Ten Virtues of Structured Graphs

(or why structured graphs can be better than flat ones)

Roberto Bruni

also based on join work with F. Gadducci, A. Lluch-Lafuente, U. Montanari, E. Tuosto

Department of Computer Science, University of Pisa
bruni@di.unipi.it

Research supported by SENSORIA

March 28-29, 2009, GT-VMT'09, York, UK

Outline

GT-VMT 2009

- Graph-based techniques
  - formal semantics, concurrency, logics, verification, tools
- Visual modelling
  - project planning, network management, traffic control, business processes, software architectures, www site design, and many more...
- Modern software and Sensoria project (service-oriented computing)
  - key issues such as scalability, representation distance, open-endedness, dynamicity, distribution
  - within specification, design, validation and verification phases

Sensoria Poster Collage (http://www.sensoria-ist.eu)

Software Engineering for Service-Oriented Overlay Computers www.sensoria-ist.eu

develops

semantically well-founded languages, novel theories, methods and tools for constructing and analysing the new generation of high-quality service-oriented systems

integrates

foundational theories, techniques, and methods with pragmatic software engineering

researches

- linguistic primitives for modelling and programming service-oriented systems
- qualitative and quantitative analysis methods for global services
- development and deployment techniques for systems services

offers

- model-driven approach for service-oriented software engineering
- modelling of service-oriented systems
- analysis of behaviour, security and quality of service properties
- suite of tools and techniques for
  - deploying service-oriented systems
  - reengineering legacy software into services

case studies

in automotive, finance, telecommunications and and e-learning domains

SIXTH FRAMEWORK PROGRAMME

Information Society Technologies

List of partners

Coordinator: Prof. Dr. Martin Wirsing, Ludwig-Maximilians-Universität München, Germany
Università di Trento | University of Leicester | Warsaw University | TU Denmark at Lyngby | Università di Pisa
Università di Firenze | Università di Bologna | ISTI Pisa | Universidade de Lisboa | University of Edinburgh | ATX S
Telecom Italia Lab | Imperial College London | FAST GmbH | Budapest University of Technology and Economics
S&N AG | University College London | Politecnico di Milano

All That Graphs

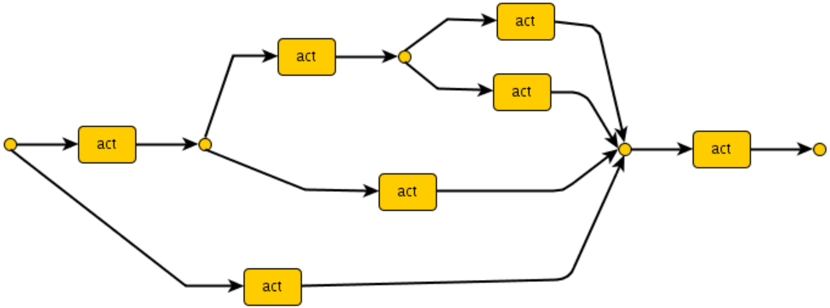
	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
n_0	a_1		a_2			a_3		a_4		
n_1		a_5	a_6	a_7			a_8		a_9	a_{10}
n_2	a_{11}				a_{12}		a_{13}		a_{14}	a_{15}
n_3	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}		a_{24}
n_4				a_{25}			a_{26}		a_{27}	a_{28}
n_5		a_{29}	a_{30}			a_{31}	a_{32}			a_{33}
n_6			a_{34}		a_{35}					
n_7					a_{36}	a_{37}			a_{38}	
n_8			a_{39}		a_{40}		a_{41}	a_{42}		a_{43}
n_9	a_{44}	a_{45}	a_{46}		a_{47}	a_{48}				

All That Graphs


```
- <graphml xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml http://www.yworks.com/xml/schema/graphml/1.0/ygraphml.xsd">
    <key for="node" id="d0" yfiles.type="nodegraphics"/>
    <key attr.name="description" attr.type="string" for="node" id="d1"/>
    <key for="edge" id="d2" yfiles.type="edgegraphics"/>
    <key attr.name="description" attr.type="string" for="edge" id="d3"/>
    <key for="graphml" id="d4" yfiles.type="resources"/>
  - <graph edgedefault="directed" id="G" parse.edges="18" parse.nodes="18" parse.order="free">
    - <node id="n0">
      - <data key="d0">
        - <y:ShapeNode>
            <y:Geometry height="30.0" width="47.0" x="-251.0" y="0.0"/>
            <y:Fill color="#FFCC00" transparent="false"/>
            <y:BorderStyle color="#000000" type="line" width="1.0"/>
            <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="Arial Black" fontSize="12" fontStyle="plain" hasBackgroundColor="false" hasLineColor="false"
            height="17.962890625" modelName="internal" modelPosition="c" textColor="#000000" visible="true" width="34.03515625" x="6.482421875"
            y="6.0185546875">2hub</y:NodeLabel>
            <y:Shape type="roundrectangle"/>
          </y:ShapeNode>
        </data>
        <data key="d1"/>
      </node>
    - <node id="n1">
```

All That Graphs

Our choice



A Scenario: Software Architectures as Graphs

- D. Garlan & D. Perry, 1995
  - "*... the structure of the components of a program / system, their interrelationship, and principles and guidelines governing their design and evolution over time.*"
- components (and connectors) as hyper-edges
  - (here represented as boxes of various shapes)
- ports (and roles) as tentacles
  - (here represented as arrows)
- attachments as nodes
  - (here represented as smaller circles)
- connectors and attachments are sometimes omitted

Why "Spaghetti" Graphs are Considered Harmful

- When GT applied to large case studies, graphs better be structured in order to be comprehensible
- Analogies with structured programming and type theory
  - it is helpful to use graphs that are conveniently formatted and annotated
  - discard / ignore non-conformant graphs
- Analogies with process calculi
  - containment and links (as in bigraphs)
  - dynamics and reconfiguration via inductive, conditional rewrite rules

Our proposal

- From graphs to hierarchical hypergraphs
  - certain hyperdeges can contain hypergraphs that can be hierarchical themselves
  - arbitrary depth of nesting
- ADR (Architectural Design Rewriting)
  - graphs + their blueprint (like binaries + source templates)
  - exploit blueprint for applying formal methods
  - please visit `http://www.albertolluch.com/research/adr` to know more

Outline

Introduction

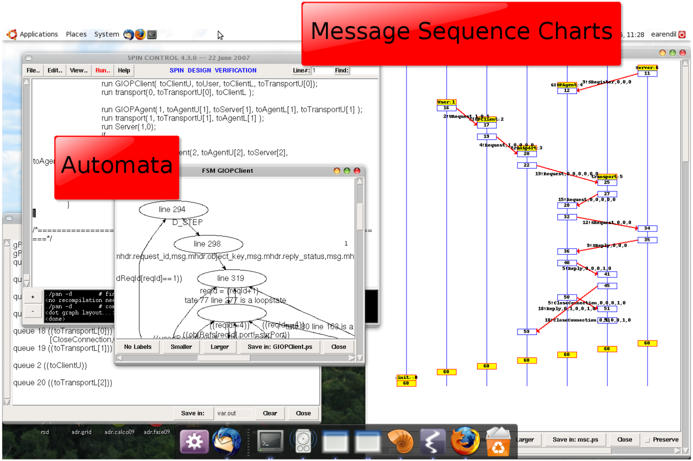
Styles for Visual Support

Dynamics

ADR

Concluding Remarks

Visualization can Support Formal Methods

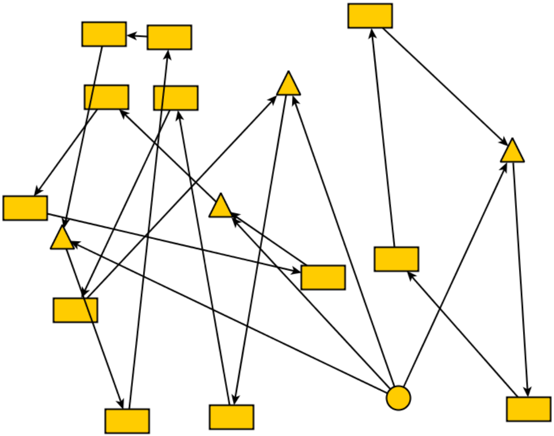


Architectural Styles

- IEEE standard 1471
  - *"... a set of patterns or rules for creating one or more architectures in a consistent fashion."*
- Style = Vocabulary + Rules
  - Used to construct and document
  - Used to describe / explain
  - Used to understand
  - Used to validate
  - Used for conformance check
  - Used to reason about
  - To be reused

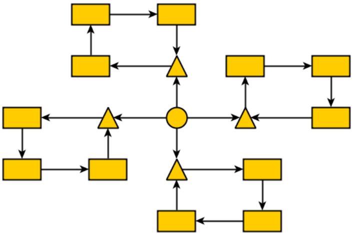
Uhm...

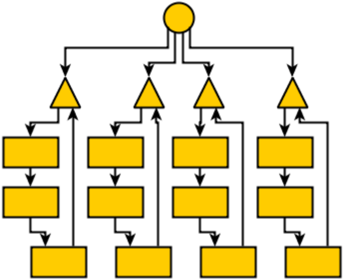
Can you spot some "regularity"?



Graph Re-drawing

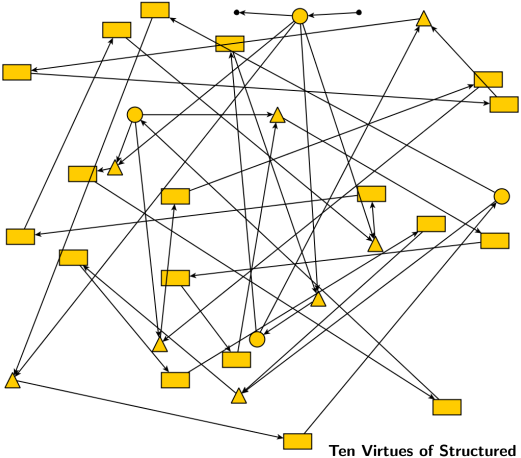
And now?





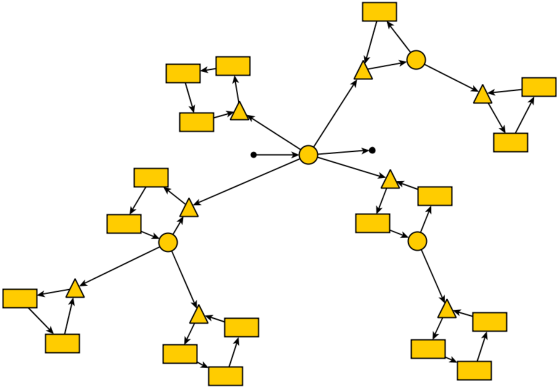
Well...

Another try?



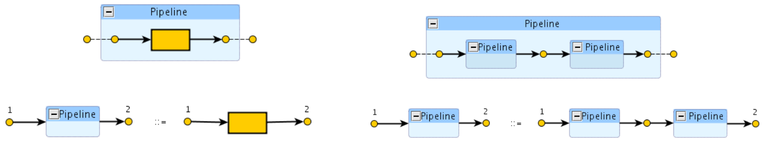
Another Graph Re-drawing

Can you describe its “shape” (or style)?



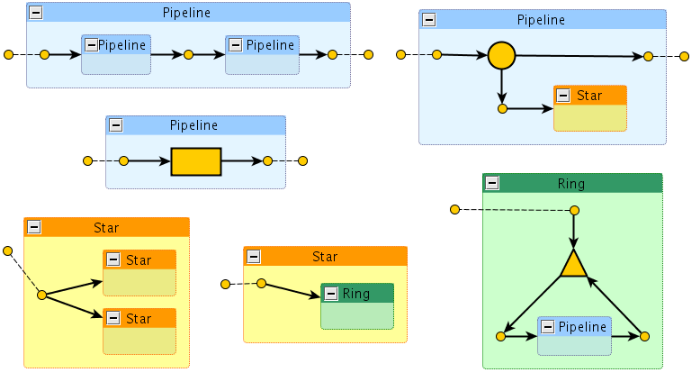
Styles from Productions

- Legenda: titled boxes as non-terminals, ordinary boxes as terminals

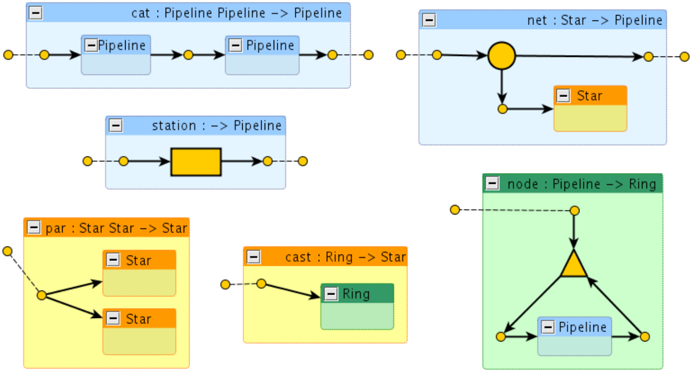


- Several readings are possible:
  - Refinement
  - Types (`Pipeline`) and ops (*station* and $cat(\cdot)$, based on hyperedge replacement)
    - $\mathit{station} :\longrightarrow \texttt{Pipeline}$
    - $\mathit{cat} : \texttt{Pipeline} \times \texttt{Pipeline} \longrightarrow \texttt{Pipeline}$
  - Abstraction

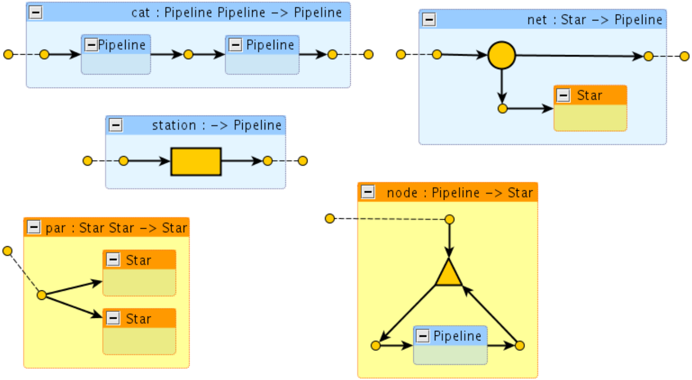
Types for Pipelines, Rings and Stars



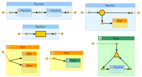
Types and Ops for Pipelines, Rings and Stars

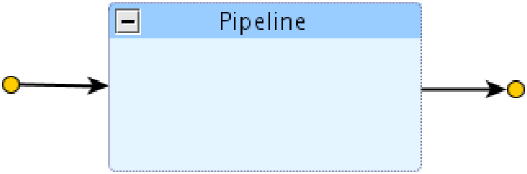


Simplified Memberships (for Pipelines and Stars)

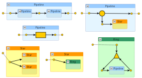


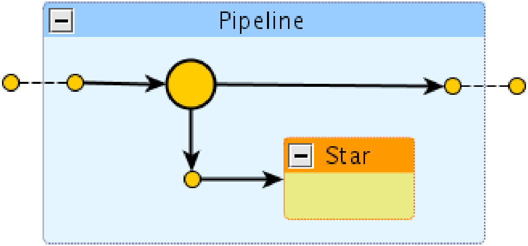
An Example of Derivation (with “Blueprint”)



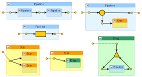


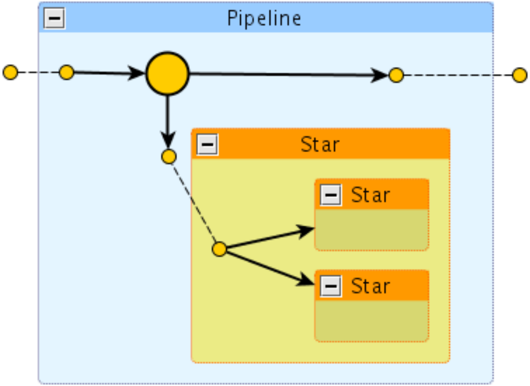
An Example of Derivation (with “Blueprint”)



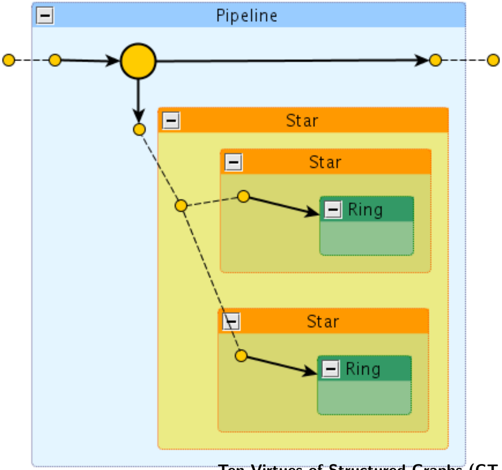


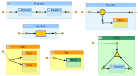
An Example of Derivation (with “Blueprint”)



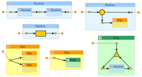


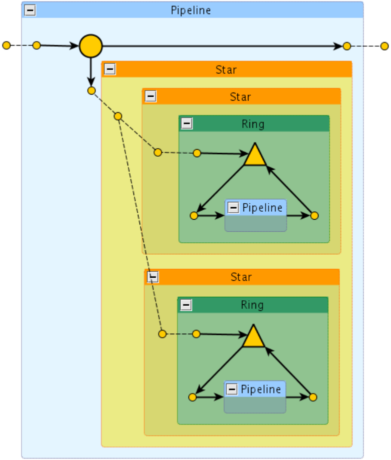
An Example of Derivation (with “Blueprint”)



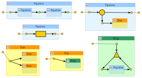


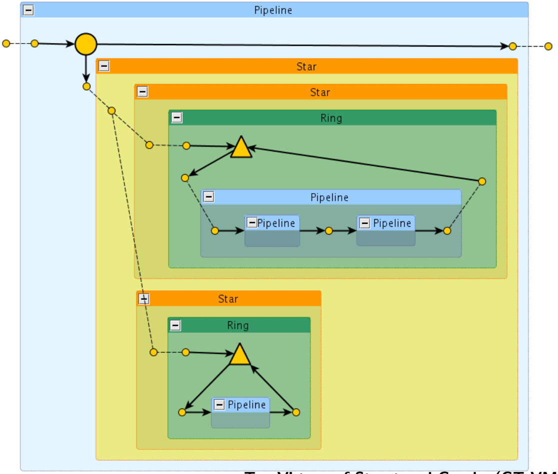
An Example of Derivation (with "Blueprint")



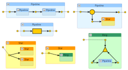


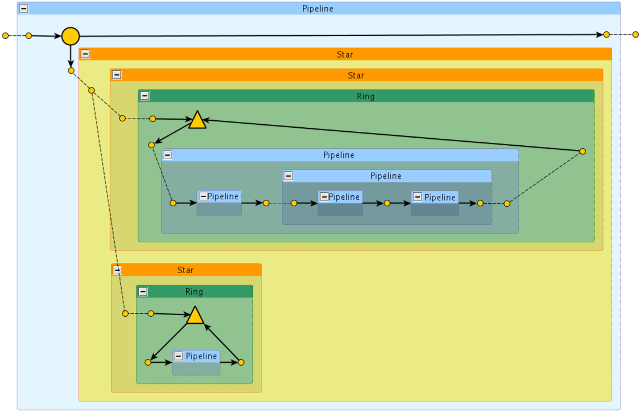
An Example of Derivation (with "Blueprint")



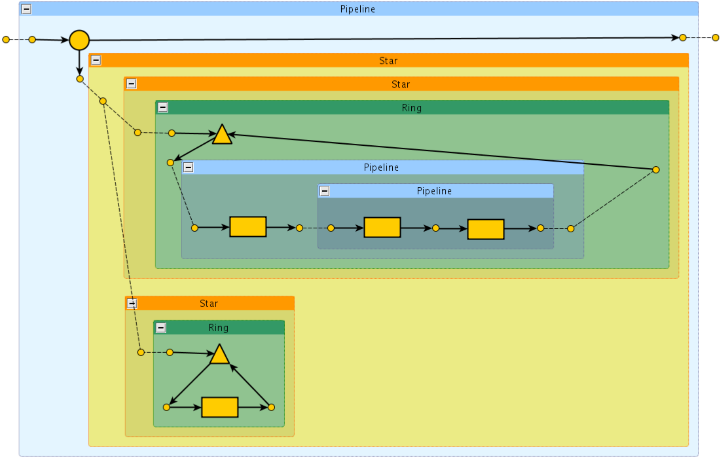


An Example of Derivation (with “Blueprint”)

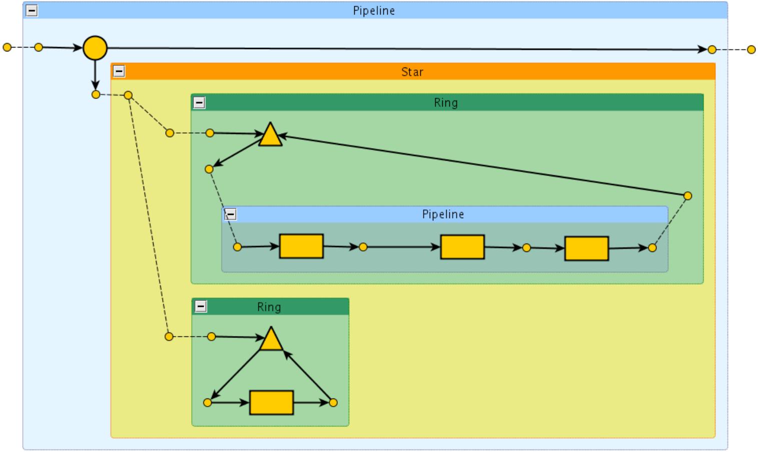




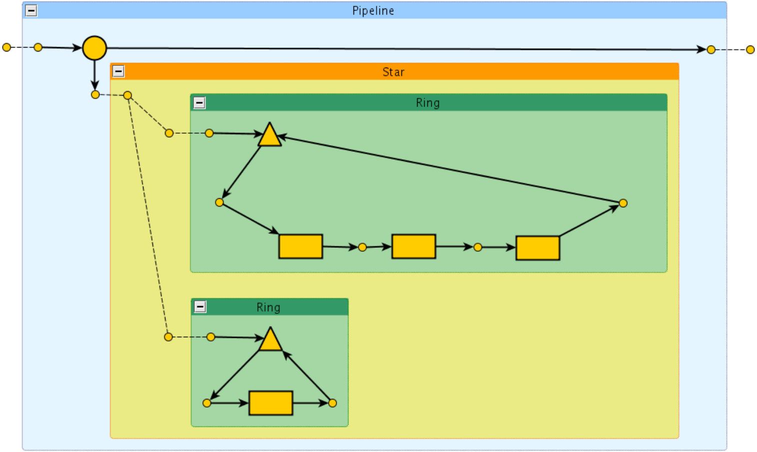
An Example of Derivation (with “Blueprint”)



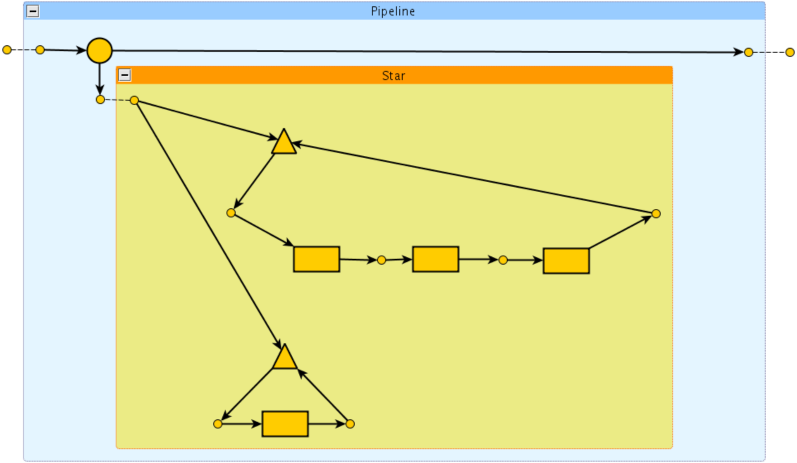
Simplified Typing and Drawing (“Flattening”)



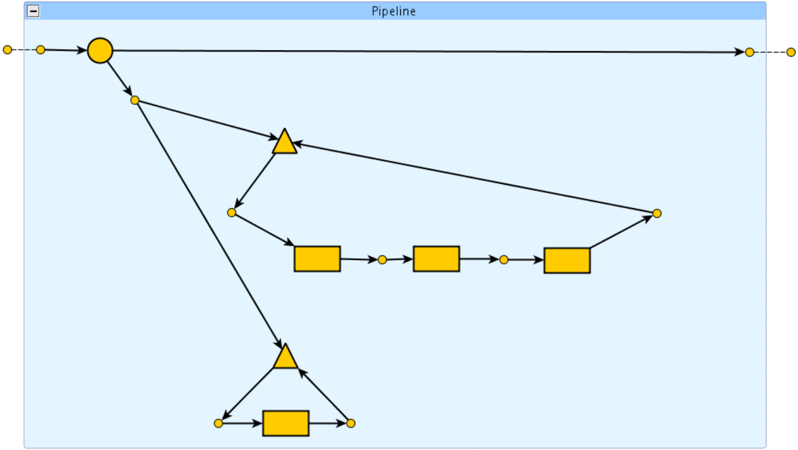
Simplified Typing and Drawing (“Flattening”)



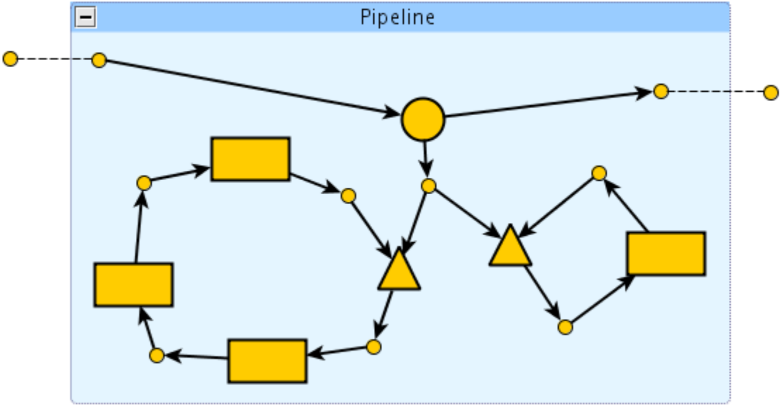
Simplified Typing and Drawing ("Flattening")



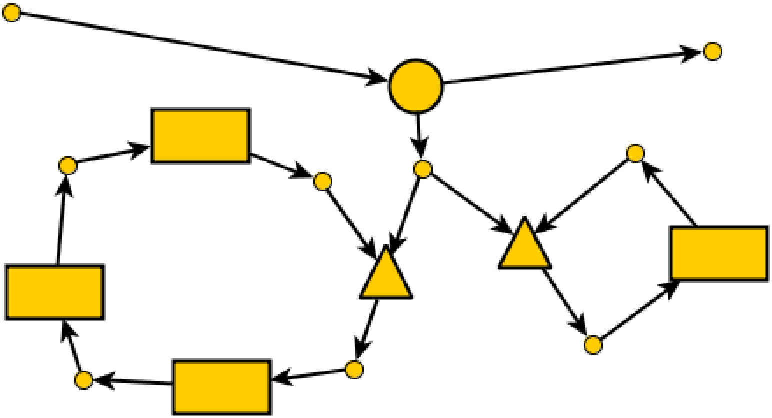
Simplified Typing and Drawing (“Flattening”)



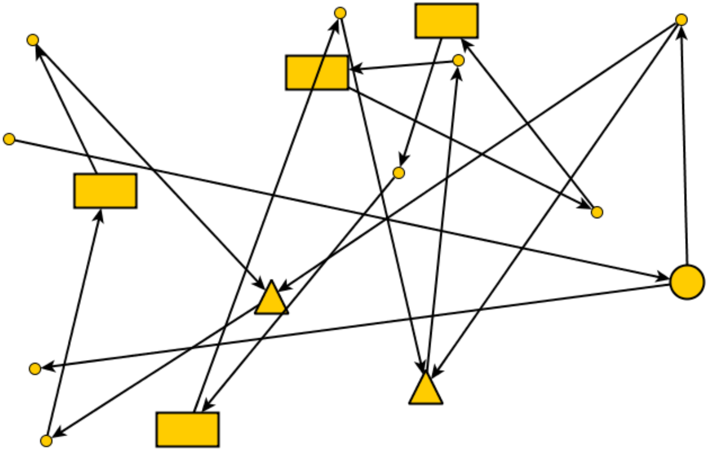
Simplified Typing and Drawing (“Flattening”)



Simplified Typing and Drawing ("Flattening")



Simplified Typing and Drawing (“Flattening”)



Simplified Typing and Drawing (“Flattening”)

The corresponding proof term is

```
net(par(cast(node(cat(station,
                          cat(station, station)))),
        cast(node(station))
      )
    )
```

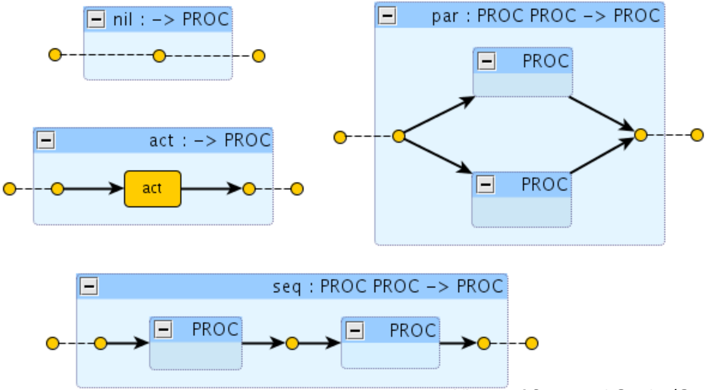
Or just

```
net(par(node(cat(station,station,station)),
        node(station)))
```

Note that nodes need not be mentioned

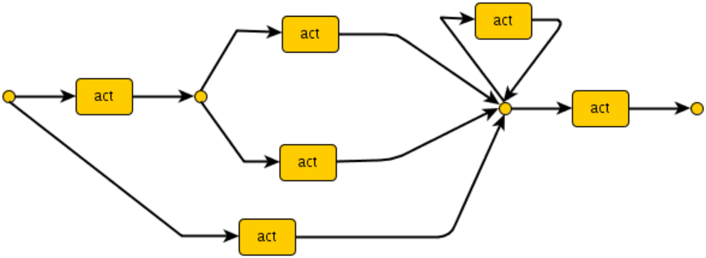
Another Example: Workflows

Activities composable in series and in parallel (fork & join): disconnected activity and cyclic parts are not allowed



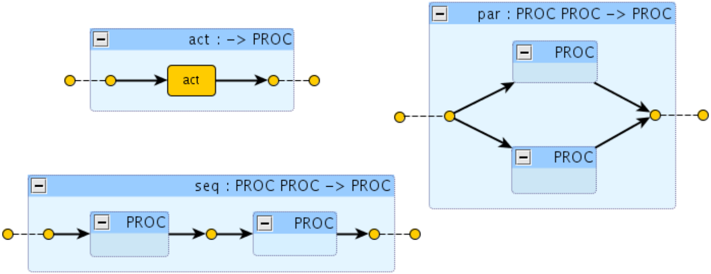
Another Example: Workflows

Is this a well-formed workflow?



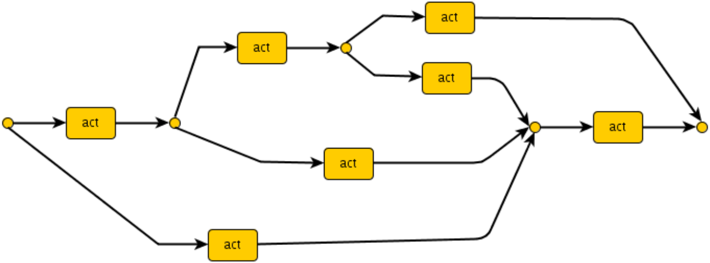
Another Example: Workflows

Activities composable in series and in parallel (fork & join): disconnected activity and cyclic parts are not allowed



Another Example: Workflows

Is this a well-formed workflow?



Six Virtues of Structured Graphs

- Requirements
  - Type graphs are ok (and synergic to our approach) but limited
  - Additional logic languages often needed
  - We can account for many patterns in a natural way
- Parsing and browsing
  - Large graphs are hard to "understand" and navigate
  - Their blueprint (if any available) helps quite a lot
- Model Construction and Model conformance
  - Conformance is guaranteed by construction
  - Otherwise hard to recover from scratch (proof-carrying graphs)
- Compositionality and Abstraction & Refinement
  - Interfaces are needed to constrain composition, but hard to recover in flat graphs
  - The hierarchical approach makes them available at any level
  - Different levels of granularity can be considered

Six Virtues of Structured Graphs

- Requirements
  - Type graphs are ok (and synergic to our approach) but limited
  - Additional logic languages often needed
  - We can account for many patterns in a natural way
- Parsing and browsing
  - Large graphs are hard to “understand” and navigate
  - Their blueprint (if any available) helps quite a lot
- Model Construction and Model conformance
  - Conformance is guaranteed by construction
  - Otherwise hard to recover from scratch (proof-carrying graphs)
- Compositionality and Abstraction & Refinement
  - Interfaces are needed to constrain composition, but hard to recover in flat graphs
  - The hierarchical approach makes them available at any level
  - Different levels of granularity can be considered

Six Virtues of Structured Graphs

- Requirements
  - Type graphs are ok (and synergic to our approach) but limited
  - Additional logic languages often needed
  - We can account for many patterns in a natural way
- Parsing and browsing
  - Large graphs are hard to “understand” and navigate
  - Their blueprint (if any available) helps quite a lot
- Model Construction and Model conformance
  - Conformance is guaranteed by construction
  - Otherwise hard to recover from scratch (proof-carrying graphs)
- Compositionality and Abstraction & Refinement
  - Interfaces are needed to constrain composition, but hard to recover in flat graphs
  - The hierarchical approach makes them available at any level
  - Different levels of granularity can be considered

Six Virtues of Structured Graphs

- Requirements
  - Type graphs are ok (and synergic to our approach) but limited
  - Additional logic languages often needed
  - We can account for many patterns in a natural way
- Parsing and browsing
  - Large graphs are hard to "understand" and navigate
  - Their blueprint (if any available) helps quite a lot
- Model Construction and Model conformance
  - Conformance is guaranteed by construction
  - Otherwise hard to recover from scratch (proof-carrying graphs)
- Compositionality and Abstraction & Refinement
  - Interfaces are needed to constrain composition, but hard to recover in flat graphs
  - The hierarchical approach makes them available at any level
  - Different levels of granularity can be considered

Outline

Style-Preserving Reconfiguration

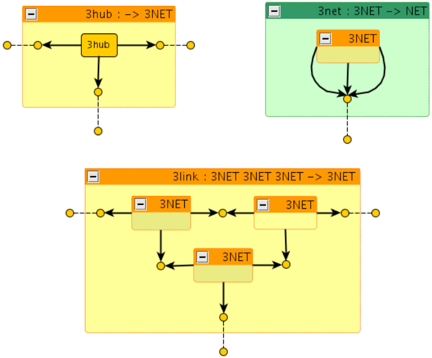
- A reconfiguration is a change in an architecture
  - static? e.g. for deployment on different platforms, improvements, updates, upgrades, model-driven transformation
  - partially specified? e.g. some components are not known at design time, except for their types
  - run-time? e.g. triggered by security policies, load balancing, mobility, QoS assurance, components joining and leaving the system, dynamic binding, wrapping, self-* architectures
- Style-preservation is relevant
  - from well-formed graphs to well-formed graphs (but possibly with different shapes)
- Examples
  - reverse all actions in a pipeline, serialize a workflow, star to ring transformation, migrate all clients of a server, close all sub-sessions upon termination of their parents

How to Write Reconfiguration Rules

- Using graph transformation
  - direct manipulation of flat graphs
  - applicable in non well-formed graphs
  - well-formedness of results must be proved
  - in the flat case: rules manipulate components (many steps required)
  - in the hierarchical case: rules manipulate groups of components (one step can suffice)
- Exploiting structured graphs
  - rules manipulate well-formedness proofs
  - inductive localization of the least part of the proof where the change is needed
  - style-preserving by construction

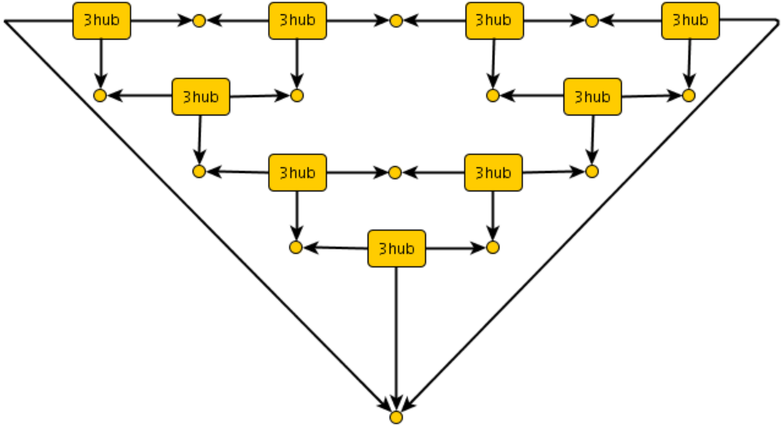
An Example: 3hub Network

Network hubs have three degrees of connectivity and connections are driven by the style (only allowed: some sort of reversed pyramids)



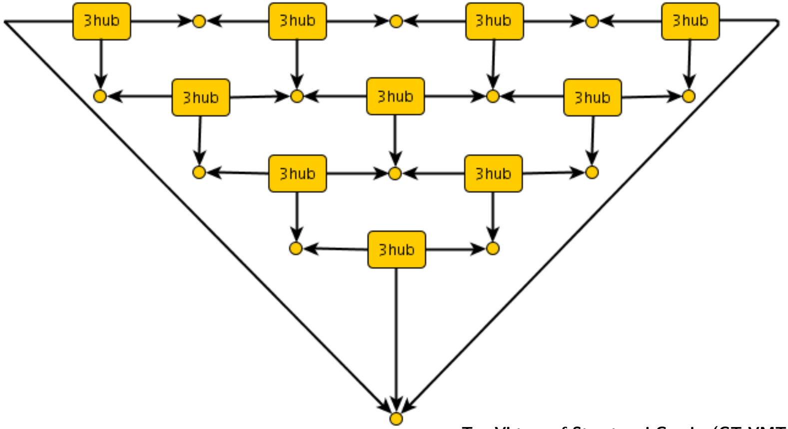
An Example: 3hub Network

A valid 3hubs network



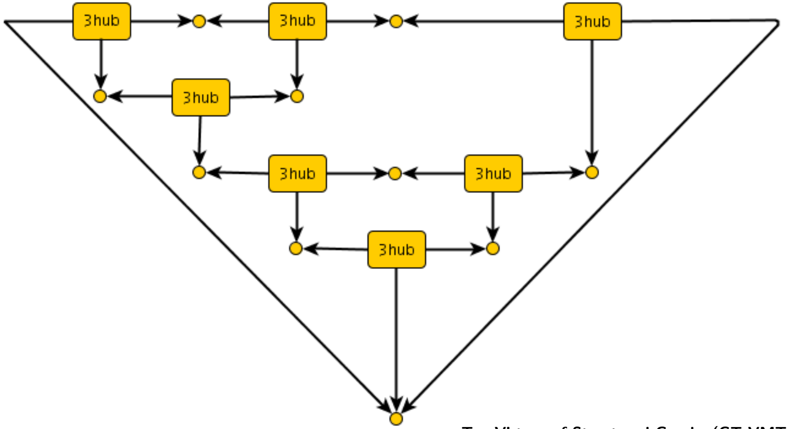
An Example: 3hub Network

A valid 3hubs network? or maybe not?



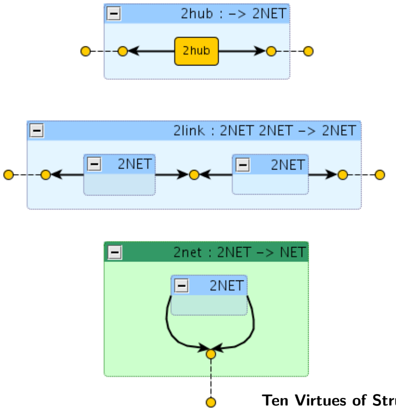
An Example: 3hub Network

A valid 3hubs network? or maybe not?



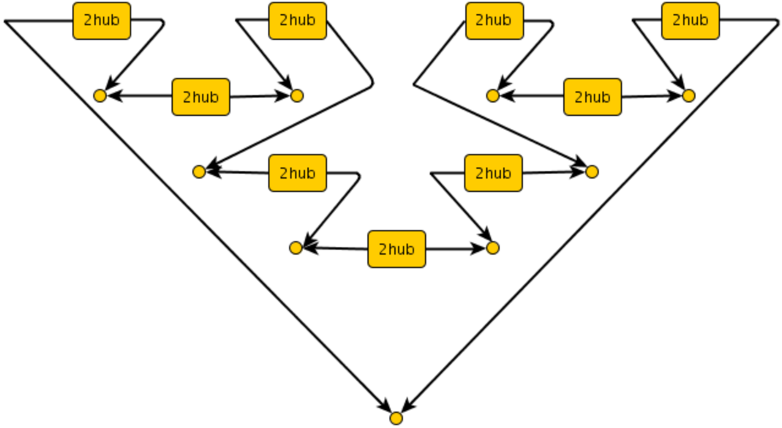
An Example: 2hub Network

Network hubs have just two degrees of connectivity and connections are driven by the style (only allowed: rings)



An Example: 2hub Network

A valid 2hubs network

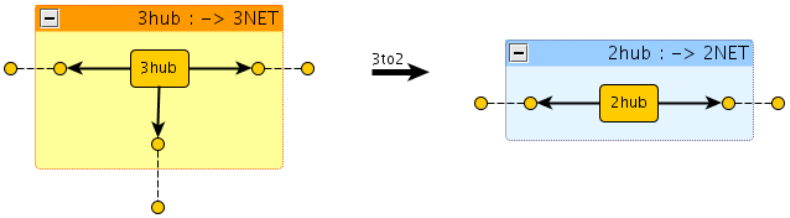


An Example: From 3hub Networks to 2hub Networks

- Under certain circumstances, it is required to reconfigure any valid 3hub network to a valid 2hub network
  - the whole network must be reconfigured (not just part of it)
  - total number of hubs is unchanged
  - 2hubs must form a ring
- Idea:
  - exploit blueprint, not the flat graph
  - reconfiguration is defined inductively on the structure of the network
  - exploit conditional rewrite rules

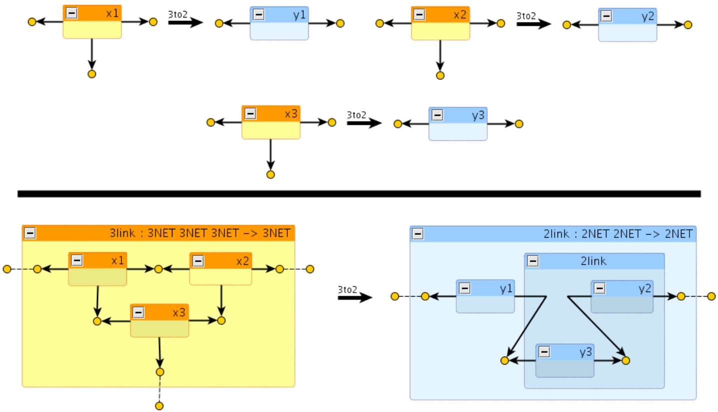
An Example: From 3hub Networks to 2hub Networks

Reconfigure a single 3hub (note that type is changed: some sort of transduction, context must be adapted)



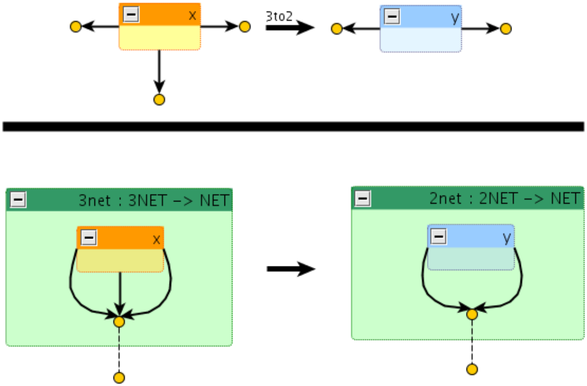
An Example: From 3hub Networks to 2hub Networks

Reconfigure the link structure (a transduction, again)



An Example: From 3hub Networks to 2hub Networks

Reconfigure the whole network (note that type is preserved, rewrite is silent, applicable in any larger context)



An Example: Rewrite Rules for Network Transformation

$$\texttt{3hub} \xrightarrow{\texttt{3to2}} \texttt{2hub}$$

$$\frac{x_1 \xrightarrow{\texttt{3to2}} y_1 \quad x_2 \xrightarrow{\texttt{3to2}} y_2 \quad x_3 \xrightarrow{\texttt{3to2}} y_3}{\texttt{3link}(x_1, x_2, x_3) \xrightarrow{\texttt{3to2}} \texttt{2link}(y_1, \texttt{2link}(y_3, y_2))}$$

$$\frac{x \xrightarrow{\texttt{3to2}} y}{\texttt{3net}(x) \longrightarrow \texttt{2net}(y)}$$

Three More Virtues of Structured Graphs

- Reconfiguration and Evolution
  - (flat) graph transformation requires ad-hoc studies and techniques (e.g., negative application conditions, interfaces, atomicity issues), augmenting the representation distance (high expertize, technology transfer more difficult)
  - structured graph rewrites can be more handy and efficient (e.g. graph matching not necessarily required)
  - style preservation: to be proved vs guaranteed by proofs
  - concurrency? special cases (edge to edge rules)?
- Graphical encoding
  - seamless grouping of item through the hierarchy (e.g. for representing nested sessions, transactions, scopes)
  - in the case of process calculi, facilitated by suitable graph algebras (see next part of the talk)
  - Encoding properties (soundness, completness) shown by structural induction

Three More Virtues of Structured Graphs

- Reconfiguration and Evolution
  - (flat) graph transformation requires ad-hoc studies and techniques (e.g., negative application conditions, interfaces, atomicity issues), augmenting the representation distance (high expertize, technology transfer more difficult)
  - structured graph rewrites can be more handy and efficient (e.g. graph matching not necessarily required)
  - style preservation: to be proved vs guaranteed by proofs
  - concurrency? special cases (edge to edge rules)?
- Graphical encoding
  - seamless grouping of item through the hierarchy (e.g. for representing nested sessions, transactions, scopes)
  - in the case of process calculi, facilitated by suitable graph algebras (see next part of the talk)
  - Encoding properties (soundness, completeness) shown by structural induction

Outline

Introduction

Styles for Visual Support

Dynamics

ADR

Concluding Remarks

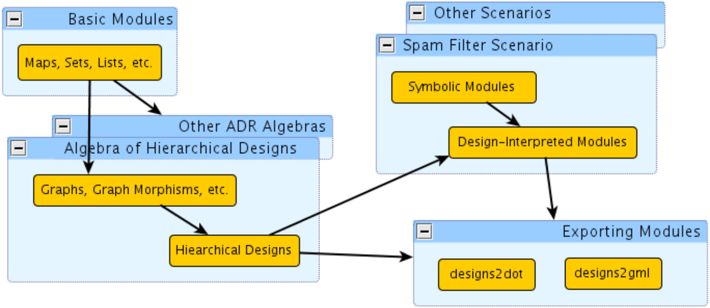
ADR in a Nutshell

- ADR formulas:
  - ADR = Designs + Term Rewriting
  - Designs = Typed Hierarchical Graphs (with Interfaces)
- ADR ingredients:
  - Sorts: Vocabulary, Types (edge and node labels)
  - Values: Designs (hierarchical graphs with interfaces)
  - Operations: Graph-grammar-like rules
  - Terms: proofs of construction
  - Terms (with variables): partial Designs, partial proofs
  - Axioms: properties of operations
  - Membership predicates: additional style rules
  - Rewrite rules: behaviour, reconfigurations
  - Rewrite strategies: style conformance, style analysis, etc.

A Flexible Unifyig Framework for Design, Execution, Reconfiguration

- Not necessarily in the spirit of universal models:
  - node as names + hyper-edge as ops + parallel composition + name fusion + name hiding = any graph can be obtained
  - node as names + hyper-edge as ops + type annotation + tailored constructors = only well-formed designs are described
- Some other ADR features:
  - Membership equational theory (e.g. ACI1, subsorting, overloading)
  - Flattening axioms (e.g. not all operators are hierarchic)
- Some ADR caveats:
  - different proof terms for the same graph are possible
  - constraints not fully integrated yet
  - concurrency aspects not addressed yet

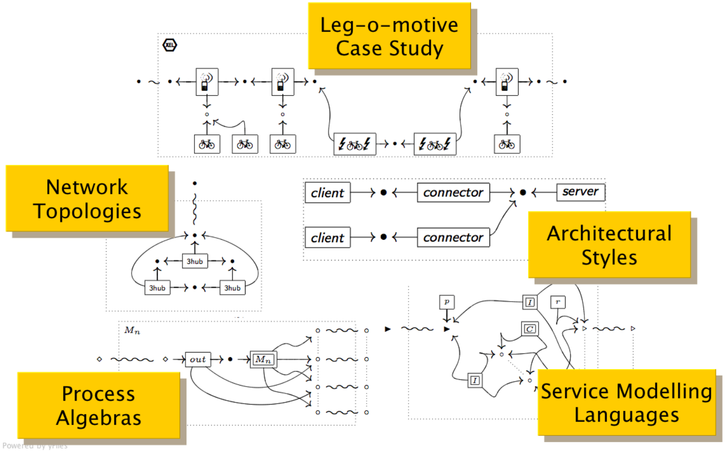
Maude Prototype for ADR



Maude Prototype for ADR

- Why Maude?
  - built-in membership equational theories (e.g. to support style conformance check)
  - conditional rewrite rules supported
  - standard encoding of LTS
  - built-in search strategies (e.g. to support model finding)
  - built-in LTL model-checker
  - defineable logic languages (within the same framework): e.g. graph logics (Courcelle's MSO), modal logics, spatial logics

ADR Case Studies



An Example: From Process Calculi to Graphs

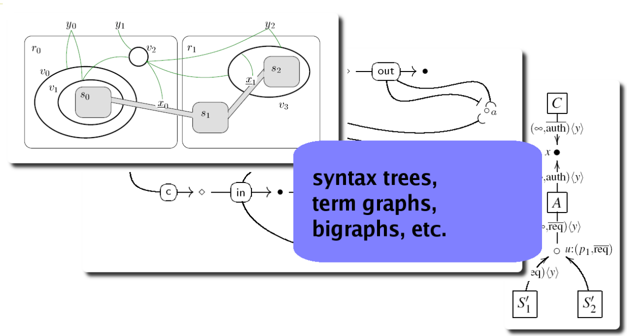
The syntax of process calculi (with name handling)

$P, Q ::=$	$\sum_{i \in I} \pi_i P_i$	Guarded Sum
$\mid$	$s.P$	Service Definition
$\mid$	$\overline{s}.P$	Service Invocation
$\mid$	$r \triangleright P$	Sessio
$\mid$	$P > Q$	Pipeli
$\mid$	$P \vert Q$	Parallel Composition
$\mid$	$(\nu n)P$	Restriction
$\mid$	$!P$	Replication

Algebraic form:
- grammar
- structural congruence

An Example: From Process Calculi to Graphs

Terms as graphs



An Example: From Process Calculi to Graphs

The syntax of graphs

Definition 22 (bigraph) A *bigraph* over the signature $\mathcal{K}$ takes the form $G = (V, ctrl, G^{\mathsf{T}}, G^{\mathsf{M}}) : I \to J$ where: $I = \langle m, X\rangle$ and $J = \langle n, Y\rangle$ are its *inner and outer interfaces*, each combining a *width* (a finite ordin… each … and 3…

Definition 7 (hypergra… …isms). *A hypergraph G is a triple $\langle E_G, N_G, t_G\rangle$ suc…* …es, N_G *is the set of nodes, and $t_G : E_G \to N_G^*$ is the* … of … fun…

Definition 2.1 (Grap… …nt *has form $\Gamma \vdash G$ where:*

1. *$\Gamma \subseteq \mathcal{N}$ is a finite set of names (the free nodes of the graph);*
2. *G is a* graph term *generated by the grammar*

$$G ::= L(\mathbf{x}) \quad | \quad G|G \quad | \quad \nu y\, G \quad | \quad nil$$

where $\mathbf{x}$ is a tuple of names, $L \in \mathcal{L}$, $\mathrm{rank}(L) = |\mathbf{x}|$ and y is a name.

tuples,
sets,
morphisms, etc.

An Example: From Process Calculi to Graphs

Encoding can become cumbersome

$$[\![(\nu a)P]\!]_\Gamma = \begin{cases} [\![P]\!]_\Gamma & \textit{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_c \otimes id_\Gamma) \circ [\![P\{^c/_a\}]\!]_{\{c\}\uplus\Gamma} & \textit{otherwise} \end{cases}$$

$$[\![P \mid Q]\!]_\Gamma = [\![P]\!]_\Gamma \otimes [\![Q]\!]_\Gamma \qquad [\![a(b).P]\!]_\Gamma = (in_{a,c} \otimes id_\Gamma) \circ [\![P\{^c/_b\}]\!]_{\{c\}\uplus\Gamma}$$

$$[\![0]\!]_\Gamma = 0_p \otimes 0_\Gamma \qquad [\![\overline{a}b.P]\!]_\Gamma = (out \ldots$$

$$[\![\mathbf{0}]\!]_X = 1 \curlywedge X \qquad [\![P|Q]\!]_X = [\![P]\!]_X \curlywedge [\![Q]\!]_X \ldots$$

$$[\![zx.P]\!]_X = \mathsf{get}^{x,z} \circ [\![P]\!]_X \qquad [\![\bar{z}x.P]\!]_X = \mathsf{se}\ldots$$

$$[(\nu a)P]^p_n = \mathtt{hide}_n([P\{\ldots/_a \ldots$$

$$[P \mid Q]^p_n = \mathtt{par}_n([P]^p_n, [Q]^p_n) \qquad \ldots = \mathtt{in}_{i,n}([P\{^{\ldots+1}/_y\}]^p_{n+1})$$

$$[0]^s_n = \mathtt{nil}_n \qquad [M + N]^s_n = \mathtt{choice}_n([M]^s_n, [N]^s_n)$$

operations are parametric, complex, ad hoc

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$
\begin{array}{llll}
\text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\
\text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle
\end{array}
$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$
\begin{array}{llll}
\text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\
\text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle
\end{array}
$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$\begin{array}{llll} \text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\ \text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle \end{array}$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$\begin{array}{llll} \text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\ \text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle \end{array}$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$\begin{array}{llll} \text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\ \text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle \end{array}$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

A Re-usable Graph Algebra for Process Calculi

Components as edges $l(\vec{x})$, types as design labels L.

$$\begin{array}{llll} \text{(designs)} & \mathbb{D} & ::= & \mathsf{L}_{\vec{x}}[\mathbb{G}] \\ \text{(graphs)} & \mathbb{G} & ::= & \mathbf{0} \mid x \mid l(\vec{x}) \mid \mathbb{G}|\mathbb{G} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\vec{y}\rangle \end{array}$$

- In $\mathsf{L}_{\vec{x}}[\mathbb{G}]$, the nodes $\vec{x}$ in $\mathbb{G}$ are bound by the interface (as arguments), the other free names of $\mathbb{G}$ are global.
- We write $\mathsf{L}_{\langle\vec{y}\rangle}[\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}]$ as a shorthand for $\mathsf{L}_{\vec{x}}[\mathbb{G}]\langle\vec{y}\rangle$
- A flattening axiom for some inessential design label L takes the form $\mathsf{L}_{\vec{x}}\mathbb{G}\langle\vec{y}\rangle \equiv \mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ (but $\mathbb{G}\{^{\vec{y}}/_{\vec{x}}\}$ has still type L)
- Structural equivalence as graph isomorphism

Some Sketches of Encoding

π-calculus in ADR (**P** process type, **G** guarded sums type)

$$
\begin{aligned}
[\![(\nu x)Q]\!] &= \mathbf{P}_p[\ (\nu x)[\![Q]\!]\langle p\rangle\ ] \\
[\![N + M]\!] &= \mathbf{G}_p[\ [\![N]\!]\langle p\rangle \ \mid\ [\![M]\!]\langle p\rangle\ ] \\
[\![Q \mid R]\!] &= \mathbf{P}_p[\ [\![Q]\!]\langle p\rangle \ \mid\ [\![R]\!]\langle p\rangle\ ]
\end{aligned}
$$

CaSPiS in ADR (**P** process type, **S** session type)

$$
\begin{aligned}
[\![Q \mid R]\!] &= \mathbf{P}_{p,i,o,r}[\ p \mid i \mid o \mid r \mid [\![Q]\!]\langle p,i,o,r\rangle \ \mid\ [\![R]\!]\langle p,i,o,r\rangle\ ] \\
[\![s^{+} \rhd Q]\!] &= \mathbf{P}_{p,i,o,r}[\ i \mid o \mid \mathbf{S}_{\langle p,r\rangle}[\ [\![Q]\!]\langle p,s^{+},s^{-},r\rangle\ ]\ ] \\
[\![s^{-} \rhd Q]\!] &= \mathbf{P}_{p,i,o,r}[\ i \mid o \mid \mathbf{S}_{\langle p,r\rangle}[\ [\![Q]\!]\langle p,s^{-},s^{+},r\rangle\ ]\ ]
\end{aligned}
$$

Some Sketches of Encoding

π-calculus in ADR (**P** process type, **G** guarded sums type)

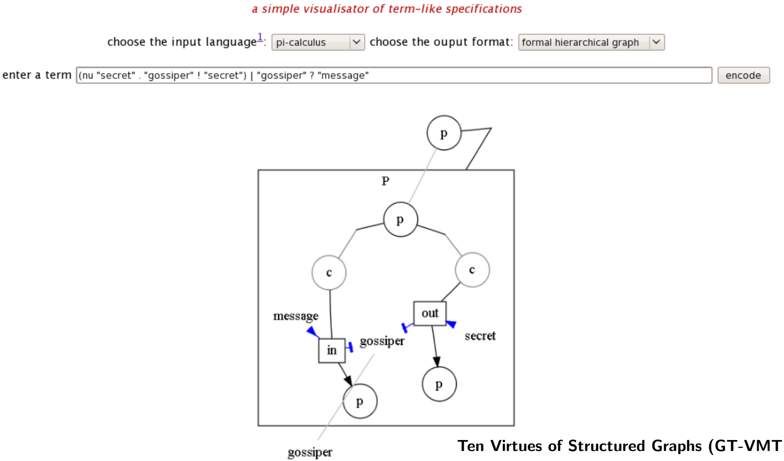
$$
\begin{aligned}
[\![(\nu x)Q]\!] &= \mathbf{P}_p[\ (\nu x)[\![Q]\!]\langle p \rangle\ ] \\
[\![N + M]\!] &= \mathbf{G}_p[\ [\![N]\!]\langle p \rangle \mid [\![M]\!]\langle p \rangle\ ] \\
[\![Q \mid R]\!] &= \mathbf{P}_p[\ [\![Q]\!]\langle p \rangle \mid [\![R]\!]\langle p \rangle\ ]
\end{aligned}
$$

CaSPiS in ADR (**P** process type, **S** session type)

$$
\begin{aligned}
[\![Q \mid R]\!] &= \mathbf{P}_{p,i,o,r}[\ p \mid i \mid o \mid r \mid [\![Q]\!]\langle p,i,o,r \rangle \mid [\![R]\!]\langle p,i,o,r \rangle\ ] \\
[\![s^+ \rhd Q]\!] &= \mathbf{P}_{p,i,o,r}[\ i \mid o \mid \mathbf{S}_{\langle p,r \rangle}[\ [\![Q]\!]\langle p,s^+,s^-,r \rangle\ ]\ ] \\
[\![s^- \rhd Q]\!] &= \mathbf{P}_{p,i,o,r}[\ i \mid o \mid \mathbf{S}_{\langle p,r \rangle}[\ [\![Q]\!]\langle p,s^-,s^+,r \rangle\ ]\ ]
\end{aligned}
$$

Visualization: adr2graphs (Early Prototype)

Please have a try at http://www.albertolluch.com/adr2graphs



One Last Virtue of Structured Graphs

- Logical specification and verification
  - ad-hoc spatial logics: from “general” to “derived” modalities
  - formulas closer to visualization (easier to use)
  - types as properties: a property $\mathcal{P}$ demonstrated by structural induction on type T show that all graphs of type T satisfy $\mathcal{P}$.
  - re-use existing (efficient) tools whenever possible

Outline

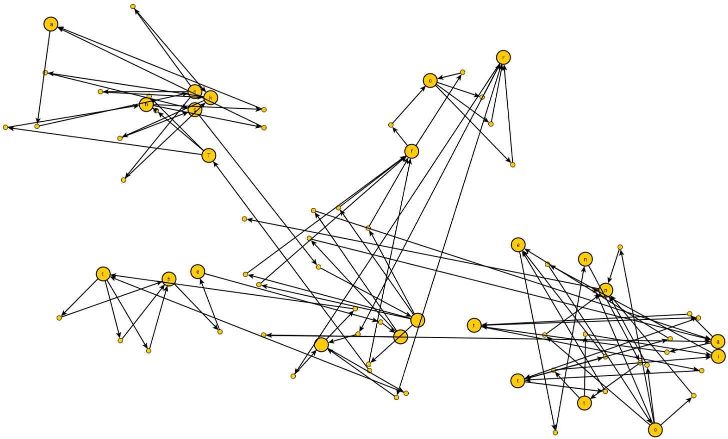
Where ADR can help

- Design of software architectures
  - drop & bind components + check + correct: tedious, error prone
  - bounded FO/SAT (Alloy): performant, but trial & error, no hint, no guidance
- Guaranteed reconfiguration
  - prove theorems on GT: ad-hoc, manual, limited re-use
  - model checking on GT: validate a particular instance, scalability issues, undecidable in general
  - monitor & repair: no guarantees
- Usability
  - other integrated environment require acquaintance with many different languages and theories

Related work

- Ordinary GT:
  - nice theory of concurrency, but structure must be encoded somehow in flat graphs,
  - problems with grouping and atomicity
- Hierarchical graphs:
  - main difference relies on interfaces
- Alloy:
  - highly specialized SAT solver, but Maude is more flexible

End of Talk (Graphs Powered by yEd)



End of Talk (Graphs Powered by yEd)