Graphical Languages for Modeling Complex Reactive Systems

Or: Three proposals to argue with . . .

Reinhard von Hanxleden

Christian-Albrechts Universität Kiel
Faculty of Engineering
Dept. of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
www.informatik.uni-kiel.de/~rvh

SYNCHRON 2003, December 2003

Graphical Modeling—A Good Thing!

> *Today, we see with some surprise that visual notations for synchronous languages have found their way to successful industrial use with the support of commercial vendors. This probably reveals that building a visual formalism on the top of a mathematically sound model gives actual strength to these formalisms and makes them attractive to users.*
>
> Benveniste et. al.

Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone.
The Synchronous Languages Twelve Years Later.
In *Proceedings of the IEEE, Special Issue on Embedded Systems*, volume 91, pages 64–83, January 2003.

Graphical Modeling—A Good *Enough* Thing?

Observation 1:

- Graphical languages are convenient to browse
- . . . but are can be a pain to *edit*!

Observation 2:

- Graphical languages are appealing
- . . . but not necessarily *effective* in conveying technical information!

Observation 3:

- Graphical models already work well to visualize complex structures
- . . . but are still limited for visualizing complex *behaviors*!

Overview

State of the Practice

- WYSIWYG editors to create graphical models
- Some editors offer alignment tools
- Creating graphical models slow
- In initial phase, often resort to paper and pencil
- Maintaining graphical models time consuming

State of the Practice

> *I quite often spend an hour or two just moving boxes and wires around, with no change in functionality, to make it that much more comprehensible when I come back to it.*
>
> One practitioner, according to Petre

Marian Petre.
Why looking isn't always seeing: readership skills and graphical programming.
Communications of the ACM, 38(6):33–44, June 1995.

What is the Problem?

- Inserting elements tedious
  - Text: Just enter new elements!
  - Graphics: Often must "make room" first;
    Often need to modify adjacent elements as well
- Deleting elements also tedious
  - Text: Just delete unneeded elements!
  - Graphics: Often leaves distracting "holes"
    Again may need to modify adjacent elements as well

What is the *Real* Problem?

Non-linearity

- Text: 1-D
- Graphics: 2-D

Context entanglement

- Transitions
- State hierarchy, concurrency

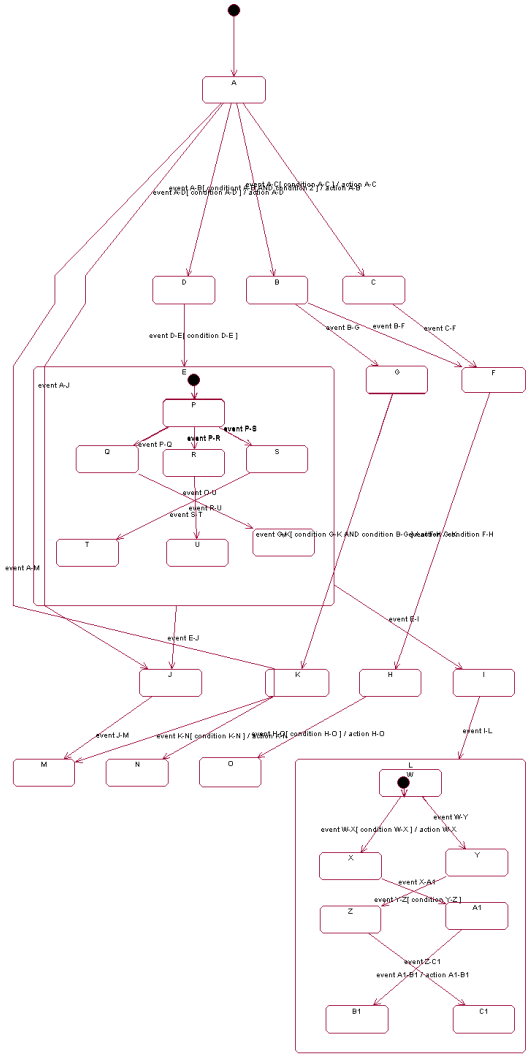
The conclusion

- Let the computer do the layout!
- User provides quick & dirty graphical model (WYSIWIG)
- Alternative: User provides textual model
  ... thus opening up another world of possibilities!
  - Automated model-derivation
  - Configurability
  - Scalability
  - ...
- Question: Why is it not already done?

Automated Graph Layout

Answer: It's not trivial

Example: Autolayout from Rational Rose



Castelló et al.

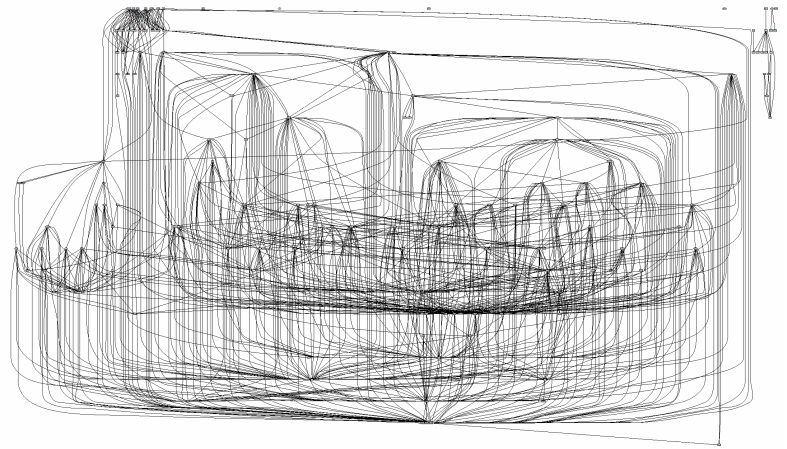
Automated Graph Layout

- . . . an established discipline of computer science
- Numerous applications
  - Circuit design
  - Flow charting
  - Motion planning
- Often formulated as optimization problem
  - Minimize cost function (edge lengths, crossings)
  - Meet constraints (no elements overlap)

Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis.
Algorithms for drawing graphs: An annotated bibliography.
Computational Geometry: Theory and Applications, 4:235–282, June 1994.

Typical Graph Problem Characteristics

- Relatively simple problem formulation
  - Nodes of no/fixed spatial extent
  - Edges straight/rectilinear
  - No labels
  - No hierarchy
- Want to solve large problem instances
  - Often $> 10^3$ nodes, $\gg 10^3$ edges



Gansner/North

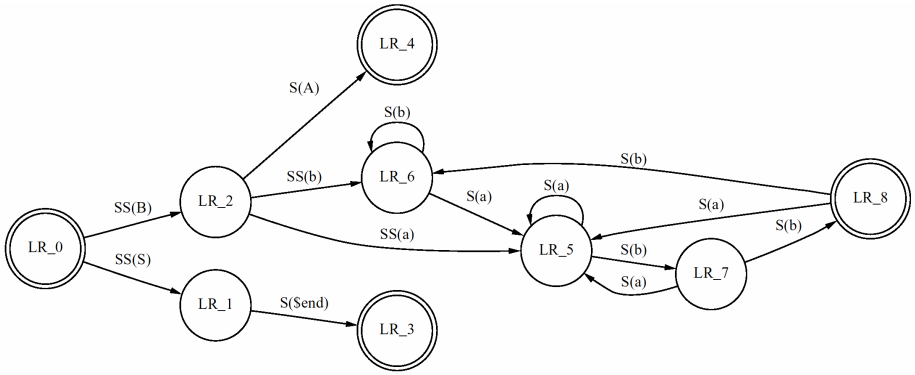
Statechart Layout Characteristics

- Problem formulation rather complicated
  - Nodes of varying spatial extent
  - Transitions (edges) may be splines
  - Labels for nodes and transitions; comments
  - Hierarchy
- Problem instances relatively small
  - Hierarchy decomposes problem
  - Typically, < 20 states/level

Previous Work

The graphviz system

- General scheme for drawing directed graphs
- Layering, Simplex algorithm to minimize back-edges
- Generates splines, positions labels
- `http://www.research.att.com/sw/tools/graphviz/`

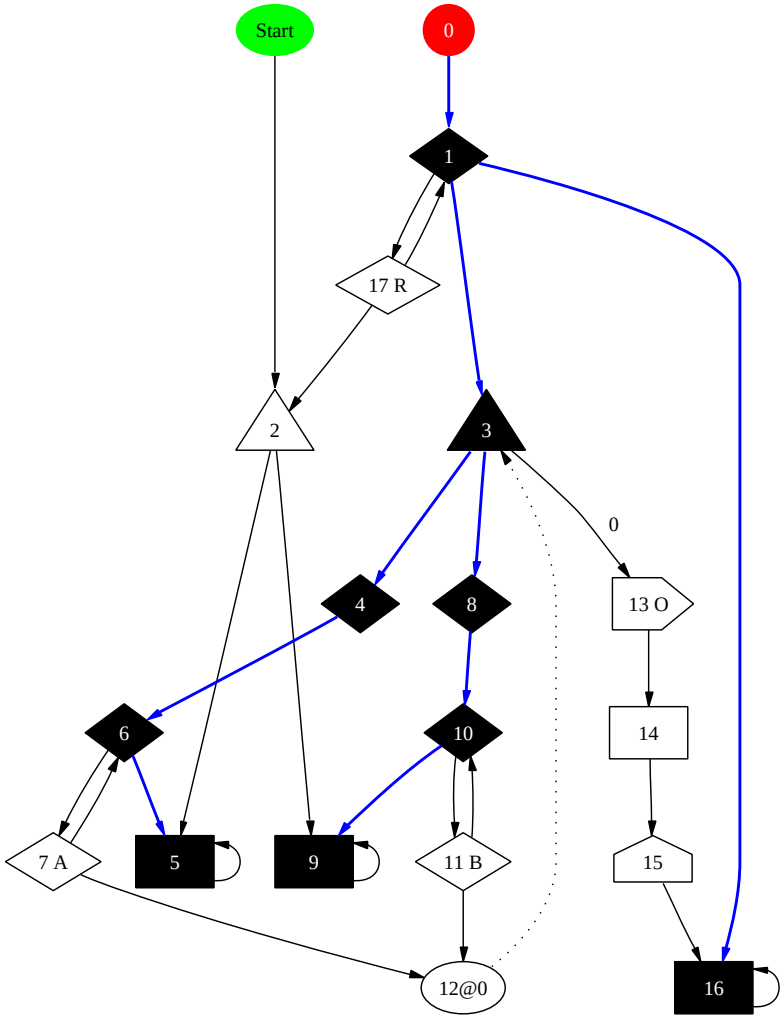


Emden R. Gansner and Stephen C. North.
An open graph visualization system and its applications to software engineering.
Software—Practice and Experience, 30(11):1203–1233, September 2000.

Previous Work

ic2dot [Edwards]

- Based on Esterel IC format
- Generates input for graphviz system
- `http://www1.cs.columbia.edu/~sedwards/software.html`

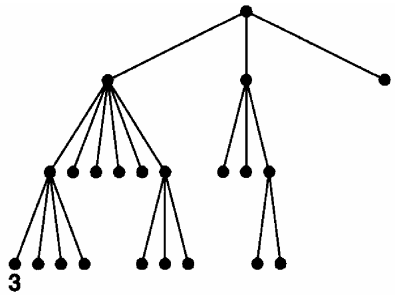


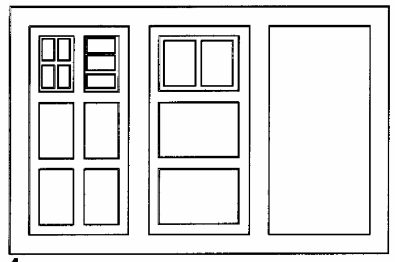
Previous Work

Harel et al.

- Blobs: nested rectangles
- No transition or label placement

D. Harel and G. Yashchin.
An Algorithm for Blob Hierarchy Layout.
The Visual Computer, 18:164–185, 2002.





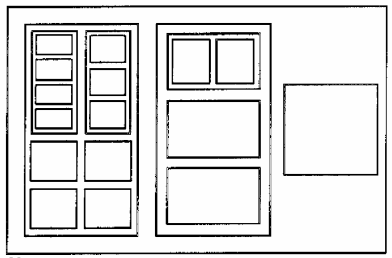


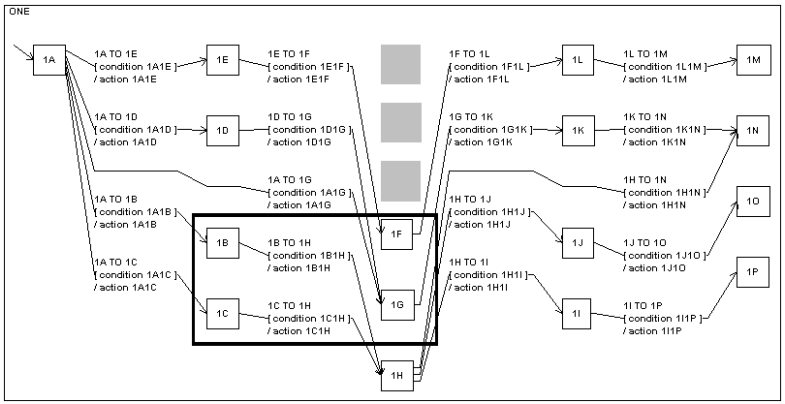
Fig. 3. Tree representation of a hierarchy

Fig. 4a,b. Uniformity of blob dimensions: **a** uniform size siblings; **b** uniform size similar siblings and leaves

Previous Work

Castelló et al.

- Targets Statecharts
- Layering + floor planning
- Heuristics to minimize “edit disruption”



Rodolfo Castelló, Rym Mili, and Ioannis G. Tollis.
A Framework for the Static and Interactive Visualization for Statecharts.
Journal of Graph Algorithms and Applications, 6(3):313–351, 2002.

The KIEL Statechart Layouter

- Kiel Integrated Environment for Layout
- Uses simple linear horizontal/vertical layout heuristic
- Uses layering scheme for placing transitions
- Implemented in Java
- Plugins for ArgoUML (UML Statecharts) and Esterel Studio (Safe State Machines)

The KIEL Statechart Layouter

Options:

- State alignment (center, top/left, bottom/right)
- Transition shape (splines, polygons)
- Layout depth (deep, shallow)
- Select and order layout criteria
  - Minimal number of transition crossings
  - Minimal number of transition layers
  - Balanced number of transition layers
  - Positioning of initial and final states

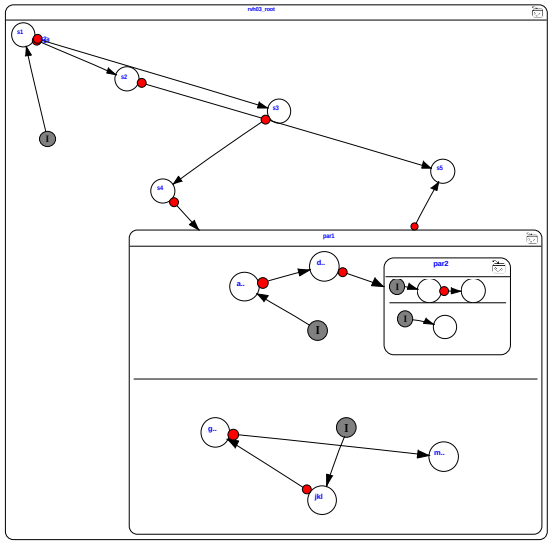
Horizontal/Vertical Layout

- Optimization problem is very constrained
  - Must choose (linear) permutation of states per level
  - Rest follows mechanically, bottom-up
- Currently use highly un-optimized brute force algorithm

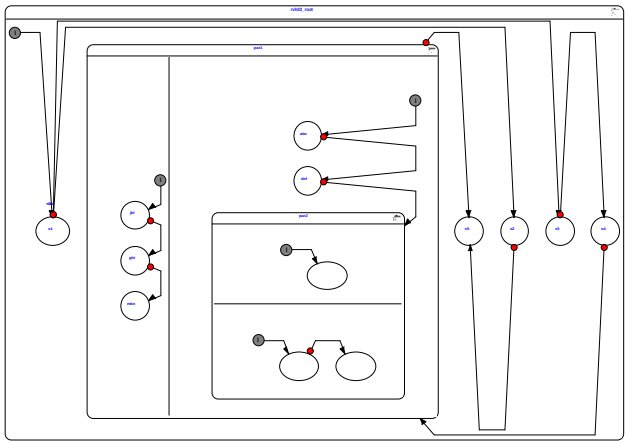
```
for all charts
  for all hierarchy levels
    pick some permutation P_Opt of states
    for all other permutations P
    for all criteria C, in order of ranking
      if P better than P_Opt under C
        P_Opt := P
      else if P worse than P_Opt under C
        break
```

Layout Example 1

Original SSM:

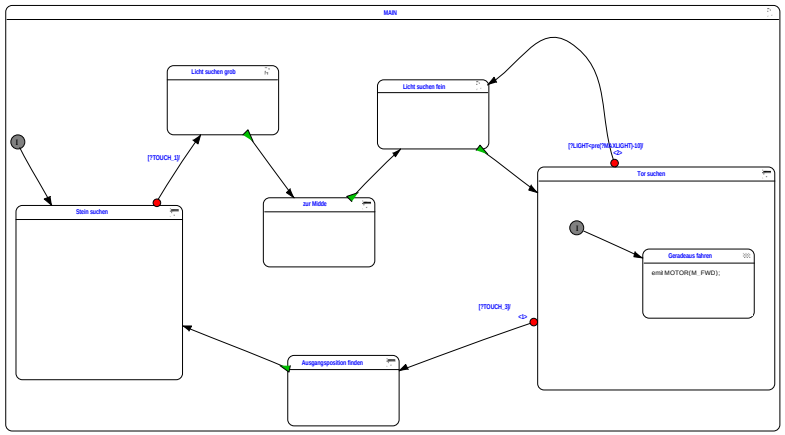


Layout Result:

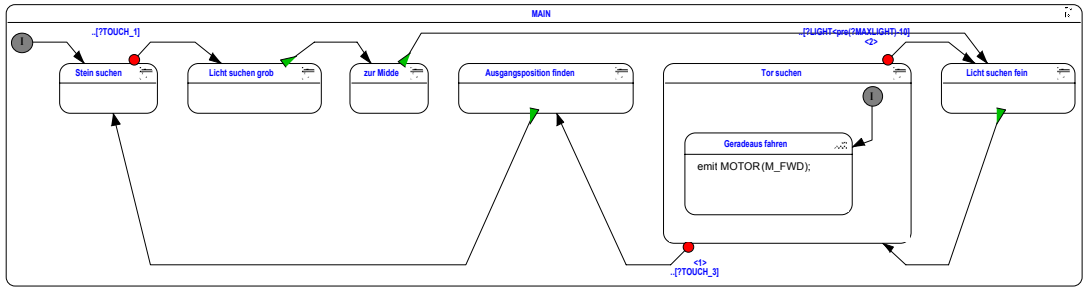


Layout Example 2

Original SSM:

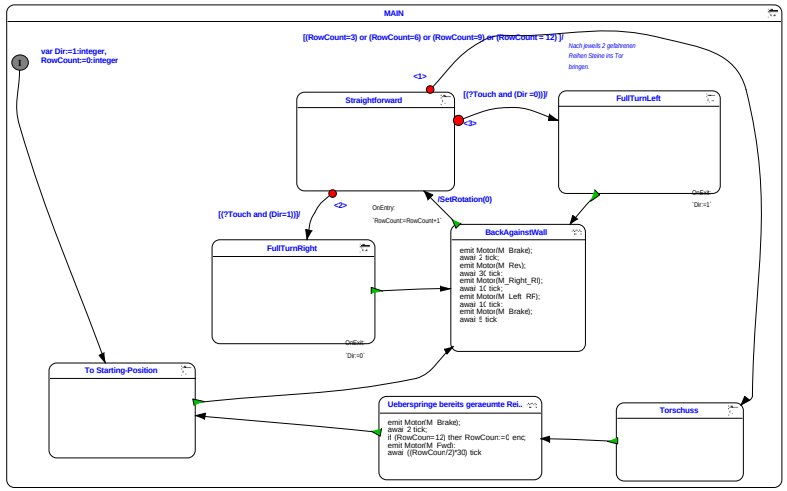


Layout Result:

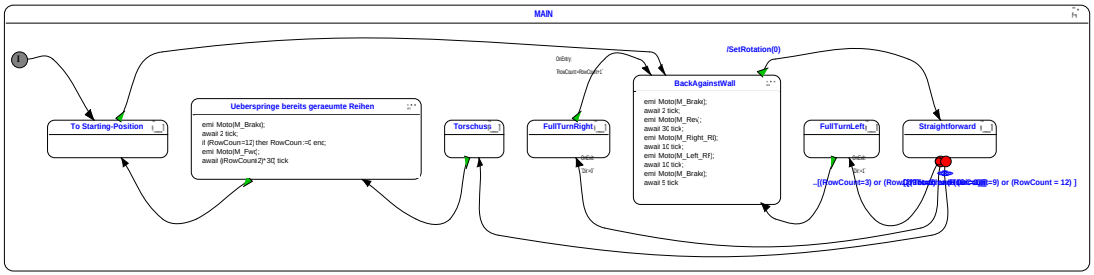


Layout Example 3

Original SSM:



Layout Result:



Assessment

Layout results

- State placement generally satisfactory
- Transition and label placement should be improved
- Computation times OK in most cases (couple of seconds)
- However, larger instances slow down significantly

Side effects

- Initial entry/modification of model accelerated
- Lower threshold to use modeling tool—more exploratory process

Overview

How effective are graphical languages?

Recall Observation 2, Part 1:

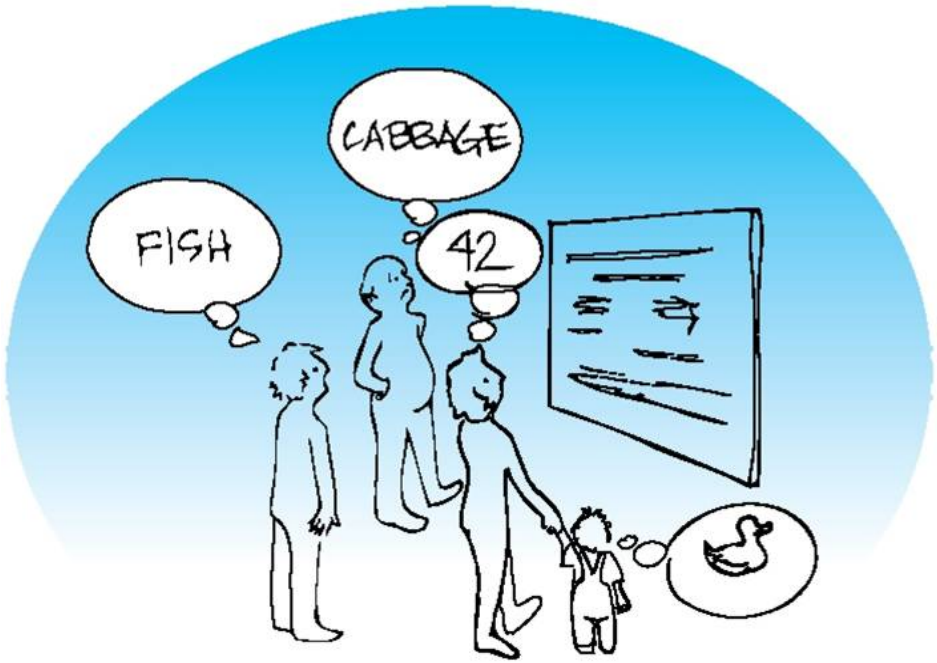
Graphical languages are *appealing*

- Richness
- 'Gestalt'-effect
- High-level of abstraction
- Mapping to the domain
- Accessibility, comprehensibility
- Fun

How effective are graphical models?

But also recall Part 2:

Graphical languages are not necessarily *effective* in conveying technical information!



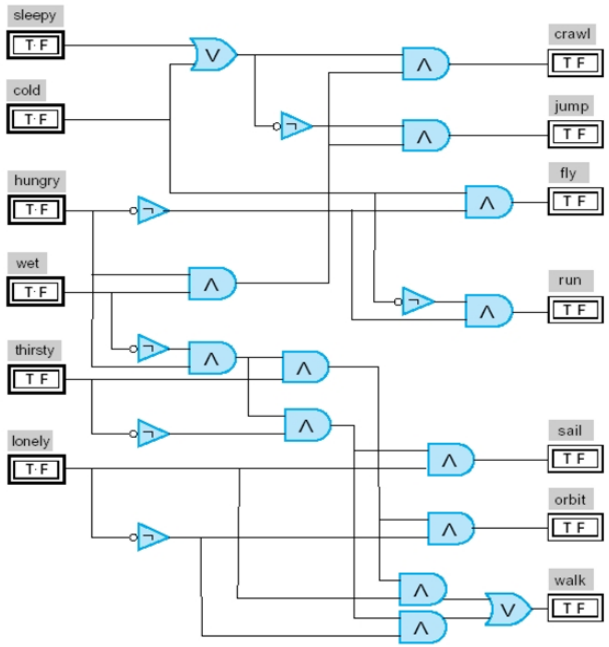
Petre/Communications of the ACM

An Experiment on Reading Comprehension

An empirical study [Petre 1995]

- Subjects: design experts
- Representations: nested structure, max cues
- Questions like "What does this do?"

```
howl:   if honest & tidy &
(lazy | sluggish)
laugh:  if honest & tidy &
!lazy & !sluggish
whisper: if honest & !tidy
& (nasty & greedy |
!nasty & !greedy)
bellow: if honest & !tidy &
nasty & !greedy
groan:  if honest & !tidy &
!nasty & greedy
mutter: if !honest & sluggish
shout:  if !honest & !sluggish
```



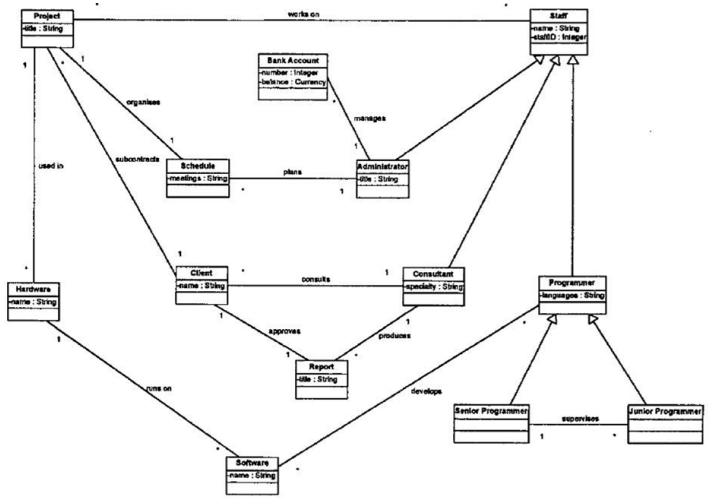
An Experiment on Reading Comprehension

Results

- Graphic slower than text, **in all conditions**
- Uniform - for every single tested subject
- Subjects text as guide for graphics if possible

An Experiment on UML Comprehension

Aim: Verify that layout rules proposed for diagrams in general are effective for UML class diagrams



Helen C. Purchase, Matthew McGill, Linda Colpoys, and David Carrington. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study.
In *ACM International Conference Proceeding Series archive, Australian symposium on Information visualisation*, pages 129–137, 2001.

Tested Criteria

- Minimized number of bends
- Evenly distributed nodes
- Uniform lengths of edges
- Consistent direction of flow
- Orthogonality

The Experiment

- Subjects: 30 students 3rd year Computer Science
- No prior knowledge of UML
- Preparation: 15 minutes UML tutorial
- Four examples of task to perform
- Reference: Textual specification in simple English
- Task: decide whether diagram (in)correct
- Measured speed and accuracy
- Within subject analyses

Results

- Nearly all answers correct
- Response time varied significantly
- Several surprising results
  - Low number of bends worsened performance
  - Medium edge variation better than small edge variation
- **None of the expectations satisfied**
- Conclusion: Must consider semantics as well
- Beyond generic aesthetic criteria, secondary notation is critical, *specific* to modeling language

Secondary Notation

- Typically not part of notation
- Provide additional hints to reader
  - Adjacency
  - Clustering
  - White space
  - Labeling ...
- Effectively result in language sub-setting

Secondary Notation

- Example of poor SN:

```
while ((used!=1) || (a[0] !=1)) { if (a[0] & 0x1)
{ k=1; for (c = 0; c <= used; c++)
{ a[c] = 3 * a[c] + k; k = a[c] / 10; a[c] = a[c] % 10;}
if (a[used]) { used++; if (used >= 72)
{ printf ("Run out of space\n"); exit(1);}} }
else {k = 0; for (c = used - 1; c >= 0; c--)
{ a[c] = a[c] + 10*k; k = a[c] & 0x1; a[c] = a[c] >>1;}
if (a[used - 1] == 0) used--; }count++; }
```

Secondary Notation

- Example of better SN:

```
while ((used!=1) || (a[0] != 1)) {
  if (a[0] & 0x1) {
    k=1;
    for (c = 0; c <= used; c++) {
      a[c] = 3 * a[c] + k;
      k = a[c] / 10;
      a[c] = a[c] %10;
    }
    if (a[used]) {
      used++;
      if (used >= 72) {
        printf ("Run out of space\n");
        [...]
```

Experts vs. Novices

- Presence of SN not enough
- Good use of SN required *learned* skills
- Novice users
  - make little use of structure and SN
  - have difficulties to determine relevant elements
- Expert users
  - make good use of SN in drawing and reading
  - use consistent strategies as a group
  - search in reduced space—focus on relevant parts

The Proposal

- Develop catalogue of efficient secondary notations for Statecharts
  (Style Guide; Normal Forms)
- Provide support for conformance checking
  (Style Checker)
- Provide support for generating conformant diagrams
  (Pretty Printer)

Proposing Some Secondary Notations for Statecharts

Placement of initial and final state

- Goal: Aid identification of initial/final state
- Example: Top/left, bottom/right, respectively

Placement of remaining states

- Goal: Support understanding of state sequencing
- Example: Minimize back transitions

Shape of transitions

- Goals: State sequencing; prominent source/sink states
- Example: Clock-wise orientation

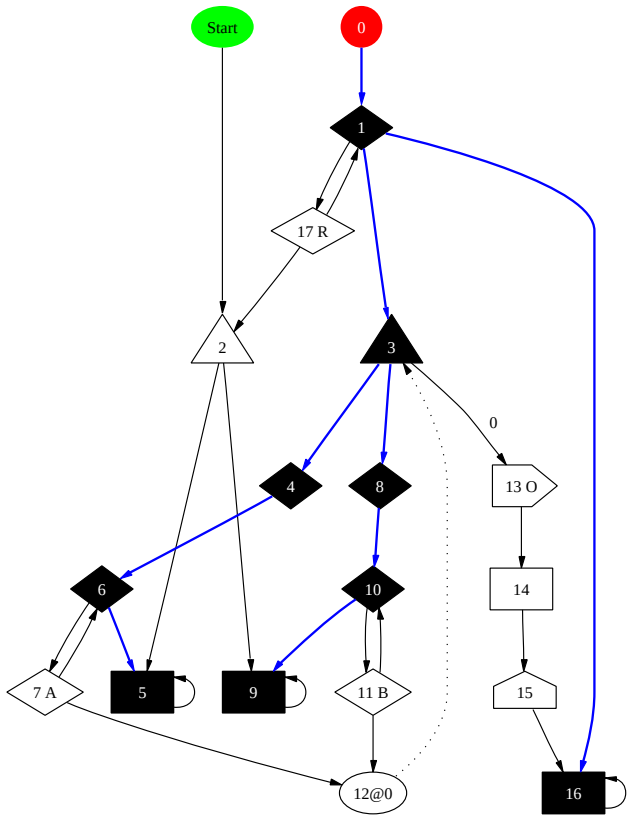
Proposing Some Secondary Notations for Statecharts

Placement of labels

- Goal: Easy matching of labels and transitions
- Example: Left of transition, relative to direction

Exploitation of symmetry

- Goal: Highlight design regularities
- Example: ???



Secondary Notation in KIEL

- Place initial states top/left
- Place final states bottom/right
- Clock-wise orientation of transitions
- Consistent placement of labels
- Try to put successive states adjacently
- Linear layout
- Not yet: minimize back transitions

Secondary Notation in KIEL—Assessment

- Clock-wise orientation of transitions surprisingly helpful
- Have effectively defined a (loose) normal form
- *After getting used to having normal form, other diagrams look unnecessarily unordered/distracting!*

Overview

Visualizing Complex Behaviors

Recall Observation 3:

- Graphical models already work well to visualize complex *structures*
- . . . but are still limited for visualizing complex *behaviors*!

But wait—isn't behavior what Statecharts are all about?
Yes, but . . .

Visualizing Complex Behaviors with Statecharts

- View of single Statechart rarely suffices—need to see several active Statecharts at once
- Set of active Statecharts changes dynamically
- Keeping active charts in foreground requires significant additional user effort during simulation
- Could envision scheme that automatically brings active charts in foreground
- However, this appears
  - non-trivial
  - probably still not sufficient
- . . . and what exactly does "active" mean, anyway?

What is the Problem?

- Concurrency
- Fixed level of detail
- Spreading system across several charts (windows) aids model creation and maintenance
  . . . but results in fragmented overall picture
- CONCURRENCY

The Proposal

1. Provide overview of whole system in single picture (Deep Layout)
2. Allow level of detail to vary
3. Dynamic Statecharts

Deep Layout—Previous Work

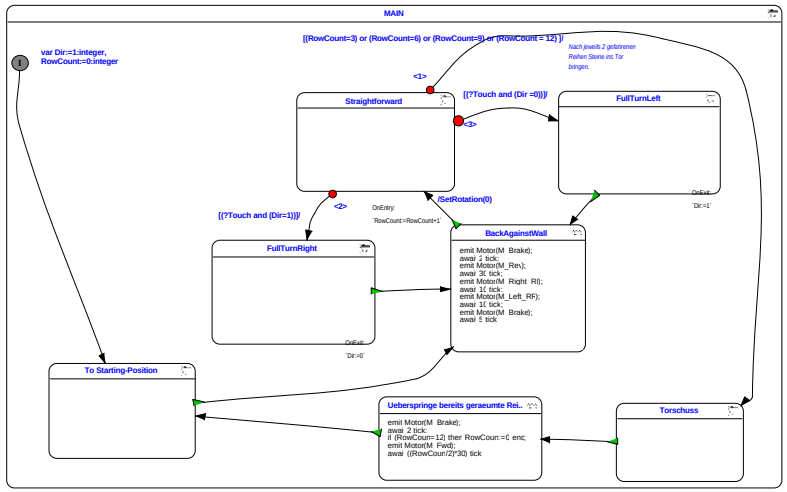
Jaffe et al.

- RSML (Requirement State Machine Language)
- Combines graphical (states + transitions) with textual (transition triggers, data dictionary) syntax
- Synthesizes graphical system overview from textual input language

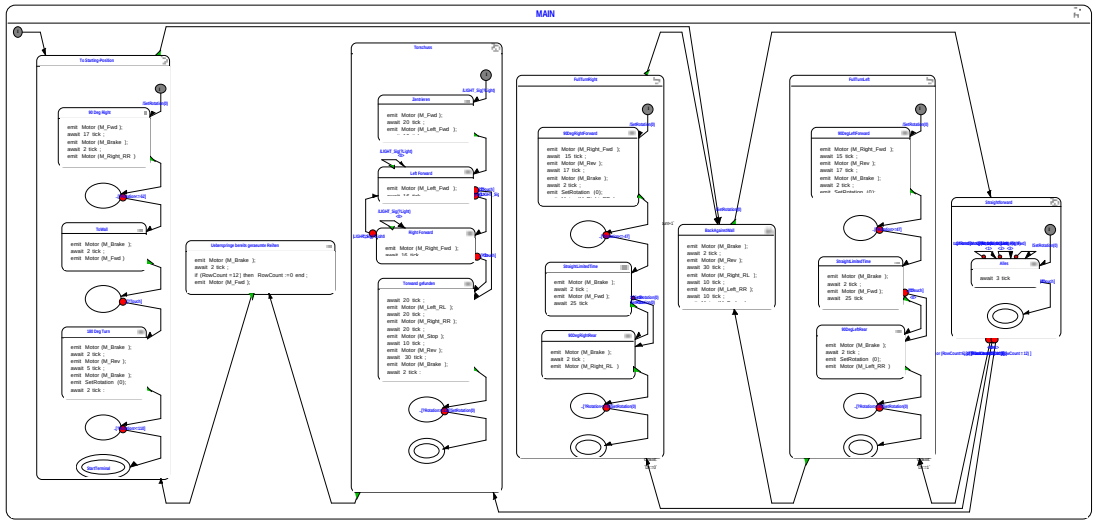
Matthew S. Jaffe, Nancy G. Leveson, Mats P. E. Heimdahl, and Bonnie E. Melhart.
Software requirements analysis for real-time process-control systems.
IEEE Transactions on Software Engineering, 17(3):241–258, March 1991.

KIEL—Example of Deep Layout

Original SSM:



After Layout:



Is (Static) Deep Layout Good Enough?

Example: Statemate model of airbag control

- Individually, 18 Activity Charts + 26 Statecharts
- After instantiation, $6 + 17 * 2 + 21 = 61$ Activity Charts + $12 + 17 * 15 + 21 = 288$ Statecharts!

Dynamic Statecharts

- Introduce different system views, defining
  - visible parts of the system
  - visible level of detail
- Present dynamically changing views dependent on
  1. Simulation state
  2. User requests
- A dynamic extension to semantic focus-and-context representation

Oliver Köth.
Semantisches Zoomen in Diagrammeditoren am Beispiel von UML.
Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2001.

Dynamic Statecharts

Idea: Views should hide in-active sub-states

Construction of views:

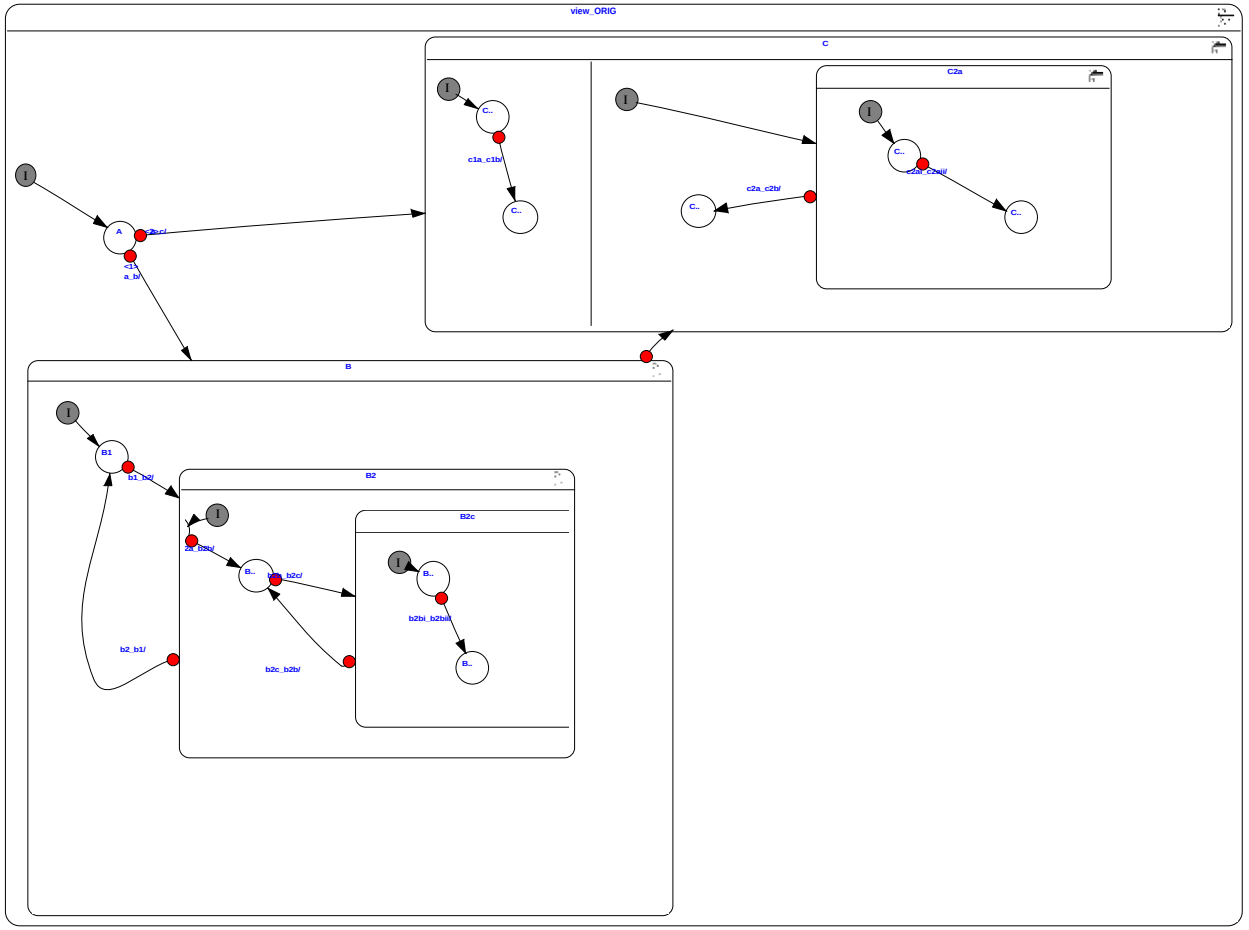
- Define views for all final (bottom-most) OR states
- Compose views of AND states
- Build up views hierarchically
- Note: How to do this in a semantically clean fashion is not always obvious!
  - Example: Transitions entering/exiting in-active sub-states
- Results in as many views as composite states (Macrostates)
- However, each view shows complete system!

During simulation:

- Enter view associated with *current* parent (composite) state

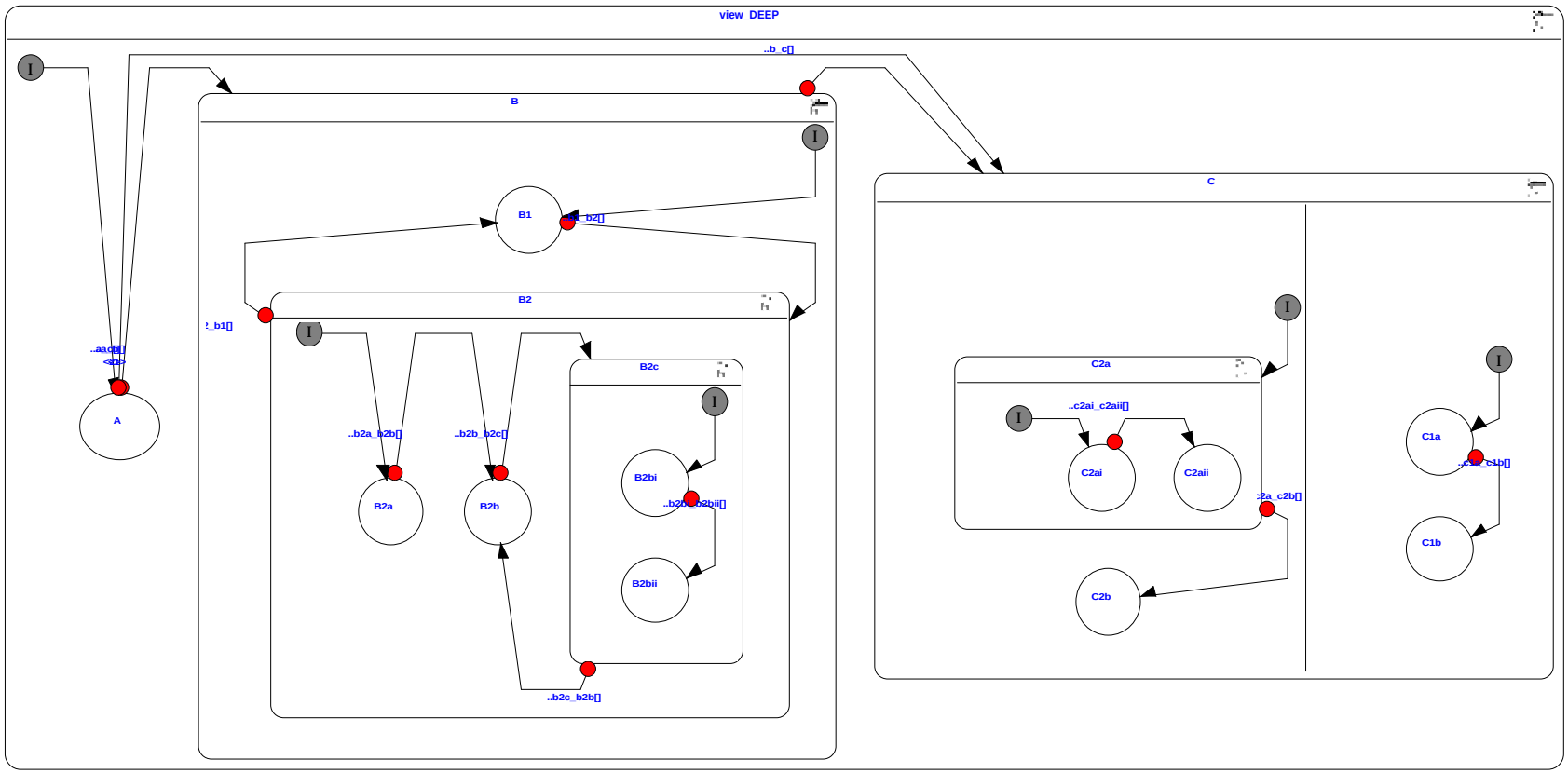
Example of Dynamic Statecharts

Static view, before layout:



Example of Dynamic Statecharts

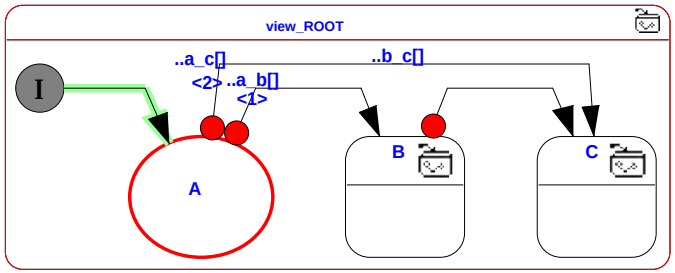
Static view, after layout:



Example of Dynamic Statecharts

After start of simulation, change to dynamic view, to visualize details of currently active states while retaining overview of complete system

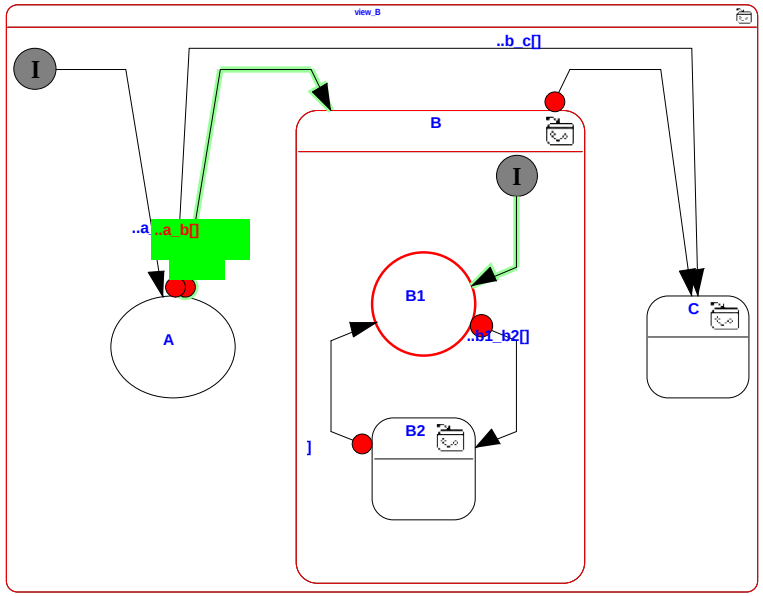
- Enter state `A`
- Present view associated with parent of `A`—which is the `ROOT` state
- Details of states `B` and `C` are hidden so far



Example of Dynamic Statecharts

Transition from A to B:

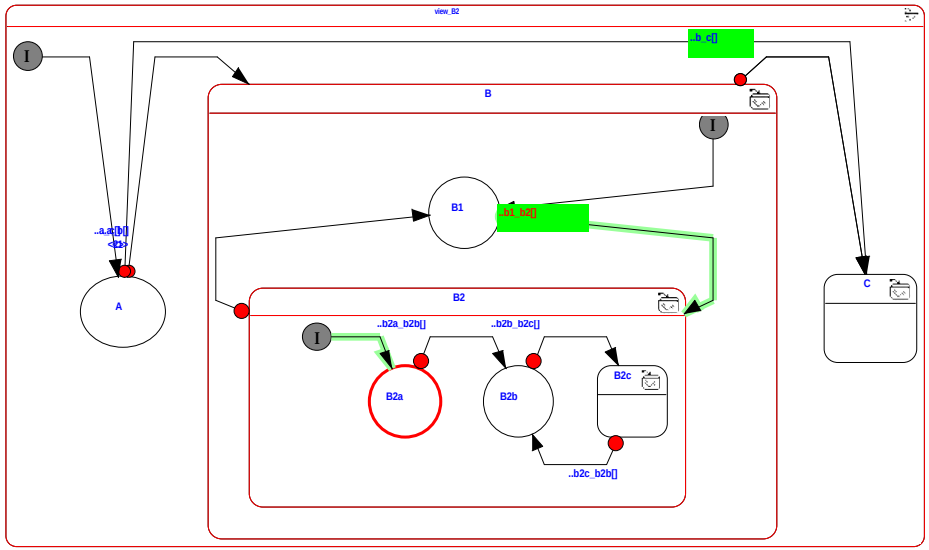
- Enter sub-state B1
- Present view associated with parent of B1—which is B
- Exhibit details of state B
- Still hide details of sub-states of B
- During simulation, should provide smooth blending between views to preserve mental map



Example of Dynamic Statecharts

Transition from B1 to B2:

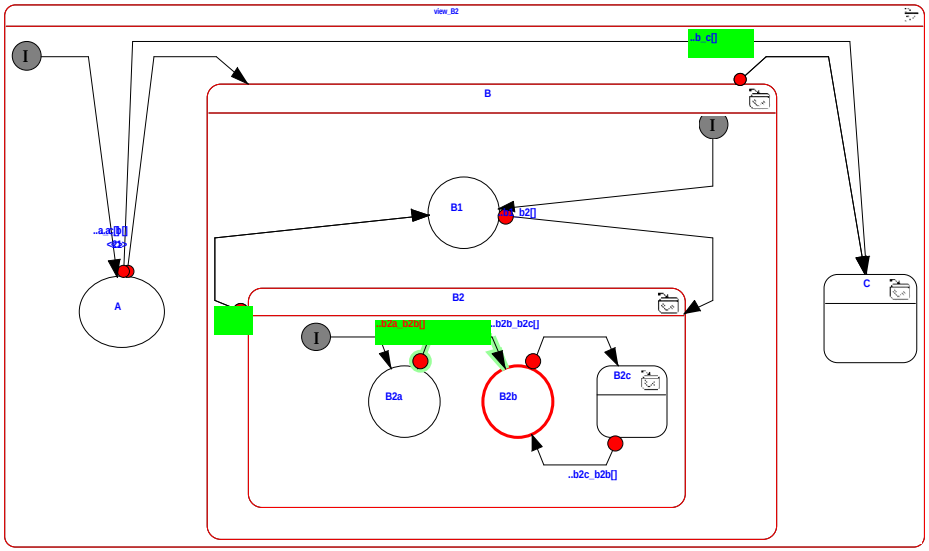
- Enter sub-state B2a
- Present view associated with B2



Example of Dynamic Statecharts

Transition from B2a to B2b:

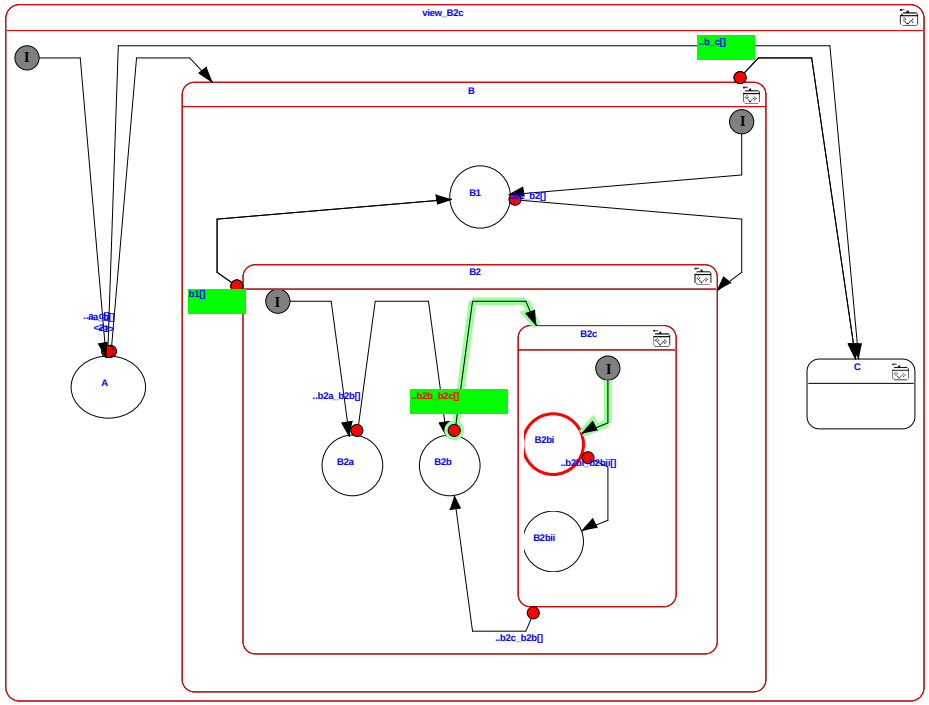
- Stay in view associated with B2



Example of Dynamic Statecharts

Transition from `B2b` to `B2c`:

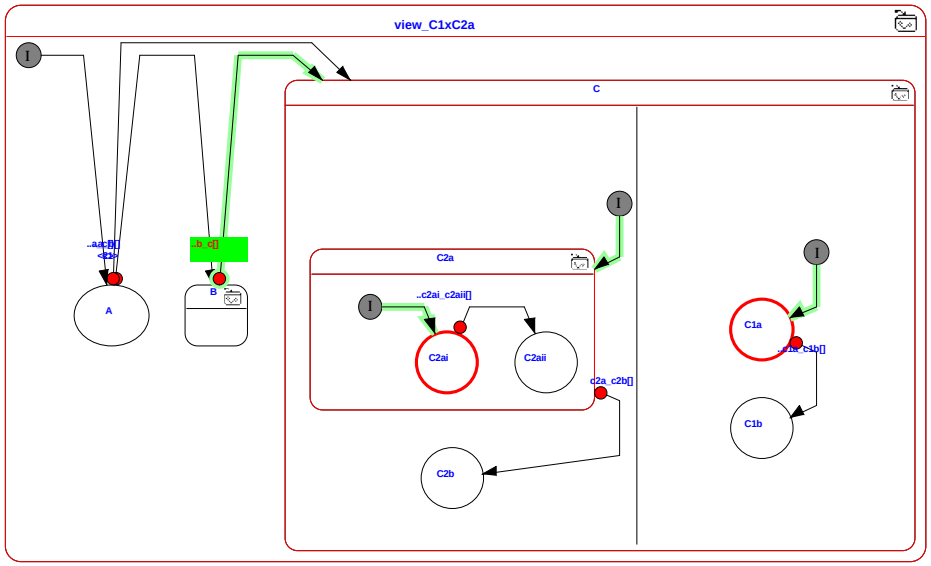
- Enter sub-state `B2ci`
- Present view associated with `B2c`



Example of Dynamic Statecharts

Transition from `B` to `C`:

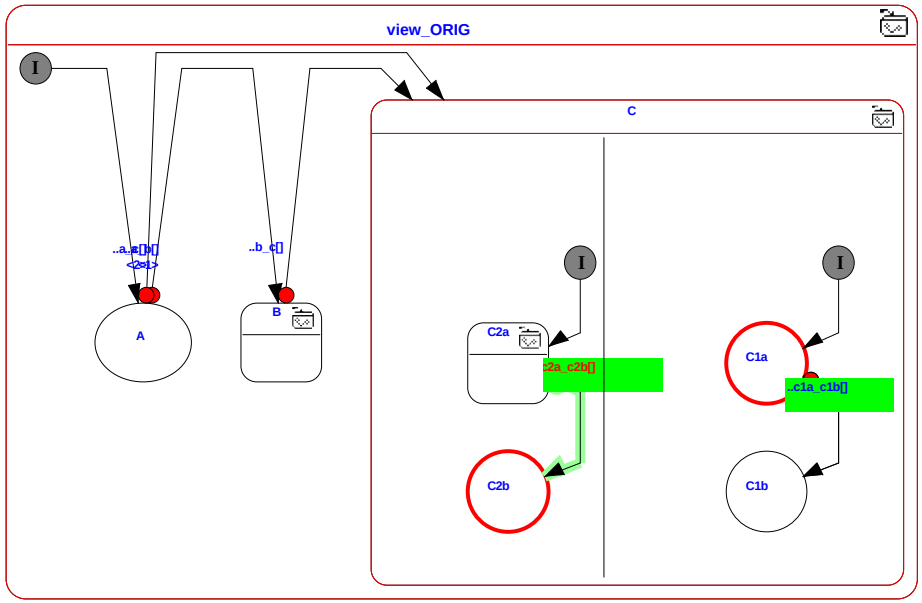
- Enter parallel sub-states `C1a` and `C2ai`
- Present view associated with `C1` and `C2a`



Example of Dynamic Statecharts

Transition from C2a to C2b:

- Present view associated with C1 and C2



Statistics for this example:

- 6 Macrostates + 11 simple states
- 6 views
- 12 state configurations

Dynamic Statecharts

Beyond the *highlight current transition/next state* paradigm

- Show last n transitions/states
- Show (future) enabled transitions
- Restrict scope to set of user-selected variables
- Restrict scope to set of dynamically-selected variables
- ...

Summary

> *It appears that graphical notations can have a greater capacity to 'go wrong' than textual notations. . . . It is time to recognize the impact of 'bad graphics'—of haphazard use of perceptual cues and secondary notations—mis-cueing, misleading, misreading, and misunderstanding.*
>
> Petre

- We have developed clean semantics—*but have we paid enough attention to a clean graphical representation and its proper usage?*
- This question particularly pressing for the application domains the synchronous languages are concerned with

Summary

Three Observations and Proposals:

1. Graphical models nice to browse, but hard to write—so let the computer help more!
2. Graphical languages appealing, but not effective enough—should develop and consciously use secondary notations!
3. Graphical languages good for understanding structures, but (so far) bad for analyzing dynamics—use dynamic charts!

Summary

The KIEL Prototype

- Automatic layout of Statecharts
- Simple horizontal/vertical heuristic
- Simple layering scheme
- Not optimized
- Can do flat or deep layout
- Interfaces to Esterel Studio and ArgoUML
- Layout of states already quite satisfactory
- Layout of transitions and labels need to be improved
- Simple set of secondary notations already appears effective

Going further

- KIEL
  - Examine alternative layout schemes
  - Extend supported input languages (Esterel!)
  - Interface to Statemate and Rhapsody
  - Consider move to C++/OpenGL
- Explore dynamic Statecharts
- Refine secondary notations for Statecharts (et al.)
- Cognitive experiments

thanks!

questions or comments?