Software Models and Representations
Part 3
Graphs Other than DFGs

Formalizing DFGs as Relations

- \((i, j) \in \text{DataFlow}_G\) if node \(i\) creates data that node \(j\) uses in \(G\)
- \(\text{INPUT}_G\) -- set of nodes (input) such that input is a provider of input to \(G\) from an external source
- \(\text{OUTPUT}_G\) -- set of nodes (output) such that output is a conveyor of artifacts computed by \(G\) to an external source

- Preferably the data artifact is defined rigorously

- \((n, \text{text}) \in \text{NodeAnnotation}_G\) if the string \(\text{text}\) describes the functioning of node \(n\) in \(G\)

- This may imply different semantics (e.g. abstraction)

InputAnnotation, OutputAnnotation are similar

Questions this helps answer:
- Why create this data? Who uses this data? What results does the end user see? What does the end user have to input?
- Questions this can’t answer: What is the exact sequence of events? How does a node do its job?

Hierarchy

- Enables incrementally adding detail
- Increased precision too
- Draws upon innate human mental capability
  - Abstraction
  - Encapsulation
- A typical solution to the problem of needing detail, but needing to avoid overload
- But creates potential problems

Consistency is a principal concern

- Are the diagrams consistent with each other?
- Top view consistent with elaborations?
  - Arrows consistent
  - Data flows consistent
  - Other semantics?
- Are we seeing different shadows of the same object?
- Invitation to subtle errors

Simple Hierarchical Elaboration

![Simple Hierarchical Elaboration Diagram]

Something a Little More Complex

![Something a Little More Complex Diagram]
(Some) Consistency definitions

- Let \( (n, \text{somenode}) \in \text{NodeAnnotation} \backslash \text{ParentGraph} \) for some \( n, \text{somenode} \in N \), where \( \text{ParentGraph} = (N, E) \)
- Let \( \text{somenode} = (N', E') \)
  - This is the DFG elaborating on “somenode”
- Some consistency properties
  - If \( \{(m, \text{somenode}) \in \emptyset \), \( m, \text{somenode} \in N \), then \( \text{Cardinality} (\text{INPUT}(\text{somenode})) = 0 \)
  - If \( \{|(\text{somenode}, m)| \in \emptyset \), \( m, \text{somenode} \in N \), then \( \text{Cardinality} (\text{OUTPUT}(\text{somenode})) = 0 \)
- Maybe some others(?)
  - If \( \text{Cardinality} (\{m, \text{somenode}\}) = k \), \( m, \text{somenode} \in N \), then \( \text{Cardinality} (\text{INPUT}(\text{somenode})) = k \)
  - If \( \text{Cardinality} (\{|(\text{somenode}, m)|\} = k \), \( m, \text{somenode} \in N \), then \( \text{Cardinality} (\text{OUTPUT}(\text{somenode})) = k \)
Elaboration of “Select New Floor” Suggests Artifact Issues

IDEF0

• Commercial DFG formalism
• Some formality and rigor behind it
• Primarily pictorial
• In wide use
• Additional semantics in IDEF1, IDEF2, etc.

IDEF0

• Commercial DFG formalism
• Some formality and rigor behind it
• Primarily pictorial
• In wide use
• Additional semantics in IDEF1, IDEF2, etc.

Broadening DFG Semantics

• Node cannot begin until data arrives along all in-edges
  – Let DFG = (N, E), where N is set of nodes, (n_i), If n_i is a function, then it must be defined on the set of all artifacts IA = (ia_{ij}), such
    ( (n_j, n_i), ia_{ij} ) \in EdgeAnnotation_{DFG}
  – How to adapt this for
    – any semantics?
    – for output semantics?
    – exactly one Output node
    – Etc.

These sorts of constraints can support additional types of reasoning:
  Eg. about parallelism
Still More Broadening

- Different shapes of boxes
- Different pictures instead of boxes
- Different colors
- Different lines
- ......

Central questions: What are the semantics?
Does this really help? Or confuse?

Scientific Workflow Graph

 Attempt to make this more precise

But iteration complicates matters

Kepler--Another DFG Technology

- Data Flow graph notation
- And elaborate diagram iconography
- Has hierarchical decomposition
- Capability for specifying DFG semantics
- For each diagram
- Can be different at each level of hierarchy (!)
- Based on Ptolemy II system
    "Modeling of Sensor Nets in Ptolemy II", in Proc. of
    Information Processing in Sensor Networks”, (IPSN),
Kepler Elaboration: Focus of difficulty with iteration

What kinds of questions are well addressed by DFGs?

- Overall structure of functional capabilities
  - What does this piece do?
- System outputs and inputs
- How might changes be made?
- What functions create what data entities
- Problems with iteration, though

Given that Precision is essential

- What about the other three dimensions?
- Detail
  - Gained from hierarchical elaboration
- Breadth
  - Difficult to specify many kinds of iteration
  - Difficult to be clear about concurrency
- Clarity
  - Seems to be reduced by increased Detail and attempts at Breadth
  - With the need for Precision

Considerable Appeal, but Limited Value, to most stakeholders

- Users think they have sufficient understanding
  - But have trouble being able to see easy things (iteration)
- Developers have same problem
- Managers may only care to see easy things (!)
  - Although they should be interested in more
- Bystanders may be shown only easy things
  - Which could be a real problem

Final observations

- Very primitive representation
  - Very limited semantics
- But actually more a family of model types
  - Different sets of semantics
- The actual relation(s) are rarely made clear and precise
- Powerful aid to intuition and efficiency of communication
  - Clear advantages over natural language
- But is intuition misled by ambiguity, misinterpretation?
- Does not help explain HOW things get done

Control Flow Diagrams

- Semantics are different from DFD’s semantics
  - Arrows represent flow of control, rather than flow of data
  - Different shapes/types of boxes with different semantics
- Basic control flow graph:
  - Boxes represent functions
  - Some other shape represents control flow alternation
  - Arrows represent control flow:
    - If there is an arrow from circle A to circle B, it means ImmFol: “the execution of B can immediately follow the execution of A for some execution”
  - Different semantics for arrows between different shapes
  - Still other shapes represent Start and Stop
  - Use of hierarchy for elaborating boxes
- Usual enhancements:
  - Annotate edges with predicates
  - Special symbols for branching, concurrency control....
Control Flow Diagrams

- Semantics are different from DFD's semantics
  - Arrows represent flow of control, rather than flow of data
  - Different shapes/types of boxes with different semantics

- Basic control flow graph:
  - Boxes represent functions
  - Some other shape represents control flow alternation
  - Arrows represent control flow:
    - If there is an arrow from circle A to circle B, it means
      ImmFol: "the execution of B can immediately follow
      the execution of A for some execution"
  - Different semantics for arrows between different shapes
  - Still other shapes represent Start and Stop
  - Use of hierarchy for elaborating boxes

- Usual enhancements:
  - Annotate edges with predicates
  - Special symbols for branching, concurrency control...

Example Control Flow Graph Semantics

- Square boxes: Functions
  - Any number of inedges
  - One outedge
- Edges: ImmFol relation
  - Function at head executes immediately after
    function at tail concludes
  - Round ovals: Branches/decisions
    - Each oval represents a Boolean function
    - For each oval there are exactly two outedges, labeled True and False

Example Control Flow Graph

More Detail through Hierarchical Elaboration

- The challenges and solutions here are similar to those for DFGs
- Need for consistency between levels
- Semantics can be hard to define, stick to
More Breadth from more (sub)types of CFG

- Concurrency
- Timing
- Data flow augmentations
- Etc.

Concurrency graph with “fork and join”

Rendezvous Graph

- Rendezvous graph: $RG = (N, E)$, where $N = \text{COMP} \cup \text{SYNC}$
- $S \in \text{SYNC} \Rightarrow S$ executes only after the execution of all $C \in \text{COMP}$ such that $(C, S) \in E$
- If $(S, C) \in E$, where $S \in \text{SYNC}$, then $C$ cannot execute until $S$ has executed

Trace Flow Graph

- Explicitly represents interleaved execution
- The green edges are the ImmFol relation
- Double-headed arrows represent one arrow in each direction
- Sync nodes are filled in black
- Synch edges are blue

Different variations to address different stakeholders and their needs

- More semantic issues
  - Different graph types for different issues
- More detail
  - Hierarchy helps here
- More clarity
  - Does more elaboration and more graph types help clarity or impede it?

Focus on Clarity

- CFG has more procedural detail
- Does this help?
- What about combining DFG and CFG?
  - Iconography must be clear (?)
What is Control Flow good/not good for?

- Sense of what algorithms to use
- Constraints on data appearing
  - eg. Assuming that the Request List is maintained in sorted order
- Can estimate running speed
- Can reason about functionality
  - Possible to strand requesters
  - Possible to take riders in the wrong direction
- Drawbacks:
  - What about safety?
  - What about data?

What Stakeholders does this address?

- Developers
- ???
Control Flow vs. Data Flow Graphs

- Both shed light on similar questions
- One focuses on data evolution, the other on functional development
- Both are useful, neither removes the need for the other
- Control flow graphs map closely to implementation code written in procedural languages.
  - Good basis for determining consistency of code with ideas expressed as data flow
- Data flow graphs focus more on the product itself, seem better at helping understand if and how it gets evolved
  - Seem better adapted to studying earlier formulations of the problem to be solved, and ways of solving it

Finite State Machines (FSM's)

- FSM's describe behavior of a system:
  - The sequence of stages/steps/conditions that the system goes through
  - FSM shows how a system acts/reacts to inputs
  - Does this by showing progress through different states

- Hypothesis:
  - The universe in which the system being described must operate can be accurately modeled as always being in exactly one of a finite number of states (situations)
  - There are only a finite number of possible system inputs

Finite State Machines (FSM's)

- FSM = (Q, I, \( \delta \)), where
  - \( Q = \{ q_i \} \), the set of all possible system states
  - \( I = \{ i_j \} \), the set of all events that can affect the state of the system
  - \( \delta \) is a function, \( \delta : Q \times I \rightarrow Q \) such that if the system is in state \( q \) and event \( i \) occurs, then the system transitions to state \( \delta(q, i) \)

- Use of hierarchy can help add detail
  - But presents familiar consistency problems

- Other definitions:
  - Start state
  - Accepting state
  - Trap state

Why FSM's?

- Primary appeal is visualizability–clarity
  - Circles represent states
  - (Curved) arrows represent transitions
  - Arrows are annotated with inputs

- Intuitively: Can "watch" a stream of inputs "drive" the behavior of the system as a sequence of movements from state to state

Kinds of Questions FSMs seem adept at helping answer:
"What is a good way to think about the problem to be solved?"
"What is the solution approach?"
"How does this program work?"

Finite State Machine for Digital Watch

- Time display mode
- Datebook mode
- Phone book mode
- Alarm display mode
- Press button A

Finite State Machine
More Finite State Machine Details

- Time display mode
- Datebook mode
- Phone book mode
- Alarm display mode
- Press button A
- Press button B
- Press button C

State also specifies activities
- Leans towards a CFG
- Transition may involve computation
  - Considered to be "instantaneous"
- Transition may be conditional
  - Event as well as a condition
- Transition may emit events
  - To drive other FSMs
- FSMs may be hierarchical
  - What exactly are semantics?

More Complex FSMs

Statecharts: More Complex FSMs

- Initialize: Initialize course
- Open: Register a student
- Closed: Report course is full
- Canceled: Send cancellation notices
- RegistrationComplete: Generate class roster
- CancelCourse: ...
Statecharts: More Complex FSMs

- Initialize
- Initialize course object
- Assign professor to course
- Open entry: Register a student
- Closed do: Report course is full
- Canceled do: Send cancellation notices
- addStudent/numStudents = 0
- cancelCourse
- RegistrationComplete do: Generate class roster
- [numStudents = 10]
- cancelCourse
- registration closed [numStudents >= 3]
- registration closed [numStudents < 3]
- Unassigned
- activity on this transition
- transition
- next state
- exit state

Statechart with Nested States

- superstate
- substate
- Initialize
- Register
- Open entry: Register a student
- Unassigned
- do: Assign professor to course
- Open
- Closed do: Report course is full
- Canceled do: Send cancellation notices
- addStudent/numStudents = 0
- cancelCourse
- RegistrationComplete do: Generate class roster
- [numStudents = 10]
- cancelCourse
- registration closed [numStudents >= 3]
- registration closed [numStudents < 3]
- Unassigned
- activity on this transition
- transition
- next state
- exit state

What is FSM good/not good for?

- Focus on specific issue: safety concern
  - Model unsafe state
  - Model state transitions
  - Can unsafe state be reached?
- Drawbacks
  - No sense of functionality
    - Unless additional semantics
  - No sense of how functionality achieved
    - Except perhaps hierarchy
  - Hard to deal with concurrency
    - Without additional semantics
    - Impossible to reason about timing
      - Unless additional semantics

Petri Nets

- More powerful and intuitive depiction of control flow strong on depiction of parallelism and concurrency
- A Petri Net structurally consists of
  - A finite number of places
  - A finite number of transitions
  - A finite set of arrows that connect places to transitions (or vice versa)
- If an arrow goes from a place to a transition, then place is said to be an input place of the transition.
- If an arrow goes from a transition to a place, then place is said to be an output place of the transition.

Marking and Firing Petri Nets

- A Petri Net place can be marked by the presence of a token
  - Any collection of places can be marked.
  - Any such marking is said to define a state of the Petri Net
- Petri Nets proceed from one state to another by means of a firing
  - Occurs only when every input place of a transition is marked with a token.
- The effect of the firing of a transition is to
  - Remove all of the tokens from the transition's input places
  - Put tokens in all of the transition's output places

Scenarios Definition with Petri Nets

- A marking represents a scenario
- Applying firing rules creates a simulation
- Different markings support exploring dynamic behavior of a system
More Details in
Ghezzi, et al., Chapter 5.5.3

Unmarked Petri Net

A Scenario: Car going up
with UP button pressed

A Scenario: Car going up
with UP button pressed

A Scenario: Car going up
with UP button pressed

Unmarked Petri Net—Two Elevator Cars
Petri Net--Two Elevator Cars
Car 2 Wins Race

Car 1 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 2 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 2 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 1 approaching Floor n, going up
Up Button
Press at Floor n
No Press

Stop at Floor
Go to Next Floor
Up

Up Button
Pressed at Floor n
Car 2 approaching Floor n, going up
No Press
Stop at Floor
Go to Next Floor
Up

Car 1 approaching Floor n, going up
No Press
Stop at Floor
Go to Next Floor
Up

Car 2 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 1 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 2 approaching Floor n, going up
Up Button
Press at Floor n
No Press
Car 1 approaching Floor n, going up
Up Button
Press at Floor n
No Press

Car 2 Wins Race

Petri Net for a Different Elevator

Car going up
Up button
higher
No Up Button Higher
Move up to Nearest Up Button
Car going down
Up button
lower
No Down Button Lower
Move to nearest Down Button
Visit floor, Turn off button
Visit floor, Turn off button

Marking for moving up to pick up a passenger

Car going up
Up button
higher
No Up Button Higher
Move up to Nearest Up Button
Car going down
Up button
lower
No Down Button Lower
Move to nearest Down Button
Visit floor, Turn off button
Visit floor, Turn off button

Visit floor, Turn off button
Visit floor, Turn off button

Marking for moving up to pick up a passenger

Car going up
Up button higher
No Up Button Higher
Move up to Nearest Up Button

Car going down
Down button lower
No Down Button Lower
Visit floor, Turn off button
Move to nearest Down Button
Visit floor, Turn off button

Marking when no passengers higher, but passenger lower

Car going up
Up button higher
No Up Button Higher
Move up to Nearest Up Button

Car going down
Down button lower
No Down Button Lower
Visit floor, Turn off button
Move to nearest Down Button
Visit floor, Turn off button

Queues at places?

Car going up
Up button higher
No Up Button Higher
Move up to Nearest Up Button

Car going down
Down button lower
No Down Button Lower
Visit floor, Turn off button
Move to nearest Down Button
Visit floor, Turn off button

Marking when no passengers higher, but passenger lower

Car going up
Up button higher
No Up Button Higher
Move up to Nearest Up Button

Car going down
Down button lower
No Down Button Lower
Visit floor, Turn off button
Move to nearest Down Button
Visit floor, Turn off button
Marking when no passengers higher, but passenger lower

- Car going up
- Up button higher
- No Up Button Higher
- Move up to Nearest Up Button
- Visit floor, Turn off button
- Move to nearest Down Button
- Visit floor, Turn on button

Marking for both up and down buttons

- Car going up
- Up button higher
- No Up Button Higher
- Move up to Nearest Up Button
- Visit floor, Turn off button
- Move to nearest Down Button
- Visit floor

Marking for neither up nor down buttons

- Car going up
- Up button higher
- No Up Button Higher
- Move up to Nearest Up Button
- Visit floor, Turn off button
- Move to nearest Down Button
- Visit floor

Petri Nets

- More powerful and intuitive depiction of control flow strong on depiction of parallelism and concurrency
- A Petri Net structurally consists of:
  - A finite number of places
  - A finite number of transitions
  - A finite set of arrows that connect places to transitions (or vice versa)
- If an arrow goes from a place to a transition, then place is said to be an input place of the transition.
- If an arrow goes from a transition to a place, then place is said to be an output place of the transition.

Some Semantics

- PN = (Places, Transitions, Edges)
  - Places (PN) = \{place\}
  - Transitions (PN) = \{transition\}
  - Edges (PN) = (Inedges U Outedges)
    - Inedges (PN) = ((place, transition))\ x\ Outedges (PN) = \{(transition, place)\}
    - where place, \epsilon\ Places (PN), transition, \epsilon\ Transitions (PN)
- Marked: PN x Places \rightarrow \{True, False\}
  - If Marked (PN, place) = True we say that place is marked
- A transition, \epsilon\ Transitions (PN) can fire if for all of its inedges, (place, \epsilon\) \ Marked (place) = True
  - After a transition \epsilon\ Transitions (PN) fires
    - Marked (p) = False for all places, p, such that (p, \epsilon\) \notin\ Edges (PN)
    - Marked (place,\epsilon) = True, for all places, such that \epsilon\ \notin\ Edges (PN)
Many Extensions

- Bi-Logic Nets
  - Allows Oring inputs and outputs
- Colored Petri Nets
  - Tokens can now be "typed"
- Hierarchical Petri Nets
  - Usual advantages of hierarchical decomposition
- Queues at places
- Timed Petri Nets
  - For estimating performance
- And various combinations of these

Evaluation of Petri Nets

What are Petri Net representations good for?:

- How do things get done?
- Especially parallelism and nondeterminism
- Helps spot races and deadlocks

What kinds of things are Petri Nets not good for?

- Too little focus on product (tokens represent it)
- Don't scale very well
- Petri Nets get large and complicated fast
- Many extensions address shortcomings
- Too many extensions confuse the picture?