An algebra of hierarchical graphs
(and its applications)

Alberto Lluch-Lafuente
(based on a collaboration Pisa/Leicester within the Sensoria project)

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

Job Market Seminars
IMT Lucca, May 25 2009
Outline

Introduction
On structural issues
A simple scenario
Goal statement

An algebra of hierarchical graphs
A syntax for hierarchical graphs
Identifying equivalent graphs
Expressing typical structures
Hiding the complexity of hierarchical graphs

Conclusion
The structure of data, programs and all that

We observe 1) composition, 2) containment and 3) references.
The structure of data, programs and all that

We observe 1) composition, 2) containment and 3) references.

- Programs (e.g. Pascal)
  1. control flow
  2. scopes
  3. variables
The structure of data, programs and all that

We observe 1) composition, 2) containment and 3) references.

- Programs (e.g. Pascal)
  1. control flow
  2. scopes
  3. variables

- Data (e.g. XML)
  1. element list
  2. tag hierarchy
  3. references
The structure of data, programs and all that

We observe 1) composition, 2) containment and 3) references.

- Programs (e.g. Pascal)
  1. control flow
  2. scopes
  3. variables

- Data (e.g. XML)
  1. element list
  2. tag hierarchy
  3. references

- Other examples:
  - file system navigation
  - workflows (BPEL)
  - diagrams (UML)
  - etc.
The structure of *modern* data, programs and all that

Modern systems increase the relevance of containment and the interplay with composition and references becomes more subtle.
The structure of *modern* data, programs and all that

Modern systems increase the relevance of containment and the interplay with composition and references becomes more subtle.

*E.g. Nested...*

▶ Transactions

![Diagram of transaction flow]
The structure of *modern* data, programs and all that

Modern systems increase the relevance of containment and the interplay with composition and references becomes more subtle.

*E.g.* *Nested*...

- Transactions
- Locations
The structure of *modern* data, programs and all that

Modern systems increase the relevance of containment and the interplay with composition and references becomes more subtle.

*E.g.* Nested...

- Transactions
- Locations
- Sessions
The structure of *modern* data, programs and all that

Modern systems increase the relevance of containment and the interplay with composition and references becomes more subtle.

*E.g.* *Nested*...

- Transactions
- Locations
- Sessions
- Membranes
- Etc.
Outline

Introduction
- On structural issues
- A simple scenario
- Goal statement

An algebra of hierarchical graphs
- A syntax for hierarchical graphs
- Identifying equivalent graphs
- Expressing typical structures
- Hiding the complexity of hierarchical graphs

Conclusion
Networking scenario

Let us consider a simple networking scenario with some *structure*:

- topology (e.g. line, bus, ring, etc.)
- nesting (e.g. home sub-network, etc.)
- references (e.g. file sharing, services, etc.)
Networking scenario: visual approach

bus
Networking scenario: visual approach

bus

line
Networking scenario: visual approach

bus

ring

table

line
Networking scenario: visual approach

- Bus
- Ring
- Line
- Subnet
Networking scenario: visual approach

bus

ring

line

subnet
Networking scenario: visual approach

- bus + refs
- ring + refs
- line + refs
- subnet + refs
Networking scenario: textual approach

host | host | host
| host | host
Networking scenario: textual approach

```plaintext
host | host | host
    | host | host
```

```
host ; host ; host
```
Networking scenario: textual approach

host | host | host
| host | host

< host ; host ; host ; host ; host >
Networking scenario: textual approach

host | host | host
| host | host
< host ; host ; host ; host ; host >

host ; host ; host ; host

[ host ; host ]
Networking scenario: textual approach

< host ; host(a) ; host ; host(a) ; host >
Outline

Introduction
   On structural issues
   A simple scenario
   Goal statement

An algebra of hierarchical graphs
   A syntax for hierarchical graphs
   Identifying equivalent graphs
   Expressing typical structures
   Hiding the complexity of hierarchical graphs

Conclusion
Two trends to formal textual and visual specifications

### Algebraic
- Terms
  - $\text{host}(a) \mid \text{host}(b)$

### Graph-based
- Graphs (diagrams)
  - flat, hierarchical, etc.
Two trends to formal textual and visual specifications

**Algebraic**
- Terms
  \[ \text{host}(a) \mid \text{host}(b) \]
- Operations
  \[ \cdot : \text{Bus} \times \text{Bus} \rightarrow \text{Bus} \]

**Graph-based**
- Elements
- Vocabulary
- Graphs (diagrams)
  - flat, hierarchical, etc.
- Graph compositions
  - Union, tensor, etc.
Two trends to formal textual and visual specifications

**Algebraic**
- Terms
  \[ \text{host}(a) \mid \text{host}(b) \]
- Operations
  \[ \cdot \mid \cdot : \text{Bus} \times \text{Bus} \rightarrow \text{Bus} \]
- Axioms
  \[ x \mid y \equiv y \mid x \]

**Graph-based**
- Graphs (diagrams)
  - flat, hierarchical, etc.
- Graph compositions
  - Union, tensor, etc.
- Homomorphisms
  - isomorphism, etc.
## Two trends to formal textual and visual specifications

<table>
<thead>
<tr>
<th>Algebraic</th>
<th>Graph-based</th>
</tr>
</thead>
</table>
| - Terms   | - Graphs (diagrams)  
 | \( \text{host}(a) \mid \text{host}(b) \) |  
 | - Operations | - flat, hierarchical, etc.  
 | \( \cdot \mid \cdot : \text{Bus} \times \text{Bus} \to \text{Bus} \) |  
 | - Axioms | - Graph compositions  
 | \( x \mid y \equiv y \mid x \) | - Union, tensor, etc.  
 | - Rewrite rules | - Homomorphisms  
 | \( \text{host}(x) \longrightarrow \text{host} \) | - isomorphism, etc.  
 | | - Transformation rules |
Goal statement

The spirit of our research is

"to conciliate algebraic and graph-based specifications"
Goal statement

The spirit of our research is
"to conciliate algebraic and graph-based specifications"

The work presented in this talk has the goal to
"Equip algebraic specifications with a graphical representation that is

- Intuitive
- Easy to define
- Easy to prove correct
Main technical goal: mapping coherent wrt. equivalence

network1

host(a)
| host
| [ host | host(a)]
Main technical goal: mapping coherent wrt. equivalence

network1

```
host(a)
| host
| [ host | host(a)]
```
Main technical goal: mapping coherent wrt. equivalence

network1

host(a)

| host
| [ host | host(a)]

network2

host

| [ host | host(a)]
| host(a)

graph1

congruent
Main technical goal: mapping coherent wrt. equivalence

network1
host(a)
| host
| [ host | host(a)]

congruent

network2
host
| [ host | host(a)]
| host(a)

graph1

graph2
Main technical goal: mapping coherent wrt. equivalence
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

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

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

where $a, b \in \mathcal{U}$.

Grammar, structural congruence, etc.

Very different syntax!

**Definition 22 (bigraph).** Let $I = (m, X)$ and $J = (n, Y)$ be two ordered sets. A bigraph is a triple $(E_G, \text{ctrl}, G^T, G^M) : I \to J$ where $E_G$ is the set of each edge in the bigraph, and $\text{ctrl}$ is a function combining a width (a finite morphisms). A hypergraph $G$ is a triple $\langle E_G, \text{ctrl}, G^T, G^M \rangle : I \to J$,

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.

Adjacency matrix, tuples, sets, morphisms
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

\[
P := 0 \mid M \mid (va)P \mid P \cdot P
\]

where \( a, b \in \mathcal{U} \).

solution: graph algebras

\[
\| (va)P \|_R = \begin{cases} \| P \|_R & \text{if } a \notin \text{fn}(P) \\ \left( \text{id}_P \otimes \nu_v \otimes \text{id}_R \right) \circ \| P \{^c/_{a} \} \|_{\omega R} & \text{otherwise} \end{cases}
\]

\[
\| P \cdot Q \|_R = \| P \|_R \otimes \| Q \|_R
\]

\[
\| a(b).P \|_R = \left( \text{in}_{a,c} \otimes \text{id}_R \right) \circ \| P \{^c/_{b} \} \|_{\omega R}
\]

\[
\| 0 \|_R = 0_P \otimes 0_R
\]

\[
[0]_X = 1 \otimes X
\]

\[
[x]_X = \text{get}_X \circ \| P \|_X
\]

\[
\exists x.P)_X = \text{send}_X \circ \| P \|_X
\]

\[
\| (\nu a)P \|_n = \text{niden}_n([1 \cdots / a]_{n+1})
\]

\[
[P \cdot Q]_n = \text{par}_n([P]_n, [Q]_n)
\]

\[
[0]_n = \text{nil}_n
\]

\[
[i(y).P]_n = \text{in}_{i,n}([P]_{n+1}/y)_{n+1}
\]

\[
[M + N]_n = \text{choice}_n([M]_n, [N]_n)
\]

**Definition 22 (bighraph)**

where: \( I = (m, X) \) and \( J = (n, Y) \) are ordered pairs of each node \( i \) and \( j \) in the bighraph.

**Definition 7 (hypergraph)**

a triple \( (E_G, N_G, t_G) \) 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.
Outline

Introduction
On structural issues
A simple scenario
Goal statement

An algebra of hierarchical graphs
A syntax for hierarchical graphs
Identifying equivalent graphs
Expressing typical structures
Hiding the complexity of hierarchical graphs

Conclusion
The syntax of the graph algebra

\[ G, H ::= 0 \]

the empty graph
The syntax of the graph algebra

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

\[ x \]

\[ a \text{ node called } x \]
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \mid t(\overline{x}) \]

an edge labelled with \( t \) attached to \( \overline{x} \)
The syntax of the graph algebra

\[ G, H :::= \ 0 \ | \ x \ | \ t(x) \ | \ G \ || \ H \]

parallel composition: disjoint union up to common nodes
The syntax of the graph algebra

\[ G, H ::= 0 \mid x \mid t(x) \mid G \parallel H \mid (\nu x)G \]

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

\[
D \ ::= \ T_{\overline{x}}[G] \\
G, H \ ::= \ 0 \mid x \mid t(\overline{x}) \mid G \parallel H \mid (\nu x)G
\]

graph $G$ with interface of type $T$ exposing $\overline{x}$
The syntax of the graph algebra

\[ D ::= T_x[G] \]
\[ G, H ::= 0 | x | t(x) | G || H | (\nu x)G | D(y) \]

a nested graph attached to \( \bar{y} \)
The syntax of the graph algebra

\[ D ::= T_{\bar{x}}[G] \]
\[ G, H ::= 0 \mid x \mid t(\bar{x}) \mid G \mid H \mid (\nu x)G \mid D(\bar{y}) \]

a nested graph attached to $\bar{y}$
Outline

Introduction
On structural issues
A simple scenario
Goal statement

An algebra of hierarchical graphs
A syntax for hierarchical graphs
Identifying equivalent graphs
Expressing typical structures
Hiding the complexity of hierarchical graphs

Conclusion
Hierarchical graph isomorphism
Hierarchical graph isomorphism
Structural axioms characterise graph isomorphism

\[
\begin{align*}
G || H & \equiv H || G \quad \text{(PARALLEL1)} \\
G || (H || I) & \equiv (G || H) || I \quad \text{(PARALLEL2)}
\end{align*}
\]
Structural axioms characterise graph isomorphism

\[
\begin{align*}
G \parallel H & \equiv H \parallel G \quad \text{(PARALLEL1)} \\
G \parallel (H \parallel I) & \equiv (G \parallel H) \parallel I \quad \text{(PARALLEL2)} \\
G \parallel 0 & \equiv G \quad \text{(NODES1)} \\
(\nu x)(\nu y)G & \equiv (\nu y)(\nu x)G \quad \text{(NODES2)} \\
(\nu x)0 & \equiv 0 \quad \text{(NODES5)} \\
(\nu x)G & \equiv (\nu y)G\{y/x\} \quad \text{if } y \notin \text{fn}(G) \quad \text{(NODES3)} \\
L_x[G] & \equiv L_y[G\{y/x\}] \quad \text{if } |y| \cap \text{fn}(G) = \emptyset \quad \text{(NODES4)} \\
G \parallel (\nu x)H & \equiv (\nu x)(G \parallel H) \quad \text{if } x \notin \text{fn}(G) \quad \text{(NODES5)} \\
L_x[(\nu y)G](\bar{z}) & \equiv (\nu y)L_x[G](\bar{z}) \quad \text{if } y \notin |\bar{x}| \cup |\bar{z}| \quad \text{(NODES6)} \\
x \parallel G & \equiv G \quad \text{if } x \in \text{fn}(G) \quad \text{(NODES7)}
\end{align*}
\]

These axioms are rather standard and thus intuitive to those familiar with algebraic specifications.
Outline

Introduction
  On structural issues
  A simple scenario
  Goal statement

An algebra of hierarchical graphs
  A syntax for hierarchical graphs
  Identifying equivalent graphs
  Expressing typical structures
  Hiding the complexity of hierarchical graphs

Conclusion
Typical structures are derived operators

$[X] \overset{\text{def}}{=} SubBus_p[X(p)]$, with $X : Bus$
Typical structures are derived operators

(network) parallel composition

\[ X \parallel Y \overset{\text{def}}{=} Bus_p[X(p)\parallel Y(p)] \]

Axiom \( Bus_x[G](\overline{y}) \equiv G\{\overline{y}/x\} \) gets rid of associativity and commutativity.
Typical structures are derived operators

(network) sequential composition

\[ X; Y \overset{\text{def}}{=} Line_{in, out}[(\nu mid) X(in, mid) \parallel Y(mid, out)] \]
Outline

Introduction
On structural issues
A simple scenario
Goal statement

An algebra of hierarchical graphs
A syntax for hierarchical graphs
Identifying equivalent graphs
Expressing typical structures

Hiding the complexity of hierarchical graphs

Conclusion
The model of hierarchical graphs

intuitive visual representation

complex textual representation we are hiding

```result tEhi: {morphism(graph("in","mid","out"), e(θ), e(θ) --> "in" "out"),
in" --> "Node","mid" --> "Node","out" --> "Node"), e(θ) --> e("Line"), graph(
node", e("Line"), e("Line") --> "Node" "Node" "Node")),e(θ) --> {morphism(graph 
"in","mid","mid","out"), (e(θ),e(1), (e(θ) --> "in" "mid1",e(1) --> "mid" "o ")}, ("in" --> "Node","mid" --> "Node","mid1" --> "Node","out" --> "Node"), (e 
) --> e("host"),e(1) --> e("SubLine"), graph("Node", e("SubLine"),e("host"))
(e("SubLine") --> "Node" "Node",e("host") --> "Node" "Node")),e(1) --> {morphism(
graph("in","mid","out"), e(θ),e(1), (e(θ) --> "in" "mid",e(1) --> "mid" "o ")}, ("in" --> "Node","mid" --> "Node","out" --> "Node"), (e(θ) --> e("host")
e(1) --> ("mid" --> "in","out" --> "out"),"mid1",(nil).List{Node},e(0) --> (" 
" --> "in","mid" --> "mid1","out" --> "out"),"mid1","in" "out")```
The model of hierarchical graphs

intuitive visual representation

complex textual representation we are hiding

\[
\text{Line}_{in,out}[(\nu \mid mid) \\text{host}(in, mid); \\text{SubLine}_{in,out}[(\nu \mid mid) \\text{host}(in, mid); \\text{host}(mid, out); \ (mid, out)]}
\]
From graph terms to graphs

c eq [ g | h ] = \{ morphism(graph((nj,ni),(ej,ei),(mj,mi)),
         deterministic(apply2dom(nm2,tnm1), apply2dom(nm1,tnm2))),
         deterministic(apply2dom(em2,tem1), apply2dom(em1,tem2))),
         graph((tng, tnh), (tet, teh), deterministic((tmg, tmh)))),
         (tig, apply2dom(em1,tih)), (apply2dom(em2,em1,type2em2(nm2,xg)),
         apply2dom(em1,apply2dom(nm1,xh))), (fg, fh), nil \}

if \( g \neq \text{nilg} \) \( h \neq \text{nilg} \)
\( \{ \text{morphism}(\text{graph}(\text{n1,\text{eg,mg},\text{tnm1,tem1,graph}(\text{tng,\text{tet,tmg})}, \text{tig}, \text{xg}, \text{fg}, \text{vg}) \}, \text{tnm2,tem2,graph}(\text{tnh,\text{teh,tmh})}, \text{tih}, \text{xh}), \text{nil}) \}
*** \text{refresh} [g] \text{wrt to free nodes in} \text{fh not in fg}
\{/ \text{morphism}(\text{graph}(\text{ng,\text{eg,mg},\text{nm2,em2,graph}(\text{nj,ej,mj})) := \text{refreshG}(\text{graph}(\text{ng,\text{eg,mg},\text{nm2,em2,graph}(\text{nj,ej,mj}))))},
*** \text{refresh} [h] \text{wrt to} [g] \text{after the above refreshing) except for nodes}
\{/ \text{morphism}(\text{graph}(\text{nh,\text{eh,mh},\text{nm1,em1,graph}(\text{ni,ej,mi})) := \text{refreshG}(\text{graph}(\text{nh,\text{eh,mh},\text{nm1,em1,graph}(\text{ni,ej,mi}))))},

\text{ceq [ d1 \{ nl1 \} ] = \{ morphism(graph((\text{makeSet(nl1),fg}),e(\theta),e(\theta) |\rightarrow nl1)),
         (\text{compose(nm1,tnm1), restrictmap(tnm1,fg)}),
         \text{tem1},
         \text{graph}(\text{tng,\text{tet,tmg}}),
         \text{tig}, \text{apply2dom(nm2,xg)}, (\text{fg, makeSet(nl1)}), \text{nil}\} \}
\text{if} \{ \text{morphism}(\text{graph}(\text{ng,\text{eg,mg},\text{nm1,tem1,graph}(\text{tng,\text{tet,tmg})}, \text{tig}, \text{xg}, \text{fg}, \text{vg}) \} := [\{\}
\{/ \text{nm1} := \text{deterministic(buildmap(nl1,vg))}
\{/ \text{nm2} := \text{deterministic(buildmap(vg,nl1})) .
From graph terms to graphs

**Formal definition**

\[
\begin{align*}
[x] &= \langle \langle x, \emptyset, \bot \rangle, \bot, \emptyset, \{x\}, \emptyset \rangle \\
[l(\overline{x})] &= \langle \langle |\overline{x}|, e, e \mapsto \overline{x} \rangle, \bot, \emptyset, |\overline{x}|, \emptyset \rangle \\
[(\nu x)G] &= \langle G_G, I_G, X_G, F_G \setminus x, \emptyset \rangle \\
\emptyset &= \langle \emptyset, \bot, \bot, \emptyset, \emptyset \rangle \\
[G || H] &= \langle G_G \oplus H_H, I_G \oplus I_H, X_G \oplus X_H, F_G \cup F_H, \emptyset \rangle \\
[L_x[G]] &= \langle \langle F_G, e', e' \mapsto \langle \overline{x} \rangle, e' \mapsto G_G, I_G, X_G \rangle, e' \mapsto id_{F_G}, F_G \setminus \overline{x}, \overline{x} \rangle \\
[D(\overline{x})] &= \langle G_D \{V_D/\overline{x}\}, I_D, X_D \{V_D/\overline{x}\}, F_D \cup |\overline{x}|, \emptyset \rangle \\
\text{if } D : L \land \text{flat}_L \not\equiv_d \\
[D(\overline{x})] &= \langle I_D(e)\{\overline{x}/F_D(e)\}, I_D(e), X_D(e), F_D(e) \cup |\overline{x}|, \emptyset \rangle \\
\text{if } D : L \land \text{flat}_L \equiv_d 
\end{align*}
\]
From graph terms to graphs

the algebra is offering...

\[ \text{eq } X \mid Y = \text{Bus}[p . \ p \mid X\{p\} \mid Y\{p\}] \]

1 self-contained line of code

vs

13 lines full of auxiliary functions!
Main result: coherence for the graph algebra

<table>
<thead>
<tr>
<th>graph1</th>
<th>congruent</th>
<th>graph2</th>
<th>isomorphic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus[ p . host(p,a)</td>
<td>host(p)</td>
<td>host(p,a) ]... ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>graphterm1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus[ p . host(p,a)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>graphterm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus[ p . host(p,a)</td>
</tr>
</tbody>
</table>
Outline

Introduction
  On structural issues
  A simple scenario
  Goal statement

An algebra of hierarchical graphs
  A syntax for hierarchical graphs
  Identifying equivalent graphs
  Expressing typical structures
  Hiding the complexity of hierarchical graphs

Conclusion
Main application of the result: encodings are facilitated
Main application of the result: encodings are facilitated.

network1
- host(a)
  - host
    - [ host | host(a) ]

graph1
- Bus[ p .
  - host(p,a)
    - host(p)
    - host(a)
  ...
 ]

network2
- host
  - [ host | host(a) ]
- host(a)

graph2
- Bus[ p .
  - host(p,a)
    - host(p)
    - host(a)
  ...
 ]

graphterm1
- congruent
graphterm2
- congruent
graph1
- isomorphic
The algebra facilitates a modular implementation

Specification languages

- networks
- pi-calculus
- caspis
- etc.

Graph formats

- dot
- GraphML

External Tools
The algebra facilitates a modular implementation
Implementation snapshot (a simple visualiser)

Available at www.albertolluch.com/adr2graphs
Applications (general)

Modelled with the algebra

- Network topologies [BL09]
Applications (general)

Modelled with the algebra

- Network topologies [BL09]
- Process calculi [GLB]
Applications (general)

Modelled with the algebra

- Network topologies [BL09]
- Process calculi [GLB]
- Workflows [GLB]
Applications (general)

Modelled with the algebra

- Network topologies [BL09]
- Process calculi [GLB]
- Workflows [GLB]

Modelled without the algebra

- Service modelling language [BLME07]
Applications (general)

Modelled with the algebra

- Network topologies [BL09]
- Process calculi [GLB]
- Workflows [GLB]

Modelled without the algebra

- Service modelling language [BLME07]
- UML4SOA profile [BLME07]
Applications (general)

Modelled with the algebra

▶ Network topologies [BL09]
▶ Process calculi [GLB]
▶ Workflows [GLB]

Modelled without the algebra

▶ Service modelling language [BLME07]
▶ UML4SOA profile [BLME07]
▶ Architectural styles [BLM08]
Applications (service oriented calculi)

CaSpiS (sessions)
- Nesting of sessions
- Sharing of session channels

Activity A has invoked two services S1, S2 creating two nested sessions with channels \( a, b \).
Applications (service oriented calculi)

CaSpiS (sessions)
- Nesting of sessions
- Sharing of session channels

Sagas (transactions)
- Nesting of transactions
- Workflow constructs

A saga as an ordinary workflow compensated with another workflow.

A workflow as saga without compensation flow.
Related work

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

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

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

GS-Graphs [CG99]
- 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
Concluding remarks

The graph algebra . . .

- Grounds on widely-accepted models;
- Hides the complexity of hierarchical graphs;
- Enables proofs by structural induction;
- Extends ADR with node sharing and serves as primitive algebra for ADR;
- Simplifies the modelling of process calculi;
- Offers a technique for complementing textual and visual notations in formal tools;
- Has been evaluated on calculi, networks, etc.
- Natural implementation in Maude (support for theorem proving, model checking, simulation, etc.)
Thanks for your attention
Credits and references I

Roberto Bruni and Alberto Lluch Lafuente.
Ten virtues of structured graphs.
In Invited paper at the 8th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT’09), Electronic Communications of the EASST, 2009.
To appear.

Roberto Bruni, Alberto Lluch Lafuente, and Ugo Montanari.
Hierarchical Design Rewriting with Maude.
In Proceedings of the 7th International Workshop on Rewriting Logic and its Applications (WRLA’08),
To appear.

Roberto Bruni, Alberto Lluch Lafuente, Ugo Montanari, and Emilio Tuosto.
Service Oriented Architectural Design.

Andrea Corradini and Fabio Gadducci.
An algebraic presentation of term graphs, via gs-monoidal categories. Applied Categorical Structures.

Andrea Corradini, Ugo Montanari, and Francesca Rossi.
An abstract machine for concurrent modular systems: CHARM.

Frank Drewes, Berthold Hoffmann, and Detlef Plump.
Hierarchical graph transformation.
Credits and references II

Fabio Gadducci.
Term graph rewriting for the pi-calculus.

Fabio Gaducci, Alberto Lluch Lafuente, and Roberto Bruni.
Graphical representation of process calculi via an algebra of hierarchical graphs.

Bigraphs and mobile processes.

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