An Algebra of Hierarchical Graphs

Roberto Bruni
(joint-work with Fabio Gadducci and Alberto Lluch)

Department of Computer Science, University of Pisa
Software Engineering for Service-Oriented Overlay Computers

TGC 2010
5th Symposium on Trustworthy Global Computing
Munich (Germany), February 24-26, 2010.
Outline

Introduction

A Running Example

Hierarchical Graphs

An Algebra of Hierarchical Graphs

Conclusion
Graphs are pervasive to Computer Science

Some advantages of graphs (up to isomorphism):
▶ names are helpful but inessential;
▶ element placement is helpful but inessential;
▶ connections between elements are essential
Algebras vs Graphs in distributed systems

Goal

Flexible Graph-based representation of Service oriented systems

Mobile systems (names in $\pi$-calculus vs nodes in graphs)

Service oriented systems

sessions, transactions, ambients: which graphs for containment?
Calculi vs Graphs

**Algebraic**
- Terms
  - $a \mid b$

**Graph-based**
- Graphs (diagrams)
  - flat, hierarchical, etc.

**Elements**
Calculi vs Graphs

**Algebraic**
- Terms
  - $a \mid b$
- Operations
  - $\cdot : W \times W \rightarrow W$

**Graph-based**
- Elements
- Vocabulary
- Graphs (diagrams)
  - flat, hierarchical, etc.
- Graph compositions
  - $Union$, $tensor$, etc.
### Calculi vs Graphs

<table>
<thead>
<tr>
<th>Algebraic</th>
<th>Graph-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terms</td>
<td>Graphs (diagrams)</td>
</tr>
<tr>
<td>$a \mid b$</td>
<td><em>flat, hierarchical, etc.</em></td>
</tr>
<tr>
<td>Operations</td>
<td>Graph compositions</td>
</tr>
<tr>
<td>$\cdot \mid \cdot : \mathcal{W} \times \mathcal{W} \rightarrow \mathcal{W}$</td>
<td><em>Union, tensor, etc.</em></td>
</tr>
<tr>
<td>Axioms</td>
<td>Homomorphisms</td>
</tr>
<tr>
<td>$x \mid y \equiv y \mid x$</td>
<td><em>isomorphism, etc.</em></td>
</tr>
</tbody>
</table>

**Elements**

**Vocabulary**

**Equivalence**

**Homomorphisms**

**Transformation rules**
Calculi vs Graphs

**Algebraic**
- Terms
  \[ a | b \]
- Operations
  \[ \cdot | : W \times W \rightarrow W \]
- Axioms
  \[ x | y \equiv y | x \]
- Rewrite rules
  \[ a \rightarrow b \]

**Graph-based**
- Elements
- Vocabulary
- Equivalence
- Dynamics
- Graphs (diagrams)
  - \textit{flat, hierarchical, etc.}
- Graph compositions
  - \textit{Union, tensor, etc.}
- Homomorphisms
  - \textit{isomorphism, etc.}
- Transformation rules
Which graphs?
Which graphs?
Which graphs?
Which graphs?
Main technical problem: representation distance

Definition 15 (processes). Let $\mathcal{U}$ be a set of names. A process $P$ is a term generated by the syntax

\[
P := 0 \mid M \mid (\nu a)P \mid P \mid P
\]

\[M := M + M \mid A.P\]

where $a, b \in U$.

Grammar, structural congruence, etc.

Definition 15 (processes). Let $\mathcal{U}$ be a set of names. A process $P$ is a term generated by the syntax

\[
P := 0 \mid M \mid (\nu a)P \mid P \mid P
\]

\[M := M + M \mid A.P\]

where $a, b \in U$.

Adjacency matrix, tuples, sets, morphisms

Definition 22 (bigraph). The bigraph $G = (V, c, (G^T, G^M)) : I \rightarrow J$ is a triple of

1. $V : \text{nodes}$
2. $c : \text{classes}$
3. $E = \{E^1, \ldots, E^n\}$
4. $\alpha : V \rightarrow \text{ordinals}$

where $I = (m, X)$ and $J = (n, Y)$ and each $E_n$ is a set of edges, $N_G$ is the set of nodes, and $\alpha$ is a bijection.

Definition 7 (morphisms). A hypergraph $G$ is a triple $\langle E_G, V_G, t_G \rangle$, where $E_G$ is the set of edges, $V_G$ is the set of nodes, and $t_G : E_G \rightarrow V_G$ is the range function.

Let $G, H$ be hypergraphs. A (hypergraph) morphism $f : G \rightarrow H$ is a pair of functions $f_E : E_G \rightarrow E_H$, $f_N : N_G \rightarrow N_H$ preserving the tentacle function.
Main technical problem: representation distance

solution: graph algebras

\[
\begin{align*}
\llbracket (\nu a)P \rrbracket_r &= \begin{cases} 
\llbracket P \rrbracket_r & \text{if } a \notin \text{fn}(P) \\
(id_r \otimes \nu a \otimes id_r) \circ \llbracket P^{\{c\}/a} \rrbracket_{\Gamma} & \text{otherwise}
\end{cases} \\
\llbracket P \mid Q \rrbracket_r &= \llbracket P \rrbracket_r \otimes \llbracket Q \rrbracket_r \\
\llbracket a(b).P \rrbracket_r &= (in_{a,c} \otimes id_r) \circ \llbracket P^{\{c\}/a} \rrbracket_{\Gamma} \\
\llbracket 0 \rrbracket_r &= 0_r \otimes 0_r \\
\llbracket a_b.P \rrbracket_r &= (out_{a,b} \otimes id_r) \circ \llbracket P \rrbracket_r
\end{align*}
\]

\[
\begin{align*}
\llbracket 0 \rrbracket_X &= 1_X \\
\llbracket P \mid Q \rrbracket_X &= \llbracket P \rrbracket_X \land \llbracket Q \rrbracket_X \\
\llbracket (x).P \rrbracket_X &= \bigtriangleup_x \circ \llbracket P \rrbracket_{X \cup \{x\}} \\
\llbracket x.z.P \rrbracket_X &= \text{get}^{x,z} \circ \llbracket P \rrbracket_X \\
\llbracket x.z.P \rrbracket_X &= \text{send}^{x,z} \circ \llbracket P \rrbracket_X
\end{align*}
\]

\[
\begin{align*}
\llbracket (\nu a)P \rrbracket_n &= \text{nil}_{\text{deg}(\llbracket P^{\{\cdots, \cdots \}/a} \rrbracket_{n+1})} \\
\llbracket (P \mid Q) \rrbracket_n &= \text{par}^n(\llbracket P \rrbracket_n, \llbracket Q \rrbracket_n) \\
\llbracket i(y).P \rrbracket_n &= \text{in}_{i,n}(\llbracket P^{\{n+1/y\}}_{n+1} \rrbracket_{n+1}) \\
\llbracket 0 \rrbracket_n &= \text{nil}_n \\
\llbracket M + N \rrbracket_n &= \text{choice}^n(\llbracket M \rrbracket_n, \llbracket N \rrbracket_n)
\end{align*}
\]

Definition 22 (bigraph) Let \( I = (m, X) \) and \( J = (n, Y) \) be ordered sets, then a bigraph \( G = (E_G, N_G, t_G) \) is a triple such that \( E_G \) is the set of edges, \( N_G \) is the set of nodes, and \( t_G : E_G \to N_G^* \) is the tentacle function.

Let \( G, H \) be hypergraphs. A (hypergraph) morphism \( f : G \to H \) is a pair of functions \( f_E : E_G \to E_H, f_N : N_G \to N_H \) preserving the tentacle function.
Main result: a flexible, general intermediate language

- workflow language
- process calculus
- architecture description language
- etc.

- nested graphs
- gs-graphs
- bigraphs
- etc.
Main result: a flexible, general intermediate language

- workflow language
- process calculus
- architecture description language
- suitable graph algebra
- nested graphs
- gs-graphs
- bigraphs
- etc.
- etc.
Outline

Introduction

A Running Example

Hierarchical Graphs

An Algebra of Hierarchical Graphs

Conclusion
Running Example: Long running transactions

We shall consider a simple language for transactions with

- sequential composition;
- parallel (split-join) composition;
- compensating activity;
- scope of compensation.

Analogous to the *Nested Sagas* of [BMM05].
Process terms and their graphical representations

\[ \text{task1} \rightarrow \text{task2} \rightarrow \text{task3} \]

\text{task1} ; \text{task2} ; \text{task3}
Process terms and their graphical representations

task1 ; task2 ; task3

task1 | task2 | task3
Process terms and their graphical representations

task1 ; task2 ; task3

ordinary flow %
compensation flow

task1 | task2 | task3
Process terms and their graphical representations

1. task1
2. task2
3. task3

- Ordinary flow %
- Compensation flow

[Nested flow]
Main technical goal: mapping coherent wrt. equivalence

flow1

a
| b
| [ c % d ]

graph1

graph2
Main technical goal: mapping coherent wrt. equivalence

**flow1**

- a
  - b
  - [ c % d ]

**flow2**

- b
  - [ c % d ]
  - a

**graph1**
Main technical goal: mapping coherent wrt. equivalence

flow1

a
| b
| [ c % d ]

flow2

b
| [ c % d ]
| a

graph1

graph2
Outline

Introduction

A Running Example

Hierarchical Graphs

An Algebra of Hierarchical Graphs

Conclusion
Graph layers

\( \mathcal{N} \) universe of nodes
\( \mathcal{A} = \mathcal{A}_\mathcal{E} \uplus \mathcal{A}_\mathcal{D} \) universe of edges

The set \( \mathcal{L} \) of graph layers is the set of tuples \( G = \langle N_G, E_G, t_G, F_G \rangle \) where

1. \( E_G \subseteq \mathcal{A} \) is a (finite) set of edges,
2. \( N_G \subseteq \mathcal{N} \) a (finite) set of nodes,
3. \( t_G : E_G \to N_G^* \) a tentacle function, and
4. \( F_G \subseteq N_G \) a set of free nodes.

The set \( \mathcal{P} \subseteq \mathcal{L} \) of plain graphs contains those graph layers \( G \) such that \( E_G \subseteq \mathcal{A}_\mathcal{E} \).
(standard notion of hypergraph plus a chosen set of free nodes)
Hierarchical graphs

The set $\mathcal{H}$ of \textit{hierarchical graphs} is the least set containing all the tuples $G = \langle N_G, E_G, t_G, i_G, x_G, r_G, F_G \rangle$ where

1. $\langle N_G, E_G, t_G, F_G \rangle$ is a graph layer;
2. $i_G : E_G \cap \mathcal{A}_D \rightarrow \mathcal{H}$ (embedding function);
3. $x_G : E_G \cap \mathcal{A}_D \rightarrow \mathcal{N}^*$ (exposure function), s.t. for all $e \in E_G \cap \mathcal{A}_D$
   
   \begin{enumerate}
   \item $|x_G(e)| = |t_G(e)|$, (same arity for exposure and tentacle func.)
   \item $\forall n, m \in \mathbb{N}, \ x_G(e)[n] = x_G(e)[m] \text{ iff } t_G(e)[n] = t_G(e)[m]$;
   \end{enumerate}
4. $r_G : E_G \cap \mathcal{A}_D \rightarrow (N_G \leftrightarrow \mathcal{N})$ is a renaming function, s.t. for all $e \in E_G \cap \mathcal{A}_D$, $r_G(e)(N_G) = F_{i_G(e)}$.

(for this talk, we can assume $r_G(e)$ is the ordinary inclusion)
A hierarchical graph and its simplified representation

A graph layer (free nodes $x$ and $y$)
A hierarchical graph and its simplified representation

embedding function $i_G$
A hierarchical graph and its simplified representation

exposure function $x_G$
A hierarchical graph and its simplified representation

renaming function $r_G$
A hierarchical graph and its simplified representation

informal notation (free nodes $x$ and $y$)
Hierarchical graph isomorphism

The actual model of hierarchical graphs has a suitable notion of isomorphism.
Hierarchical graph isomorphism

The actual model of hierarchical graphs has a suitable notion of isomorphism.
Hierarchical graph morphism (formally)

Let $G$, $H$ be graphs such that $F_G \subseteq F_H$.

A graph morphism $\phi : G \to H$ is a tuple $\langle \phi_N, \phi_E, \phi_I \rangle$ where

1. $\phi_N : N_G \to N_H$ is a node morphism,
2. $\phi_E : E_G \to E_H$ an edge morphism, and
3. $\phi_I = \{ \phi^e \mid e \in E_G \cap \mathcal{A}_D \}$ a family of graph morphisms $\phi^e : i_G(e) \to i_H(\phi_E(e))$ such that
   3.1 $\forall e \in E_G, \phi_N(t_G(e)) = t_H(\phi_E(e))$, i.e. the tentacle function is respected;
   3.2 $\forall e \in E_G \cap \mathcal{A}_D, \phi^e_N(x_G(e)) = x_H(\phi_E(e))$, i.e. the exposure function is respected;
   3.3 $\forall e \in E_G \cap \mathcal{A}_D, \forall n \in N_G, \phi^e_N(r_G(e)(n)) = r_H(\phi_E(e))(\phi_N(n))$, i.e. the renaming function is respected;
   3.4 $\forall n \in F_G, \phi_N(n) = n$, i.e. the free nodes are preserved.
Outline

Introduction

A Running Example

Hierarchical Graphs

An Algebra of Hierarchical Graphs

Conclusion
The syntax of the graph algebra

\[ G, H ::= 0 \]

the empty graph
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \]

a node called \( x \)
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \mid l\langle \bar{x} \rangle \]

an (hyper)edge labelled with \( l \) attached to \( \bar{x} \)

for instance, \( a\langle p, q, r \rangle \)
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \mid l\langle x \rangle \mid G \parallel H \]

parallel composition: disjoint union up to common nodes

for instance, \( a\langle p, q, r \rangle \parallel a\langle p, q, r \rangle \)
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \mid l\langle x \rangle \mid G \parallel H \]

parallel composition: disjoint union up to common nodes

for instance, \( a\langle p, q, r \rangle \mid a\langle p, q, r \rangle \)
The syntax of the graph algebra

\[ \mathcal{G}, \mathcal{H} ::= \ 0 \ | \ x \ | \ l\langle x \rangle \ | \ G\mathcal{H} \ | \ (\nu x)\mathcal{G} \]

declaration of a new node \( x \)

for instance, \((\nu s) \ (a\langle p, s, r \rangle \ | \ b\langle s, q, r \rangle)\)
The syntax of the graph algebra

\[ D ::= L_x[G] \]

\[ G, H ::= 0 \mid x \mid l\langle x\rangle \mid G \mid H \mid (\nu x)G \]

graph \( G \) with interface of type \( L \) exposing \( \overline{x} \)

for instance, \( S_{p, q, s}[(\nu r)\text{flow}\langle p, q, r, q, s\rangle] \)
The syntax of the graph algebra

\[
\mathcal{D} ::= \quad L_x[G] \\
G, H ::= \quad 0 \mid x \mid I\langle x \rangle \mid G|H \mid (\nu x)G \mid \mathcal{D}\langle \overline{y} \rangle
\]

a nested graph attached to \(\overline{y}\)

for instance, \(\mathcal{D}\langle a, b, c \rangle\)
The syntax of the graph algebra

\[ D ::= L_x[G] \]
\[ G, H ::= 0 \mid x \mid l\langle x \rangle \mid G \mid H \mid (\nu x)G \mid D\langle y \rangle \]

a nested graph attached to \( y \)

for instance, \( D\langle a, b, c \rangle \), with \( D = S_{p,q,s}[(\nu r)\text{flow}\langle p, q, r, q, s \rangle] \)
Structural congruence axioms

Isomorphism is elegantly captured by structural axioms.

\[ G \parallel H \equiv H \parallel G \quad \text{(DA1)} \]

\[ G \parallel (H \parallel I) \equiv (G \parallel H) \parallel I \quad \text{(DA2)} \]

\[ G \parallel 0 \equiv G \quad \text{(DA3)} \]
Structural congruence axioms

Isomorphism is elegantly captured by structural axioms.

\[ \begin{align*}
G | H & \equiv H | G \quad \text{(DA1)} \\
G | (H | I) & \equiv (G | H) | I \quad \text{(DA2)} \\
G | 0 & \equiv G \quad \text{(DA3)}
\end{align*} \]

\[ \begin{align*}
(\nu x)(\nu y)G & \equiv (\nu y)(\nu x)G \quad \text{(DA4)} \\
(\nu x)0 & \equiv 0 \quad \text{(DA5)} \\
G | (\nu x)H & \equiv (\nu x)(G | H) \quad \text{if } x \not\in \text{fn}(G) \quad \text{(DA6)}
\end{align*} \]
Structural congruence axioms

Isomorphism is elegantly captured by structural axioms.

\[
\begin{align*}
G | H & \equiv H | G \quad \text{(DA1)} \\
G | (H | I) & \equiv (G | H) | I \quad \text{(DA2)} \\
G | 0 & \equiv G \quad \text{(DA3)}
\end{align*}
\]

\[
\begin{align*}
(\nu x)(\nu y)G & \equiv (\nu y)(\nu x)G \quad \text{(DA4)} \\
(\nu x)0 & \equiv 0 \quad \text{(DA5)} \\
G | (\nu x)H & \equiv (\nu x)(G | H) \quad \text{if } x \notin \text{fn}(G) \quad \text{(DA6)}
\end{align*}
\]

\[
\begin{align*}
L_x[G] & \equiv L_y[G\{\bar{y}/x\}] \quad \text{if } |\bar{y}| \cap \text{fn}(G) = \emptyset \quad \text{(DA7)} \\
(\nu x)G & \equiv (\nu y)G\{\bar{y}/x\} \quad \text{if } y \notin \text{fn}(G) \quad \text{(DA8)}
\end{align*}
\]

Axioms DA1–DA8 are rather standard and thus intuitive to those familiar with (nominal) process calculi.
Structural congruence axioms

Isomorphism is elegantly captured by structural axioms.

\[ G | H \equiv H | G \quad \text{(DA1)} \]
\[ G | (H | I) \equiv (G | H) | I \quad \text{(DA2)} \]
\[ G | 0 \equiv G \quad \text{(DA3)} \]
\[ (νx)(νy)G \equiv (νy)(νx)G \quad \text{(DA4)} \]
\[ (νx)0 \equiv 0 \quad \text{(DA5)} \]
\[ G | (νx)H \equiv (νx)(G | H) \quad \text{if} \ x \notin fn(G) \quad \text{(DA6)} \]
\[ L_x[G] \equiv L_y[G\{y/x\}] \quad \text{if} \ |y| \cap fn(G) = \emptyset \quad \text{(DA7)} \]
\[ (νx)G \equiv (νy)G\{y/x\} \quad \text{if} \ y \notin fn(G) \quad \text{(DA8)} \]
\[ x | G \equiv G \quad \text{if} \ x \in fn(G) \quad \text{(DA9)} \]
\[ L_x[z | G]\langle y \rangle \equiv z | L_x[G]\langle y \rangle \quad \text{if} \ z \notin |x| \quad \text{(DA10)} \]
Structural congruence axioms

Isomorphism is elegantly captured by structural axioms.

$$
G \mid H \equiv H \mid G \quad (DA1)
$$

$$
G \mid (H \mid I) \equiv (G \mid H) \mid I \quad (DA2)
$$

$$
G \mid 0 \equiv G \quad (DA3)
$$

$$
(\nu x)(\nu y)G \equiv (\nu y)(\nu x)G \quad (DA4)
$$

$$
(\nu x)0 \equiv 0 \quad (DA5)
$$

$$
G \mid (\nu x)H \equiv (\nu x)(G \mid H) \quad \text{if } x \notin \text{fn}(G) \quad (DA6)
$$

$$
L_x[G] \equiv L_y[G\{\bar{y}/x\}] \quad \text{if } |\bar{y}| \cap \text{fn}(G) = \emptyset \quad (DA7)
$$

$$
(\nu x)G \equiv (\nu y)G\{y/x\} \quad \text{if } y \notin \text{fn}(G) \quad (DA8)
$$

$$
x \mid G \equiv G \quad \text{if } x \in \text{fn}(G) \quad (DA9)
$$

$$
L_x[z \mid G]\langle y \rangle \equiv z \mid L_x[G]\langle y \rangle \quad \text{if } z \notin [\bar{x}] \quad (DA10)
$$

Axioms DA1–DA8 are rather standard and thus intuitive to those familiar with (nominal) process calculi.
The encoding $\llbracket \cdot \rrbracket$, mapping (well-formed) terms into graphs, is the function inductively defined as (letting $\llbracket G \rrbracket = \langle N_G, E_G, t_G, i_G, x_G, r_G, F_G \rangle$)

\[
\begin{align*}
\llbracket 0 \rrbracket &= \langle \emptyset, \emptyset, \bot, \bot, \bot, \bot, \emptyset \rangle \\
\llbracket x \rrbracket &= \langle \{x\}, \emptyset, \bot, \bot, \bot, \{x\} \rangle \\
\llbracket I\langle x \rangle \rrbracket &= \langle \{x\}, \{e\}, e \mapsto x, \bot, \bot, \bot, \llbracket x \rrbracket \rangle \\
\llbracket G \mid H \rrbracket &= [G] \oplus [H] \\
\llbracket (\nu x)G \rrbracket &= \langle N_G, E_G, t_G, i_G, x_G, r_G, F_G \setminus x \rangle \\
\llbracket L_x[G]\langle \bar{y} \rangle \rrbracket &= \langle N_G, \{e'\}, e' \mapsto \bar{y}, e' \mapsto [G] \oplus [\bar{y}], e' \mapsto \bar{x}, e' \mapsto id_{N_G}, (F_G \setminus [\bar{x}]) \cup [\bar{y}] \rangle
\end{align*}
\]

where $e \in A_I$ and $e' \in A_L$. 

Encoding
Main Result

It is worth to remark that the encoding is surjective, i.e. every graph can be denoted by a term of the algebra.

**Theorem**

Let $G$ be a graph. Then, there exists a well-formed term $G$ generated by the design algebra such that $G$ is isomorphic to $[G]$.

Moreover, our encoding is sound and complete, meaning that equivalent terms are mapped to isomorphic graphs and vice versa.

**Theorem**

Let $G_1$, $G_2$ be well-formed terms generated by the design algebra. Then, $G_1 \equiv G_2$ if and only if $[G_1]$ is isomorphic to $[G_2]$. 
Concluding remarks

The approach...

- Grounds on widely-accepted models;
- Simplifies the graphical representation of complex systems;
- Hides the complexity of hierarchical graphs;
- Enables proofs by structural induction;
- Has been evaluated on various kinds of languages;
- Nesting and sharing features suitable for modelling SOC features such as transactions or sessions;
- Experimental implementation in RL/Maude (support for theorem proving, model checking, simulation, etc.);
- Offers a technique for complementing textual and visual notations in formal tools.
Visualizer: adr2graphs

adr2graphs

*a simple visualiser of algebraic specifications*

1) Choose the input language: network topologies (alpha)
2) Choose the output format: formal hierarchical graph
3) Enter a term in the box below.

```
< host % host("a") ; host >
< host % { host ; host("a") } >
```

1 Use the following syntax (blanks are mandatory)

Bus ::= Host (single host)
Bus | Bus (bus union)
{ Bus } (bus nesting)

Line ::= Host (single host)
Line ; Line (line concatenation)
{ Line } (line nesting)

Ring ::= < Line > (closed line)
{ Ring } (ring nesting)

Host ::= host (disconnected host)
host(x) (connected host)

where x is a channel name given as a doubly-quoted string

example: host | < host % host("a") ; host > | < host % { host ; host("a") } >

Done
Related work

GS-Graphs [CG99, FM00]
- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)

Ranked Graphs [Gad03]
- node sharing, calculi encoding
- no composition interface, flat

Hierarchical Graphs [DHP02]
- basic model, composition interface
- no node sharing, no algebraic syntax
Related work

GS-Graphs [CG99, FM00]
- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)

Ranked Graphs [Gad03]
- node sharing, calculi encoding
- no composition interface, flat
Related work

GS-Graphs [CG99, FM00]
- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)

Ranked Graphs [Gad03]
- node sharing, calculi encoding
- no composition interface, flat

Hierarchical Graphs [DHP02]
- basic model, composition interface
- no node sharing, no algebraic syntax
Related Work

Bigraphs [JM03]

- nesting + linking
- 2 overlapping structures, complex syntax, no composition interface, flat
Related Work

Bigraphs [JM03]
- nesting + linking
- 2 overlapping structures, complex syntax, no composition interface, flat

Graph Algebra, SHR [CMR94]
- basic algebra
- flat, no composition interface
Credits and references I

[BMM05] Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari.
Theoretical foundations for compensations in flow composition languages.

[CG99] Andrea Corradini and Fabio Gadducci.
An algebraic presentation of term graphs, via gs-monoidal categories. applied categorical structures.

An abstract machine for concurrent modular systems: CHARM.

[DHP02] Frank Drewes, Berthold Hoffmann, and Detlef Plump.
Hierarchical graph transformation.

Tile formats for located and mobile systems.

[Gad03] Fabio Gadducci.
Term graph rewriting for the pi-calculus.
In Atsushi Ohori, editor, *Proceedings of the 1st Asian Symposium on Programming Languages and

Bigraphs and mobile processes.

Note: Some figures have been borrowed from the referred papers.