho,\chi)g^\pi, A)$ and $((\rho,\chi)g^\pi, A)$ is a scoped event path.

Proof: By the constraints on the transition rules, noting that $recep((\rho,\chi)g^\pi, i) = recep(K_i)$ and $extern((\rho,\chi)g^\pi, i) = extern(K_i)$. □

8.3. Component algebra on computations

The definitions of composability and parallel composition for computations is an adaption of the definitions given in [14] to account for the addition of the delivered messages 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 computations are composable just if corresponding configurations are composable and the associated scoped event paths are composable.

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

For composable paths, parallel composition is defined by

$$\begin{aligned}
&\pi_0 \parallel \pi_1 = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}] \\
&\quad \text{where} \\
&K_0 = K_0^0 \parallel K_0^1 \quad \text{and} \quad l_i = \begin{cases} l_i^j & \text{if } l_i^{1-j} = \mathtt{idle} \\ \mathtt{idle} & \text{if } l_i^0 \text{ and } l_i^1 \text{ are complements} \end{cases}
\end{aligned}$$

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. Restriction and renaming are defined in the obvious manner and the details are omitted. Restriction, when defined, or renaming of a computation yield a computation of the restriction or renaming of the initial configuration and again fairness is preserved. Analogs of (**ep.id.commute**) and (**ep.assoc**) hold for computations since the additional information in a computation does not affect the arguments.

Sets of computation paths with a common initial configuration form an actor component algebra. The interface of a component is the interface of the common initial configuration. Operations on computations are lifted to sets of computations in the usual manner. For example, if $K_0 \bowtie K_1$ and $P_j \subseteq \mathit{Path}(K_j)$ for $j < 2$ then parallel composition is defined by

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

The proof that these operations obey the component algebra laws is similar to the proofs for other components and again we omit the details. It is easy to check that composition of computation paths of composable configurations gives a computation path of the composition of the configurations, and fairness is preserved.

Lemma (path.cmps): If $K_0 \bowtie K_1$ and $P_j \subseteq \mathit{Path}(K_j)$ for $j < 2$ then

$$P_0 \parallel P_1 \in \mathit{Path}(K_0 \parallel K_1)$$

and if $P_j \subseteq \mathit{Fair}(K_j)$ for $j < 2$, then

$$P_0 \parallel P_1 \in \mathit{Fair}(K_0 \parallel K_1).$$

Theorem (path.morph): ([14],§ 6) The maps *Path* and *Fair* from configurations to computation path sets are a component algebra morphisms.

Note that the configuration based component algebra is not quite a component algebra since we have taken components to be single configurations and thus composability at the component level is not just disjointness of receptionists. This can be fixed in a number of ways. We could take components to be equivalence classes of configurations under renamings that fix the interface and packet preserving bijections. We could also consider arbitrary sets of configurations with a common interface. In either case the algebraic laws and morphism results hold with similar proofs.

8.4. Mapping Computations to OEDs

Now we define a mapping *c2oed* from $\mathit{Path}(K)$ to $\mathit{OED}(\mathit{iface}(K))^\dagger$, which preserves finiteness and fairness. We can think of the event diagram $\mathit{c2oed}(\pi)$ as being obtained by taking the limit of event diagrams corresponding to finite initial segments of π, however we will not make this process explicit. The global time is just g^π. In §9 we will define the set $\mathit{OED}(K)$ of OEDs representing computations of K and show that the image of $\mathit{Path}(K)$ is $\mathit{OED}(K)$. Let

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

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

$$\mathit{c2oed}(\pi) = (\mathit{oed}, g^\pi) \quad \text{where} \quad \mathit{oed} = (\rho, \chi)(A, E, \mathit{activator}, \xrightarrow{\text{arr}}, \mathit{acqB}, \mathit{Cr}, P)$$

where

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

$$A = \bigcup_{i \in \mathbf{N}} \mathit{InAct}(I_i)$$

$$E = \bigcup_{i \in \mathbf{N}} E_i$$

$$\mathit{activator}(e) = \begin{cases} ! & \text{if } e \in \mathit{Undel}(I_0) \\ \textbf{?} & \text{if } (\exists i)(l_i = \mathtt{in}(e)) \\ e' & \text{if } (\exists i, \vec{a}, \vec{e})(l_i = \mathtt{d}(e', \vec{a}, \vec{e}) \ \wedge \ e \in \vec{e}) \end{cases}$$

$$\xrightarrow{\text{arr}} = \bigcup_{a \in A} \xrightarrow{\text{arr}}_a$$

$$\text{where} \quad e \xrightarrow{\text{arr}}_a e' \ \Leftrightarrow \ (\{e, e'\} \subseteq E/a \, \wedge \\ (\exists k, k')(k < k' \ \wedge \ l_k = \mathtt{d}(e, \ldots) \ \wedge \ l_{k'} = \mathtt{d}(e', \ldots)))$$

$$\mathit{acqB}(e) = \mathit{acq}(\mathit{aState}(I_k, \mathit{target}(e))) \quad \text{if } l_k = \mathtt{d}(e, \vec{a}, \vec{e})$$

$$\mathit{Cr}(a) = e \quad \text{if } (\exists k, \vec{a}, \vec{e})(l_k = \mathtt{d}(e, \vec{a}, \vec{e}) \ \wedge \ a \in \vec{a})$$

$$P = \{e \mid (\exists i)(\forall k)(e \in \mathit{Undel}(I_{i+k}))\}$$

Note that *oed* is well defined because each event occurs at most once as the main event of a label, and at most once as an element of $\vec{e}$ in a delivery label.

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

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

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

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

Proof : (0) is easy to see. Thus $\mathrm{Dom}(g) = \widetilde{E}(\mathit{oed})$. For the rest we note that by (**HBCax**) the locality laws hold for *oed*, and by (**path.ep**) $\mathit{EPLaw}((\rho, \chi)g, A)$ holds. Thus if π is finite we are done. If π is fair then $P = \emptyset$ and again we are done. $\square$

Continuing the example of §3.1.1, we see that $\mathit{oed}^{\mathrm{T}} = \mathit{c2oed}(\pi^{\mathrm{T}})$.

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 $\pi_j \in \mathit{Path}(K_j)$, then
(a) $\pi_0 \bowtie \pi_1$ iff $\mathit{c2oed}(\pi_0) \bowtie \mathit{c2oed}(\pi_1)$
(b) if $\pi_0 \bowtie \pi_1$, then $\mathit{c2oed}(\pi_0) \parallel \mathit{c2oed}(\pi_1) = \mathit{c2oed}(\pi_0 \parallel \pi_1)$

Proof : Part (a) is direct from the definitions. For part (b) assume composability and let

$$(\mathit{recep}(K_j), \mathit{extern}(K_j)) = (\rho_j, \chi_j)$$

$$(\rho, \chi) = (\rho_0, \chi_0) \parallel (\rho_1, \chi_1)$$
$$c2oed(\pi_0) \parallel c2oed(\pi_1) = (\rho, \chi)(A, E, activator, \xrightarrow{\text{arr}}, acqB, Cr, P)$$
$$c2oed(\pi_0 \parallel \pi_1) = (\rho, \chi)(A', E', activator', \xrightarrow{\text{arr}}{}', acqB', Cr', P')$$

Since internal actors, arrival order, acquaintance and creation functions, and pending events are determined by internal aspects not effected by composition, it is easy to see that $A = A'$, $\xrightarrow{\text{arr}} = \xrightarrow{\text{arr}}{}'$, $acqB = acqB'$, $Cr = Cr'$, and $P = P'$. E is the set of internal and output events of π_0 and π_1 with outputs from j to an eventual receptionist of $1 - j$ removed, and this is just the internal and output events of $\pi_0 \parallel \pi_1$, which is E'. Finally from the definition of activator computed from a path we see that *activator* and *activator'* are the same. □

The map *c2oed* induces a natural equivalence relation on computations with the same initial configuration:

$$\pi \sim_{oed} \pi' \;\Leftrightarrow\; c2oed(\pi) = (oed, g) \wedge c2oed(\pi') = (oed, g')$$

In fact equivalent paths correspond to the different possible global times of the underlying event diagrams.

Lemma (c2oed.eq): If $\pi \in Path(K)$, $c2oed(\pi) = (oed, g)$, and $(oed, g') \in OED^\dagger$, then there is $\pi' \in Path(K)$, such that $c2oed(\pi') = (oed, g')$.

Proof : Let $\pi = [K_i \xrightarrow{l_i} K_{i+1} \mid i \in \mathbf{N}]$ and define l'_i by

$$l'_i = \begin{cases} \mathtt{in}(e) & \text{if } g'^{-1}(i) = \{\mathtt{in}(e)\} \\ \mathtt{out}(e) & \text{if } g'^{-1}(i) = \{\mathtt{out}(e)\} \\ l_j & \text{if } g'^{-1}(i) = \{e\} \text{ and } g(e) = j \\ \mathtt{idle} & \text{if } g'^{-1}(i) = \{\ \} \end{cases}$$

Then π' is the computation determined by K_0 and l'_i for $i \in \mathbf{N}$. □

The map *c2oed* is extended to computation path components $P \subseteq Path(K)$ as follows.

$$c2oed(P) = \{oed \mid (\exists \pi \in P)(c2oed(\pi) = (oed, g^\pi))\}$$

To establish the morphism property of *c2oed* we must restrict attention to the sub-algebra path sets closed under $\sim_{oed}$.

Theorem (c2oed.morph): The map *c2oed* is a component algebra morphism. In fact, for starting configurations K, K_0, K_1

(1) If $K_0 \bowtie K_1$ and $P_j \subseteq Path(K_j)$, with P_j closed under $\sim_{oed}$ for $j < 2$, 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 $c2oed(\widehat{\alpha}(P)) = \widehat{\alpha}(c2oed(P))$.

Proof : We prove (1), (2) and (3) are easy consequences of the definitions. Assume $K_0 \bowtie K_1$ and $P_j \subseteq \mathit{Path}(K_j)$. For $\subseteq$, assume $\mathit{oed} \in \mathit{c2oed}(P_0 \parallel P_1)$ and show $\mathit{oed} \in \mathit{c2oed}(P_0) \parallel \mathit{c2oed}(P_1)$. Pick $\pi_j \in P_j$ for $j < 2$ such that $\pi_0 \bowtie \pi_1$ and $(\mathit{oed}, g^{\pi_{01}}) = \mathit{c2oed}(\pi_0 \parallel \pi_1)$ where $\pi_{01} = \pi_0 \parallel \pi_1$. By (**c2oed.pcmps**) $\mathit{c2oed}(\pi_0) \bowtie \mathit{c2oed}(\pi_1)$ and $\mathit{oed}, g^{\pi_{01}} = \mathit{c2oed}(\pi_0) \parallel \mathit{c2oed}(\pi_1)$ hence $\mathit{oed} \in \mathit{c2oed}(P_0) \parallel \mathit{c2oed}(P_1)$.
For $\supseteq$, assume $\mathit{oed} \in \mathit{c2oed}(P_0) \parallel \mathit{c2oed}(P_1)$ and show $\mathit{oed} \in \mathit{c2oed}(P_0 \parallel P_1)$. Pick $\mathit{oed}_j \in \mathit{c2oed}(P_j)$ for $j < 2$ and g_j such that $(\mathit{oed}_0, g_0) \bowtie (\mathit{oed}_1, g_1)$ and $(\mathit{oed}, g_{01}) = (\mathit{oed}_0, g_0) \parallel (\mathit{oed}_1, g_1)$. By closure under equivalence, we can pick $\pi_j \in P_j$ such that $\mathit{c2oed}(\pi_j) = (\mathit{oed}_j, g_j)$. Then, by (**c2oed.pcmps**) $\pi_0 \bowtie \pi_1$ and $(\mathit{oed}, g_{01}) = \mathit{c2oed}(\pi_0 \parallel \pi_1)$. □

9. 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 the OED semantics, $\mathit{OED}(K)$, of a configuration, K, and showing that $\mathit{OED}(K) = \mathit{c2oed}(\mathit{Path}(K))$. The rest follows from the various morphism results.

9.1. *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. We first define an auxiliary relation that makes explicit the various associations required. Let

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

$$\mathit{iState} : A \to \mathbf{S}$$

$$\mathit{ae} : \mathit{Dlv}(\mathit{oed}) \to \mathbf{A}^*$$

$$\mathit{me} : \mathit{Dlv}(\mathit{oed}) \to \mathbf{E}^*$$

$$\mathit{sAfter} : \mathit{Dlv}(\mathit{oed}) \to \mathbf{S}$$

We say that *oed* is an event diagram for K via the associations $(\mathit{ae}, \mathit{me}, \mathit{iState}, \mathit{sAfter})$, (written $\mathit{oed} \in \mathit{OED}(K, \mathit{ae}, \mathit{me}, \mathit{iState}, \mathit{sAfter})$) iff the following conditions hold:

(iface) $\mathit{iface}(\mathit{oed}) = \mathit{iface}(K)$

(!) $\mathit{Cr}^{-1}(!) = \mathit{InAct}(K)$

$\mathit{activator}^{-1}(!) = \mathit{Undel}(K)$

$\mathit{iState}(a) = \mathit{aState}(K, a)$ if $\mathit{Cr}(a) = \,!$

and for $e \in \mathit{Dlv}(\mathit{oed})$

(ae) $\quad$ ($ae(e)$ enumerates $Cr^{-1}(e)$)

(me) $\quad$ ($me(e)$ enumerates $\mathit{activator}^{-1}(e)$)

(d) $\quad$ $\mathit{deliver}(\mathit{target}(e), s, \mathit{message}(e))(ae(e)) =$

$$(\mathit{sAfter}(e), \mathit{iState}(ae(e)), \mathit{pkt}(me(e)))$$

$$\text{where} \quad s = \begin{cases} \mathit{iState}(\mathit{target}(a)) & \text{if } \mathit{rank}(e) = 0 \\ \mathit{sAfter}(e') & \text{if } \mathit{rank}(e) > 0 \text{ and} \\ & e' \text{ is the immediate predessor of } e \\ & \text{in the arrival ordering.} \end{cases}$$

and where we lift *iState* and *pkt* pointwise to sequences. $\mathit{iState}(a)$ is the initial state of actor a. For an internal event with target a, $\mathit{sAfter}(e)$ is the state of a just after message delivery represented by e.

(!) says that the actors and messages in K are those initially present in the event diagram, and if a is initially present, then $\mathit{iState}(a)$ is its state in K. (ae) says that $ae(e)$ is an enumeration of the actors created by e. (me) says that $me(e)$ is an enumeration of the events activated by e — events corresponding to messages sent in response to e. (d) says that for each internal event, e, the state of the target after the event, $\mathit{sAfter}(e)$, and the initial states of the newly created actors, $\mathit{iState}(a_i)$ for $a_i \in ae(e)$, and the messages sent, $\mathit{pkt}(me(e))$, are those given by the *deliver* function. Note that if $\mathit{oed} \in \mathit{OED}(K, ae, me, \mathit{iState}, \mathit{sAfter})$ then $\mathit{iState}, \mathit{sAfter}$ are uniquely determined.

For K a starting configuration, the set, $\mathit{OED}(K) \subseteq \mathit{OED}(\mathit{iface}(K))$, of event diagrams for K is defined as follows.

$$\begin{aligned} &\mathit{oed} = (\rho, \chi)(A, E, \mathit{activator}, \xrightarrow{\text{arr}}, \mathit{acqB}, Cr, P) \in \mathit{OED}(K) \Leftrightarrow \\ &\quad (\exists ae : \mathit{Dlv}(\mathit{oed}) \to \mathbf{A}^*, me : \mathit{Dlv}(\mathit{oed}) \to \mathbf{E}^*, \\ &\quad\quad \mathit{iState} : A \to \mathbf{S}, \mathit{sAfter} : \mathit{Dlv}(\mathit{oed}) \to \mathbf{S}) \\ &\quad\quad\quad \mathit{oed} \in \mathit{OED}(K, ae, me, \mathit{iState}, \mathit{sAfter}) \end{aligned}$$

For K a starting configuration define $c\mathit{OED}(K) = \mathit{OED}(K) \cap c\mathit{OED}$

Example: Ticker OEDs As an example of OED semantics we describe the set of event diagrams for the Ticker system of §3. Recall the starting ticker configuration $\mathit{Ticker}[t] = (t, \emptyset)([T(0)]_t \,,\, t \triangleleft \mathtt{tick} \mid \emptyset)$. Then

$$\begin{aligned} &\mathit{OED}(\mathit{Ticker}[t]) = \\ &\quad \{(t, \emptyset)(\{t\}, E, \mathit{activator}, \xrightarrow{\text{arr}}, \ \lambda e \in E/t.\{t\}, \ \lambda a \in \{t\}.!, P) \in \mathit{OED} \mid \Phi\} \end{aligned}$$

where Φ holds just if

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

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

(2.1) if $message(e) = \mathtt{tick}$, then there is some $e' \in E \cup P$ with $activator^{-1}(e) = \{e'\}$ and $pkt(e') = t \triangleleft \mathtt{tick}$

(2.2) if $message(e) = c$ with $c \in \mathbf{A}$, then there is some $e' \in E \cup P$ such that $activator^{-1}(e) = \{e'\}$, $pkt(e') = c \triangleleft k$
where $k = |\{e'' \in E/t \mid message(e'') = \mathtt{tick} \wedge rank(e'') < rank(e)\}|$

(2.3) otherwise $activator^{-1}(e) = \emptyset$

Lemma (Ticker OED properties): $OED(Ticker[t])$ is closed under renamings that fix the interface, and 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 $t \triangleleft \mathtt{tick}$,

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

9.2. 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) $OED(K) = \{c2oed(\pi) \mid \pi \in Path(K)\}$
(2) $cOED(K) = \{c2oed(\pi) \mid \pi \in Fair(K)\}$

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 $\mathtt{d}(e, \vec{a}, \vec{e})$ 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

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

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

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

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

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

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

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

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

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

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

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

(1) $\quad \pi \in \mathit{Path}(K)$

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

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

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.

9.3. ID and IP semantics

The semantics $\mathit{ID}(K)$ and $\mathit{IP}(K)$ are defined by

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

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

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.

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

where Ψ holds just if, $\{t\} \in A$, $\mathit{message}(e) \in \mathbf{N}$ for $\mathtt{out}(e) \in O$, and letting $AE = \{\mathtt{in}(e) \in I \mid \mathit{message}(e) \in \mathbf{A} - \{t\}\}$, there exists a total order $\xrightarrow{\mathrm{arr}}_t$ 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 $\mathit{message}(e) = \mathit{target}(B(\mathtt{in}(e))$

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

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

(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 $message(e) = n'$

The bijection B corresponds to the 'in response to' relation, and the total ordering $\xrightarrow{\text{arr}}_t$ corresponds to the restriction of the arrival ordering of the underlying event diagram to messages that generate messages to external actors. Again the set $ID(Ticker[t])$ is closed under renamings that fix the interface and under packet preserving isomorphisms on $\mathbf{E}$.

9.4. 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

Proof : for $\mathcal{S}$ one of *Path*, *Tree*, or *Fair* this is just (**c2oed.morph**). 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.

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**) $(oed, g) = c2oed(\pi_0) \parallel c2oed(\pi_1) \in {Oed_l}^\dagger$.

□

9.5. Distinctions

For each of the semantic maps $\mathcal{S}$ there is a corresponding notion of equivalence of configurations:

$$K_0 \sim_{\mathcal{S}} K_1 \;\Leftrightarrow\; \mathcal{S}(K_0) = \mathcal{S}(K_1).$$

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 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 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

$$K_0 = (a, c)([N]_a, [S(c)]_b, b \lhd b \mid \emptyset)$$
$$K_1 = (a, c)([S(b)]_a, [S(c)]_b, b \lhd b \mid \emptyset)$$

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 $\mathtt{in}(e) \prec_1 \mathtt{out}(e')$ where $\mathtt{out}(e')$ is the single output event, because the message from a to b caused by the input arrives before the initially present message.

10. 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 distributed 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 [1, 4], are simple and intuitive, but too concrete for many purposes. Operational models based on rewriting logic [14] provide a first level of abstraction, defining an equivalence relation on computations that hides the ordering of independent transitions. Event diagrams [8, 7] 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 [15] (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 a 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 promising approach is the graphical language of specification diagrams [13].

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 [3]. 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 the replacement of existing components.

Notes

1. We use the Baker-Hewitt notion of participant, which differs from the Clinger formulation in that it includes the created actors.

2. the event path portion of an interaction paths is just a path of the interaction semantics [15] presented using injective functions from input/output events to numbers rather than functions from numbers to input/output events and silent events (corresponding to numbers not in the range). The two forms of presentation are equivalent. The set of reserved names is added here to be able to define a parallel composition operation that is associative. It can be suppressed when algebraic considerations are not relevant.
3. We focus on HBC-actor theories as they have a simple relation to our event diagrams. It is straightforward to modify event diagrams to accommodate a wider class of actor theories.

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Mass., 1986.
2. G. Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–141, September 1990.
3. G. Agha. Abstracting interaction patterns: A programming paradigm for open distributed systems. In E. Najm and J-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 135–153. Chapman & Hall, 1997.
4. 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.
5. Henry G. Baker. Actor systems for realtime computation. Technical Report MIT LCS Technical Report 197, MIT, March 1978.
6. Henry G. Baker and Carl Hewitt. Laws for communicating parallel processes. In *IFIP Congress*, pages 987–992. IFIP, August 1977.
7. W. D. Clinger. *Foundations of Actor Semantics.* PhD thesis, MIT, 1981. MIT Artificial Intelligence Laboratory AI-TR-633.
8. I. Greif. Semantics of communicating parallel processes. Technical Report 154, MIT, Project MAC, 1975.
9. C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8(3):323–364, 1977.
10. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
11. J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Concur96*, 1996.
12. I. A. Mason and C. L. Talcott. A semantically sound actor translation. In *ICALP'97*, number 1256 in Lecture Notes in Computer Science, pages 369–378, 1997.
13. S. F. Smith. Specification diagrams for actor systems. In *Proceedings of Higher-Order Operational Techniques in Semantics, HOOTS II*, volume 10 of *Electronic Notes in Theoretical Computer Science*, 1998.
14. C. L. Talcott. An actor rewriting theory. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in Electronic Notes in Theoretical Computer Science, pages 360–383. North Holland, 1996.
15. 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.