Components as Location Graphs

Jean-Bernard Stefani

Components as Location Graphs

Jean-Bernard Stefani

INRIA

Abstract. This paper presents a process calculus framework for modeling ubiquitous computing systems and dynamic component-based structures as location graphs. A key aspect of the framework is its ability to model nested locations with sharing, while allowing the dynamic reconfiguration of the location graph, and the dynamic update of located processes.

1 Introduction

Motivations. Computing systems are increasingly built as distributed, dynamic assemblages of hardware and software components. Modelling these assemblages requires capturing different kinds of dependencies and containment relationships between components. The software engineering literature is rife with analyses of different forms of whole-part, aggregation or composition relationships, and of their attendants characteristics such as emergent property, overlapping lifetimes, and existential dependency [2]. These analyses explicitly consider the possibility for a component to be shared at once between different wholes, an important requirement in particular if one expects to deal with multiple architectural views of a system.

Consider, for instance, a software system featuring a database DB and a client of the database C. The database comprises the following (sub)components: a cache CC, a data store DS and a query engine QE. Both data store and query engine reside in the same virtual machine V_0, for performance reasons. Client and cache reside in another virtual machine V_1, also for performance reasons. We have here a description which combines two architectural views, in the sense of [17]: a *logical* view, that identifies two high-level components, the database DB and its client C, and the sub-components CC, QE and DS of the database, and a *process* view, that maps the above components on virtual machines V_0 and V_1. We also have two distinct containment or whole-part relationships: being placed in a virtual machine, and being part of the DB database. A virtual machine is clearly a container: it represents a set of resources dedicated to the execution of the components it contains, and it manifests a failure dependency for all the components it executes (should a virtual machine fail, the components it contains also fail). The database DB is clearly a composite: it represents the result of the composition of its parts (cache, query engine, and data store) together with their attendant connections and interaction protocols; it encapsulates the behavior of its subcomponents; and its lifetime constrains those of its parts (e.g. if the

database is destroyed so are its subcomponents). The cache component CC in this example is thus part of two wholes, the database DB and the virtual machine V_1.

Surprisingly, most formal models of computation and software architecture do not provide support for a direct modelling of containment structures with sharing. On the one hand, one finds numerous formal models of computation and of component software with strictly hierarchic structures, such as Mobile Ambients and their different variants [7, 6], the Kell calculus [25], BIP [4], Ptolemy [26], or, at more abstract level, Milner's bigraphs [20]. In bigraphs, for instance, it would be natural to model the containment relationships in our database example as instances of sub-node relationships in bigraphs, because nodes correspond to agents in a bigraph. Yet this is not possible because the sub-node relation in a bigraph is restricted to form a forest. To model the above example as a bigraph would require choosing which containment relation (placement in a virtual machine or being a subcomponent of the database) to represent by means of the sub-node relation, and to model the other relation by means of bigraph edges. This asymetry in modelling is hard to justify for both containment relations are proper examples of whole-part relationships.

On the other hand, one finds formal component models such as Reo [1], π-ADL [22], Synchronized Hyperedge Replacement (SHR) systems [14], SRM-Light [15], that represent only interaction structures among components, and not containment relationships, and models that support the modeling of non-hierarchical containment structures, but with other limitations. Our own work on the Kell calculus with sharing [16] allows to model non-hierarchical containment structures but places constraints on the dependencies that can be modelled. For instance, the lifetime dependency constraints associated with the virtual machines and the database in our example above (if the aggregate or composite dies so do its sub-components) cannot be both easily modeled. The reason is that the calculus still enforces an ownership tree between components for the purpose of passivation: components can only passivate components lower down in the tree (i.e. suspend their execution and capture their state). The formal model which comes closer to supporting non strictly hierarchical containment structures is probably CommUnity [28], where component containment is modelled as a form of superposition, and can be organized as an arbitrary graph. However, in CommUnity, possible reconfigurations in a component assemblage are described as graph transformation rules that are separate from the behavior of components, making it difficult to model reconfigurations initiated by the component assemblage itself.

To sum up, we are missing a model of computation that allows us to directly model both different forms of interactions and different forms of containment relationships between components; that supports both planned (i.e. built in component behaviors) and unplanned (i.e. induced by the environment) dynamic changes to these relationships, as well as to component behaviors.

Contribution. In this paper, we introduce a model of computation, called *G-Kells*, which meets these requirements. We develop our model at a more concrete

level than bigraph theory, but we abstract from linguistic details by developing a process calculus framework parameterized by a notion of process and certain semantical operations. Computation in our model is carried out by located processes, i.e. processes that execute at named *locations*. Locations can be nested inside one another, and a given location can be nested inside one or more locations at once. Locations constitute scopes for interactions: a set of processes can interact when situated in locations nested within the same location. Behaviors of located processes encompass interaction with other processes as well as reconfiguration actions which may change the structure of the location graph and update located processes. In addition, the G-Kells framework supports a notion of dynamic priority that few other component models support, apart from those targeting real-time and reactive systems such as BIP and Ptolemy.

Outline. The paper is organized as follows. The framework we introduce can be understood as an outgrowth of our prior work with C. Di Giusto [13], in which we proposed a process calculus interpretation of the BIP model. We recall briefly in Section 2 the main elements of this work, to help explain the extensions we introduce in our G-Kells framework. Section 3 presents the G-Kells framework and its formal operational semantics. Section 4 discusses the various features of the G-Kells framework, and related work. Section 5 concludes the paper.

2 CAB: a process calculus interpretation of BIP

One way to understand the G-Kells model we introduce in the next section, is to see it as a higher-order, dynamic extension of the CAB model [13], a process calculus interpretation of the BIP model. We recall briefly in this section the main elements of CAB.

The CAB model captures the key features of the BIP model: (i) hierarchical components; (ii) composition of components via explicit "glues" enforcing multiway synchronization constraints between sub-components; (iii) priority constraints regulating interactions among components. Just as the BIP model, the CAB model is parameterized by a family $\mathcal{P}$ of primitive behaviors. A CAB component, named l, can be either a primitive component $C = l[P]$, where P is taken from $\mathcal{P}$, or a composite component $C = l[C_1, \ldots, C_n \star G]$, built by composing a set of CAB components $\{C_1, \ldots, C_n\}$ with a glue process G. When l is the name of a component C, we write $C.\mathtt{nm} = l$. We note $\mathcal{C}$ the set of CAB components, $\mathcal{N}_l$ the set of location names, and $\mathcal{N}_c$ the set of channel names. Behaviors in $\mathcal{P}$ are defined as labelled transition systems, with labels in $\mathcal{N}_c$.

In CAB, the language for glues G is a very simple language featuring:

- *Action prefix* $\xi.G$, where ξ is an action, and G a continuation glue (in contrast to BIP, glues in CAB can be stateful).
- *Parallel composition* $G_1 \mid G_2$, where G_1 and G_2 are glues. This operator can be interpreted as an *or* operator, that gives the choice of meeting the priority and synchronization constraints of G_1 or of G_2.
- *Recursion* $\mu X.\, G$, where X is a process variable, and G a glue.

Actions embody synchronization and priority constraints that apply to subcomponents in a composition. An action ξ consists of a triplet $\langle \pi \cdot a \cdot \sigma \rangle$, where π is a priority constraint, σ is a synchronization constraint, and a is a channel name, signalling a possibility of synchronization on channel a. Priority and synchronization constraints take the same form: $\{l_i : a_i \mid i \in I\}$, where l_i are location names, and a_i are channel names. A synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ requires each sub-component l_i to be ready to synchronize on channel a_i. Note that in a synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ we expect each l_i to appear only once, i.e. for all $i, j \in I$, if $i \neq j$ then $l_i \neq l_j$. A priority constraint $\pi = \{l_i : a_i \mid i \in I\}$ ensures each subcomponent named l_i is *not* ready to synchronize on channel a_i.

Example 1. A glue G of the form $\langle \{l : a\}; c; \{l_1 : a_1, l_2 : a_2\} \rangle.G'$ specifies a synchronization between two subcomponents named l_1 and l_2: if l_1 offers a synchronization on channel a_1, and l_2 offers a synchronization on a_2, then their composition with glue G offers a synchronization on c, provided that the subcomponent named l does not offer a synchronization on a. When the synchronization on a takes place, implying l_1 and l_2 have synchronized with composite on a_1 and a_2, respectively, a new glue G' is put in place to control the behavior of the composite.

Note that the same component l can appear in both the priority and the synchronization constraint of the same action ξ.

Example 2. An action of the form $\langle \{l : a\} \cdot c \cdot \{l : b, l' : b\} \rangle$ specifies that a synchronization on c is possible provided both subcomponents l and l' offer a synchronization on b, and component l does not offer a synchronization on a.

The operational semantics of the CAB model is defined as the labeled transition system whose transition relation, $\rightarrow \subseteq \mathcal{C} \times (\mathcal{N}_l \times \mathcal{N}_c) \times \mathcal{C}$, is defined by the inference rules in Figure 2, where we use the following notations:

- $\mathbf{C}$ denotes a finite (possibly empty) set of components
- $\mathbf{C}_\sigma$ denotes the set $\{C_i \mid i \in I\}$, i.e. the set of subcomponents involved in the multiway synchronization directed by the synchronization constraint σ in rule Comp. Likewise, ${\mathbf{C}'}_\sigma$ denotes the set $\{C'_i \mid i \in I\}$.
- $\mathbf{C} \models_p \{l_i : a_i \mid i \in I\}$ denotes the fact that $\mathbf{C}$ meets the priority constraint $\pi = \{l_i : a_i \mid i \in I\}$, i.e. for all $i \in I$, there exists $C_i \in \mathbf{C}$ such that $C_i.\mathtt{nm} = l_i$ and $\neg(C_i \xrightarrow{l_i:a_i})$, meaning there are no C' such that $C_i \xrightarrow{l_i:a_i} C'$.

The Comp rule in Figure 2 relies on the transition relation between glues defined as the least relation verifying the rules in Figure 1.

The transition relation is well defined despite the presence of negative premises, for the set of rules in Figure 2 is stratified by the height of components, given by the function `height`, defined inductively as follows:

$$\mathtt{height}(l[P]) = 0 \qquad \mathtt{height}(l[\mathbf{C} \star G]) = 1 + \mathtt{max}\{\mathtt{height}(C) \mid C \in \mathbf{C}\}$$

Indeed, in rule Comp, if $\mathtt{height}(l[\mathbf{C} \star G]) = n$, then the components in $\mathbf{C}$ that appear in the premises of the rule have a maximum height of $n - 1$. The

$$\textsc{Act}\ \ \xi.G \xrightarrow{\xi} G \qquad\qquad \textsc{Rec}\ \ \frac{G\{^{\mu\, X.\, G}/_{X}\} \xrightarrow{\xi} G'}{\mu\, X.\, G \xrightarrow{\xi} G'}$$

$$\textsc{Parl}\ \ \frac{G \xrightarrow{\xi} G'}{G \mid G_2 \xrightarrow{\xi} G' \mid G_2} \qquad\qquad \textsc{Parr}\ \ \frac{G \xrightarrow{\xi} G'}{G_2 \mid G \xrightarrow{\xi} G_2 \mid G'}$$

Fig. 1. LTS semantics for CAB glues

$$\textsc{Prim}\ \ \frac{P \xrightarrow{\alpha} P'}{l[P] \xrightarrow{l:\alpha} l[P']}$$

$$\textsc{Comp}\ \ \frac{G \xrightarrow{\langle \pi \,\cdot\, a \,\cdot\, \sigma \rangle} G' \qquad \sigma = \{l_i : a_i \mid i \in I\} \qquad \forall i \in I,\ C_i \xrightarrow{l_i:a_i} C_i' \qquad \mathbf{C} \models_p \pi}{l[\mathbf{C} \star G] \xrightarrow{l:a} l[(\mathbf{C} \setminus \mathbf{C}_\sigma) \cup \mathbf{C'}_\sigma \star G']}$$

Fig. 2. LTS semantics for CAB($\mathcal{P}$)

transitions relation $\rightarrow$ is thus the transition relation associated with the rules in Figure 2 according to Definition 3.14 in [5], which is guaranteed to be a minimal and supported model of the rules in Figure 2 by Theorem 3.16 in [5].

We now give some intuition on the operational semantics of CAB. The evolution of a primitive component $C = l[P]$, is entirely determined by its primitive behavior P, following rule Prim. The evolution of a composite component $C = l[\mathbf{C}\star G]$ is directed by that of its glue G, which is given by rules Act, Parl, Parr and Rec. Note that the rules for glues do not encompass any synchronization between branches G_1 and G_2 of a parallel composition $G_1 \mid G_2$. Rule Comp specifies how glues direct the behavior of a composite (a form of superposition): if the glue G of the composite $l[\mathbf{C}\star G]$ offers action $\langle \pi \cdot a \cdot \sigma \rangle$, then the composite offers action $l : a$ if both the priority ($\mathbf{C} \models_p \pi$) and synchronization constraints are met. For the synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ to be met, there must exist subcomponents $\{C_i \mid i \in I\}$ ready to synchronize on channel a_i, i.e. such that we have, for each i, $C_i \xrightarrow{l_i:a_i} C_i'$, for some C_i'. The composite can then evolve by letting each each C_i perform its transition on channel a_i, and by letting untouched the components in $\mathbf{C}$ not involved in the synchronization (in the rule Comp, the components C_i in $\mathbf{C}$ are simply replaced by their continuation C_i' on the right hand side of the conclusion).

The CAB model is simple but already quite powerful. For instance, it was shown in [13] that CAB($\emptyset$), i.e. the instance of the CAB model with no primi-

tive components, is Turing complete[1]. Priorities are indispensable to the result, though: as shown in [13], CAB($\emptyset$) without priorities, i.e. where glue actions have empty priority constraints, can be encoded in Petri nets. We now turn to the G-Kells model itself.

3 G-Kells: a framework for location graphs

3.1 Syntax

The G-Kells process calculus framework preserves some basic features of the CAB model (named locations, actions with priority and synchronization constraints, multiway synchronization within a location) and extends it in several directions at once:

- First, we abstract away from the details of a glue language. We only require that a notion of process be defined by means of a particular kind of labelled transition system. The G-Kells framework will then be defined by a set of transition rules (the equivalent of Figure 2 for CAB) that takes as a parameter the transition relation for processes.
- We do away with the tree structure imposed by CAB for components. Instead, components will now form directed graphs between named locations, possibly representing different containment relationships among components.
- In addition, our location graphs are entirely dynamic, in the sense that they can evolve as side effects of process transitions taking place in nodes of the graphs, i.e. locations.
- CAB was essentially a pure synchronization calculus, with no values exchanged between components during synchronization. The G-Kells framework allows higher-order value passing between locations: values exchanged during synchronization can be arbitrary, including names and processes.

The syntax of G-Kells components is quite terse, and is given in Figure 3. The set of G-Kells components is called $\mathcal{K}$. Let us explain the different constructs:

- $\mathsf{0}$ stands for the null component, which does nothing.
- $l[P]$ is a *location* named l, which hosts a *process* P. As we will see below, a process P can engage in interactions with other processes hosted at other locations, but also modify the graph of locations in various ways.
- $l.r \multimap h$ denotes an *edge* in the location graph. An edge $l.r \multimap h$ connects the *role* r of a location l to another location h.
- $C_1 \parallel C_2$ stands for the parallel composition of components C_1 and C_2, which allows for the independent, as well as synchronized, evolution of C_1 and C_2.

[1] The CAB model is defined in [13] with an additional rule of evolution featuring silent actions. For simplicity, we have not included such a rule in our presentation here, but the stated results still stand for the CAB model presented in this paper.

$$C ::= \mathsf{0} \quad | \quad l[P] \quad | \quad l.r \twoheadrightarrow h \quad | \quad C \parallel C$$

$$l, h \in \mathcal{N}_l \quad r \in \mathcal{N}_r$$

Fig. 3. Syntax of G-Kells components

A role is just a point of attachment to nest a location inside another. A role r of a location l can be *bound*, meaning there exists an edge $l.r \twoheadrightarrow h$ attaching a location h to r, or unbound, meaning that there is no such edge. We say likewise that location h is bound to location l, or to a role in location l, if there exists an edge $l.r \twoheadrightarrow h$. Roles can be dynamically attached to a location, whether by the location itself or by another location. One way to understand roles is by considering a location $l[P]$ with unbound roles $r_1, \ldots, r_n$ as a frame for a composite component. To obtain the composite, one must complete the frame by binding all the unbound roles $r_1, \ldots, r_n$ to locations $l_1, \ldots, l_n$, which can be seen as subcomponents. Note that several roles of a given location can be bound to the same location, and that a location can execute with unbound roles.

Locations serve as scopes for interactions: as in CAB, interactions can only take place between a location and all the locations bound to its roles, and a location offers possible interactions as a result. Unlike bigraphs, all interactions are thus local to a given location. One can understand a location in two ways: either as a composite glue superposing, as in CAB, priority and synchronization constraints on the evolution of its subcomponents, i.e. the locations bound to it, or as a connector, providing an interaction conduit to the components it binds, i.e. the locations bound to it. More generally, one can understand intuitively a whole location graph as a component C, with unbound locations acting as external interfaces for accessing the services provided by C, locations bound to these interfaces corresponding to subcomponents of C, and unbound roles in the graph to possible places of attachment of missing subcomponents.

We do not have direct edges of the form $l \twoheadrightarrow h$ between locations to allow for processes hosted in a location, say l, to operate without knowledge of the names of locations bound to l through edges. This can be leveraged to ensure a process is kept isolated from its environment, as we discuss in Section 4. We maintain two invariants in G-Kells components: at any one point in time, for any location name l, there can be at most one location named l, and for any role r and location l, there can be at most one edge of the form $l.r \twoheadrightarrow k$.

Example 3. Let us consider how the example we discussed in the introduction can be modeled using G-Kells. Each of the different elements appearing in the configuration described (database DB, data store DS, query engine QE, cache CC, client C, virtual machines V_0 and V_1) can be modeled as locations, named accordingly. The database location has three roles s, q, c, and we have three edges $DB.s \twoheadrightarrow DS$, $DB.q \twoheadrightarrow QE$, $DB.c \twoheadrightarrow CC$, binding its three subcomponents DS, QE and CC. The virtual machines locations have two roles each, 0 and 1, and we have four edges $V_1.0 \twoheadrightarrow C$, $V_1.1 \twoheadrightarrow CC$,

$V_0.0 \twoheadrightarrow DS$, $V_0.1 \twoheadrightarrow QE$, manifesting the placement of components C, CC, DS, QE in virtual machines. Now, the database location hosts a process supporting the semantics of composition between its three subcomponents, e.g. the cache management protocol directing the interactions between the cache and the other two database subcomponents[2]. The virtual machine locations host processes supporting the failure semantics of a virtual machine, e.g. a crash failure semantics specifying that, should a virtual machine crash, the components it hosts (bound through roles 0 and 1) should crash similarly. We will see in Section 4 how this failure semantics can be captured.

3.2 Operational semantics

We now formally define the G-Kells process calculus framework operational semantics by means of a labelled transition system.

Names, values and environments

Notations. We use boldface to denote a finite set of elements of a given set. Thus if S is a set, and s a typical element of S, we write $\mathbf{s}$ to denote a finite set of elements of S, $\mathbf{s} \subseteq S$. We use ϵ to denote an empty set of elements. We write $\wp_f(S)$ for the set of finite subsets of a set S. If S_1, S_2, S are sets, we write $S_1 \uplus S_2 = S$ to mean $S_1 \cup S_2 = S$ and S_1, S_2 are disjoint, i.e. S_1 and S_2 form a partition of S. We sometimes write $s, \mathbf{s}$ to denote $\{s\} \cup \mathbf{s}$. If C is a G-Kell component, $\Gamma(C)$ represents the set of edges of C. Formally, $\Gamma(C)$ is defined by induction as follows : $\Gamma(\mathtt{0}) = \emptyset$, $\Gamma(l[P]) = \emptyset$, $\Gamma(l.r \twoheadrightarrow h) = \{l.r. \twoheadrightarrow h\}$, $\Gamma(C_1 \parallel C_2) = \Gamma(C_1) \cup \Gamma(C_2)$.

Names and Values. We use three disjoint, infinite, denumerable sets of names, namely the set $\mathcal{N}_c$ of channel names, the set $\mathcal{N}_l$ of location names, and the set $\mathcal{N}_r$ of role names. We set $\mathcal{N} = \mathcal{N}_c \cup \mathcal{N}_l \cup \mathcal{N}_r$. We note $\mathcal{P}$ the set of processes. We note $\mathcal{V}$ the set of values. We posit the existence of three functions $\mathtt{fcn} : \mathcal{V} \to \mathcal{N}_c$, $\mathtt{fln} : \mathcal{V} \to \mathcal{N}_l$ $\mathtt{frn} : \mathcal{V} \to \mathcal{N}_r$ that return, respectively, the set of free channel names, free location names, and free role names occurring in a given value. The restriction of $\mathtt{fcn}$ (resp. $\mathtt{fln}, \mathtt{frn}$) to $\mathcal{N}_c$ (resp. $\mathcal{N}_l, \mathcal{N}_r$) is defined to be the identity on $\mathcal{N}_c$ (resp. $\mathcal{N}_l, \mathcal{N}_r$). The function $\mathtt{fn} : \mathcal{V} \to \mathcal{N}$, that returns the set of free names of a given value, is defined by $\mathtt{fn}(V) = \mathtt{fcn}(V) \cup \mathtt{fln}(V) \cup \mathtt{frn}(V)$. The sets $\mathcal{N}$, $\mathcal{P}$ and $\mathcal{V}$, together with the functions $\mathtt{fcn}$, $\mathtt{fln}$, and $\mathtt{frn}$, are parameters of the G-Kells framework. We denote by $\mathcal{E}$ the set of edges, i.e. the set of triples $l.r \twoheadrightarrow h$ of $\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_l$. We stipulate that names, processes, edges and finite sets of edges are values: $\mathcal{N} \cup \mathcal{P} \cup \mathcal{E} \cup \wp_f(\mathcal{E}) \subseteq \mathcal{V}$. We require the existence of a relation $\mathtt{match} \subseteq \mathcal{V}^2$, used in ascertaining possible synchronization.

[2] Notice that the database location does not run inside any virtual machine. This means that, at this level of abstraction, our *process* architectural view of the database composite is similar to a network connecting the components placed in the two virtual machines.

Environments. Our operational semantics uses a notion of *environment*. An environment Γ is just a subset of $\mathcal{N} \cup \mathcal{E}$, i.e. a set of names and a set of edges. The set of names in an environment represents the set of *already used* names in a given context. The set of edges in an environment represents the set of edges of a location graph. the set of environments is noted $\mathcal{G}$.

Processes

Process transitions. We require the set of processes to be equipped with a transition system semantics given by a labelled transition system where transitions are of the following form:

$$P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'$$

The label $\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle$ comprises four elements: a priority constraint π (an element of Pr), an offered interaction α, a synchronization constraint σ (an element of S), and an effect ω (an element of A). The first three are similar in purpose to those in CAB glues. The last one, ω, embodies queries and modifications of the surrounding location graph.

An offered interaction α takes the form $\{a_i\langle V_i \rangle \mid i \in I\}$, where I is a finite index set, a_i are channel names, and V_i are values.

Evalution functions. We require the existence of evaluation functions on priority constraints ($\texttt{eval}_\pi$), and on synchronization constraints ($\texttt{eval}_\sigma$) of the following types $\texttt{eval}_\pi : \mathcal{G} \times \mathcal{N}_l \times \mathsf{Pr} \to \wp_f(\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_c)$, and $\texttt{eval}_\sigma : \mathcal{G} \times \mathcal{N}_l \times \mathsf{S} \to \wp_f(\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_c \times \mathcal{V})$. The results of the above evaluation functions are called *concrete (priority or synchronization) constraints*. The presence of evaluation functions allows us to abstract away from the actual labels used in the semantics of processes, and to allow labels used in the operational semantics of location graphs, described below, to depend on the environment and the surrounding location graph.

Example 4. One can, for instance, imagine a kind of broadcast synchronization constraint of the form $* : a\langle V \rangle$, which, in the context of an environment Γ and a location l, evaluates to a constraint requiring all the roles bound to a locations in Γ to offer an interaction on channel a, i.e.: $\texttt{eval}_\sigma(\Gamma, l, * : a\langle V \rangle) = \{l.r : a\langle V \rangle \mid \exists h,\, l.r \twoheadrightarrow h \in \Gamma\}$.

We require the existence of an evaluation function on effects ($\texttt{eval}_\omega$) with the type $\texttt{eval}_\omega : \mathcal{G} \times \mathcal{N}_l \times \mathsf{A} \to \wp_f(\mathsf{E})$, where E is the set of *concrete effects*. A concrete effect can take any of the following forms, where l is a location name:

- $l : \texttt{newl}(h, P)$, $l : \texttt{newch}(c)$, $l : \texttt{newr}(r)$, respectively to create a new location named h with initial process P, to create a new channel named c, and to create a new role named r.
- $l : \texttt{add}(h, r, k)$, $l : \texttt{rmv}(h, r, k)$, respectively to add and remove a graph edge $h.r \twoheadrightarrow k$ to and from the surrounding location graph.
- $l : \texttt{gquery}(\Gamma)$, to discover a subgraph Γ of the surrounding location graph.
- $l : \texttt{swap}(h, P, Q)$, to swap the process P running at location h for process Q.

– $l : \texttt{kill}(h)$, to remove location h from the surrounding location graph.

Concrete effects embody the reconfiguration capabilities of the G-Kells framework. Effects reconfiguring the graph itself come in pairs manifesting introduction and elimination effects: thus, adding and removing a node (location) from the graph ($\texttt{newl}, \texttt{kill}$), and adding and removing an edge from the graph ($\texttt{add}, \texttt{rmv}$). Role creation ($\texttt{newr}$) is introduced to allow the creation of new edges. Channel creation ($\texttt{newch}$) allows the same flexibility provided by name creation in the π-calculus. The swap effect ($\texttt{swap}$) is introduced to allow the atomic update of a located process. The graph query effect ($\texttt{gquery}$) is perhaps a bit more unorthodox: it allows a form of reflection whereby a location can discover part of its surrounding location graph. It is best understood as an abstraction of graph navigation and query capabilities which have been found useful for programming reconfigurations in component-based systems [11].

Operational semantics of location graphs

Transitions. The operational semantics of location graphs is defined by means of a transition system, whose transition relation $\rightarrow$ is defined, by means of the inference rules presented below, as a subset of $\mathcal{G} \times \mathcal{K} \times \mathcal{L} \times \mathcal{K}$. Labels take the form $\langle \pi \cdot \sigma \cdot \omega \rangle$, where π and σ are *located priority and synchronization constraints*, respectively, and ω is a finite set of *located effects* (for simplicity, we reuse the same symbols than for constraints and effects). The set of labels is noted $\mathcal{L}$.

A located priority constraint π is just a concrete priority constraint, and takes the form $\{l_i.r_i : a_i \mid i \in I\}$, where I is a finite index set, with l_i, r_i, a_i in location, role and channel names, respectively. A located synchronization constraint takes the form $\{u_j : a_j\langle V_j \rangle\}$, where a_j are channel names, u_j are either location names l_j or pairs of location names and roles, noted $l_j.r_j$, and V_j are values. Located effects can take the following forms: $l : \texttt{rmv}(h, r, k)$, $l : \texttt{swap}(h, P, Q)$, $l : \texttt{kill}(h)$, $h : \overline{\texttt{rmv}}(h, r, k)$, $h : \overline{\texttt{swap}}(h, P, Q)$, and $h : \overline{\texttt{kill}}(h)$, where l, h, k are location names, r is a role name, P, Q are processes. The predicate $\texttt{located}$ on E identifies located effects. In particular, for a set $\omega \subset \mathsf{E}$, we have $\texttt{located}(\omega)$ if and only if all elements of ω are located.

A transition $\langle \Gamma, C, \langle \pi \cdot \sigma \cdot \omega \rangle, C' \rangle \in \rightarrow$ is noted $\Gamma \triangleright C \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C'$ and must obey the folowing constraint:

– If $\Gamma = \mathbf{u} \cup \Delta$, where $\mathbf{u}$ is a set of names, and Δ is a set of edges, then $\texttt{fn}(\Delta) \subseteq \mathbf{u}$ and $\texttt{fn}(\Gamma) = \mathbf{u}$.
– $\texttt{fn}(C) \subseteq \texttt{fn}(\Gamma)$, i.e. the free names occurring in C must appear in the already used names of Γ.
– If $l.r \twoheadrightarrow h$ and $l.r \twoheadrightarrow k$ are in $\Gamma \cup \Gamma(C)$, then $h = k$, i.e. we only have a single edge binding a role r of a given location l to another location h.

Auxiliary relation $\rightarrow_\bullet$. The definition of $\rightarrow$ makes use of an auxiliary relation $\rightarrow_\bullet \subseteq \mathcal{G} \times \mathcal{K} \times \mathcal{L}_\bullet \times \mathcal{K}$, where $\mathcal{L}_\bullet$ is a set of elements of the form $\langle \pi \cdot \sigma \cdot \omega \rangle$ where π and σ are located priority and synchronizaton constraints, and where ω is a finite

set of concrete effects. Relation $\rightarrow_\bullet$ is defined as the least relation satisfying the rules in Figure 4, where we use the following notation: if $\alpha = \{a_i\langle V_i\rangle \mid i \in I\}$, then $l : \alpha = \{l : a_i\langle V_i\rangle \mid i \in I\}$, and if $\alpha = \epsilon$, then $l : \alpha = \epsilon$.

Rule Act simply expresses how process transitions are translated into location graph transitions, and process level constraints and effects are translated into located constraints and concrete effects, via the evaluation functions introduced above. Rule NewL specifies the effect of an effect $l : \texttt{newl}(h, Q)$, which creates a new location $h[Q]$. Notice how effect $l : \texttt{newl}(h, Q)$ is removed from the set of effects in the transition label in the conclusion of the rule: auxiliary relation $\rightarrow_\bullet$ is in fact used to guarantee that a whole set of concrete effects are handled atomically. All the rules in Figure 4 except Act, which just prepares for the evaluation of effects, follow the same pattern. Rules NewC and NewR specify the creation of a new channel name and of a new role name, respectively. The rules just introduce the constraint that the new name must not be an already used name in the environment Γ. Our transitions being in the early style, the use of the new name is already taken into account in the continuation location graph C (in fact in the continuation process P' appearing on the left hand side of the instance of rule Act that must have led to the building of C). This handling of new names is a bit unorthodox but it squares nicely with the explicitly indexed labelled transition semantics of the π-calculus given by Cattani and Sewell in [8]. Rule AddE specifies the effect of adding a new edge to the location graph. Rule Gquery allows the discovery by processes of a subgraph of the location graph. In the rule premise, we use the notation Γ_l to denote the set of edges reachable from location l, formally: $\Gamma_L = \{h.r \twoheadrightarrow k \in \Gamma \mid l \twoheadrightarrow^+_\Gamma h\}$, where $l \twoheadrightarrow^+_\Gamma h$ means that there exists a non-empty chain $l.r \twoheadrightarrow l_1$, $l_1.r_1 \twoheadrightarrow l_2, \ldots, l_{n-1}.r_{n-1} \twoheadrightarrow l_n$, $l_n.r_n \twoheadrightarrow h$, with $n \geq 1$, linking l to h in the location graph Γ. As in the case of name creation rules, the exact effect on processes is left unspecified, the only constraint being that the discovered graph be indeed a subgraph of the location graph in the environment.

Transition relation $\rightarrow$. The transition relation $\rightarrow$ is defined by the rules in Figure 5. We use the following notations and definitions in Figure 5:

- Function $\texttt{seval}$ is defined by induction as follows (for any $l, r, h, V, \sigma, \sigma'$):

$$\begin{array}{ll} \texttt{seval}(\Gamma, \sigma) = \texttt{seval}(\Gamma, \sigma') & \textit{if } \sigma = \{l.r : a\langle V\rangle, h : a\langle W\rangle\} \cup \sigma' \\ & \quad \wedge\, l.r \twoheadrightarrow h \in \Gamma \wedge \texttt{match}(V, W) \\ \texttt{seval}(\Gamma, \sigma) = \sigma & \textit{otherwise} \end{array}$$

- Function $\texttt{aeval}$ is defined by induction as follows (for any $l, r, h, k, P, Q, \sigma, \sigma'$):

$$\begin{array}{ll} \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{swap}(h, P, Q), \overline{\texttt{swap}}(h, P, Q)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{rmv}(h, r, k), \overline{\texttt{rmv}}(h, r, k)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{kill}(h), \overline{\texttt{kill}}(h)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \sigma & \textit{otherwise} \end{array}$$

$$\textsc{Act}_\bullet \quad \frac{P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P' \qquad \pi_\bullet = \mathtt{eval}_\pi(\Gamma, l, \pi) \qquad \sigma_\bullet = \mathtt{eval}_\sigma(\Gamma, l, \sigma) \qquad \omega_\bullet = \mathtt{eval}_\omega(\Gamma, l, \omega)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi_\bullet \cdot l:\alpha \cup \sigma_\bullet \cdot \omega_\bullet \rangle}_\bullet l[P']}$$

$$\textsc{NewL} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newl}(h,Q), \omega \rangle}_\bullet C \qquad h \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \parallel h[Q]}$$

$$\textsc{NewC} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newch}(c), \omega \rangle}_\bullet C \qquad c \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

$$\textsc{NewR} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newr}(r), \omega \rangle}_\bullet C \qquad r \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

$$\textsc{AddE} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{add}(h,r,k), \omega \rangle}_\bullet C \qquad \neg(\exists k,\, h.r \rightarrowtriangle k \in \Gamma)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \parallel h.r \rightarrowtriangle k}$$

$$\textsc{GQuery} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{gquery}(\Delta), \omega \rangle}_\bullet C \qquad \Delta \subseteq \Gamma_l}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

Fig. 4. Rules for auxiliary relation $\rightarrow_\bullet$

$$\textsc{Swap} \quad \Gamma \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{swap}}(l,P,Q) \rangle} l[Q] \qquad\qquad \textsc{Kill} \quad \Gamma \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{kill}}(l) \rangle} \mathsf{0}$$

$$\textsc{Rmv} \quad \Gamma \triangleright h.r \rightarrowtriangle k \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{rmv}}(h,r,k) \rangle} \mathsf{0}$$

$$\textsc{Act} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \qquad \mathtt{located}(\omega)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C}$$

$$\textsc{Comp} \quad \frac{\begin{array}{c}\Gamma' = \Gamma \cup \Gamma(C_1) \cup \Gamma(C_2) \qquad \pi = \pi_1 \cup \pi_2 \qquad \mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N} \\ \Gamma' \triangleright (\!|C_1 \parallel C_2|\!)_\pi \models \pi \qquad \sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2) \qquad \omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2) \\ \Gamma' \cup \Delta_1 \triangleright C_1 \xrightarrow{\langle \pi_1 \cdot \sigma_1 \cdot \omega_1 \rangle} C_1' \qquad \Gamma' \cup \Delta_2 \triangleright C_2 \xrightarrow{\langle \pi_2 \cdot \sigma_2 \cdot \omega_2 \rangle} C_2'\end{array}}{\Gamma \triangleright C_1 \parallel C_2 \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C_1' \parallel C_2'}$$

Fig. 5. Rules for transition relation $\rightarrow$

– Assume $\pi = \{l_i.r_i : a_i \mid i \in I\}$, then $(\!|C|\!)_\pi$ is obtained by replacing in C all the locations $\{l_i[P_i] \mid i \in I\}$ with locations $\{l_i[(\!|P_i|\!)] \mid i \in I\}$, where $(\!|P|\!)$ is defined by the LTS obtained from that of P by the following rule:

$$\textsc{Trim}\ \frac{P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'}{(\!|P|\!) \xrightarrow{\langle \epsilon \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'}$$

  In other terms, $(\!|C|\!)_\pi$ is obtained by disregarding the priority constraints that are generated by locations l_i mentioned in the priority constraint π.
– $\Gamma \triangleright C \models \{l_i.r_i : a_i \mid i \in I\}$ means that, for all $i \in I$, $\Gamma \triangleright C \models l_i.r_i : a_i$. We write $\Gamma \triangleright C \models l.r : a$ to mean that

$$\neg(\exists h, \pi, \sigma, \omega, V, C',\ l.r \twoheadrightarrow h \ \wedge\ \Gamma \triangleright C \xrightarrow{\langle \pi \cdot l:a\langle V \rangle, \sigma \cdot \omega \rangle} C')$$

Rule Swap specifies that at any point in time a location can see its process swapped for another. Likewise, rules Kill and Rmv specify that at any point in time a location or an edge, respectively, can be removed from a location graph. Rule Act specifies the termination of the atomic execution of a set of concrete effects. All the effects remaining must be located effects, which expect some counterpart (provided by the rules Swap, Kill, and Rmv, when invoking rule Comp) to proceed.

Rule Comp is the workhorse of our operational semantics of location graphs. It specifies how to determine the transitions of a parallel composition $C_1 \parallel C_2$, by combining the priority constraints, synchronization constraints and effects obtained from the determination of contributing transitions from C_1 and C_2. The latter takes place in extended environment Γ', that contains the original environment Γ, but also the edges present in C_1 and C_2, defined as $\Gamma(C_1)$ and $\Gamma(C_2)$, respectively. To ensure the names created as side effects of C_1 and C_2 transitions are indeed unique, the determination of the contributing transition of C_1 takes place in an environment where the already used names include those in Γ as well as those in Δ_1, which gathers names that may be created as a side effect of the contributing transition of C_2. Likewise for determining the contributing transition of C_2. The constraint $\mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N}$ ensures that names in Δ_1 and Δ_2 are disjoint, as well as disjoint from the already used names of $\mathtt{fn}(\Gamma')$.

The original aspect of rule Comp lies with the computation of located synchronization contraints and effects resulting from the parallel composition of G-Kells components: we allow it to be dependent on the global environment Γ' with the clauses $\sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2)$ and $\omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2)$, which, in turn, allows to enforce constraints dependent on the location graph as in the definition of function `seval`. In fact, as we discuss in Section 4 below, we can envisage instances of the framework where different types of location-graph-dependent constraints apply.

The use of environments in rule Comp to obtain a quasi-compositional rule of evolution is reminiscent of the use of environment and process *frames* in the parallel rule of the ψ-calculus framework [3]. We say our rule Comp is *quasi-compositional* for the handling of priority is not compositional: it relies on the

global condition $\Gamma' \triangleright (\!|C_1 \parallel C_2|\!)_\pi$, which requires computing with (an altered view of) the whole composition. The use of the $(\!|\cdot|\!)_\_$ operator in rule Comp is reminiscent of the handling of priorities in other works [9]. An easy way to turn rule Comp into a completely compositional rule would be to adopt a more "syntactic" approach, defining $(\!|C|\!)_\pi$ to be obtained by replacing *all* locations $l[P]$ in C by their trimmed version $l[(\!|P|\!)]$, and defining environments to include information on possible actions by trimmed locations. On the other hand, we can also adopt a purely "semantic" (albeit non-compositional) variant by defining $(\!|C|\!)_\pi$ to be just C. However, in this case, we don't know whether the completeness result in Theorem 1 below still stands.

Example 5. To illustrate how the rules work, consider the following process transitions:

$$P \xrightarrow{\langle \epsilon \cdot \epsilon \cdot *:a\langle V\rangle \cdot \mathtt{newl}(h,P)\rangle} P' \qquad P_1 \xrightarrow{\langle \epsilon \cdot a\langle V_1\rangle \cdot \epsilon \cdot \{\mathtt{newl}(h_1,P_1)\}\rangle} P_1' \quad P_2 \xrightarrow{\langle \epsilon \cdot a\langle V_2\rangle \cdot \epsilon \cdot \epsilon\rangle} P_2'$$

Let $\Gamma = \{l, l_1, l_2\} \cup \{l.r_1 \twoheadrightarrow l_1, l.r_2 \twoheadrightarrow l_2\}$, and let Δ, Δ' be such that $\Delta \uplus \Delta' = \mathcal{N} \setminus \{l, l_1, l_2, h, h_1\}$. We assume further that $\mathtt{match}(V, V_1)$ and $\mathtt{match}(V, V_2)$, that all names l, l_1, l_2, h, h_1 are distinct, and that

$$\begin{aligned}
&\mathtt{eval}_\sigma(\Gamma, l, * : a\langle V\rangle) = \{l.r_1 : a\langle V\rangle, l.r_2 : a\langle V\rangle\}\\
&\mathtt{eval}_\omega(\Gamma, l, \{\mathtt{newl}(h,P)\}) = \{l : \mathtt{newl}(h,P)\}\\
&\mathtt{eval}_\omega(\Gamma, l_1, \{\mathtt{newl}(h_1,P_1)\}) = \{l_1 : \mathtt{newl}(h_1,P_1)\}\\
&\mathtt{eval}_\omega(\Gamma, l_2, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l_1, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l_2, \epsilon) = \epsilon
\end{aligned}$$

Applying rule Act$_\bullet$, we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \{l:\mathtt{newl}(h,P)\}\rangle}_\bullet l[P']\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot l_1:a\langle V_1\rangle \cdot \{l_1:\mathtt{newl}(h_1,P_1)\}\rangle}_\bullet l_1[P_1']\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_2[P_2] \xrightarrow{\langle \epsilon \cdot l_2:a\langle V_2\rangle \cdot \epsilon\rangle}_\bullet l_2[P_2']
\end{aligned}$$

Applying rule NewL, we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \epsilon\rangle}_\bullet l[P'] \parallel h[P]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot \{h_1:a\langle V_1\rangle\} \cdot \epsilon\rangle}_\bullet l_1[P_1'] \parallel h_1[P_1]
\end{aligned}$$

Applying rule Act we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \epsilon\rangle} l[P'] \parallel h[P]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot \{l_1:a\langle V_1\rangle\} \cdot \epsilon\rangle} l_1[P_1'] \parallel h_1[P_1]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_2[P_2] \xrightarrow{\langle \epsilon \cdot \{l_2:a\langle V_2\rangle\} \cdot \epsilon\rangle} l_2[P_2']
\end{aligned}$$

Finally, applying rule Comp we get

$$\begin{aligned}
\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \parallel l_2[P_2] &\xrightarrow{\langle \epsilon \cdot \{l_1:a\langle V_1\rangle, l_2:a\langle V_2\rangle\} \cdot \epsilon\rangle} l_1[P_1'] \parallel h_1[P_1] \parallel l_2[P_2']\\
\Gamma \triangleright l[P] \parallel l_1[P_1] \parallel l_2[P_2] &\xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon\rangle} l[P'] \parallel h[P] \parallel l_1[P_1'] \parallel h_1[P_1] \parallel l_2[P_2']
\end{aligned}$$

Transition relation $\rightarrow$ as a fixpoint. Because of the format of rule Comp, which does not *prima facie* obey known SOS rule formats [21], or conform to the standard notion of transition system specification [5], the question remains of which relation the rules in Figures 4 and 5 define. Instead of trying to turn our rules into equivalent rules in an appropriate format, we answer this question directly, by providing a fixpoint definition for $\rightarrow$. We use the fixpoint construction introduced by Przymusinski for the three-valued semantics of logic programs [23].

Let $\sqsubseteq$ be the ordering on pairs of relations in $\mathcal{T} = \mathcal{G} \times \mathcal{K} \times \mathcal{L} \times \mathcal{K}$ defined as:

$$\langle R_1, R_2 \rangle \sqsubseteq \langle T_1, T_2 \rangle \iff R_1 \subseteq T_1 \wedge T_2 \subseteq R_2$$

As products of complete lattices, $(\mathcal{T}, \subseteq)$ and $(\mathcal{T}^2, \sqsubseteq)$ are complete lattices [10]. One can read the Comp rule in Figures 5 as the definition of an operator $\mathcal{F} : \mathcal{T}^2 \rightarrow \mathcal{T}^2$ which operates on pairs of sub and over-approximations of $\rightarrow$. Let $\rightarrow_0$ be the relation in $\mathcal{T}^2$ obtained as the least relation satisfying the rules in Figures 4 and 5, with rule Comp omitted. Operator $\mathcal{F}$ is then defined as follows:

$$\begin{aligned}
\mathcal{F}(R_1, R_2) &= (\rightarrow_0 \cup R_1 \cup \mathsf{r}(\rightarrow_0 \cup R_1, R_2), R_2 \cap \mathsf{r}(R_2, \rightarrow_0 \cup R_1)) \\
\mathsf{r}(R_1, R_2) &= \{t \in \mathcal{T} \mid t = (\Gamma, C_1 \parallel C_2, \langle \pi \cdot \sigma \cdot \omega \rangle, C_1' \parallel C_2') \\
&\qquad \wedge \mathtt{comp}(\Gamma, C_1, C_2, \pi, \sigma, \omega, C_1', C_2', R_1, R_2)\}
\end{aligned}$$

where the predicate `comp` is defined as follows:

$$\begin{aligned}
\mathtt{comp}(\Gamma, C_1, C_2, \pi, \sigma, \omega, C_1', C_2', R_1, R_2) \iff & \\
& \exists \pi_1, \pi_2, \sigma_1, \sigma_2, \omega_1, \omega_2, \Delta_1, \Delta_2, \Gamma', \\
& \Gamma' = \Gamma \cup \Gamma(C_1) \cup \Gamma(C_2) \\
& \wedge \pi = \pi_1 \cup \pi_2 \\
& \wedge \mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N} \\
& \wedge \sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2) \\
& \wedge \omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2) \\
& \wedge (\Gamma' \cup \Delta_1, C_1, \langle \pi_1 \cdot \sigma_1 \cdot \omega_1 \rangle, C_1') \in R_1 \\
& \wedge (\Gamma' \cup \Delta_2, C_2, \langle \pi_2 \cdot \sigma_2 \cdot \omega_2 \rangle, C_2') \in R_1 \\
& \wedge \Gamma' \triangleright (\!| C_1 \parallel C_2 |\!)_\pi \models_{R_2} \pi
\end{aligned}$$

where $\Gamma \triangleright C \models_R \{l_i : a_i \mid i \in I\}$ means that, for all $i \in I$, $\Gamma \triangleright C \models_R l_i : a_i$, and where $\Gamma \triangleright C \models_R l : a$ stands for:

$$\neg(\exists \pi, \sigma, \omega, V, C', (\Gamma, C, \langle \pi \cdot \{l : a\langle V \rangle\} \cup \sigma \cdot \omega \rangle, C') \in R)$$

The definition of the predicate `comp` mimics the definition of rule Comp, with all the conditions in the premises appearing as clauses in `comp`, but where the positive transition conditions in the premises are replaced by transitions in the sub-approximation R_1, and negative transition conditions (appearing in the $\Gamma' \triangleright (\!| C_1 \parallel C_2 |\!)_\pi \models \pi$ condition) are replaced by equivalent conditions with transitions not belonging to the over-approximation R_2. With the definitions above, if R_1 is a sub-approximation of $\rightarrow$, and R_2 is an over-approximation of

$\rightarrow$, then we have $R_1 \subseteq \pi_1(\mathcal{F}(R_1, R_2))$ and $\pi_2(\mathcal{F}(R_1, R_2)) \subseteq R_2$, where π_1, π_2 are the first and second projections. In other terms, given a pair of sub and over approximations of $\rightarrow$, $\mathcal{F}$ computes a pair of better approximations.

From this definition, if it is easy to show that $\mathcal{F}$ is order-preserving:

Lemma 1. *For all $R_1, T_1, R_2, T_2 \in \mathcal{T}^2$, if $(R_1, R_2) \sqsubseteq (T_1, T_2)$, then $\mathcal{F}(R_1, R_2) \sqsubseteq \mathcal{F}(T_1, T_2)$.*

Since $\mathcal{F}$ is order-preserving, it has a least fixpoint, $\mathcal{F}_* = (D, U)$, by the Knaster-Tarski theorem. We can then define $\rightarrow$ to be the first projection of $\mathcal{F}_*$, namely D. With the definition of $(\!|C|\!)_\pi$ we have adopted, and noting that it provides a form of stratification with the number of locations in a location graph with non-empty priority constraints, it is also possible to show that $\rightarrow = U$, meaning that $\rightarrow$ as just defined is *complete*, using the terminology in [27]. In fact, using the terminology in [27], we can prove the theorem below, whose proof we omit for lack of space:

Theorem 1. *The relation $\rightarrow$ as defined above is the least well-supported model of the rules in Figures 4 and 5. Moreover $\rightarrow$ is complete.*

4 Discussion

We discuss in this section the various features of the G-Kells framework and relevant related work.

Introductory example. Let's first revisit Example 3 to see how we can further model the behavior of its different components. We can add, for instance, a crash action to the virtual machine locations, which can be triggered by a process transition at a virtual machine location of the form $P \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon \cdot \mathtt{kill}(*) \rangle} \mathsf{0}$ with the following evaluation function:

$$\mathtt{eval}_\omega(\Gamma, l, \mathtt{kill}(*)) = \{\mathtt{kill}(h) \mid \exists r,\, l.r \twoheadrightarrow h \in \Gamma\}$$

yielding, for instance, the following transition (where $\mathbf{u}$ includes all free names in $V_0[P] \parallel C[P_C] \parallel CC[P_{CC}]$):

$$\mathbf{u}, \{V_0.0 \twoheadrightarrow C, V_0.1 \twoheadrightarrow CC\} \triangleright V_0[P] \parallel C[P_C] \parallel CC[P_{CC}] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon \rangle} V_0[\mathsf{0}] \parallel \mathsf{0} \parallel \mathsf{0}$$

This crash behavior can be extended to an arbitrary location graph residing in a virtual machine, by first discovering the location graph inside a virtual machine via the `gquery` primitive, and then killing all locations in the graph as illustrated above.

Early style. Our operational semantics for location graphs is specified in an early style [24], with values in labels manifesting the results of successful communication. This allows us to remain oblivious to the actual forms of synchronization used. For instance, one could envisage pattern matching as in the ψ-calculus [3],

$$[\![l[P]]\!] = l[P]$$
$$[\![l[C_1, \ldots, C_n \star G]]\!] = l_1[C_1] \parallel l.1 \rightarrowtriangle l_1 \parallel \ldots \parallel l_n[C_n] \parallel l.n \rightarrowtriangle l_n \parallel l[[\![G]\!]]$$
$$[\![\mathsf{0}]\!] = \mathsf{0}$$
$$[\![\langle \pi \cdot l : a \cdot \sigma \rangle.G]\!] = \langle [\![\pi]\!] \cdot \{l : a\} \cdot [\![\sigma]\!] \rangle.[\![G]\!]$$
$$[\![G_1 \mid G_2]\!] = [\![G_1]\!] \mid [\![G_2]\!]$$
$$[\![\mu X.\, G]\!] = \mu X.\, [\![G]\!]$$
$$[\![\{l_i : a_i \mid i \in I\}]\!] = \{i : a_i \mid i \in I\}$$

$$\mathtt{eval}_\pi(\Gamma, l, i : a_i) = l.i : a_i \qquad \mathtt{eval}_\sigma(\Gamma, l, i : a_i) = l.i : a_i$$

Fig. 6. Encoding CAB in the G-Kells framework

or even bi-directional pattern matching: for instance we could have a process synchronization constraint $r : a\langle x, V\rangle$ matching an offered interaction $h : a\langle W, y\rangle$, which translate into matching located synchronization constraints $l.r : a\langle W, V\rangle$ and $h : a\langle W, V\rangle$.

Mobility vs higher-order. Our operational semantics comprises both mobility features with location binding, and higher-order features with swapping and higher-order interactions. One could wonder whether these features are all needed as primitives. For instance, one could argue that mobility features are enough to model higher-order phenomena as in the π-calculus [24]. Lacking at this point a behavioral theory for the G-Kells framework, we cannot answer the question definitely here. But we doubt that mobility via location binding is sufficient to faithfully encode higher-order communication. In particular, note that we have contexts (location graphs) that can distinguish the two cases via the ability to kill locations selectively.

Directed graphs vs acyclic directed graphs. Location graphs form directed graphs. One could wonder whether to impose the additional constraints that such graphs be acyclic. While most meaningul examples of ubiquitous systems and software structures can be modeled with acyclic directed graphs, our rules for location graphs function readily with arbitrary graphs. Enforcing the constraint that all evolutions of a location graph keep it acyclic does not seem necessary.

Relationship with CAB. The G-Kells model constitutes a conservative extension of CAB. A straightforwward encoding of CAB in the G-Kells framework can be defined as in Figure 6, with translated glues $[\![G]\!]$ defined with the same LTS, mutatis mutandis, as CAB glues. The following proposition is then an easy consequence of our definitions:

Proposition 1. *Let C be a CAB component. We have $C \xrightarrow{l:a} C'$ if and only if $[\![C]\!] \xrightarrow{\langle \epsilon \cdot \{l:a\} \cdot \epsilon \rangle} [\![C']\!]$.*

Graph constraints in rules. For simplicity, the evaluation functions `seval` and `aeval` have been defined above with only a simple graph constraint in the first clause of the `seval` definition. One can parameterize these definitions with additional graph constraints to enforce different policies. For instance, one could constrain the use of the swap, kill and and edge removal operations to locations dominating the target location by adding a constraint of the form $l \rightarrowtriangle^* h$ to each of the clauses in the definition of `aeval`, where $l \rightarrowtriangle^*_\Gamma h$ means that there exists a (possibly empty) chain $l.r \rightarrowtriangle l_1$, $l_1.r_1 \rightarrowtriangle l_2, \ldots, l_{n-1}.r_{n-1} \rightarrowtriangle l_n$, $l_n.r_n \rightarrowtriangle h$ linking l to h in the location graph Γ. Similar constraints could be added to rule AddE. Further constraints could be added to rule Gquery to further restrict the discovery of subgraphs, for instance, preventing nodes other than immediate children to be discovered.

Types and capabilities. The framework presented in this paper is an untyped one. However, introducing types similar to i/o types capabilities in the π-calculus [24] would be highly useful. For instance, edges of a location graph can be typed, perhaps with as simple a scheme as different colors to reflect different containment and visibility semantics, which can be exploited in the definition of evaluation functions to constrain effects and synchronization. In addition, location, role and channel names can be typed with capabilities constraining the transfer of rights from one location to another. For instance, transferring a location name l can come with the right to swap the behavior at l, but not with the right to kill l, or with the right to bind roles of l to locations, but not with the ability to swap the behavior at l. We believe these capabilities could be useful in enforcing encapsulation and access control policies.

Relation with the ψ-calculus framework and SCEL. We already remarked that our use of environments is reminiscent of the use of frames in the ψ-calculus framework [3]. An important difference with the ψ-calculus framework is the fact that we allow interactions to depend on constraints involving the global environment, in our case the structure of the location graph. Whether one can faithfully encode the G-Kells framework (with mild linguistic assumptions on processes) with the ψ-calculus framework remains to be seen.

On the other hand, it would seem worthwhile to pursue the extension of the framework presented here with ψ-calculus-like assertions. We wonder in particular what relation the resulting framework would have with the SCEL language for autonomous and adaptive systems [12]. The notion of *ensemble*, being assertion-based, is more fluid than our notion of location graph, but it does not have the ability to superimpose on ensembles the kind of control actions, such as swapping and killing, that the G-Kells framework allows.

Relation with SHR. The graph manipulation capabilities embedded in the G-Kells framework are reminiscent of synchronized hyperdege replacement (SHR) systems [18]. In SHR, multiple hyperedge replacements can be synchronized to yield an atomic transformation of the underlying hypergraph in conjunction with information exchange. Intuitively, it seems one can achieve much the same effects

with G-Kells: located effects can atomically build a new subgraph and modify the existing one, and they can be synchronized across multiple locations thanks to synchronization constraints. In contrast, SHR systems lack priorities and the internalization of hyperedge replacement rules (the equivalent of our processes) in graph nodes to account for inherent dynamic reconfiguration capabilities. We conjecture that SHR systems can be faithfully encoded in the G-Kells framework.

5 Conclusion

We have introduced the G-Kells framework to lift limitations in existing computational models for ubquitous and reconfigurable software systems, in particular the ability to describe dynamic structures with sharing, where different aggregates or composites can share components. Much work remains to be done, however. We first intend to develop the behavioral theory of our framework. Indeed we hope to develop a first-order bisimulation theory for the G-Kells framework, avoiding the difficulties inherent in mixing higher-order features with passivation described in [19]. We also need to formally compare G-Kells with several other formalisms, including SHR systems and the ψ-calculus framework. And we definitely need to develop a typed variant of the framework to exploit the rich set of capabilities that can be attached to location names.

Ackowledgements

Damien Pous suggested the move to an early style semantics. The paper was much improved thanks to comments by Ivan Lanese on earlier versions.

References

1. C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2), 2006.
2. F. Barbier, B. Henderson-Sellers, A. Le Parc, and J.M. Bruel. Formalization of the whole-part relationship in the unified modeling language. *IEEE Trans. Software Eng.*, 29(5), 2003.
3. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.
4. S. Bliudze and J. Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR*, volume 5201 of *LNCS*. Springer, 2008.
5. R. N. Bol and J. F. Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996.
6. M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: the calculus of boxed ambients. *ACM. Trans. Prog. Languages and Systems*, 26(1), 2004.
7. L. Cardelli and A. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1), 2000.

8. G. L. Cattani and P. Sewell. Models for name-passing processes: interleaving and causal. *Information and Computation*, 190(2), 2004.
9. R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In *Handbook of Process Algebra*. Elsevier, 2001.
10. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2nd ed.)*. Cambridge University Press, 2002.
11. P.C. David, T. Ledoux, M. Léger, and T. Coupaye. Fpath and fscript: Language support for navigation and reliable reconfiguration of fractal architectures. *Annals of Telecommunications*, 64(1-2), 2009.
12. R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *ACM Trans. on Autonomous and Adaptive Systems*, 9(2), 2014.
13. C. Di Giusto and J.-B. Stefani. Revisiting glues for component-based systems. In *COORDINATION 2011*, volume 6721 of *LNCS*. Springer, 2011.
14. G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, and E. Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *FMCO 2005*, volume 4111 of *LNCS*. Springer, 2005.
15. J. L. Fiadeiro and A. Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and System Modeling*, 12(2), 2013.
16. D. Hirschkoff, T. Hirschowitz, D. Pous, A. Schmitt, and J.-B. Stefani. Component-Oriented Programming with Sharing: Containment is not Ownership. In *GPCE 2005*, volume 3676 of *LNCS*. Springer, 2005.
17. P. Kruchten. Architectural blueprints – The 4+1 view model of software architecture. *IEEE Software*, 12(6), 1995.
18. I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Theory and Practice of Logic Programming*, 7(1-2), 2007.
19. S. Lenglet, A. Schmitt, and J.B. Stefani. Characterizing contextual equivalence in calculi with passivation. *Information and Computation*, 209(11), 2011.
20. R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.
21. M. R. Mousavi, M. A. Reniers, and J. F. Groote. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3), 2007.
22. F. Oquendo. π-ADL: An Architecture Description Language based on the Higher-Order π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29(4), 2004.
23. T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13, 1990.
24. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
25. A. Schmitt and J.-B. Stefani. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In *Global Computing*, volume 3267 of *LNCS*. Springer, 2005.
26. S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee. A modular formal semantics for ptolemy. *Mathematical Structures in Computer Science*, 23(4), 2013.
27. R. J. van Glabbeek. The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.*, 60-61, 2004.
28. M. Wermelinger and J. L. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Sci. Comput. Program.*, 44(2), 2002.