Bigraphical models for Container-based Systems

Marino Miculan, Fabio Burco

MADS Lab, Department of Mathematics, Computer Science and Physics,
University of Udine, Italy. marino.miculan@uniud.it

1 Introduction

Microservices-oriented architectures are emerging in the digitalization of business processes, since their flexibility and openness support easier integration, scalability, dynamic deployment and reconfiguration. In these architectures, a central role is played by *containers*, with their clear separation of tasks between developers and system administrators. Containers provide development, packaging and deployment mechanisms, such that applications can be developed by abstracting from the environment in which they are executed. Thus, containers can be seen as the lowest level of a microservice-oriented architecture, holding the running application, libraries, and their dependencies, and which can be composed and connected together.

Collecting, connecting and coordinating (*orchestrating*) containers into a complete working system is not an easy task. Some of the duties of the administrator are: provisioning and deployment of containers; providing each container the resources it needs (e.g. volumes for file storage); establishing networks between containers; exposing services running in a container to the outside world; remapping port addresses in order to avoid conflicts; etc.

In order to simplify this task, the administrator is provided with tools and utilities supporting the deployment and management of containers; some examples are Docker's `docker-compose` and Kubernetes' `kubectl` functions for handling *pods*. Still, these tools are complex and error-prone: in their configuration files (for example, `docker-compose.yml`) it is easy to overlook a required resource, or to mismatch service names, or to misconfigure a network connection hindering communications between containers or (even worse) allowing for unexpected (and possibly dangerous) communications. These problems may be caught at the system's start-up (e.g., a missing image), but may arise suddenly at any time during the system lifetime, causing service malfunctions and interruptions. In fact, at the moment the designed containers are tested directly in a running system, and if errors occur some corrections are made; this operation is repeated several times before the deployment becomes operational, and with no guarantee of further misbehaviours.

This situation would greatly benefit from *formal models* of container-based architectures. These models should abstract from the application level details, but still detailed enough to capture the aspects concerning the composition and connections of containers. In these models we should be able to express formally important properties of these systems, such as security ones; moreover, they should support the development of tools for the analysis, verification and manipulation of system's architectures and configurations.

In this work, we propose such a model, within the framework of *Bigraphical Reactive Systems* (BRSs) [7, 9]. Introduced as a meta-model for ubiquitous, distributed, mobile systems, BRSs have been successfully applied to the formalization of a broad variety of domain-specific models, including context-aware systems and web-service orchestration languages; a non exhaustive list is [1, 3, 4, 7, 8]. Beside their normative power, BRSs are appealing because they provide a range of interesting general results and tools, which can be readily instantiated with the specific model under scrutiny: libraries for bigraph manipulation (e.g., [2] and jLibBig), simulation tools, graphical editors [5], model checkers [12], modular composition [11], etc..

A key feature of BRSs is that system configurations are represented by *bigraphs*, which are graph-like data structures capable to describe at once both the locations and the logical connections of (possibly nested) components of a system. Since bigraphs can be naturally composed, this model allows for *modular design* of container-based services. Then, the evolution of a system can be defined in a declarative way by means of a set of *(parametric) graph rewriting rules*, which can replace and change components' positions and connections.

The rest of the paper is structured as follows. In Section 2 we introduce *local directed bigraphs* and their application as models for container based systems. Some applications of this model, especially to security aspects, are briefly described in Section 3. Finally, in Section 4 we draw some conclusions and sketch directions for further works.

2 Bigraphs for containers

In order to define the formal model for containter-based systems, we introduce *local directed bigraphs*, a variant of directed bigraphs [6] which allows to deal with *localized* resources.

2.1 Local directed bigraphs

A *(local) interface* $X = \langle (X_0^+, X_0^-), (X_1^+, X_1^-), \ldots, (X_n^+, X_n^-) \rangle$ is a list of *polarized interfaces*, each of which is defined as a pair of disjoint finite sets of names (X_i^+, X_i^-). The pair (X_0^+, X_0^-) contains the global names. We define $X^+ \triangleq \biguplus_{i=1}^n X_i^+$ and $X^- \triangleq \biguplus_{i=1}^n X_i^-$, $width(X) \triangleq n$.

Two interfaces X, Y can be juxtaposed yielding a new interface, as follows:

$$X \otimes Y \triangleq \langle (X_0^+ \uplus Y_0^+, X_0^- \uplus Y_0^-), (X_1^+, X_1^-), \ldots, (X_n^+, X_n^-), (Y_1^+, Y_1^-), \ldots, (Y_m^+, Y_m^-) \rangle$$

Let $\mathcal{K}$ be a signature, that is, a set of elements which form the syntactic basis of bigraphs. Each type $c \in \mathcal{K}$, c has *polarized arity* $c = \langle n^+, m^- \rangle$. A *local directed bigraph (ldb)* $B : I \to O$ is a tuple $B = (V, E, ctrl, prnt, link)$, where

- I and O are the *inner* and *outer* interfaces respectively;
- V and E are the sets of *nodes* and *edges* respectively;
- $ctrl : V \to \mathcal{K}$ is the control map;
- $link : Pnt(B) \to Lnk(B)$ is the link map;
- $prnt : width(I) \uplus V \to V \uplus width(O)$ is the parent map;

where ports, points and links are defined as follows, for $i \in \{1, \ldots, n\}$:

$$link \triangleq \biguplus_{i=1}^{n} link_i \qquad Pnt \triangleq \biguplus_{i=1}^{n} Pnt_i \qquad Lnk \triangleq \biguplus_{i=1}^{n} Lnk_i$$

$$Prt_i^+(B) \triangleq \sum_{v \in V | prnt^*(v) = i} \pi_1(ctrl(v)) \qquad Prt_i^-(B) \triangleq \sum_{v \in V | prnt^*(v) = i} \pi_2(ctrl(v))$$

$$link_i : Pnt_i(B) \to Lnk_i(B)$$

$$Pnt_i(B) \triangleq I_0^+ \uplus O_0^- \uplus I_i^+ \uplus O_i^- \uplus Prt_i^+(B) \qquad Lnk_i(B) \triangleq I_0^- \uplus O_0^+ \uplus I_i^- \uplus O_i^+ \uplus E \uplus Prt_i^-(B)$$

such that $\forall x \in link(I_i^+) \cap I_i^- : |link^{-1}(x)| = 1$ and $\forall y \in link(O_i^+) \cap O_i^- : |link^{-1}(y)| = 1$.

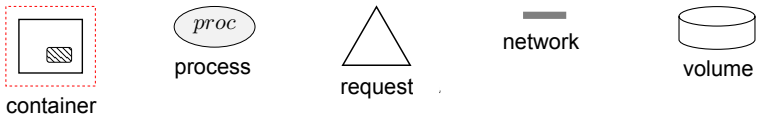

Figure 1: Signature for container bigraphs.

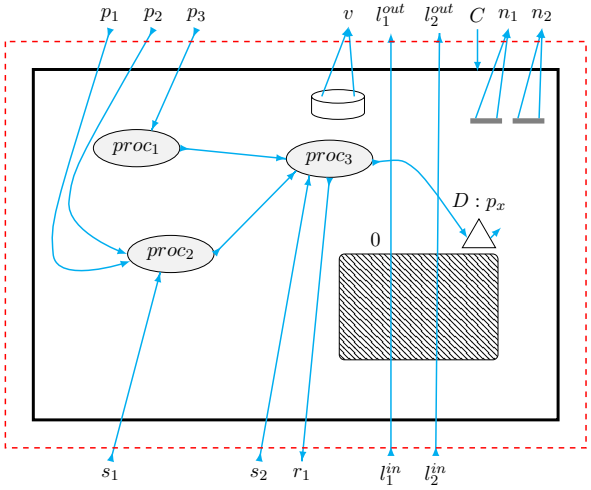

Figure 2: A bigraph representing a container. Inner interface is $\langle(\emptyset,\emptyset),(\{s_1,s_2,l_1^{in},l_2^{in}\},\{r_1\})\rangle$, and outer interface is $\langle(\emptyset,\emptyset),(\{v,l_1^{out},l_2^{out},n_1,n_2\},\{p_1,p_2,p_3,C\})\rangle$.

Bigraphs can be combined in two orthogonal ways: composition and juxtaposition. Let $B_1 : I_1 \to O_1$, $B_2 : I_2 \to O_2$ be two ldb. Their juxtaposition (tensor product) $B_1 \otimes B_2 : I_1 \otimes I_2 \to O_1 \otimes O_2$ is defined as $B_1 \otimes B_2 = (V_1 \uplus V_2, E_1 \uplus E_2, ctrl_1 \uplus ctrl_2, prnt_1 \uplus prnt_2, link_1 \uplus link_2)$.

Let $B_1 : X \to Y$, $B_2 : Y \to Z$ be two ldb. Their composition $B_2 \circ B_1 : X \to Z$ is defined as the bigraph $(V_1 \uplus V_2, E_1 \uplus E_2, ctrl_1 \uplus ctrl_2, prnt, link)$, where $prnt : width(X) \uplus V \to V \uplus width(Z)$ and $link : Pnt(B_2 \circ B_1) \to Lnk(B_2 \circ B_1)$ are defined as expected.

Therefore, given a signature $\mathcal{K}$, the local interfaces and local directed bigraphs over $\mathcal{K}$ form the objects and arrows of a monoidal category $\mathcal{LDB}(\mathcal{K})$.

2.2 A signature for containers

Once we have defined the algebraic framework of our model, we can introduce a signature for containers. The signature we consider in this paper is the following:

$$\mathcal{K} = \{container : (0^+, 1^-), process_{r,s} : (r^+, s^-),$$
$$request : (1^+, 1^-), network : (1^+, 1^-), volume : (1^+, 1^-)\}$$

This signature is graphically depicted in Figure 1. These basic elements can be nested and connected, as defined in Section 2.1, yielding bigraphs such as the one in Figure 2. This bigraph has one *root*, represented by the red dotted rectangle. Under this root there is one container node, which contains three process nodes, one volume node, two network nodes, a

```
version: '2'
services:
  wp:
    image: wordpress
    links:
      - db
    ports:
      - "8080:80"
    networks:
      - front
    volumes:
      - datavolume:/var/www/data:ro
  db:
    image: mariadb
    expose:
      - "3306"
    networks:
      - front
      - back
  pma:
    image: phpmyadmin/phpmyadmin
    links:
      - db:mysql
    ports:
      - "8181:80"
    volumes:
      - datavolume:/data
    networks:
      - back
networks:
  front:
    driver: bridge
  back:
    driver: bridge
volumes:
  datavolume:
    external: true
```

Figure 3: Example of `docker-compose.yml` configuration file.

request node, and one *site* (the gray area). Arrows connect node ports and names, respecting their polarity. The intended meaning of arrows is that of "resource accesses", or dependencies. In this example, the container offers services to (i.e., accepts requests from) the surrounding environment on port p_1, p_2, p_3, and needs to access a volume v and two networks n_1, n_2. The site is a "hole", meaning that it can be filled by adding another bigraph containing processes (which can access to services offered inside the container through s_1, s_2) and resources (which $proc_3$ can access through r_1). Filling a hole with another bigraph corresponds to the composition defined in the previous subsection, and as such it is subject to precise formal conditions, similar to composition of typed functions; in particular, a name of one interface can be connected to that of another interface if their polarity is the same.

2.3 Composition of containers is composition of bigraphs

An important example of bigraph composition is represented by the composition of containers, as performed by, e.g., `docker-compose`. In this case, the context bigraph can be obtained automatically from the `docker-compose.yml` file.

As an example, let us consider the `docker-compose.yml` in Figure 3. Its corresponding "context" bigraph is shown in Figure 4(a). This bigraph has one root (representing the whole resulting system), as many holes as components ("services") to be assembled, the (possibly shared) networks and volumes that each container requires, and exposes the (possibly renamed) ports to the external environment. Three bigraphs with the correct interfaces (Figure 4(b)) can be composed into the environment, yielding the system in Figure 4(c). This resulting system can be seen as a "pod", and which can be composed into the site (of the right interface) of other bigraphs, in a modular fashion.

The correspondence between `docker-compose.yml` files and environment bigraphs can be made formal; in fact, we have implemented a tool which translates `docker-compose` YAML files into composition bigraphs, taking advantage of the library jLibBig for bigraph manipulation.

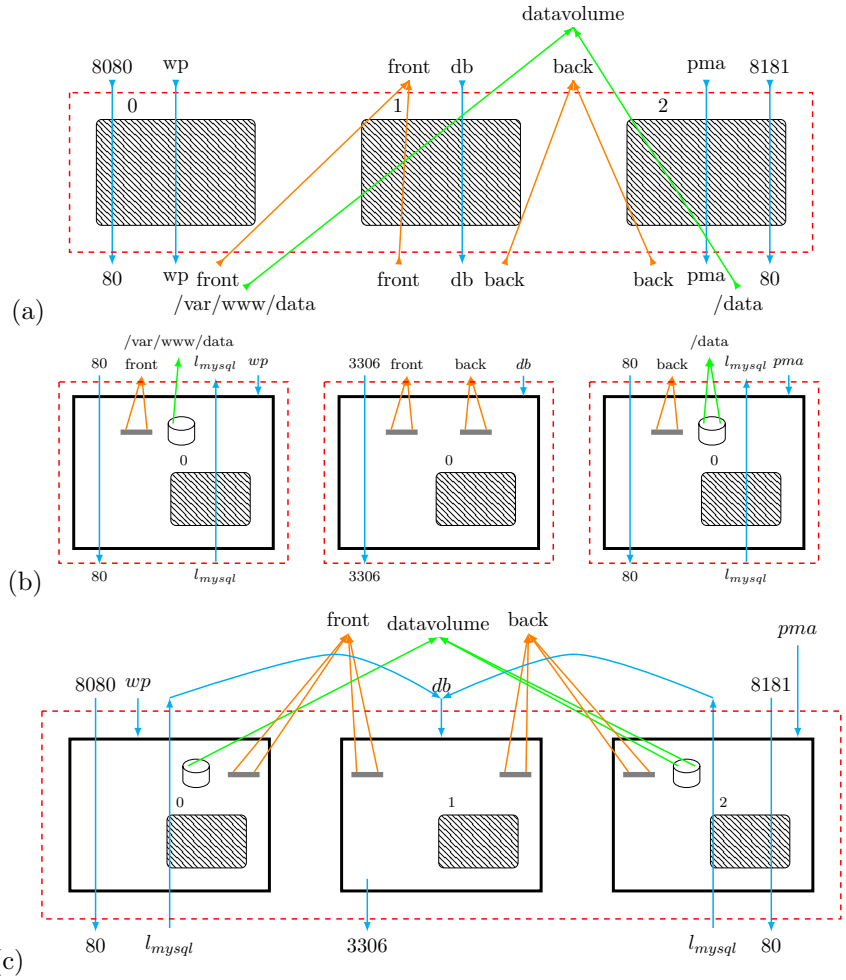

Figure 4: Example of composition: (a) the composition environment (corresponding to the YAML file in Figure 3); (b) the containers to be assembled; (c) the result of the composition.

3 Application: analysis of containers architectures

Once a container architecture is represented as a bigraph, it can be easily analysed and manipulated. Many properties about containers can be formalized as properties of the corresponding bigraphic representations and hence verified using well-known techniques from static analysis and model checking. In this section we briefly mention two checks that we have implemented in a prototype checker (and which are not performed by `docker-compose`).

Links correctness Given a bigraph modeling a container composition (e.g., obtained from a YAML file), we can check that no container requires a `links` to containers that can not be reached through any shared networks. This property is simply verified by checking that if a link connects two nodes of two different containers, then there must be an edge which is connected to by network nodes from both containers.

Network security levels safety Let us suppose that networks connecting the containers are ordered according to a *security hierarchy*, specified by the user as a set of ordering assertions of the form $n > m$. A simple security isolation policy can require that information from (more confidential) network n should not leak to a (lower secure) network m—but the flow in the other direction is allowed. A composition configuration violates this policy if there is a "channel", through containers, volumes, and networks, connecting two networks of different security levels.

We can verify if a given configuration violates this policy by looking for order-violating paths between networks nodes in the corresponding bigraph. To this end, the bigraph B modelling the system is visited in order to build a bipartite graph G as follows:

- nodes of G are each container, network and volume of B;
- for each c container and r a network from which c can read, the arc (r, c) is added to G; if c can also write to r, then the arc (c, r) is added to G. The same applies to volumes.

Then, for each pair of nodes h, l such that $h > l$ (in the transitive closure of the order), we check that there is no directed path from h to l in the graph G. If this is the case, then the configuration respects the separation policy. Otherwise, an information leakage is possible and hence a warning is raised.

4 Conclusions

In this paper we have introduced a bigraphic model of container-based systems. In this model, containers and container configurations are represented by *bigraphs*, which are graph-like data structures capable to describe at once both the locations and the logical connections of (possibly nested) components of a system. Composition of bigraphs correspond precisely to the composition of containers, as performed, e.g., by `docker-compose`. This representation can be used for analysing and verifying properties of container-based.

Up to our knowledge, this is one of the first formal models for container-based systems. Maybe the closest work is [10], where a model-driven management of Docker containers has been proposed. This model enables the verification of containers architecture at design time. One difference with our work is that our model is compositional by construction, allowing naturally a modular design and analysis of containers and components.

In this paper we have considered only the static aspects of container based systems. As future work, we intend to extend our approach to *dynamic* properties of the systems. Evolution of bigraphic systems are usually specified by means of a set of graph-rewriting rules. In our setting, these rewriting rules can be used to model various system reconfigurations, e.g. scaling up/down or component replacement. Correspondingly, we will be able to take into account other properties, such as safety invariants and liveness properties.

References

[1] G. Bacci, D. Grohmann, and M. Miculan. Bigraphical models for protein and membrane interactions. In G. Ciobanu, editor, *Proc. MeCBIC*, volume 11 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–18, 2009.

[2] G. Bacci, D. Grohmann, and M. Miculan. Dbtk: a toolkit for directed bigraphs. In *CALCO 2009 Conference Proceedings*, volume 5728 of *Lecture Notes in Computer Science*. Springer, 2009.

[3] M. Bundgaard, A. J. Glenstrup, T. T. Hildebrandt, E. Højsgaard, and H. Niss. Formalizing higher-order mobile embedded business processes with binding bigraphs. In *Proc. COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 83–99. Springer, 2008.

[4] T. C. Damgaard, E. Højsgaard, and J. Krivine. Formal cellular machinery. *Electronic Notes in Theoretical Computer Science*, 284:55–74, 2012.

[5] A. J. Faithfull, G. Perrone, and T. T. Hildebrandt. BigRed: A development environment for bigraphs. *ECEASST*, 61, 2013.

[6] D. Grohmann and M. Miculan. Directed bigraphs. In *Proc. MFPS*, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 121–137. Elsevier, 2007.

[7] O. H. Jensen and R. Milner. Bigraphs and transitions. In A. Aiken and G. Morrisett, editors, *POPL*, pages 38–49. ACM, 2003.

[8] A. Mansutti, M. Miculan, and M. Peressotti. Multi-agent systems design and prototyping with bigraphical reactive systems. In K. Magoutis and P. Pietzuch, editors, *Proc. DAIS*, volume 8460 of *Lecture Notes in Computer Science*, pages 201–208. Springer, 2014.

[9] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[10] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle. Model-driven management of docker containers. In *9th IEEE International Conference on Cloud Computing*, pages 718–725, 2016.

[11] G. Perrone, S. Debois, and T. T. Hildebrandt. Bigraphical refinement. In J. Derrick, E. A. Boiten, and S. Reeves, editors, *Proc. REFINE*, volume 55 of *Electronic Proceedings in Theoretical Computer Science*, pages 20–36, 2011.

[12] G. Perrone, S. Debois, and T. T. Hildebrandt. A model checker for bigraphs. In S. Ossowski and P. Lecca, editors, *Proc. SAC*, pages 1320–1325. ACM, 2012.