An EMOF-Compliant Abstract Syntax for Bigraphs

Timo Kehrer, Christos Tsigkanos and Carlo Ghezzi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy

Bigraphs are an emerging modeling formalism for structures in ubiquitous computing. Besides an algebraic notation, which can be adopted to provide an algebraic syntax for bigraphs, the bigraphical theory introduces a visual concrete syntax which is intuitive and unambiguous at the same time; the standard visual notation can be customized and thus tailored to domain-specific requirements. However, in contrast to modeling standards based on the Meta-Object Facility (MOF) and domain-specific languages typically used in model-driven engineering (MDE), the bigraphical theory lacks a precise definition of an abstract syntax for bigraphical modeling languages. As a consequence, available modeling and analysis tools use proprietary formats for representing bigraphs internally and persistently, which hampers the exchange of models across tool boundaries. Moreover, tools can be hardly integrated with standard MDE technologies in order to build sophisticated tool chains and modeling environments, as required for systematic engineering of large systems or fostering experimental work to evaluate the bigraphical theory in real-world applications. To overcome this situation, we propose an abstract syntax for bigraphs which is compliant to the Essential MOF (EMOF) standard defined by the Object Management Group (OMG). We use typed graphs as a formal underpinning of EMOF-based models and present a canonical mapping which maps bigraphs to typed graphs in a natural way. We also discuss application-specific variation points in the graph-based representation of bigraphs. Following standard techniques from software product line engineering, we present a framework to customize the graph-based representation to support a variety of application scenarios.

1 Introduction

Bigraphs and bigraphical reactive systems have been proposed by Milner [20] as a fundamental theory for structures in ubiquitous computing [27]. A concise model of space, which forms the context of a ubiquitous system, serves as a starting point in the bigraphical modeling methodology. The idea of bigraphs is based on two fundamental concepts of discrete spaces: locality and connectivity, called *placing* and *linking* in bigraphical terminology. In essence, a bigraph consists of a *place graph*, a forest defined over a set of nodes which is intended to represent entities and their locality in terms of a containment structure, and a *link graph*, a hypergraph composed over the same set of nodes representing arbitrary linking among those entities. Possible local reconfigurations of the space are expressed as a set of reaction rules yielding a Bigraphical Reactive System (BRS). A reaction rule is a rewriting rule which consists of two bigraphs, called *redex* and *reactum*. Roughly speaking, the redex specifies a bigraphical pattern whose occurrence found in a "host bigraph" enables replacement by the reactum.

The bigraphical theory has been shown to be general enough as a meta-calculus to embed and thus unify reasoning over a set of existing formalisms and calculi targeting different facets of computation, such as Petri nets [18], the Ambient Calculus [5] and the pi-calculus [4]. Bigraphs have found application in ubiquitous systems in diverse domains such as cyber-physical [25, 26] and biological systems [17]. The use of bigraphs and BRS as underlying supporting formalisms enables a spectrum of reasoning approaches spanning from modeling to simulation and verification. However, as argued by Milner [20], much experimental work has to be done in order to evaluate whether the idea of bigraphs serves as a

Alexander Heußner, Aleks Kissinger, Anton Wijs (Eds.):
2nd Graphs as Models workshop (GaM 2016)
EPTCS 231, 2016, pp. 16–30, doi:10.4204/EPTCS.231.2

suitable foundation for building large-scale ubiquitous systems. In this regard, the bigraphical modeling approach shall be not only used for the purpose of analysis, but even as a language for simulation and "programming". Following this line of research, the theory of bigraphs actually serves as a foundation for domain-specific languages. Besides an algebraic notation, which can be adopted to provide a textual syntax, the bigraphical theory introduces a visual concrete syntax which can be customized to domain-specific requirements.

Several approaches utilizing bigraphs exist, focusing on different aspects. Visual design of bigraphical models has been considered [10], while facilities for simulation, verification and execution are provided by a plethora of approaches utilizing different forms of bigraphs and their dynamics. For example, DBtk[1] targets a form of bigraphs with directed edges; SBAM [14] extends bigraphs with stochastic semantics; finally, BigMC [21] performs model checking of matching properties.

Systematic engineering of systems using bigraphs, however, is hindered by obstacles to modeling, integration, evaluation, and the overall development process. Bigraphical tool support is still in its infancy and available research prototypes are yet poorly integrated. Methods supporting development of large-scale models are sorely missing. This situation hampers empirical assessments with real users to evaluate expressive power, generality, and utility of the bigraphical theory in real-world applications. An important reason for this unfortunate situation in actual engineering of systems is that the notion of bigraphs as a domain-specific modeling language yet lacks a precise and commonly accepted definition of an abstract syntax. The available modeling and analysis tools use proprietary formats for representing bigraphs internally and persistently, which hampers the exchange of models across tool boundaries and the development of sophisticated tool chains.

Contributions. In this paper, we propose an abstract syntax for bigraphs which is compliant with the Essential Meta-Object Facility (EMOF) standard defined by the Object Management Group (OMG) [11]. This facilitates the integration of bigraphical modeling and analysis tools with mainstream technologies for model-driven engineering (MDE), typically based on the EMOF standard. We use typed graphs as a formal underpinning of EMOF-based models and present a canonical mapping which maps bigraphs to typed graphs in a natural way. Thereupon, we discuss application-specific variation points in the graph-based representation of bigraphs. Following techniques from software product line engineering, we present a framework to customize the graph-based representation to support a variety of applications. We implemented the approach as an extension of the Eclipse IDE integrating a bigraph modeling tool [10] with the Eclipse Modeling Framework [24], the de-facto reference implementation of EMOF.

Paper structure. Section 2 presents bigraphs, along with a motivating example of a cyber-physical space which is used throughout the paper. Section 3 briefly introduces our notion of typed graphs which is used in Section 4 for a canonical mapping of bigraphs to typed graphs. Section 5 bootstraps a set of application-specific variation points of a bigraphical abstract syntax, aimed to support different forms of bigraphs. Section 6 outlines how to integrate the approach into an existing MDE environment and presents an example application. Section 7 concludes the paper along with an outlook on future work.

2 The Idea of Bigraphs: Informal Introduction and Visual Syntax

A *Bigraph* consists of two graphs. A *place graph* is a forest, a set of rooted trees defined over a set of nodes. A *link graph* is a hypergraph composed over the same set of nodes and a set of edges, each linking an arbitrary number of nodes. Connections of an edge with its nodes are called ports. Place and

link graphs are orthogonal, and edges between nodes can cross locality boundaries. The types of nodes, called controls in bigraphical terminology, are defined by a *signature*. To avoid misunderstandings, hereafter we use the terms and definitions introduced by Milner [20], although we acknowledge that sometimes they may sound unconventional and unintuitive.

A bigraph B is a structure $(V_B, E_B, ctrl_B, prnt_B, link_B) : \langle k, X\rangle \to \langle m, Y\rangle$ where V_B is a set of nodes, E_B is a set of edges, $ctrl_B$ is the control map that assigns controls to nodes, $prnt_B$ is the parent map that defines the nesting of nodes, and $link_B$ is the link map that defines a link structure. The notation $\langle k, X\rangle \to \langle m, Y\rangle$ indicates that the bigraph has k sites which denote the presence of other unspecified nodes, m roots, a set X of inner names and a set Y of of outer names. $\langle k, X\rangle$ and $\langle m, Y\rangle$ are called the inner and outer interfaces (or faces) of the bigraph.

The bigraphs presented here are pure (non binding) and concrete; placing and linking are independent structures, and nodes and edges have discrete identifiers [19]. Moreover, signatures are basic signatures abstracting from dynamic aspects, namely activeness and passiveness of controls. We present only essential structural definitions; the interested reader can refer to [20]. Names, node-identifiers and edge-identifiers are drawn from infinite sets, respectively $\mathscr{X}$, $\mathscr{V}$ and $\mathscr{E}$, disjoint from each other.

Definition 1 (Signature)**.** *A signature Σ is a pair (κ, ar). It consists of a set κ whose elements are node types called controls, and a function $ar : \kappa \to \mathbb{N}$ assigning a natural number representing arity (the number of ports) to each control.*

In the sequel, we will describe a signature as a set of pairs $control : arity$.

Definition 2 (Place Graph)**.** *A place graph B^P over a signature Σ is a structure $B^P = (V_B, ctrl_B, prnt_B) : k \to m$ having an inner face k and an outer face m, both finite ordinals $k = \{0, 1, \ldots, k-1\}$ and $m = \{0, 1, \ldots, m-1\}$; these index the sites and roots of the place graph. B^P has a finite set $V_B \subset \mathscr{V}$ of nodes, a control map $ctrl_B : V_B \to \kappa$ and a parent mapping $prnt_B : k \cup V_B \to V_B \cup m$ which is acyclic.*

Definition 3 (Link Graph)**.** *A link graph B^L over a signature Σ is a structure $B^L = (V_B, E_B, ctrl_B, link_B) : X \to Y$ with finite inner face $X \subset \mathscr{X}$ and finite outer face $Y \subset \mathscr{X}$, called the names of the link graph. B has finite sets $V_B \subset \mathscr{V}$ of nodes and $E_B \subset \mathscr{E}$ of edges, a control mapping $ctrl_B : V_B \to \kappa$ and a link mapping $link_B : X \cup P_B \to E_B \cup Y$ where $P_B \stackrel{def}{=} \{(v, i) \mid i \in ar(ctrl_B(v))\}$ is the set of ports of B; thus (v, i) is the i^{th} port of node v.*

A bigraph is then a pair of a place graph and a link graph, called its *constituents*.

Definition 4 (Bigraph)**.** *A bigraph $B = (V_B, E_B, ctrl_B, prnt_B, link_B) : \langle k, X\rangle \to \langle m, Y\rangle$ over a signature Σ consists of a place graph $B^P = (V_B, ctrl_B, prnt_B) : k \to m$ and a link graph $B^L = (V_B, E_B, ctrl_B, link_B) : X \to Y$.*

Throughout the paper, we refer to the nodes, edges, roots, sites, inner names, outer names and ports as the *elements* of a bigraph. Moreover, we use the rigorous visual notation introduced in [20] as concrete syntax for bigraphs. In this notation, roots and sites are shown as dotted and shaded boxes, respectively, with indices defined by the place graph's outer and inner face attached to them. Nodes are represented as ellipses with a control label attached where their containment relation is visually represented by nesting a place graph node inside another one. Ports of nodes are represented as black bullets; they can be linked together to form edges. Moreover, ports can be linked to outer names, and edges can be linked to inner names of the link graph interface. By convention, outer names are drawn above a bigraph figure while inner names are drawn below.

In the following, we introduce a sample bigraph serving as running example. Consider an instance of a cyber-physical space that models a printing scenario, inspired by [10]; an office environment comprised

of two rooms, where one contains a computer and a printer, while the other one contains a user who holds a print job. A user can submit the job for printing through the computer connected to the printer. Example 1 focuses on static aspects of this system, i.e. it shows a snapshot at a particular point of time.

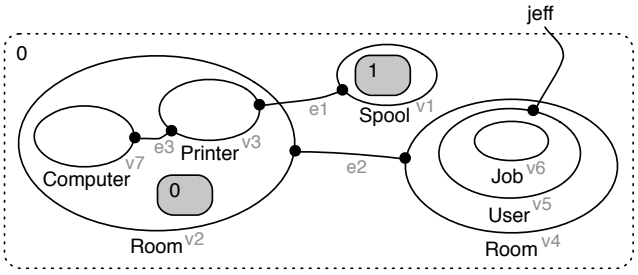

Figure 1: Bigraph model of a context-aware spooling system.

Example 1 (Snapshot of a bigraph model of a context-aware printing system)**.** *Fig. 1 shows a visual representation of a bigraph* $B_1 : \langle 2, \emptyset \rangle \to \langle 1, \{jeff\} \rangle$ *over signature* $\Sigma_1 = \{Job : 0,\ User : 1,\ Room : 1,\ Spool : 1,\ Printer : 2,\ Computer : 1\}$*. It consists of a single root and defines one outer name* $jeff$*. The root contains nodes corresponding to a print Spool and two Rooms. The rooms are connected by an edge, the ports being linked to this edge, semantically representing a connection that corresponds to room doors. The left Room contains a Printer which is connected to the print Spool, by linking the ports of the Printer and the print Spool with a common edge. Moreover, a node of control Computer is connected to the Printer. The Room on the left and the Spool contain sites, which serve as placeholders to denote the presence of unspecified nodes (representing unspecified entities). The Room on the right contains a User, in turn containing a print Job which has not yet been submitted for printing. A link from the user's port to outer name jeff identifies the User. In Fig. 1, grey labels identify nodes and edges for explanation purposes in following sections.*

3 Graph-based Representation of EMOF Models

In MDE, the conceptual contents of a model are usually considered as a typed attributed graph which is known as the abstract syntax graph (ASG) of a model. The allowed types of nodes and edges in an ASG as well as their possible relations are typically defined by a meta-model which thus serves as an abstract syntax specification. In the remainder of this section, we introduce our graph-based representation of meta-models and instance models. The formalization is based on [3, 9] and is fully compatible with the EMOF standard.

Graphs and graph morphisms. Following object-oriented modeling principles of the EMOF standard, our basic notion of a graph refers to a directed unlabelled graph $G = (G_N, G_E, src_G, tgt_G)$ which consists of a set G_N of nodes, a set G_E of edges, as well as source and target functions $src_G, tgt_G : G_E \to G_N$ associating to each edge its source and target node. Given two graphs G and H, a pair of functions (f_N, f_E) with $f_N : G_N \to H_N$ and $f_E : G_E \to H_E$, forms a graph morphism $f : G \to H$ if it preserves source and target, i.e. $f_N \circ src_G = src_H \circ f_E$ and $f_N \circ tgt_G = tgt_H \circ f_E$. In other words, for each edge $e_G \in G_E$ there is a corresponding edge $e_H = f_E(e_G) \in H_E$ such that $src_G(e_G)$ is mapped to $src_H(e_H)$ and $tgt_G(e_G)$ is mapped to $tgt_H(e_H)$.

Meta-models as type graphs. A meta-model is formally treated as a distinguished graph called *type graph*, whose nodes and edges represent *node types* and *edge types*, respectively. In addition, a type graph comprises the definition of an *inheritance hierarchy* including *abstract node types*, a *containment structure*, *opposite edges* representing bidirectional edge types, and *multiplicities* attached to edge types.

Definition 5 (Type graph)**.** *A type graph* $TG = (T, I, A, C, OE, mult)$ *is a graph* $T = (T_N, T_E, src_T, tgt_T)$*, representing node types and edge types, equipped with the definition of a type hierarchy* $I \subseteq T_N \times T_N$*, which must be an acyclic relation, a set* $A \subseteq T_N$ *identifying abstract node types, a containment structure* $C \subseteq T_E$*, and an anti-reflexive, symmetric and unique relation* opposite edges $OE \subseteq T_E \times T_E$ *representing bidirectional edge types. Moreover, multiplicities can be attached to edge types by function* $mult : T_E \to Mult$*. Mult denotes the set of multiplicities, and a multiplicity is a pair* $[lb, ub] \in \mathbb{N} \times (\mathbb{N} \cup \{*\})$ *with* $lb \leq ub$ *or* $ub = *$*.*

Given $(x, y) \in I$, x refers to the *subtype* and y to the *supertype* induced by the inheritance hierarchy. We use function $allSub : T_N \to 2^{T_N}$ with $allSub(x) = \{y \in T_N \mid (y, x) \in I^+\}$ and I^+ being the transitive closure of the inheritance relation I, to refer to all subtypes of a node type.

Models as typed graphs. We formalize the typing relation between models and meta-models by a special graph morphism, called *typing morphism*, relating a typed graph with its associated type graph.

Definition 6 (Typed graph and typing morphism)**.** *An instance graph* G *is typed over a fixed type graph* TG *if there is a typing morphism* $type_G : G \to T$ *which maps the nodes and edges of* G *to those of* T *in a suitable way, i.e. for each edge* $e_G \in G_E$ *there is a corresponding edge* $e_T = type_G(e_G) \in T_E$*, such that:*

- $src_G(e_G)$ *is mapped to* $src_T(e_T)$ *or any subtype* $x \in allSub(src_T(e_T))$*, and*
- $tgt_G(e_G)$ *is mapped to* $tgt_T(e_T)$ *or any subtype* $x \in allSub(tgt_T(e_T))$*.*

We say that graph G typed over TG is *valid* if there are *no containment cycles*, each node has *at most one container*, and *all opposite edges* are handled consistently. In the sequel, speaking about typed graphs, we assume them to be valid.

Attributed graphs. Throughout this paper, we largely neglect the formal treatment of node attributes. However, they can be handled by following the definition of attributed graphs, which can be found in [13]. The main idea of formalizing node attributes is to consider them as edges of a special kind referring to *(i)* data types in case of type graphs and *(ii)* data values in case of typed graphs.

Notational conventions. We adopt the syntax of UML class and object diagrams as a compact visual notation for type and instance graphs, respectively. Visual connections between nodes without arrowheads denote bidirectional edges. Although formally treated as edges of a special kind, attributes are shown as integral parts of nodes. For node types, we use the notation $a : DT$ to denote the declaration of an attribute a of type DT. For instance graph node attributes, we write $a = v$ to symbolize the assignment of value v to attribute a. For a given node $x \in G_N$, we write $x.a$ to refer to the value of its attribute a.

4 A Canonical Mapping of Bigraphs to Typed Graphs

In this section, we present a canonical mapping of bigraphs to typed graphs. We first introduce a basic type graph TG_{Base} which models the constituents of bigraphs in a natural way, following standard object-oriented design principles. Subsequently, we show how to extend TG_{Base} to a type graph TG_Σ for a given

bigraphical signature Σ. Having compatible type definitions Σ and TG_Σ at hand, we we can finally define the relation between a bigraph B over Σ and its corresponding graph G typed over TG_Σ.

4.1 Basic Type Graph

A basic type graph modeling the core concepts of bigraphs is shown in Fig. 2. The left-hand side of Fig. 2 models the elements of the place graph. We have an abstract node type *BPlace* for representing places; the nesting of places is represented by instances of the edge type *bPrnt* and its opposite *bChld*. There are three concrete subtypes of *BPlace*, namely *BRoot*, *BNode* and *BSite* in order to represent the roots, nodes and sites of a bigraph, respectively. We use attributes *index:int* to index the roots and sites. The right-hand side of Fig. 2 models the elements of the link graph. An instance of *BPoint* is connected to exactly one instance of *BLink* via an edge of type *bLink*. Conversely, an instance of *BLink* is connected to one or more instances of type *BPoint* via edges of type *bPoints*. Edge types *bLink* and *bPoints* are opposite to each other. Node types *BPoint* and *BLink* are abstract node types. Each of them has two subtypes, namely *BPort* and *BInnerName* as well as *BEdge* and *BOuterName*, respectively. The containment edge type *bPorts* expresses that ports are contained by nodes of type *BNode*; conversely, each node of type *BPort* has a parent node of type *BNode*, expressed by edge type *bNode*. Edge types *bPorts* and *bNode* are opposite to each other, thus defining a bidirectional edge type. We use attribute *index:int* to index the ports in the graph-based representation of a bigraphical node. We will refer to the basic type graph as $TG_{Base} = (T_{Base}, I_{Base}, A_{Base}, C_{Base}, OE_{Base}, mult_{Base})$ with $T_{Base} = (T_{Base_N}, T_{Base_E}, src_{T_{Base}}, tgt_{T_{Base}})$.

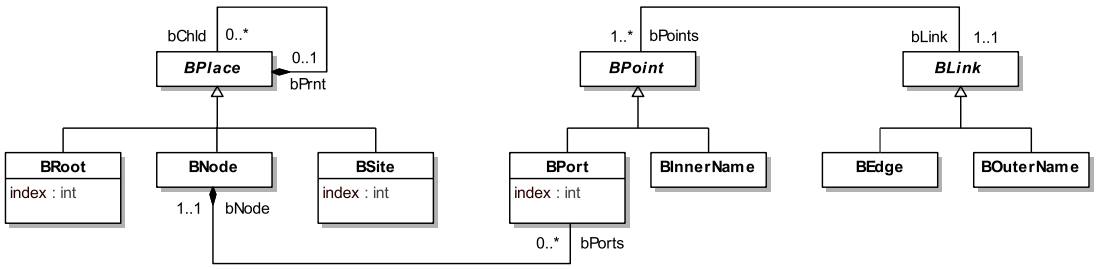

Figure 2: Basic type graph TG_{Base} modeling the anatomy of bigraphs

4.2 Mapping Bigraphical Signatures to Type Graphs

The basic type graph introduced in Fig. 2 is generic, i.e., it provides no information about bigraphical signatures: it neither models the controls nor the arities (number of ports) of nodes. To overcome this, hereafter we extend the basic type graph for a given signature. We handle controls through a *control-compatible extension* of the basic type graph w.r.t. a given signature. We then address arities via an additional well-formedness rule defined over a control-compatible extension of the basic type graph.

Controls. Our control-compatible extension of the basic type graph w.r.t. a given signature introduces for each control defined by the signature a corresponding subtype of the generic node type *BNode*.

Let $\Sigma = (\kappa, ar)$ be a bigraphical signature and TG a type graph extending TG_{Base}, i.e. we have $TG \supseteq TG_{Base}$. We say that TG extends TG_{Base} in a *control-compatible* way w.r.t. Σ, if $T_N = T_{Base_N} \cup \kappa$, $T_E = T_{Base_E}$, $src_T = src_{T_{Base}}$, $tgt_T = tgt_{T_{Base}}$, $I = I_{Base} \cup \{(x, BNode) \mid x \in \kappa\}$, $A = A_{Base}$, $C = C_{Base}$, $OE =$

OE_{Base}, and $mult = mult_{Base}$. In the following, we write TG_κ to emphasize the fact that a type graph extends the basic type graph TG_{Base} in a control-compatible way w.r.t. to a signature Σ.

Please note that bigraphical controls defined by Σ and a subset of the identifiers of node types defined by T_κ are drawn from the same set κ. To avoid confusion, we introduce an identity function $\Phi = Id_\kappa$ on set κ: given a control $K \in \kappa$, we use the notation $\Phi(K)$ to refer to its corresponding node type in T_κ.

Arities. To handle arities defined by a signature Σ, we introduce an additional well-formedness rule defined over the control-compatible extension of the basic type graph w.r.t. Σ. For each node type $\Phi(K)$ we restrict the multiplicity $[0..*]$ defined by the generic edge type *bPorts* to a fixed value $ar(K)$. We use first order logic to formalize the respective well-formedness rule.

Let Σ be a signature and TG_κ be an extension of TG_{Base}, control-compatible w.r.t. Σ. For each graph G typed over TG_Σ the following condition must hold, where $ran(\Phi)$ refers to the range of function Φ:

$$\forall n \in G_n : type_G(n) \in ran(\Phi) \implies |\{e \mid src_G(e) = n \wedge type_G(e) = bPorts\}| = ar(\Phi^{-1}(type_G(n))) \quad (1)$$

Given a node G_n with $type_G(n) \in ran(\Phi)$, $\{e \mid src_G(e) = n \wedge type_G(e) = bPorts\}$ refers to its set of outgoing edges of type *bPorts*. The cardinality of this set has to be equal to the aritiy of the corresponding control $\Phi^{-1}(type_G(n))$.

Bigraphical signatures as type graphs. Given a bigraphical signature Σ and a type graph TG, we say that TG is compatible to Σ if *(i)* it extends the basic type graph TG_{Base} in a control-compatible way and *(ii)* control arities are properly represented as additional well-formedness rule (1). In the following, we write TG_Σ to indicate the fact that a type graph is compatible w.r.t. to a signature Σ.

Example 2 (Bigraphical signature as type graph)**.** *Fig. 3 shows how the bigraphical signature Σ_1 of our running printer example is represented as type graph TG_{Σ_1}. Extensions over the basic type graph TG_{Base} of Fig. 2 are colored in light gray. Controls* Job, Printer, Room, Spool, User *and* Computer *are modeled as node types, each of them being a subtype of the basic node type* BNode*. Note that the well-formedness rule (1) which handles the arities of a signature is generic and needs not to be adjusted.*

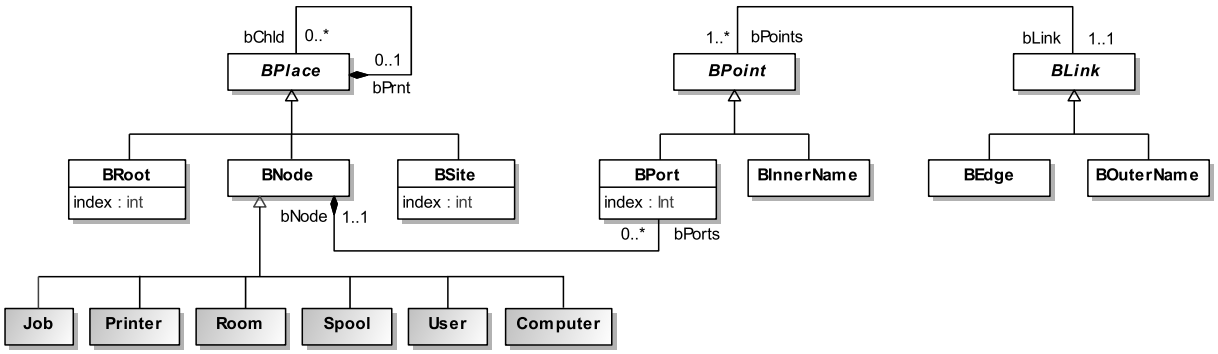

Figure 3: Type graph TG_{Σ_1} compatible to signature Σ_1 of our running printer example

4.3 Mapping Bigraphs to Typed Graphs

Now that we have specified how a bigraphical signature Σ is mapped to a typed graph TG_Σ in a compatible way, we can finally define the relation between a bigraph B over Σ and its corresponding typed

graph G over TG_Σ. First, we define an *element mapping* which bijectively maps the elements of B to the nodes of G in a suitable way. Subsequently, a set of additional *soundness criteria* will be specified over the basic element mapping.

Basic element mapping. Let us first define a basic element mapping which bijectively maps the elements of a bigraph to the nodes of a typed graph in a suitable way. The domain of this mapping is the set of elements of a bigraph, which is the union of its nodes, edges, sites, roots, inner names, outer names and ports. To establish this mapping formally, we have to recall here that the identifiers of the sites and roots of a bigraph are drawn from potentially overlapping sets, namely the finite ordinals $\{0,\ldots,k\}$ and $\{0,\ldots,m\}$, respectively (see Section 2). Thus, in order to distinguish the roots and sites, we introduce two utility functions, namely the two substitutions $\sigma_k : k \to k[\sigma_k]$ and $\sigma_m : m \to m[\sigma_m]$, both injective mappings and we have $k[\sigma_k] \cap m[\sigma_m] = \emptyset$. Analogously, identifiers for the inner names X and the outer names Y are drawn from the common infinite set $\mathscr{X}$ and thus potentially overlapping. Again, just to distinguish the inner names from the outer names, we introduce substitutions $\sigma_X : X \to X[\sigma_X]$ and $\sigma_Y : Y \to Y[\sigma_Y]$, both injective mappings and we have $X[\sigma_X] \cap Y[\sigma_Y] = \emptyset$.

We are now ready to define the mapping of the elements of a bigraph B to the nodes of an instance graph G as a bijective function

$$\phi : V_B \uplus E_B \uplus P_B \uplus k[\sigma_k] \uplus X[\sigma_X] \uplus m[\sigma_m] \uplus Y[\sigma_Y] \to G_N$$

with

$$\phi(x) = \begin{cases} Id_{V_B}(x), & \text{if } x \in V_B \\ Id_{E_B}(x), & \text{if } x \in E_B \\ Id_{k[\sigma_k]}(x), & \text{if } x \in k[\sigma_k] \\ Id_{X[\sigma_X]}(x), & \text{if } x \in X[\sigma_X] \\ Id_{m[\sigma_m]}(x), & \text{if } x \in m[\sigma_m] \\ Id_{Y[\sigma_Y]}(x), & \text{if } x \in Y[\sigma_Y]. \end{cases}$$

Proper typing of nodes. The nodes in an instance graph G must be properly typed to correctly represent a generic bigraph and the controls. Let Σ be a signature and TG_Σ its compatible type graph. Furthermore, let B be a bigraph over Σ and G be an instance graph over TG_Σ with elements of B bijectively mapped to the nodes of G by the basic element mapping ϕ. The nodes in G are properly typed if the following condition holds:

$$\begin{aligned} \forall x \in G_N : \\ \big(type_G(x) \in ran(\Phi) &\implies \phi^{-1}(x) \in V_B \wedge ctrl_B(\phi^{-1}(x)) = \Phi^{-1}(type_G(x)) \big) \wedge \\ \big(type_G(x) = BEdge &\implies \phi^{-1}(x) \in V_E \big) \wedge \\ \big(type_G(x) = BSite &\implies \phi^{-1}(x) \in k[\sigma_k] \big) \wedge \\ \big(type_G(x) = BInnerName &\implies \phi^{-1}(x) \in X[\sigma_X] \big) \wedge \\ \big(type_G(x) = BRoot &\implies \phi^{-1}(x) \in m[\sigma_m] \big) \wedge \\ \big(type_G(x) = BOuterName &\implies \phi^{-1}(x) \in Y[\sigma_Y] \big) \end{aligned}$$

Additional soundness criteria. A bijective element mapping is necessary for inducing a unique transformation from bigraphs to typed graphs (and vice versa), however, it is obviously not sufficient. To that end, we introduce a set of additional soundness criteria defined over the basic element mapping satisfying typing constraints. In the following, let again Σ be a signature and TG_Σ its compatible type graph. Furthermore, let B be a bigraph over Σ and G be an instance graph over TG_Σ with elements of

B bijectively mapped to the nodes of G by the basic element mapping ϕ, nodes of G are properly typed. We have the following additional soundness criteria:

(1) The nesting of places in B must coincide with the containment structure in G:

$$\forall p_B \in V_B \uplus k : \\ \exists e_G \in G_E : \ src_G(e_G) = \phi(p_B) \ \wedge \ tgt_G(e_G) = \phi(prnt_B(p_B)) \ \wedge \ type_G(e_G) = bPrnt$$

$$\forall e_G \in E_G : \ type_G(e_G) = bPrnt \implies \\ \exists p_B \in V_B \uplus k : \ \phi(p_B) = src_G(e_G) \ \wedge \ \phi(prnt_B(p_B)) = tgt_G(e_G)$$

(2) The linking structure in B must coincide with the linking structure in G:

$$\forall p_B \in X \uplus P_B : \\ \exists e_G \in G_E : \ src_G(e_G) = \phi(p_B) \ \wedge \ tgt_G(e_G) = \phi(link_B(p_B)) \ \wedge \ type_G(e_G) = bLink$$

$$\forall e_G \in E_G : \ type_G(e_G) = bLink \implies \\ \exists p_B \in X \uplus P_B : \ \phi(p_B) = src_G(e_G) \ \wedge \ \phi(link_B(p_B)) = tgt_G(e_G)$$

(3) The indexing of the roots in B must be consistent to the indices of their corresponding representations in G:

$$\forall i \in \{0,\ldots,m-1\}, \ n_G \in G_N \ : \ (i,n_G) \in \phi \iff n_G.index = i$$

(4) The indexing of the sites in B must be consistent to the indices of their corresponding representations in G:

$$\forall i \in \{0,\ldots,k-1\}, \ n_G \in G_N \ : \ (i,n_G) \in \phi \iff n_G.index = i$$

(5) The indexing of the ports in B must be consistent to the indices of their corresponding representations in G:

$$\forall (v,i) \in P_B, \ n_G \in G_N \ : \ ((v,i),n_G) \in \phi \iff n_G.index = i$$

Example 3 (Sample bigraph as typed graph)**.** *The sample bigraph B_1 over Σ_1 of Fig. 1 is mapped to the graph G_1 typed over the printing system type graph TG_{Σ_1} shown in Fig. 4. Note that some of the nodes of Fig. 4 have identifiers, indicated by the notation* id:NodeType*. They are aligned to the identifiers of bigraphical nodes and edges used in Fig. 1 and thus symbolize parts of the bijective mapping ϕ from bigraph elements in B_1 to the nodes in G_1. For the remaining nodes, identifiers are omitted in Fig. 4. All nodes are properly typed and additional soundness criteria regarding the nesting of places, the linking structure as well as the indexing of roots, sites and ports are fulfilled.*

5 Application-Specific Variation Points

In general, the definition of an appropriate abstract syntax of a modeling language depends on several design decisions, many of which are application-specific. The standard UML meta-model, for example, mainly targets the construction of UML editors but has several drawbacks for the construction of model differencing tools, see e.g. [16]. In this section, we address application-specific variation points of an abstract syntax for bigraphs. In order to deal with this variability, we apply basic principles known from

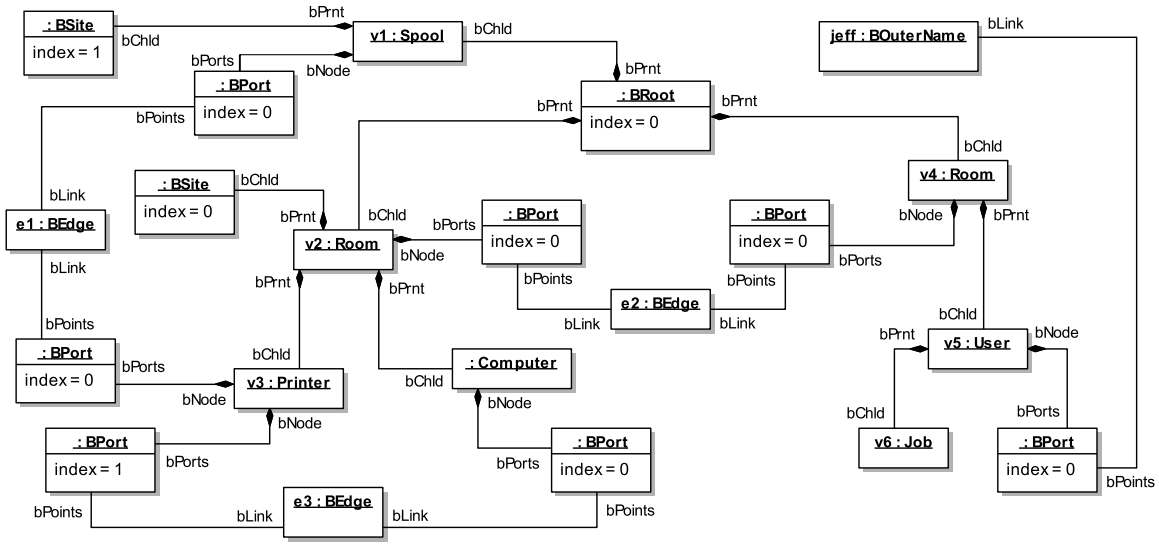

Figure 4: Sample bigraph B_1 over Σ_1 as typed graph G_1 over TG_{Σ_1}

the field of software product line engineering (SPLE) to optimally support a wide range of applications. In Section 5.1, we first identify variation points of an abstract syntax for bigraphs. The results of our variability analysis are formally documented in terms of a feature model yielding the configuration space of a bigraphical abstract syntax. Subsequently, Section 5.2 briefly discusses how to reconfigure canonical type and instance graphs using standard variability mechanisms adopted from model-based software product line engineering. We do not claim the presented variability model to be complete; however, the proposed framework can be easily extended to support additional variation points and variants.

5.1 Variability Model

Feature models are a widely used method to model variability in software product line engineering. FODA-like feature models [15] have an intuitive tree-like graphical syntax and a precise formal semantics which can be denoted as propositional formulae over Boolean feature variables [2]. The features of Fig. 5 define common and variable parts among possible variants of a bigraphical abstract syntax. The root feature, called *Bigraphical Abstract Syntax (AS)*, has four sub-features constituting the main variation points. Please note that all of these four features are abstract features (the feature names are printed in italics in the feature diagram), i.e. they are merely used to structure the feature model along the main variation points.

Typing: The variation point *Typing (T)* defines two alternatives. In the *Strongly Typed (ST)* variant, the types of graph nodes representing bigraphical nodes are defined by the typing morphism. This variant is selected for our canonical definition of an abstract syntax presented in Section 4. In a *Weakly Typed (WT)* variant, controls of bigraphical nodes are simply attached as attributes to the corresponding nodes in the typed graph, while the nodes themselves are of the generic node type *BNode*. Weak typing is sensible if applications shall dynamically retype nodes representing bigraphical nodes.

Roots: Concerning the variation point *Roots (R)*, our canonical definition of an abstract syntax for bigraphs comprises the two optional sub-features *ER* and *RI*. Firstly, the feature *Explicit Roots (ER)*

denotes that bigraphical roots are explicitly represented as dedicated nodes in the typed graph. Secondly, feature *Root Indices (RI)* means that bigraphical roots are indexed in the graph-based representation, which requires that bigraphical roots are explicitly represented. The representation of root indices is necessary if a graph-based representation needs to be re-transformed into a bigraph without the loss of any information. However, if this is not required in an application context and if we are striving for a compact representation, optional features *RI* and *ER* can be omitted.

Sites and ports: Similar variants like the one discussed for the variation point *Roots (R)* exist for the graph-based representation of sites and ports, cf. variation points *Sites (S)* and *Ports (P)* in Fig. 5. Likewise, all optional sub-features, namely *Explicit Sites (ES)* and *Site Indices (SI)* as well as *Explicit Ports (ES)* and *Port Indices (PI)* are present in the canonical abstract syntax.

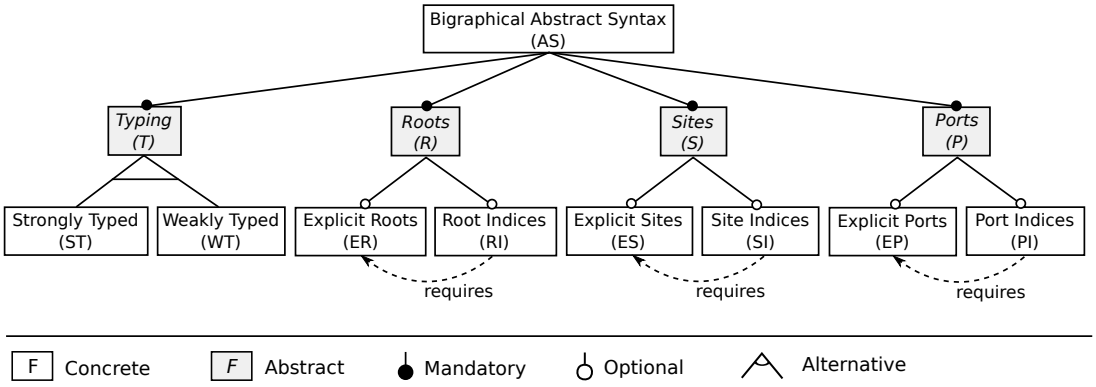

Figure 5: Variability model defining common and variable parts among possible variants of a bigraphical abstract syntax

5.2 Reconfiguring the Canonical Abstract Syntax

To reconfigure the canonical abstract syntax for bigraphs according to a selected feature configuration, suitable variability mechanisms have to be chosen on the type and on the instance level, i.e. both type and instance graphs have to be configured according to a selected feature combination.

Type-level variability. We adopt the idea of model templates [7], also known as 150% models, to establish a suitable variability mechanism on the type level; instead of having a fixed type graph derived from a bigraphical signature, we use a 150% type graph which is a superimposition of all possible variants and includes variability. A 150% type graph for our running printer example is shown in Fig. 6. It differs from the canonical type graph of Fig. 4 in that *BNode* has an additional attribute *control:String* and is defined as a subtype of *BPoint*. Moreover, some of the elements of the 150% type graph are annotated by propositional formulae over features, thus implicitly defining a mapping between features and the 150% type graph. Given a desired configuration in terms of a valid feature combination, variability is resolved as follows: Elements which are not mapped to a feature are included in any possible configuration; variable elements are included in a configuration if the annotated propositional formula over features evaluates to *true*. For example, the subtype relationship between *BNode* and *BPoint* will only be included in the final "100% type graph" if ports shall be not represented explicitly ($\neg EP$), i.e. nodes of type *BNode* can be linked to nodes of types *BEdge* and *BOuterName* directly in that case.

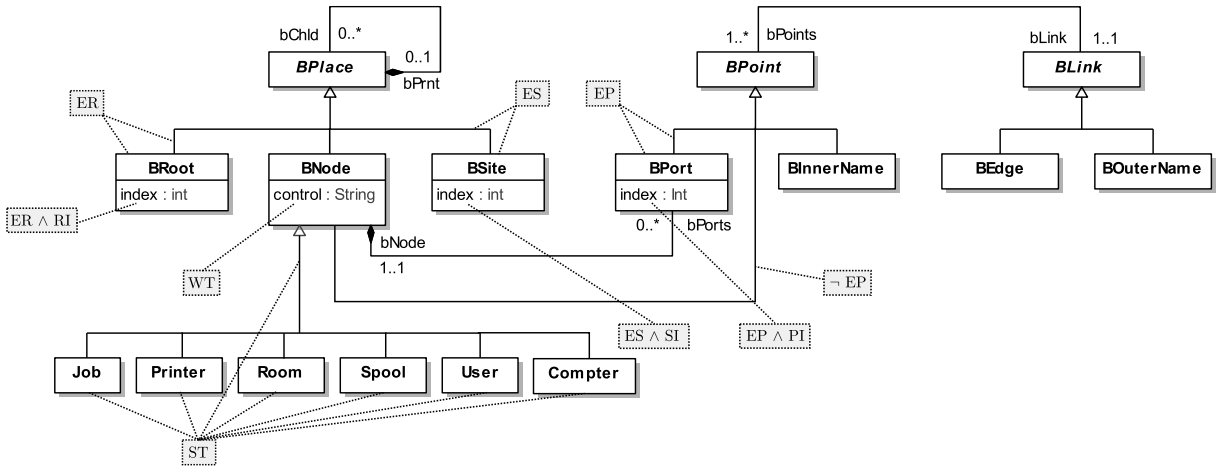

Figure 6: 150% meta-model for our running printer example

Extending the canonical type graph obtained for a given bigraphical signature to a 150% type graph is straightforward. Most of the extensions, including the respective annotations, refer to the basic type graph and can be handled in a generic way. In addition, annotation *ST* refers to each node type and the respective subtyping relationship induced by controls of the signature.

Instance-level variability. At the instance level, we use a delta-oriented approach [6, 22] as a suitable variability mechanism. The idea is that the canonical graph representation derived from a given bigraph, which can be referred to as core variant in the delta-oriented terminology, is transformed into an alternative variant by the application of a set of deltas. A delta is basically a patch removing, replacing or adding instance graph elements related to a feature. Moreover, a delta is equipped with a delta application condition. Similarly to the annotation in the 150% type graph, a delta application condition is a propositional expression over features of the feature model. A delta has to be applied to the core variant if the propositional formula evaluates to *true*. An overview of the required deltas, their application conditions (AC) and a brief description of each delta is presented in Table 1.

	AC	**Description**
$\Delta(ST, WT)$	WT	To switch from the strongly typed variant to the the weakly typed one, each node representing a bigraphical node has to be retyped to the generic node type *BNode* and the value of the attribute *control* has to be set accordingly.
$\Delta(RI, ER)$	$\neg RI$	To remove the indexing of root nodes, the attribute *index* has to be unset for each node of type *BRoot*.
$\Delta(ER, AS)$	$\neg ER$	To abstain from an explicit representation of root nodes, each node of type *BRoot* together with its incident edges has to be deleted.
$\Delta(SI, ES)$	$\neg SI$	Analogous to $\Delta(RI, ER)$.
$\Delta(ES, AS)$	$\neg ES$	Analogous to $\Delta(ER, AS)$.
$\Delta(PI, EP)$	$\neg PI$	Analogous to $\Delta(RI, ER)$.
$\Delta(EP, AS)$	$\neg EP$	Analogous to $\Delta(ER, AS)$.

Table 1: Set of deltas specifying basic reconfigurations on the instance level

6 Tool Integration and Example Application

In this section, we briefly outline how to integrate the approach into an existing MDE tool environment. We give an example application of such an integration showcasing the benefits of the proposed approach.

Integration into an MDE environment. Type and instance graphs can be implemented as meta-models and models using various technical MDE frameworks. In principle, any framework can be used as long as it supports the EMOF standard defined by the OMG. Since the available bigraphical research prototypes use tool-specific representations, input and output transformations have to be developed for interfacing with them. The mapping of bigraphs to typed graphs presented in Section 4 serves as a formal specification for this. The variability framework presented in Section 5 can be implemented using available model transformation techniques. As a proof of concept, we implemented the approach as an extension of the Eclipse IDE integrating the bigraph modeling tool Big Red [10] with the Eclipse Modeling Framework [24], the de-facto reference implementation of OMG's EMOF standard.

Example application. As an example application, we extended the bigraph modeling environment Big Red by an additional constraint checking facility which is based on the Object Constraint Language (OCL) [12]. This extension is motivated by the fact that the designer of a bigraphical domain-specific modeling language might wish to add context-sensitive constraints to the language definition restricting the set of meaningful models for the domain of interest and supporting automated constraint validation. In our running example of a context-aware printer system, we can think of the following invariants:

(iv1) A *Spool* may only contain *Jobs* and *Sites* as nested places.

(iv2) Due to capacity restrictions, a *Spool* may contain at most a pre-defined number n of Jobs. Moreover, if the maximum capacity of n *Jobs* is reached, the *Spool* may not contain a *Site* representing further unspecified *Jobs*.

(iv3) *Rooms* may be linked to each other only, i.e. they must not be linked to other nodes or outer names.

In MDE, invariants like these are typically specified using OCL expressions defined over the meta-model and used to reason on instance structures. Having an EMOF-compliant abstract syntax for bigraphs at hand, we can formally capture our sample invariants in OCL as shown below. We assume OCL expressions to be defined over the canonical meta-model of the printer example, and the capacity n of the *Spool* (iv2) to be set to 100 *Jobs*. Existing work on bigraphs does not provide a dedicated concept to express invariants like these. As a workaround, some invariants could be specified as bigraph patterns which are forbidden and use bigraph matching to verify their absence, however, very inconveniently; for invariant iv1, for instance, a bigraphical anti-pattern would consist of a *Spool* containing some node having a control different from *Job*. An anti-pattern is required for all remaining controls in the signature.

7 Conclusion and Future Work

In this paper, we presented an abstract syntax for bigraphs and reported on ongoing work to realize a complete framework for support of bigraph-enabled reasoning through state of the art model-driven engineering techniques. Our effort is motivated by the lack of a precise definition of an abstract syntax for bigraphical modeling languages, which hinders the interoperability between existing bigraphical tools, and the need to enable development support, facilitate reasoning and evaluate the bigraphical theory

```
context Spool
  inv iv1:
    self.bChld->forAll(c | c.oclIsTypeOf(BSite) or c.oclIsTypeOf(Job))
  inv iv2:
    let n : integer = 100
    self.bChld->size() <= n and
      self.bChld->size() = n implies not(self.bChld->exists( c | c.oclIsTypeOf(BSite)))
context Room
  inv iv3:
    let port : BPort = self.bPorts->first()
    port.bLink.oclIsTypeOf(BEdge) and port.bLink.bPoints->forAll(
      p | p.oclIsTypeOf(BPort) and p.oclAsType(BPort).bNode.oclIsTypeOf(Room)))
```

in real-world applications through MDE technologies and modeling environments. We proposed an abstract syntax for bigraphs which is compliant with the EMOF standard defined by the OMG. We used typed graphs as a formal underpinning of EMOF-based models and presented a canonical mapping from bigraphs to typed graphs. Since the abstract syntax of a modeling language depends on application-specific design decisions, we discussed variation points in the graph-based representation of bigraphs and showed how variability can be handled based on software product line techniques.

As for future work, to realize a complete framework, we will need to better understand the relation between bigraphical reactive systems and graph transformation systems [8]. Moreover, the mapping previously presented can be extended to other forms of bigraphs, e.g. bigraphs with sharing [23], with directed edges [1] or stochastic semantics [14]. From a technical point of view, we plan to provide a reference implementation based on Eclipse to the community, facilitating interchange of bigraphical models and enabling the integration of MDE-based techniques.

References

[1] Giorgio Bacci, Davide Grohmann & Marino Miculan (2009): *DBtk: A Toolkit for Directed Bigraphs*. In: *Proc. Intl. Conf. on Algebra and Coalgebra in Computer Science*, Springer, pp. 413–422, doi:10.1007/978-3-642-03741-2_28.

[2] Don Batory (2005): *Feature Models, Grammars, and Propositional Formulas*. In: *Proc. Intl. Conf. on Software Product Lines*, Springer, pp. 7–20, doi:10.1007/11554844_3.

[3] Enrico Biermann, Claudia Ermel & Gabriele Taentzer (2012): *Formal foundation of consistent EMF model transformations by algebraic graph transformation*. *Software & Systems Modeling* 11(2), pp. 227–250, doi:10.1007/s10270-011-0199-7.

[4] Mikkel Bundgaard & Vladimiro Sassone (2006): *Typed polyadic pi-calculus in bigraphs*. In: *Proc. Intl. Conf. on Principles and Practice of Declarative Programming*, ACM, pp. 1–12, doi:10.1145/1140335.1140336.

[5] Luca Cardelli & Andrew D. Gordon (1998): *Mobile Ambients*. In: *Proc. Intl. Conf. on Foundations of Software Science and Computation Structure*, pp. 140–155, doi:10.1007/BFb0053547.

[6] Dave Clarke, Michiel Helvensteijn & Ina Schaefer (2015): *Abstract delta modelling*. *Mathematical Structures in Computer Science* 25(3), pp. 482–527, doi:10.1017/S0960129512000941.

[7] Krzysztof Czarnecki & Michał Antkiewicz (2005): *Mapping features to models: A template approach based on superimposed variants*. In: *Proc. Intl. Conf. on Generative Programming and Component Engineering*, Springer, pp. 422–437, doi:10.1007/11561347_28.

[8] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange & Gabriele Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. Springer, doi:10.1007/3-540-31188-2.

[9] Karsten Ehrig, Jochen Malte Küster & Gabriele Taentzer (2009): *Generating instance models from meta models*. *Software & Systems Modeling* 8(4), pp. 479–500, doi:10.1007/s10270-008-0095-y.

[10] Alexander John Faithfull, Gian Perrone & Thomas T. Hildebrandt (2013): *Big Red: A Development Environment for Bigraphs*. *ECEASST* 61, doi:10.14279/tuj.eceasst.61.835.

[11] Object Management Group (2013): *Meta Object Facility (MOF) Core Specification, version 2.4.1*. OMG document number: formal/2013-06-01.

[12] Object Management Group (2014): *Object Constraint Language (OCL), version 2.4*. OMG document number: formal/2014-02-03.

[13] Reiko Heckel, Jochen Malte Küster & Gabriele Taentzer (2002): *Confluence of Typed Attributed Graph Transformation Systems*. In: *Proc. Intl. Conf. on Graph Transformation*, *Lecture Notes in Computer Science* 2505, Springer, pp. 161–176, doi:10.1007/3-540-45832-8_14.

[14] Espen Højsgaard (2012): *Bigraphical languages and their simulation*. Ph.D. thesis, IT University of Copenhagen.

[15] Kyo Kang, Sholom Cohen, James Hess, William Novak & A. Peterson (1990): *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[16] Timo Kehrer, Udo Kelter & Gabriele Taentzer (2011): *A rule-based approach to the semantic lifting of model differences in the context of model versioning*. In: *Proc. Intl. Conf. on Automated Software Engineering*, pp. 163–172, doi:10.1109/ASE.2011.6100050.

[17] Jean Krivine, Robin Milner & Angelo Troina (2008): *Stochastic bigraphs*. *Electronic Notes in Theoretical Computer Science* 218, pp. 73–96, doi:10.1016/j.entcs.2008.10.006.

[18] James Leifer & Robin Milner (2006): *Transition systems, link graphs and Petri nets*. *Mathematical Structures in Computer Science* 16(06), pp. 989–1047, doi:10.1017/S0960129506005664.

[19] Robin Milner (2006): *Pure bigraphs: Structure and dynamics*. *Information and computation* 204(1), pp. 60–122, doi:10.1016/j.ic.2005.07.003.

[20] Robin Milner (2009): *The Space and Motion of Communicating Agents*. Cambridge University Press, doi:10.1017/CBO9780511626661.

[21] Gian Perrone, Søren Debois & Thomas Hildebrandt (2012): *A model checker for bigraphs*. In: *Proc. Annual ACM Symposium on Applied Computing*, ACM, pp. 1320–1325, doi:10.1145/2245276.2231985.

[22] Christopher Pietsch, Timo Kehrer, Udo Kelter, Dennis Reuling & Manuel Ohrndorf (2015): *SiPL - A Delta-Based Modeling Framework for Software Product Line Engineering*. In: *Proc. Intl. Conf. on Automated Software Engineering*, IEEE Computer Society, pp. 852–857, doi:10.1109/ASE.2015.106.

[23] Michele Sevegnani & Muffy Calder (2015): *Bigraphs with Sharing*. *Theor. Comput. Sci.* 577, pp. 43–73, doi:10.1016/j.tcs.2015.02.011.

[24] Dave Steinberg, Frank Budinsky, Ed Merks & Marcelo Paternostro (2008): *EMF: Eclipse Modeling Framework*. Pearson Education.

[25] Christos Tsigkanos, Timo Kehrer, Carlo Ghezzi, Liliana Pasquale & Bashar Nuseibeh (2016): *Adding Static and Dynamic Semantics to Building Information Models*. In: *Proc. Intl. Workshop on Software Engineering for Smart Cyber-Physical Systems*, ACM, pp. 1–7, doi:10.1145/2897035.2897042.

[26] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi & Bashar Nuseibeh (2015): *Ariadne: Topology Aware Adaptive Security for Cyber-Physical Systems*. In: *Intl. Conf. on Software Engineering*, IEEE Computer Society, pp. 729–732, doi:10.1109/ICSE.2015.234.

[27] Mark Weiser (1991): *The computer for the 21st century*. *Scientific american* 265(3), pp. 94–104, doi:10.1145/329124.329126.