BigActors - A Model for Structure-aware Computation

Eloi Pereira
Systems Engineering
UC Berkeley, CA, USA
eloi@berkeley.edu

Christoph M. Kirsch
Dept. of Computer Sciences
Univ. of Salzburg, Austria
ck@cs.uni-salzburg.at

Raja Sengupta
Systems Engineering
UC Berkeley, CA, USA
sengupta@ce.berkeley.edu

João Borges de Sousa
School of Engineering
Univ. of Porto, Portugal
jtasso@fe.up.pt

ABSTRACT

This paper describes a model of computation for structure-aware computing called the BigActor model. The model is a hybrid. It combines the Actor model [1] and the Bigraph model [10]. The contributions of this paper are an operational semantics, an example illustrating how the model supports the concise programming of a mobile agent working in a ubiquitous computing world, a query language enabling a bigActor to observe the world around it, and a definition giving semantics to the feedback loop in control theory in the context of this model. This is followed by three theorems showing how the operational semantics supports the programming of concurrent mobile agents in the semantics of feedback control.

Categories and Subject Descriptors

D.3.1 [Programming Languages]: Formal Definitions and Theory—Semantics; F.1.1 [Computation by Abstract Devices]: Models of Computation—Relations between models, Self-modifying machines

1. INTRODUCTION

This paper describes a model of computation for structure-aware computing. The model is a hybrid. It combines the Actor model [1] and the Bigraph model [10]. Hence the name BigActor model. In this model, computation is modelled using the Actor model, enriched by three new semantics rules (Figure 8) to make it structure-aware. Structure is a bigraph. The dynamics of the structure is modelled as a Bigraph Reactive System [9]. The three extra semantic rules provide means for an actor to observe and control a bigraph, and migrate from one host to another. Hence it is called a bigActor. When the structure models a physical world as in Example 3 of Section 4, bigActors become a cyber entities in a physical world.

The BigActor model is defined over a bigraph transition system. This is formally a sequence of bigraphs generated by applying Bigraph Reaction Rules [10] to an initial bigraph. Thus, structure can be dynamic and the three new semantic rules are designed to program actors that adapt to changing structure. Context-Aware programming needs a semantics for the interaction of program and context. In this paper we borrow the one underpinning control theory (see Figure 1), because it is widely used. In control theory the system is separated into plant and controller with the controller being composed in feedback. It observes, then controls, and the control generates new observations, repeating the cycle.

Figure 1: BigActors: actors composed as a feedback loop with a bigraph.
Section 6 shows the BigActor model is able to support this semantics of adaptation. The controller is restricted to being a bigActor and the plant a bigraph. Definition 5 is the semantics of the feedback loop. We define a feedback bigActor analogous to a feedback controller in Definition 11. Theorem 1 shows that in a system with one bigActor, being a feedback bigActor is sufficient for the control-theoretic semantics. This is called correctness for want of a better term. Theorems 2 and 3 show another sufficient condition is required for correctness when there are multiple concurrent bigActors. The additional conditions required to realize the feedback control semantics are the standard ones used to exclude race conditions in concurrent programming. Theorems 1, 2, and 3 formalize the sense in which the bigActor model supports the programming of concurrent agents working in a world with changing structure by adapting to it.

We situate the BigActor model in the Actor and Bigraph literatures as follows. The contributions of this model to the Actor literature are to provide the actor with a reflection of the structure of the system. This approach has been adopted to reflect back to actor systems low-level properties of the system. Nielsen, Ren, and Agha [11, 12] provide a real-time semantics to actor systems using timed graphs to constrain the execution of actors. Agha introduces locality to actors semantics [13] for modelling multiple mobile agents. The location model used by Agha is simply a host relation between actors and hosts. The location model is flat in the sense that there is no information of where a host stands in relation to another host. Moreover, there is no sense of connectivity between hosts. Using bigraphs in the BigActor model we provide a richer model of the structure, capable of modelling nested locality of components, their connectivity and also the way the structure evolves with time.

From a bigraph research stand-point, the BigActor model provides means for embedding computation into bigraphs that model the structure of the world. This is addressed in the literature by [5, 4] which model the structure of the world and computation using different Bigraph Reaction Systems (BRS) which are composed together. It is known that querying bigraphs exclusively using bigraphs reactive systems is difficult [4]. This is addressed by [4] by introducing three different BRSs: context, proxy, and agents. Our approach differs from [5] and [4] in the sense that we model computation using the actors model, which is coupled from an observation stand point using a query language. This removes the burden of modelling queries using BRSs. Moreover, by combining the Actor model with bigraphs we intend to leverage the use of bigraphs together with the large spectrum of actor languages and frameworks (e.g. Erlang [3], Scala [7], Cloud Haskell [6], Dart, Akka, etc.).

2. BIGRAPHICAL FRAMEWORK

In this section we introduce Bigraphs. A reader familiar with Bigraphs should be able to skip this section. The examples, however, are used throughout the paper. This explanation follows [10].

As the name suggests a bigraph is a mathematical structure with two graphs, the place graph - a forest that represents nested locality of components and a link graph - a hypergraph that models connectivity between components. Figure 2 presents an example of a bigraph.

Place graphs are contained inside regions (dashed rectangles) and may also contain holes (dark grey empty rectangles) Regions and holes enable composition of placing graphs, i.e. a hole of a given bigraph can be replaced by a region of another bigraph using the composition operator. We explain composition later.

A link graph may contain hyperedges and inner names and outer names. Names are graphically represented by a line connected at one end to a port or an edge and the other end is left loose. Just as one can fit regions inside holes, one can also merge inner names and outer names using the composition operator.

A node can have ports (black dots) which are points for connections to edges or names. The kinds of nodes and their number of ports (arity) are the signature of the bigraph.

The signature takes the form \( (K, \text{ar}) \) where \( K \) is a set of kinds of nodes called controls and \( \text{ar} : K \rightarrow \mathbb{N} \) assigns an arity (i.e. a natural number) to each control. Each node in the bigraph is assigned a control. For example, the bigraph of Figure 2(a) has the following signature: \( K = \{ M : 1, Q : 2 \} \) where \( M \) has kind \( M \) with arity 1 and \( q_i \) has kind \( Q \) with arity 2. By convention we start kind names with upper-case characters and node names with lower-case characters.

A bigraph \( B \) is called concrete when each node and each edge is assigned a unique identifier (known as support). We denote the set of node identifiers of \( B \) as \( \mathcal{V} \) and the set of edge identifiers as \( E \) (e.g. \( \mathcal{V} = \{ v_0, v_1, v_2, q_0, q_1 \} \) and \( E = \{ e \} \) in the example of Figure 2).

A bigraph without support is called abstract. In abstract bigraphs, nodes are denoted by their control while edges are kept anonymous. Abstract bigraphs are defined using an algebra. In this paper we almost exclusively use concrete bigraphs and thus we devote this section to present them formally. For a formal introduction to abstract bigraphs see [10], Chapter 3.

In order to define bigraphs formally we need to introduce the concept of a bigraph interface. An interface is a pair \( \langle n, X \rangle \) where \( n \) in the interface denotes the set \( \{ 0, 1, \ldots, n - 1 \} \) of holes (or regions) and \( X \) denotes the set of inner names (or outer names). Holes and inner names are collectively called an inner face while regions and outer names are called an outer face.

**Definition 1.** A bigraph is a 5-tuple of the form:

\[
(V, E, ctrl, prnt, link) : \langle m, X \rangle \rightarrow \langle n, Y \rangle
\]

where \( V \) is a set of nodes, \( E \) is the set of hyperedges, \( ctrl : V \rightarrow K \) is a control map that assigns controls to nodes,
prnt : m ⊕ V → V ⊕ n\(^1\) is the parent map and defines the nested place structure, link : X ⊕ P → E ⊕ Y is the link map and defines the link structure. P denotes the set of ports of the bigraph and is formalized as P = \{ (v, i) | i ∈ \{0, 1, \ldots, ar(\text{ctrl}(v)) − 1\} \}. For convenience we introduce a map \( \text{Pts} : V_B → \mathcal{P}(\mathbb{N}) \) that takes a node and returns the set of ports of that node. \( \langle m, X \rangle → \langle n, Y \rangle \) provides the inner face and outer face of the bigraph, i.e. \( \langle m, X \rangle \) is the inner face and \( \langle n, Y \rangle \) is the outer face.

The composition of bigraphs is defined by matching interfaces. Like function composition, a bigraph \( A : I → J \) composed with a bigraph \( B : K → I \) is a bigraph \( C : K → J \).

**Definition 2.** Let \( A : \langle m_a, X_a \rangle → \langle n_a, Y_a \rangle \) and \( B : \langle m_b, X_b \rangle → \langle n_b, Y_b \rangle \). The **composition** of bigraphs \( A \) and \( B \), denoted by \( A \circ B \), is a bigraph \( C : \langle m_c, X_c \rangle → \langle n_c, Y_c \rangle \) where \( V_C = V_A ⊕ V_B, E_C = E_A ⊕ E_B, \text{ctrl}_C = \text{ctrl}_A ⊕ \text{ctrl}_B \). \( \text{prnt}_C \) is obtained by “filling” the holes of \( A \) with regions of \( B \) while \( \text{link}_C \) is obtained by “merging” the inner names of \( A \) with the outer names of \( B \). By convention, the regions are matched to holes with the same indices and inner names are matched with outer names with the same name. For a formal definition of \( \text{prnt}_C \) and \( \text{link}_C \) see [10], page 17. The composition \( A \circ B \) is well defined iff the outer face of \( B \) is equal to the inner face of \( A \), i.e. \( \langle n_b, Y_b \rangle = \langle m_a, X_a \rangle \).

**Example 1.** Consider the bigraphs of Figure 3. The bigraph \( \text{streetMap} : \langle 6, \emptyset \rangle → \langle 1, \emptyset \rangle \) models a street map where the grey nodes represent streets and links represent physical adjacency between street nodes. Bigraph \( \text{networkInf} : \langle 6, \emptyset \rangle → \langle 6, \emptyset \rangle \) models a network infrastructure where the blue nodes model wireless hotspots and links represent connectivity which is linked to the edge network modelling a network. The bigraph resulting from the composition of both bigraphs is \( \text{streetMap} \circ \text{networkInf} : \langle 6, \emptyset \rangle → \langle 1, \emptyset \rangle \). Note that since the \( \text{networkInf} \) keeps the holes inside street nodes one could compose \( \text{streetMap} \circ \text{networkInf} \) with other features (e.g. cars, pedestrians, utilities network, etc.). The signature for this example and subsequent ones in this paper is \( \mathcal{K} = \{ \text{Street} : 1, \text{Wlan} : 1, \text{Comp} : 1, \text{Feat} : 0 \} \).

Figure 3: Composition of the bigraph \( \text{streetMap} \) with the bigraph \( \text{networkInf} \).

\footnote{The symbol \( ⊕ \) denotes the exclusive union operator for sets.}

Figure 4: Abstract BBRs \( \text{MOVE} \) that moves a Comp node from one street to another, and \( \text{CONNECT} \) that connects the Comp node to the network infrastructure.

\footnote{Redex stands for “reducible expression.”}

In Figure 3, \( \text{streeti} \) are of kind \( \text{Street} \), and \( \text{wlan} \) are of kind \( \text{Wlan} \). The kinds \( \text{Street} \) and \( \text{Wlan} \) denote respectively streets on a city and locations with wireless connectivity. The kinds \( \text{Comp} \) and \( \text{Feat} \) are going to be used in examples later and represent, respectively computational devices and features of the world to be observed.

## 2.1 Dynamics of bigraphs

Milner defines Bigraph Reaction Rules (BRR) to create dynamics on bigraphs [10].

A **bigraph reaction rule** is a tuple \((R, R’ , η)\) where \( R \) and \( R’ \) are bigraphs called respectively redex\(^2\) and reactum. The redex is the portion of the bigraph to be matched and the reactum is the bigraph that replaces the matched portion. \( η \) is called the instantiation map and indicates how holes in \( R \) correspond to holes in \( R’ \). If \( η \) is the identity map, then we represent the rule as \( R → R’ \). For the reminder of the paper we always assume \( η \) is the identity map, i.e. hole \( i \) in \( R \) matches with hole \( i \) in \( R’ \) for all \( i ∈ \mathbb{N} \). If \( R \) and \( R’ \) are abstract bigraphs, then \( R → R’ \) is an abstract BRR. If \( R \) and \( R’ \) are concrete then the BRR is concrete. It is often convenient to define BRRs to be abstract even when working with concrete bigraphs. This defines rules that can be applied to several contexts.

Let \( r = R → R’ \) be a BRR and \( B \) a bigraph. In order to perform the reaction \( r \) in \( B \) we need first to decompose \( B \) into \( C \circ \text{Rod} \) where \( C \) represents the context and \( d \) represents the parameters inside the holes of \( R \). Assuming that \( η \) is the identity map, we compose \( C \) with the reactum \( R’ \) and with \( d \) to get the resulting \( B’ \), i.e. \( B = C \circ R \circ d ⇒ B’ = C \circ R’ \circ d \). This application of BRRs works both for abstract and concrete BRRs.

**Example 2.** Consider the two abstract BRR of Figure 4. The first rule, \( \text{MOVE} \), models a computational device (e.g. a user with a smartphone) denoted by a star with kind \( \text{Comp} \) moving from one street to another. The second rule, \( \text{CONNECT} \), models computational device to move inside a hotspot with network connectivity (denoted by a node with kind \( \text{Wlan} \)) and connecting to the network. Note that the rules are parametric, i.e. they can be applied regardless the nodes inside the street nodes. For example, this rule allows modelling a computational device to move with or without other users in a street. Figure 5 shows an example of the application of BRR \( \text{MOVE} \). The bigraphs in this example are concrete. Thus, we need to first concretize \( \text{MOVE} \). Since we want to move \( \text{sp0} \) of kind \( \text{Comp} \) from \( \text{street2} \) to \( \text{street3} \) we...
Figure 5: Example of the application of the MOVE reaction rule.

Figure 6: Actor operational semantics.

The operational semantics is defined using the five rules of Figure 6. The rule \( \text{fun : a} \) models the execution of an expression \( e \) of an actor \( a \). \( \rightarrow \lambda \) denotes the semantics of a given language that specifies the local computation of the actor. Note that the execution of \( e \) to \( e' \) can produce side-effects in the actor’s local environment \( E \). This rule is also responsible for removing \( \text{nil} \) and making the actor ready to be terminated by rule \( \text{term} \). The rule \( \text{new : a, a'} \) models the spawn of a new actor \( a' \) from \( a \). The new actor \( a' \) becomes inactive after creation. The rule \( \text{term : a} \) models the termination of the execution of a command, making the actor changing from busy to inactive. The rule \( \text{rcv : a, \langle a = v \rangle} \) models an active actor \( a \) removing a message from its mailbox and updating its local environment with the contents \( v \). The rule \( \text{snd : a, \langle a' = v \rangle} \) models an actor \( a \) sending a message \( v \) to \( a' \). The new message is included in the set of pending messages.

4. BIGACTOR MODEL

This section formalizes the BigActor Model. The operational semantics is presented in Section 4.1. We explain the semantics by deriving a trace of the program in Example 3.

Example 3. Consider the pseudo-code in Figure 7. The pseudo-code is in the style of actor languages. The program models the following scenario. An app is hosted on a smartphone \( \text{app@sp} \) in \( \text{street2} \). It wants to photograph the green boxes (Figure 10) in \( \text{street1} \). Hence the smartphone (biggraph node \( \text{sp} \) moves. It specifies the picture to be taken by a query and waits till the picture is bound to its free variable. In order to send the picture to server@srv the app

\[ \text{fun : a} \rightarrow \lambda \]

\[ \text{new : a, a'} \]

\[ \text{term : a} \]

\[ \text{rcv : a, \langle a = v \rangle} \]

\[ \text{snd : a, \langle a' = v \rangle} \]

3The original semantics includes two more rules for handling communication with external actors, i.e. actors not defined in the configuration. External actors are not an essential feature for BigActors and thus we decided to remove them for the sake of simplifying the exposition.
moves to one of the blue circles modeling wireless hotspots, and connects to the hyperlink network. It then sends the message to the server.

4.1 Operational semantics

The operational semantics is entirely in Figure 8. We use the same contextual style as the actor semantics in Section 3. A BigActor model executes by transitioning through a sequence of configurations. Definition 4 defines a configuration.

Definition 4. A BigActor configuration is a tuple \((\alpha | \mu | \eta | B)\) where \(\alpha\) is a set of bigActors, \(\mu\) is a set of pending messages, \(\eta\) is a set of pending requests, and \(B\) is a bigraph. A BigActor in \(\alpha\) is of the form \(a@h\), with \(a\) an actor, and \(h\in V_B\) a node in the bigraph \(B\). \(h\) is called the host of \(a\). The set of pending messages \(\mu\) is as in the actor semantics in Section 3. A request in the set of requests \(\eta\) is a tuple \((a@h,r)\) meaning that the operation \(r\) is requested by \(a@h\). The operation \(r\) can be any one of the expressions \(\text{send}(a@h,m)\), \(\text{observe}(q)\), \(\text{control}(u)\), \(\text{migrate}(h')\). These are the premises of semantic rule \((\text{req} : a@h,r)\) in Figure 8. \(a@h\) can be any BigActor in \(\alpha\), \(m\) any actor message, \(q\) a query resulting in an observation by a BigActor of the bigraph \(B\) as expressed by Equation 1, \(u\) any concrete bigraph reaction rule as described in Section 2, and \(h'\) another node in the bigraph \(B\).

The interpretation of a query into an observation of a given bigraph \(B\) relative to a host \(h\) is specified by the map:

\[
\llbracket \text{query} \rrbracket_h^B : \text{query} \rightarrow B
\]

(1)

Queries and observations are addressed in Section 5.

Of the four elements in the BigActor configuration, \(\alpha\) is as in the actor model except we write the \(a@h\) instead of just \(a\) for the elements of \(\alpha\), \(\mu\) is exactly as in the actor model, and \(\eta\) and \(B\) are new. Definition 5 defines an execution of the BigActor model as a sequence of configurations.

Definition 5. A BigActor execution is a sequence \(c_0,c_1,c_2,\ldots\) where each \(c_i \xrightarrow{\lambda_i} c_{i+1}\) and \(\lambda_i\) is derived by a semantic rule labelled \(\lambda_i\). The permissible \(\lambda_i\) are those in Figure 8.

The semantics can be understood as follows. A BigActor can do actor computations. This is asserted by the inclusion of the rules \((\text{fun} : a@h)\), \((\text{new} : a@h,a@h)\), \((\text{term} : a@h)\), and \((\text{rcv} : a@h,m)\). This rules are the same as their actor counterparts in Figure 6. The augmentation of an actor \(a\) to \(a@h\) and the expansion of the configuration to include \(\eta\) and \(B\) has no significance in the rules \((\text{fun},\text{term})\) and \((\text{rcv})\). In the rule \((\text{new})\), the \(a@h\) is used to assert that a new actor will have the same host as its creator. Thus to an actor programmer \((\text{new})\) has the usual actor semantics, though in the BigActor model \((\text{new})\) where the \(a@h\) has the semantics of a host, the rule prevents the creation of remote bigActors. The environment \(E\) in Figure 8 is the same as the \(E\) in the actor model in all rules.

In addition to executing expressions by \((\text{fun},\text{term},\text{new})\) and \((\text{rcv})\) as an actor would, a BigActor can also execute the expressions \((\text{send}(a@h,m))\), \((\text{observe}(q))\), \((\text{control}(u))\) and \((\text{migrate}(h'))\) as syntactically formalized in Definition 4. The semantics of these expressions are unique to this model and given first by the rule \((\text{req} : a@h,r)\) and then by the rules labelled as \(\text{snd,obs,ctr and mgrt}\), respectively. The \(\text{req}\) rule asserts that when any of the expressions \((\text{send}(a'@h',m))\), \((\text{observe}(q))\), \((\text{control}(u))\) or \((\text{migrate}(h'))\) execute in a BigActor program, the program advances exactly as an actor program by the rule \((\text{fun})\) in Figure 6 but with a side-effect. The side-effect is the addition of the request \((a@h,r)\) to the set \(\eta\). This creates a new configuration. When \(\eta\) contains a request with an expression \((\text{send}(a'@h',m))\), \((\text{observe}(q))\), \((\text{control}(u))\) or \((\text{migrate}(h'))\) a BigActor model can further advance by application of the rules \(\text{snd,obs,ctr and mgrt}\) as described next.

To make the rest comprehensible we analyze an execution trace of the program in Figure 7 from Example 3.

The \((\text{control})\), \((\text{observe})\) and \((\text{send})\) instructions have the syntax and semantics in Figure 8 and Definition 4. These instructions are encapsulated in two behaviors named the method \((\text{start})\) and method \((\text{rcv}0\text{bs})\). The syntax of the method constructor and sequentiality are not formalized. We informally assume instructions execute in the order in which they appear in the text and the BigActor \((\text{app@sp})\) starts with the method \((\text{start})\). The rest follows formally from the semantics in figure 8. Figure 9 is a formal execution of this program in the sense of definition 5.

All configurations in Figure 9 follow the notation in Definition 4, Definition 5, and two short-hands for brevity. The Figure has 9 lines. Each line has two configurations. Each line has two transitions except the last one which has only one. In each configuration the reduction context, denoted by \(R\) in the semantic rules, is abstracted with a .. The \((\text{app@sp})\) program instruction executing at each step is explicitly typed in the square brackets in each configuration. The environment \(E\) in the semantic rules is also abstracted by . In the configurations in the first 3 lines and the first configuration in line 4. The environment is a valuation of the variables of \((\text{app@sp})\) just as in the actor model. \((\text{app@sp})\) has only one variable \(B,\text{obs}\). This has no valuation in the first three lines and acquires a value in the second configuration in line 4. It is denoted with an \(E\) thereafter. \(B_0\) through \(B_2\) are bigraphs. These are visualized in Figure 10. This figure visualizes a projection of the trace in Figure 9. It projects away all transitions representing applications of \((\text{fun}),\text{term}\) and \((\text{req})\). The \((\text{fun})\) and \((\text{term})\) transitions advance the program \((\text{app@sp})\) just like an actor program and do nothing particular to this model. The \((\text{req})\) transitions advance like \((\text{fun})\) but also have the side-effect of adding a request to the set \(\eta\). Then other semantic rules can be applied to consume the elements in \(\eta\) as mentioned earlier to produce the configurations visualized in Figure 10.

The execution trace in Figure 9 can now be understood as follows. We assume the BigActor \((\text{app@sp})\) starts with the behavior \((\text{start})\) and advances to the first instruction by the rule \((\text{fun})\). The next transition is by the \((\text{req})\) rule and puts \((\text{control})(\text{move}(\text{street2},\text{street1}))\) in \(\eta\). \((\text{move})(\text{street2},\text{street1})\) is a concretization of the abstract BRR \((\text{move})\) in Figure 4. This enables use of the rule \((\text{ctr})\) to change the bigraph \(B_0\) to \(B_1\). This is visualized in the top row of Figure 10. One can see the white star move. Everything else in the two bigraphs is the same. The second transition in line 2 simply advances the actor program by rule \((\text{fun})\) to the instruction \((\text{observe})(\text{children.parent.host})\). The first transition in line 3 puts this imperative into \(\eta\) by the rule \((\text{req})\). The first behavior is now exhausted and the actor become inactive by application of the rule \((\text{fun})\) and then \((\text{term})\) resulting in the two configurations in line 4. In the second configuration
in line 4 the BigActor \text{app@sp} has advanced to the second behavior called \text{rcvObs} in Figure 7. The \text{observe(q)} imperative is still present in \eta which allows us to apply the rule \text{obs} taking us to the first configuration in line 5. The query \text{q} in the program is \text{children, parent, host} and is interpreted on bigraph \text{B1} with respect to \text{sp}. This transition is visualized in Figure 10 as the transition from the top row to the middle one. Since the host node of \text{app} is \text{sp} denoted by the white star, its parent is \text{street1} and the children are the green nodes and \text{sp} itself. The interpretation of queries is formalized in Section 5. The application of rule \text{obs} produces the bifgraph enclosed by the dashed region in the left-hand figure of the middle row of Figure 10. The transmission of the observed bigraph to the BigActor in the rule \text{obs} is handled much like an actor receives a message. The inactive actor become busy and the observation is bound to one of its free variables, \text{FV(rcvObs)} in line 5 of Figure 9. The behavior has only one free variable named \text{B.obs} in Figure 7. The environment \text{E} is now the observed bigraph being the valuation of \text{B.obs}. \eta once again becomes empty. The rest of line 5, lines 6 and 7 in Figure 9 are derived like the prior lines. The rule \text{CONNECT(wlan0)} is a concretization of the abstract BRR in Figure 4.

Lines 8 and 9 illustrate the semantics of the BigActor \text{send}. In the second configuration in line 8 the BigActor is busy at the expression \text{send(server@srv, B.obs)}. Like the actor model \text{server@srv} is intended to denote another Bi-

\text{gActor supposed to receive the message. The message is the value of \text{B.obs}. The actor semantics would put this message in} \mu. We put the expression in \eta just like the other requests and then apply \text{snd} rule to complete message delivery. We do this because the premises of the \text{snd} rule requires a link in the bigraph between the hosts of the sending and receiving BigActors. If the existence of this link can be proven, the message is transferred from \eta to \mu. It can in this case due to prior execution of the expression \text{connect(wlan0)} as illustrated by the transition connecting the middle and last rows of Figure 10. It would then be received by the other actor by application of the rule \text{rcv} exactly as in the actor model.

The BigActor rule \text{snd} is the counterpart of the actor rule with the same name, and the BigActor expression \text{send(a@h', m)} the counterpart of the actor \text{send(a'.m)}. However, the semantics is different since it requires the existence of a link between two hosts in the bigraph. This means a BigActor sending a message to a correct mailbox address could fail to communicate. This is a violation of actor semantics but not of BigActor semantics. A good analogy might be accessing a webpage in the internet, it is not sufficient to know the URL, one must also be connected to the internet by some means (e.g. wireless connection, 3G, etc.).

The example does not illustrate the rule \text{mrgt} for the expression \text{migrate}. This rule behaves much like the others. A BigActor can request a migration by the expression
The programs can change their distribution (rule \textit{mgrt}) the request can be consumed by the rule \textit{mgrt} and the BigActor and \textit{hrupt} thereafter. and locate them in the machine (locally). They could migrate up to the machine (bigraph links). The programs can communicate between the actors and the bigraph. Once again, we are thinking of the link as the physical network required for the flow of a nomadic program.

We conclude this section with a few remarks.

\textbf{Remark 1.} The program in Figure 7 is precise in the BigActor model. We see the conciseness of the program as a contribution of the model.

\textbf{Remark 2.} We see the relationship between BigActor and bigraph like the relationship between a program and its machine. Actors programs are concurrent programs and so the machine can be a distributed one. The actors model the programs and the bigraph a shared distributed machine. The programs can observe the machine (rule \textit{obs}) and change the machine (rule \textit{ctr}). Control can move, add, or remove nodes and do the same with connections (links). The hosting relationship is the distribution of the program over the machine. The programs can change their distribution (rule \textit{mgrt}). The programs can exchange messages (rules \textit{snd}, \textit{rcv}) but only up to the machine (bigraph links). The programs can compute (rules \textit{fun}, \textit{term}) and create new programs (rule \textit{new}) and locate them in the machine (locally). They could migrate thereafter.

\textbf{Remark 3.} In this semantics the coupling of BigActor and bigraph is asynchronous. Program execution puts requests in \( \eta \). The requests affect the bigraph, hosting relationship, or observations at a later instant of model time. This means a BigActor can make requests executable in the bigraph at request time but unexecutable at the later time. Section 6 addresses this issue.

\textbf{Remark 4.} Control is local. If a BigActor seeks to replace component \( R \) of a bigraph with \( R' \) then the rule \textit{ctr} requires the BigActor to be located (hosted) in \( R \). We show in Section 5 that observation is also local.

\textbf{Remark 5.} In the actor model state is local. The memory of an actor influences another only up to the messages flowing between them. In the BigActor model the Bigraph is a shared structure, and some might argue this globalizes state. For example, one may communicate between bigActors by placing and removing nodes in the bigraph and never send any inter-actor messages at all. We would consider this misuse of the model, but have nevertheless elected to provide the bigraph as a shared structure to keep the model expressive. The BigActor formalism is a model for actors operating in the physical world, which may in many applications be a shared space to the actors.

Thus the bigraph can be used as a back-channel for communication between actors. There are two ways to close this channel. The first is rather trivial. One may simply make the bigraph into an actor. Then all \textit{req}, \textit{obs}, \textit{ctr}, and \textit{mgrt} actions would appear as the flow of messages between actors and the entire physical world would become the local state of one actor. This solves the problem but does not really address any of the fundamental difficulties of coordinating observing and changing a shared physical machine. Therefore we have presented a semantics in which the bigraph is not an actor, because by making the coordination difficulties visible in the model, the programmer gets the opportunity to program for them.

However, the model does support a sufficient condition, derived from its semantics closing the bigraph as a back-channel for communication. Basically, one requires two bigActors never control and observe the same region of the bigraph. The semantics of a BigActor permits us to derive the area of influence of a BigActor as expressed in Definition 6.

\textbf{Definition 6.} Let \( a \circ h \) be a BigActor and \( R_a = \{ R_0 \rightarrow R_0', R_1 \rightarrow R_1', \ldots, R_n \rightarrow R_n' \} \) the set of all concrete BRRs used by it. The \textbf{area of influence} of \( a \circ h \) over a bigraph \( B \) is the set of nodes \( A \) such that \( v \in A \Rightarrow \exists i. B = C \circ R_i \circ d \wedge v, h \in V_{R_i} \).

\textbf{Example 4.} Consider Example 3 where the host of the app BigActor is inside \textit{street2} (bigraph \( B_0 \) of Figure 10). The area of influence under the current bigraph is:

\[ A^{appBA} = \{ \textit{street0}, \textit{street1}, \textit{street2}, \textit{street3}, \textit{street4} \} \]

Note that due to the current location of the host the only rule that can be applied is \textit{MOVE}.

Let \( c_0 \rightarrow c_1 \rightarrow \ldots \) be an execution trace where \( c_i = (\alpha_1 | \mu) \text{ or } (\eta_1 | B_i) \) is a BigActor configuration and \( \rightarrow \) is given by the BigActor semantics. Let \( A_i^{a \circ h} \) be the area of influence of \( a \circ h \) over the bigraph \( B_i \) and \( B^{a \circ h}_{obsj} \) be an observation of \( a \circ h \) over the bigraph \( B_j \). If for any two bigActors \( a \circ h \) and \( a' \circ h' \), \( A_i^{a \circ h} \) and \( V_{B^{a \circ h}_{obsj}} \) never overlap for any time \( i < j \) then the bigActors cannot use the bigraph as a back channel for communication, e.g. \( a \circ h \) might place a node in the bigraph but \( a' \circ h' \) would not observe it because the sets \( A_i^{a \circ h} \) and \( V_{B^{a \circ h}_{obsj}} \) would be disjoint.
5. QUERY LANGUAGE AND OBSERVATIONS

In this section we introduce a query language for querying the bigraph for local observations.

The query language syntax is specified by the following grammar:

\[
\text{query ::= node|children.node|linkedTo.node} \quad \text{node ::= host|parent.node}
\]

Observations are local with respect to the host node of the BigActor that is querying. We think of “local” in terms of the placing graph (parents and children of a given node) and in terms of the linking graph (nodes linked to a given node). Our intention is to introduce a query language that is expressive enough for the examples in the paper and that is compositional. Our aim is not to introduce a query language for Bigraphs.

The interpretation of a query \( q \) over a bigraph \( B \) with respect to a host \( h \) is defined such that a given observation \( B_{\text{obs}} = [[q]]^B_h \) is composable with the remaining context of \( B \), i.e. there exists a context \( C \) and parameters \( d \) such that \( B = C \circ B_{\text{obs}} \circ d \).

Example 5. Figure 11 presents an example of querying the bigraph \( B_0 \) of Figure 10 with respect to \( \text{sp} \). On the right-hand side of Figure 11 one can see the interpretation of the queries \( \text{host, parent.host, children.parent.parent.host} \).

![Figure 11: BigActor observing the street map using query children.parent.parent.host.](image)

Let \( B \) be given by \( (V_B, E_B, \text{ctrl}_B, \text{prnt}_B, \text{link}_B) : (m_B, X_B) \rightarrow (n_B, Y_B) \). The interpretation of the query language goes as follows.

\[
[[\text{host}]]^B_h \rightarrow ([h], \emptyset, \text{ctrl}_h, \text{prnt}_h, \text{link}_h) : (1, \emptyset) \rightarrow (1, Y_h),
\]

where \( \text{ctrl}_h(h) = \text{ctrl}_B(h) \),

\[
\text{prnt}_h(v) = \begin{cases} 0 & \text{if } v = h \\ h & \text{if } v = 0 \end{cases},
\]

\[
\text{link}_h(pt) = \begin{cases} \text{link}_B(pt) & \text{if } \text{link}_B(pt) \in Y_B \\ y_e & \text{if } e = \text{link}_B(pt) \in E_B \end{cases},
\]

\[
Y_h = \{y_e \mid \forall pt.e = \text{link}_h(pt) \land e \in E_B \} \cup \{y \mid \forall pt.y = \text{link}_h(pt) \land y \in Y_B \}.
\]

Edges are abstracted by unique names to make the composition \( B = C \circ B_{\text{obs}} \circ d \) valid. The edges are included in the context and are also connected to the same names. Composition would bind the names in the context and in the observation restoring the edges. Also note that the contents of node \( h \) are abstracted with a hole 0.

Let the interpretation of \([\text{node}]^B_h\) be denoted as the bigraph \( B_0 \). Note that \( B_0 \) only has one node which we denote as \( n \), i.e. \( V_{B_0} = \{n\} \). The interpretation if \text{parent.node} is given by:

\[
[[\text{parent.node}]]^B_h \rightarrow ([p], \emptyset, \text{ctrl}_p, \text{prnt}_p, \text{link}_p) : (1, \emptyset) \rightarrow (1, Y_p)
\]

where \( p = \text{prnt}_B(n) \). The remainder of the bigraph definition is provided by replacing \( p \) for \( h \) in the definition of \([\text{host}]^B_h\).

The interpretation of \text{children.node} is given by:

\[
[[\text{children.node}]]^B_h \rightarrow ([C], \emptyset, \text{ctrl}_C, \text{prnt}_C, \text{link}_C) : (1, \emptyset) \rightarrow (1, Y_C)
\]

where \( Y_C = \{v \mid v \in V_C, n = \text{prnt}_B(v)\} \) and

\[
\text{prnt}_C(v) = \begin{cases} 0 & \text{if } v \in V_C \\ v' & \text{if } v = \text{holeOf}(v') \end{cases}
\]

\text{holeOf} : V_C \rightarrow N \text{ is a function that assigns to each children node a unique hole to abstract its contents.} \text{ctrl}_C, \text{link}_C, \text{and the set of external names } Y_C \text{ are given as before although now ranging over the set of children nodes and respective ports.}

Finally, the interpretation of \text{linkedTo} is given by:

\[
[[\text{linkedTo}]]^B_h \rightarrow ([L], \emptyset, \text{ctrl}_L, \text{prnt}_L, \text{link}_L) : (1, \emptyset) \rightarrow (1, Y_L)
\]

where \( Y_L = \{v \mid \exists pt \in \text{ports}(v), \exists pt' \in \text{ports}(n). \text{link}_B(pt) = \text{link}_B(pt')\} \),

\[
\text{prnt}_L(v) = \begin{cases} \text{regionOf}(v) & \text{if } v \in V_L \\ v' & \text{if } v = \text{holeOf}(v') \end{cases}
\]

Note that for each node in \( V_L \) we create a unique region using the function \text{regionOf} : V_L \rightarrow N. By putting each node in a unique region we can restore the parenthood by composing again with the context. The remainder of the bigraph definition is as per the prior cases.

In this paper we use the query language \text{query} to define bigraphs which are local under a given bigraph and with respect to a host node.

Definition 7. Given two bigraphs \( B \) and \( B' \) and a host node \( h \), \( B' \) is local with respect to \( h \) if there exists a \( q \) such that \([q]^B_h = B' \) or \([q]^B_h = C \circ B' \circ d \) for an arbitrary context \( C \) and parameters \( d \). In other words, \( B' \) is local with respect to \( h \) if one can find a query with an interpretation containing \( B' \).

An observation \( B_{\text{obs}} \) obtained by a BigActor \( a @ h \) querying a bigraph \( B \) using the local query language is, by Definition 7, a local bigraph over \( B \) with respect to \( h \) (\([q]^B_h = C \circ B_{\text{obs}} \circ d \) where \( C \) and \( d \) are empty).

6. CORRECT EXECUTION

In this section we address the correctness of BigActors executions. Correctness is stated as a safety property over
configurations. In this section we assume that requests are served as per a First-Come First-Served (FCFS) scheduling semantics.

Definition 8. Let $c = (\alpha | \mu \mid \eta | B)$ be a BigActor configuration. $c$ is terminal if $\mu = \emptyset$, $\eta = \emptyset$, and all bigActors in $\alpha$ are inactive.

Definition 9. Let $c = (\alpha | \mu \mid \eta | B)$ be a BigActor configuration. $c$ is correct if $\exists \lambda \in \Lambda$ such that $c \rightarrow^\lambda c'$ or $c$ is terminal.

Let $\Lambda$ denote the set of labels as per the semantics stated in Figure 8. Let $\Lambda_0$ denote the set of request labels and $\Lambda_r$ denote the set of labels. Consider a function $scheid : \Lambda_0 \rightarrow \Lambda_\eta$ where, for a given execution $t = c_0 \rightarrow^\lambda_1 c_1 \rightarrow^\lambda_2 \ldots \rightarrow^\lambda_n c_n$ returns the request served by $\lambda_n$. Let $c_t \subseteq \Lambda \times \Lambda$ be a total order induced by the index $i$ of $\lambda_i$ in $t$.

Definition 10. An execution $t = c_0 \rightarrow^\lambda_0 c_1 \rightarrow^\lambda_1 \ldots$ follows a First-Come First-Served (FCFS) discipline if:

$$\lambda_i \prec_{c_t} \lambda_j \Rightarrow schd(\lambda_i) \prec_{c_t} schd(\lambda_j)$$

where $\lambda_i, \lambda_j \in \Lambda_r$.

In other words, FCFS discipline requests that requests are served in the order that are requested.

6.1 Correctness of a single bigActor

The first step towards correctness is to require that a single bigActor itself never requests operations that will lead to an incorrect configuration.

Definition 11. Let $t = c_0 \rightarrow^\lambda_0 c_1 \rightarrow^\lambda_1 \ldots$ be a BigActor execution. For each $\lambda_i$ the operational semantics identifies a unique actor $\alpha @ h$. For clarity we augment $\lambda_i$ to $\lambda_i^{\alpha @ h}$. $\alpha @ h$ is called feedback for all $\alpha @ h$ and $\alpha @ h'$ that are either control, migrate, or send the following are true:

1. $\forall j. \lambda_i^{\alpha @ h} \rightarrow (\alpha : a @ h, \lambda_i) \Rightarrow \exists i. j < i \wedge \lambda_i \Rightarrow (\alpha : a @ h, \lambda_i)$

2. $\forall i. k > i. \lambda_i^{\alpha @ h} \rightarrow (\alpha : a @ h, \lambda_i) \wedge k = (\alpha : a @ h, \lambda_i') \Rightarrow \exists j. i < j \wedge \lambda_j = (\alpha : a @ h, \lambda_i')$

3. For any $\lambda_i$ let $B_{\alpha \theta \lambda}^{\alpha @ h}$ denote the last observation of $\alpha @ h$ preceding $\lambda_i$. Let $\forall i. \lambda_i^{\alpha @ h} \rightarrow (\alpha : a @ h, \lambda_i) \Rightarrow \exists \lambda_j, \lambda_k, \theta, \alpha @ h, \lambda_i \rightarrow (\alpha : a @ h, \lambda_i)$.

The first requirement states that before any observation by $\alpha @ h$ that is not an observation, there must be an observation by $\alpha @ h$. The second requirement states that between two requests $\alpha @ h$ that are not observation there must be an observation by $\alpha @ h$. The last requirement states that, given a configuration $B_{\alpha \theta \lambda}^{\alpha @ h}$ as the biggraph and one request $\alpha @ h$ then there is a new configuration obtained by consuming $\alpha @ h$. This means that $\alpha @ h$ is the does not make a configuration with the observed biggraph incorrect.

Theorem 1. Consider a BigActor execution $t = c_0 \rightarrow^\lambda_0 c_1 \rightarrow^\lambda_1 \ldots \rightarrow^\lambda_n c_n$ that follows a FCFS discipline (Assumption 1.1). Assume that $c_i$ contains a single BigActor $\alpha @ h$ for all $i$ and $\alpha @ h$ is a feedback BigActor (Assumption 1.2). Assume that $\eta$ and $\mu$ are empty at $c_0$ (Assumption 1.3). Then $c_n$ is correct.

Proof. By mathematical induction in the the number of transitions.

Base case Given $c_0 \rightarrow^\lambda_0 c_1$, $c_1$ is correct.

By Assumption 1.3 $a @ h$ must be busy otherwise $c_0$ is terminal (Definition 8) and $\lambda_0$ can not be triggered. If $a @ h$ is busy then it can trigger any of the following rules leading to $c_1$: fun, new, term, and req. Since the overall execution is assumed to have only one BigActor new is excluded. By Assumption 1.3, the rules ctr, mgrt, obs, snd can not be executed since $\eta$ is still empty. fun, and term are internal to the actor and can not generate a transition to an incorrect configuration. By Definition 11 item 1, the rule req can only request an observation because a control, migrate, or a send request require previous observations.

Inductive step Give $c_i \rightarrow^\lambda_i c_{i+1}$ where $c_i$ is correct then $c_{i+1} \rightarrow^\lambda_{i+1}$ and $c_{i+1}$ is correct. Since $c_i$ is correct then $\lambda_i$ can be given by one of the following rules: fun, rcv, term, req, ctr, mgrt, obs, and snd (new is excluded by Assumption 2).

By the semantics definition of Figure 8, the rules fun, rcv, and term are internal to the bigActor and do not produce nor consume requests that can put the configuration in an incorrect state. The rule req can produce a new request, i.e. $\eta_{i+1} = \eta_i \cup \{r\}$. If $r$ corresponds to an observation then $\lambda_{i+1}$ is correct since the observations do not change the biggraph. If $r$ is either a control, a migration or a send request then it must produce a correct configuration under $B_{\alpha \theta \lambda_{i+1}}^{\alpha @ h}$. Let $c_i \rightarrow^\lambda_i \rightarrow^\lambda_{i+1} c_{i+1}$ where $c_i$ is decomposed as $c_i = B_{\alpha \theta \lambda}^{\alpha @ h} \circ d$ where $B_{\alpha \theta \lambda_{i+1}}^{\alpha @ h}$ is the last observation prior to $r$. By Definition 11 item 3, $r$ can be applied to $B_{\alpha \theta \lambda_{i+1}}^{\alpha @ h}$. Since $B_{\alpha \theta \lambda_{i+1}}^{\alpha @ h} \circ d$ also produces a correct configuration under $B_{\alpha \theta \lambda_i}^{\alpha @ h}$. The rules ctr, mgrt, and obs consume one request $r$ out of $\eta_i$. By Assumption 1.1, all requests of $\alpha @ h$ are served in order. Since all requests in $\eta_i$ were generated by a (single) feedback BigActor, then $\eta_{i+1} = \eta_i \setminus \{r\}$ do not make $c_{i+1}$ incorrect. $$\square$$

6.2 Correctness of concurrent bigActors

In the multiple bigActor case, the feedback property is necessary but not sufficient for correctness. A bigActor can request an operation executable at request time by the operational semantics, but not executable when the request is served. This is a consequence of the fact that in the bigActor operational semantics (Figure 8) the time which a request is generated is necessarily different from the time the request is served. We take two strategies to avoid these concurrency issues and ensure a correct execution of multiple bigActors: prevent undesired interleaving (time strategy); and/or prevent shared resources (spatial strategy). These lead to theorems 2 and 3.

6.2.1 Atomic read-write semantics

Definition 12. Let $\Lambda_\alpha$ denote the set of all ctr, mgrt, and snd labels and let $\Lambda_\alpha^{\alpha @ h}$ be the subset of labels from $\Lambda_\alpha$ produced by serving requests from $\alpha @ h$. An execution $t = c_0 \rightarrow^\lambda_0 c_1 \rightarrow^\lambda_1 \ldots$ follows an atomic read-write semantics if for any $\lambda_i = (\alpha : a @ h, \lambda_i)$ for which there exists a $\lambda_{i+1} \in \Lambda_\alpha^{\alpha @ h}$ then $\lambda_{i+1} \notin \Lambda_\alpha^{\alpha @ h}'$, $j \in \{1, \ldots, k - 1\}$ where $k = \min \{\lambda_{i+k} \in \Lambda_\alpha^{\alpha @ h}\}$ and $a @ h \neq a @ h'$.

In other words, an atomic read-write semantics requires that between any obs from $a @ h$ and subsequent ctr, mgrt, or
and from the same bigActor, no other ctr, mgrt, or snd can occur from another bigActor \( a'@h' \).

**Theorem 2.** Consider a BigActor execution \( t = c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \) that follows a FCFS discipline (Assumption 2.1) and the execution follows an atomic read-write semantics (Assumption 2.2). Assume that all bigActors are feedback (Assumption 2.3). Assume that \( \eta \) and \( \mu \) are empty at \( c_0 \) (Assumption 2.4). Then \( c_n \) is correct.

Proof. By Assumption 2.3 all bigActors are feedback. Since the feedback property is only sufficient for correct executions of single bigActor configurations we can, without loss of generality, assume that the configurations at execution \( t \) contains only two bigActors \( a@h \) and \( a'@h' \). By Assumption 2.1 and Assumption 2.2 the execution has no interleaving between observations and ctr/mgrt/snd of two different bigActors. Thus, one can partition the sequence \( t \) into subsequences \( t_0^a@h, t_0^a@h', t_1^a@h, t_1^a@h', \ldots \) where each \( t_k^a@h \) denotes the execution of a single BigActor \( a@h \) (and possibly labels regarding rules fun, term, new and recv which are internal to bigActors and thus do not make the configuration incorrect). Since each bigActor \( a@h \) is feedback, by Theorem 1, each \( t_k^a@h = c \rightarrow \ldots \rightarrow c' \), \( c' \) is correct. Thus, \( c_n \) is also correct. \( \square \)

### 6.2.2 Partitioning semantics

**Definition 13.** Consider an execution \( t = c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \). Let \( A^a@h_0 \) denote the area of influence of \( a@h \) at \( c_0 \). An execution follows a partitioning semantics if for every two bigActors \( a@h \) and \( a'@h' \)

1. \( \forall i. \forall j. A^a@h_i \cap A^a'@h'_j = \emptyset \)

2. \( \forall i. \forall j. \forall v. v \in A^a@h_i. v' \in A^a'@h'_j. Lks(v) \cap Lks(v') = \emptyset \)

where \( Lks : V_B \rightarrow \mathcal{P}(E_B \cup Y_B) \) is a function that returns the set of links connected to ports of a given node.

In other words, partitioning semantics requires the areas of influence of any two bigActors and their links be disjoint at every time.

**Theorem 3.** Consider a BigActor execution \( t = c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \) that follows a FCFS discipline (Assumption 3.1) and the execution follows a partitioning semantics (Assumption 3.2). Assume that all bigActors are feedback (Assumption 3.3). Assume that \( \eta \) and \( \mu \) are empty at \( c_0 \) (Assumption 3.4). Then \( c_n \) is correct.

Proof. Without loss of generality assume the execution of two bigActors \( a@h \) and \( a'@h' \). By Definition 13, the areas of influence and corresponding links are disjoint throughout the overall trace \( t = c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \) then for each \( B_i \) at configuration \( c_i \) can be decomposed into \( B_i = B_i^a@h \circ B_i^a@h' \) where \( B_i^a@h \) and \( B_i^a@h' \) are in two different bigraph regions without sharing any edges or names. Thus, each configuration \( c_i = \{a@h, a'@h'\} \cup \mu_i \cup \eta_i \cup B_i \) can be decomposed into two configurations \( c_i^a@h = \{a@h\} \cup \mu^a@h \cup \eta^a@h \cup B^a@h \) and \( c_i^a@h' = \{a'@h'\} \cup \mu^a@h' \cup \eta^a@h' \cup B^a@h' \) where \( \mu_i = \mu^a@h \cup \mu^a@h' \) and \( \eta_i = \eta^a@h \cup \eta^a@h' \). Since \( a@h \) and \( a'@h' \) are feedback, then by Theorem 1, there exists two traces where \( c_0^a@h \rightarrow \ldots \rightarrow c_n^a@h \) and \( c_0^a@h' \rightarrow \ldots \rightarrow c_n^a@h' \) are correct. Thus, \( c_n \) is correct. \( \square \)

### 7. CONCLUSIONS

In this paper we introduce the BigActor model for modelling structure-aware computation. The model combines the Actor model [1] and the Bigraph model [10]. We introduce an operational semantics for the BigActor model as an augmentation of the Actor model semantics. The semantics is enriched with three rules that provide means for bigActors to migrate and to observe and control the bigraph. We introduce a query language and introduce means for preventing concurrent issues that arise due to the fact that bigActors operate in a shared bigraph.

BigActor model provide means for defining agents that observe, control, and migrate over a structure of the world modelled as a bigraph. The BigActor model also introduces a model of reflection of the structure in for actor system. BigActors can also be used to embedded actors into bigraphs.

As per the future work, we are aiming at developing a theory of controllability and observability for bigActors in a control theory stand-point. We are also interested in implementing a prototype for a bigActor domain specific language for autonomous vehicles.

### 8. REFERENCES


