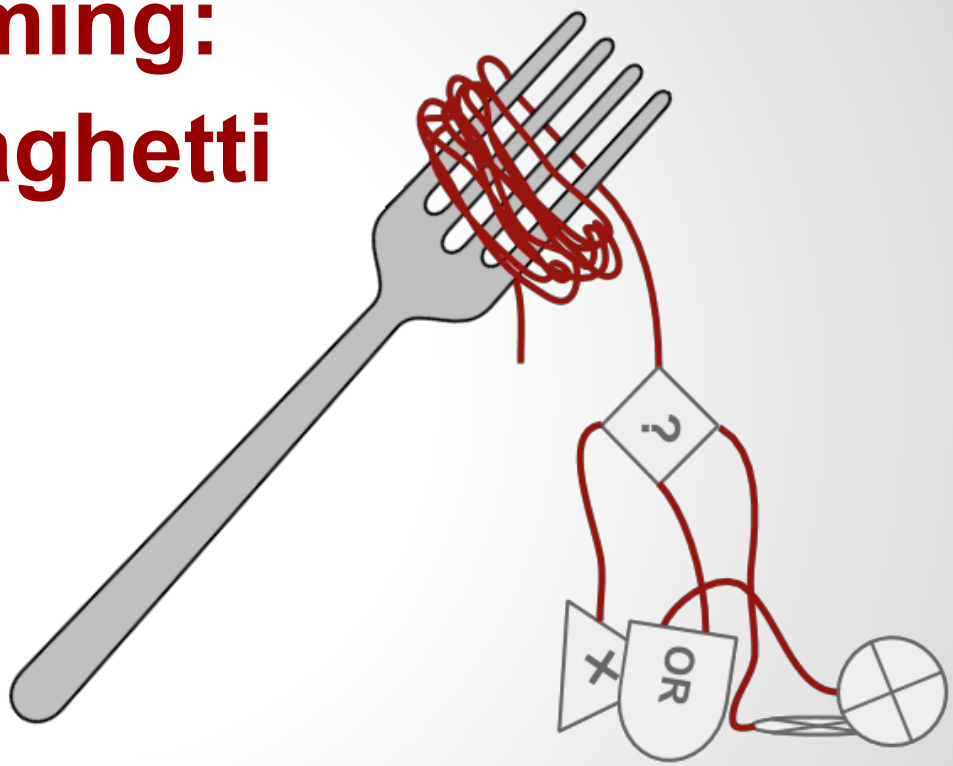
Visual Programming: Cooking the Spaghetti

GOTO Chicago '13



Rick Lane

rick.lane@tradingtechnologies.com

@R1ck_L4n3 @trading_tech @GOTOChgo

goto; conference

First things first

- Visual Programming < Regular Programming (usually)
- But, there's a time and a place
- How to make it work, when it makes sense to work

Who I am

- Chief technology officer at Trading Technologies
  - TT makes software for professional traders around the world
- Background in wargaming: terrorist network modeling and analysis, agent-based modeling, etc.
- Many similarities between these industries: a knowledge gap between the idea generators and the technologists

Lakin, Fred, Visual Grammars for Visual Languages, Proceedings of the American Association for Artificial Intelligence, Seattle, Washington, July 1987.
Keywords and Phrases: visual grammar notation, declarative programming, visual rules, formal visual languages, informal conversational graphics, visual lisp, visual emacs, vmacs, VPL-IV.C: Parsing, VPL-VI.A: Formal definition of VPLs, VPL-II. A.7: Logic languages

Meyer, Bernd, Beyond Icons: Towards New Metaphors for Visual Query Languages for Spatial Information Systems, Proceedings of the International Workshop on Interfaces to Database Systems, Glasgow, 113-135, July 1992. (published by Springer, London, 1993).
Keywords and Phrases: VPL-II.A.7: Logic languages, VPL-II.B.1: Diagrammatic languages, VPL-V.B: Database languages

Schuerr, Andreas, Rapid Programming with Graph Rewrite Rules, USENIX Symposium Proc. on Very High Level Languages (VHLL), USENIX Association, 83-100, 1994.
Keywords and Phrases: VPL-II.A.6: Imperative languages, VPL-II.A.8: Multi-paradigm languages, VPL-II.A.11: Rule-based languages, VPL-II.B.1: Diagrammatic languages

Schuerr, Andreas, A. Zuendorf, Nondeterministic Control Structures for Graph Rewriting Systems, Proc. WG'91 Workshop in Graph-theoretic Concepts in Computer Science, LNCS 570, Springer Verlag, 48-62, 1991. also: Technical Report AIB 91-17, RWTH Aachen, Germany, 1991.
Keywords and Phrases: VPL-II.A.6: Imperative languages, VPL-II.A.8: Multi-paradigm languages, VPL-II.A.11: Rule-based languages, VPL-II.B.1: Diagrammatic languages

Ambler, Allen L., Margaret M. Burnett, Integrating Algorithm Animation into a Declarative Visual Programming Language Proceedings of the IEEE Symposium on Visual Languages, Darmstadt, Germany, September 1995.
Keywords and Phrases: Forms/3, VPL-II.A.4: Form-based and spreadsheet-based languages, VPL-III.A.1: Data

goto; conference

Low-latency trading & VPL

Communication Barriers

Safety

goto; conference

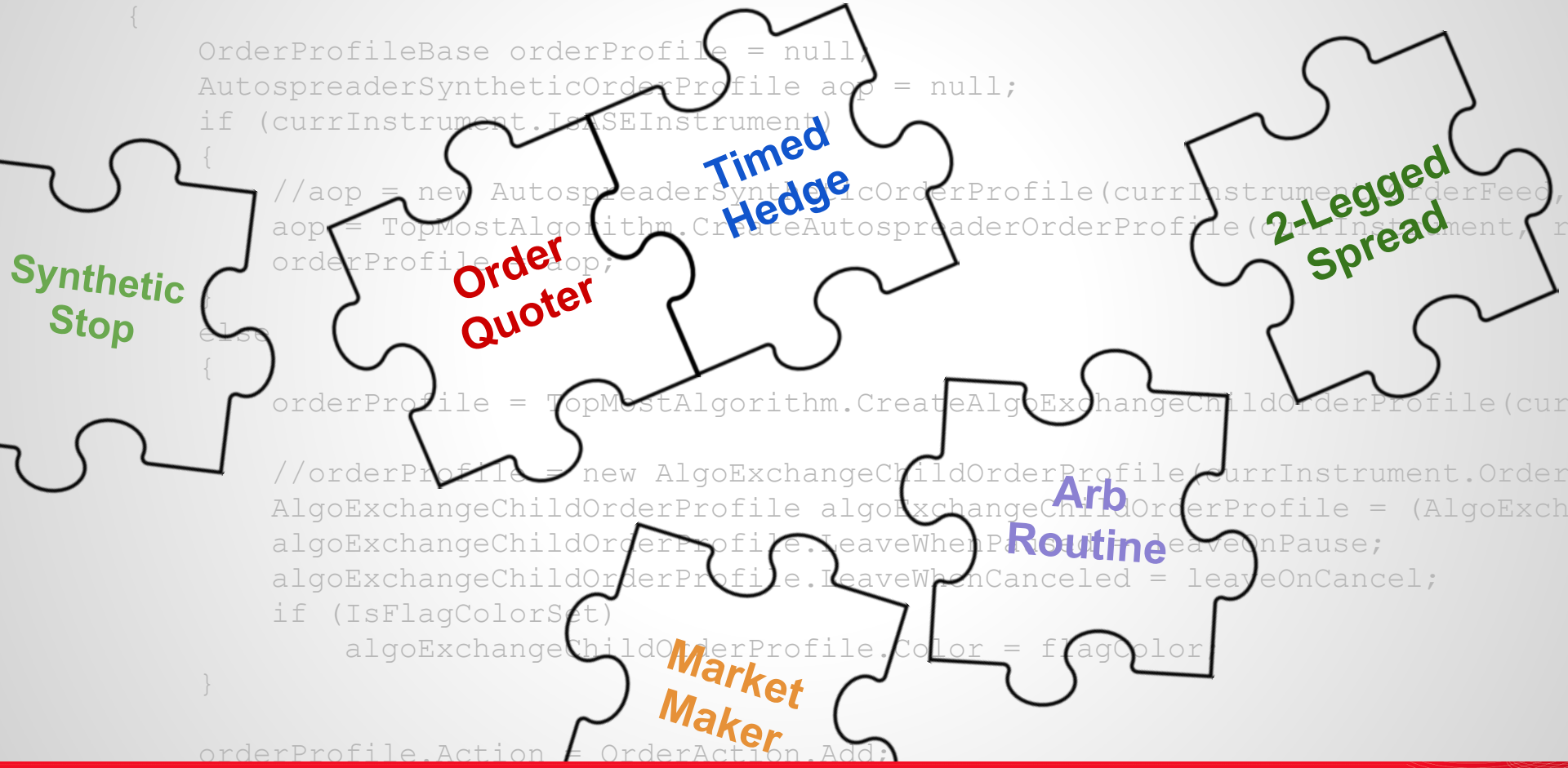
Why I decided to build a VPL

- **I didn't! -** I stumbled upon it
- I was still learning the business
- I had insanely aggressive deadlines (see: *minutes*) for writing and deploying production code

  It was built out of necessity

I got tired of these guys yelling at me

Modules made my life easier

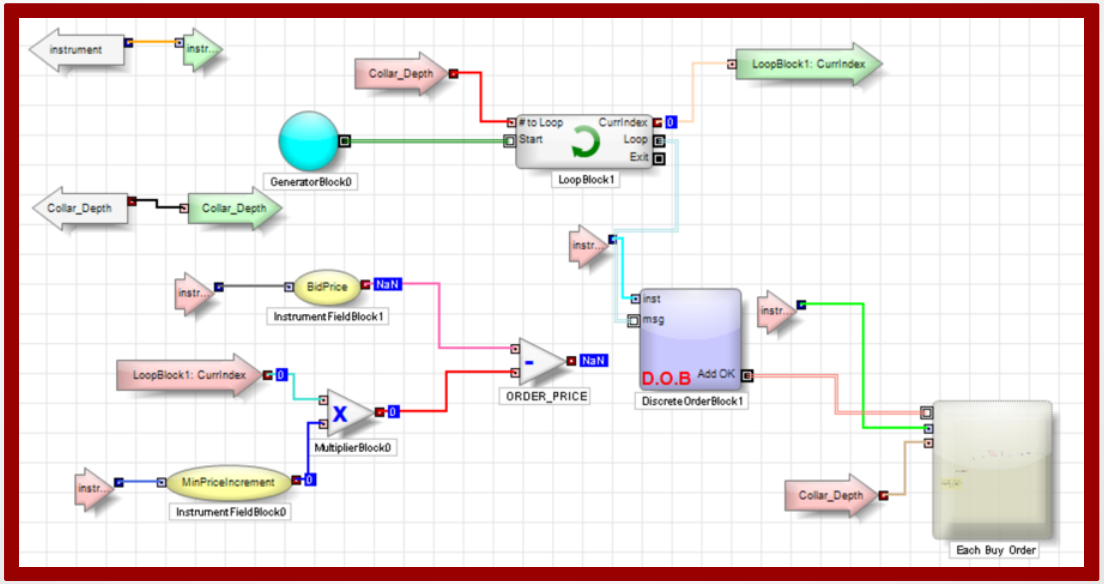


But modules weren't enough

- Bugs could still spell disaster
- I wanted *the traders* to be in control
  - Just piecing them together took work and programming know-how
  - This "work" wasn't going to add any business value

The VPL was born

- I built a GUI that let them visually link these modules together
- I didn't know I was building a VPL
- Algo Design Lab (ADL)



The "Deutsch Limit"

“The problem with visual programming languages is that you can't have more than 50 visual primitives on the screen at the same time.”

-Peter Deutsch

The "Deutsch Limit"

- The limit of 50 primitives is arbitrary
- We rely on our IDE to help us solve this problem
  - "Find References/Usages"
  - "Go to Definition/Declaration"
  - "Find in Files"
  - Refactoring
- Mental Map

The solution is threefold

1. Domain specificity
2. Hiding the "dirty stuff"
3. Mitigating the "Deutsch Limit"

1. Domain specificity

"The ability for the general public to understand a VPL is inversely proportional to the likelihood that it will actually do something useful."

-me

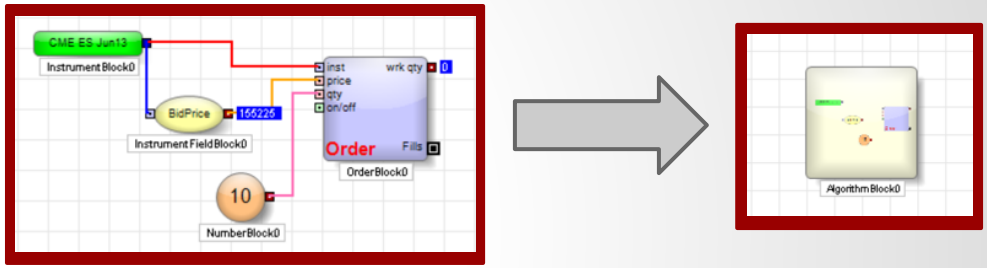
- This is really no different from a well-made API
  - Abstracts away unnecessary granularity
- Unspecific: Scratch, AppInventor, etc.
- Specific: Lego Mindstorms

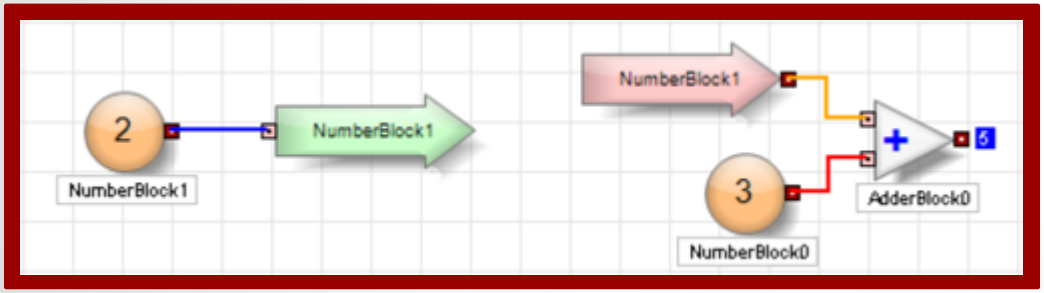
2. Hiding the "dirty stuff"

- All VPLs hide some of the generic dirty stuff
- Goes hand-in-hand with domain specificity
  - If not domain-specific, it's hard to identify the dirty stuff
- Examples of dirty stuff in automated trading
  - In-flight Orders
  - Fairness
  - "It just works"

3. Mitigating the "Deutsch Limit"

- Browser-based navigation
- Show Up/Downstream Blocks
- Block searching
- Componentization
- Breakpoints
- "Jump Blocks"





We want our users to eat fruit

Kathy Sierra - http://goo.gl/hKd95

- Cake vs. Fruit
- We want them to focus on idea generation, not implementation
- It's also about **safety**

Let's do an experiment

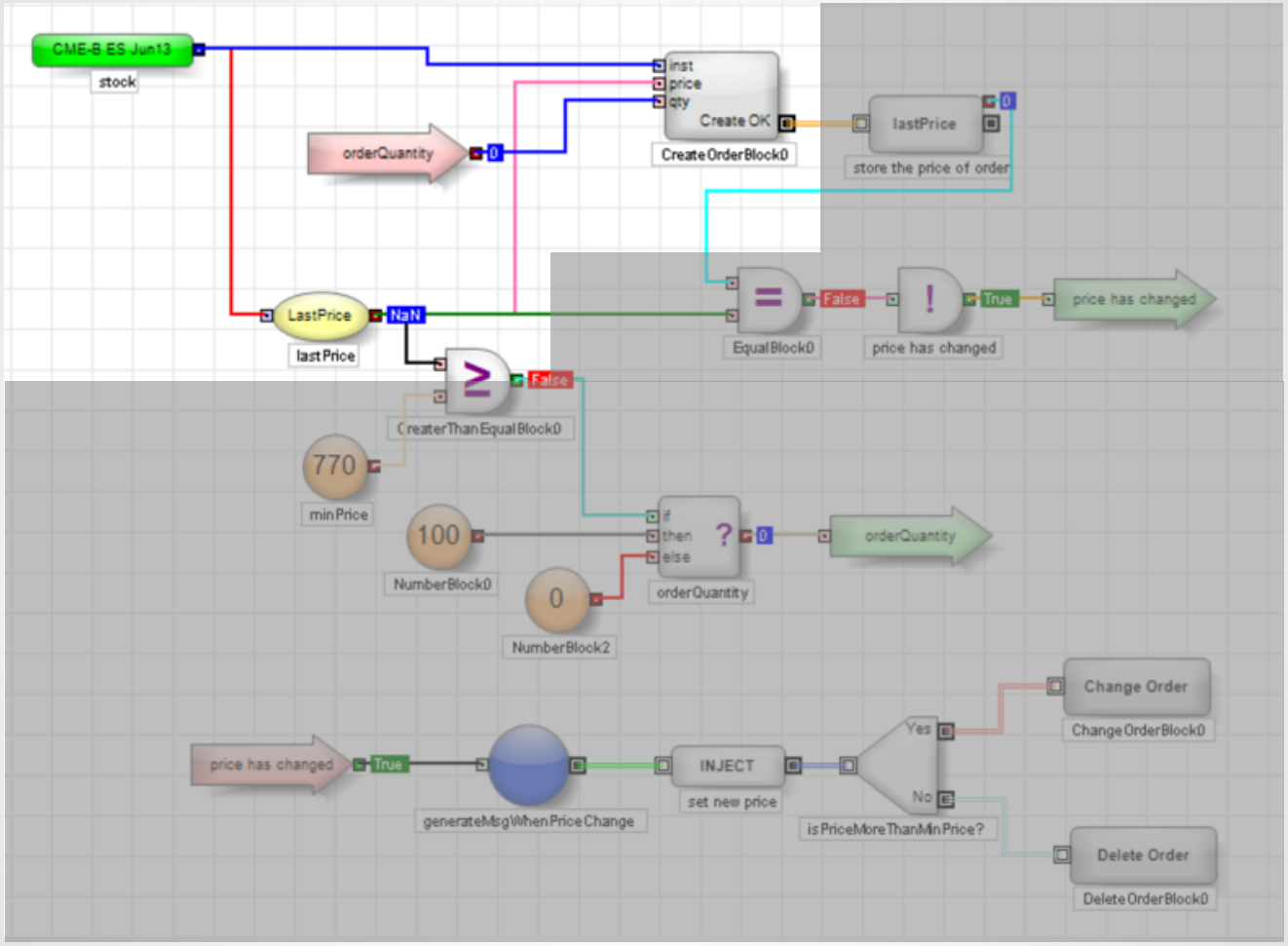
- Build a simple trading strategy:
  - Place an order to buy 100 shares of stock (e.g. GOOG) at the last-traded-price (LTP)
  - Whenever the LTP of GOOG changes, we should change our order's price to match
  - If the LTP of GOOG *ever* falls below $770, we should cancel our order

- Scratch-->Domain Specific Language-->ADL

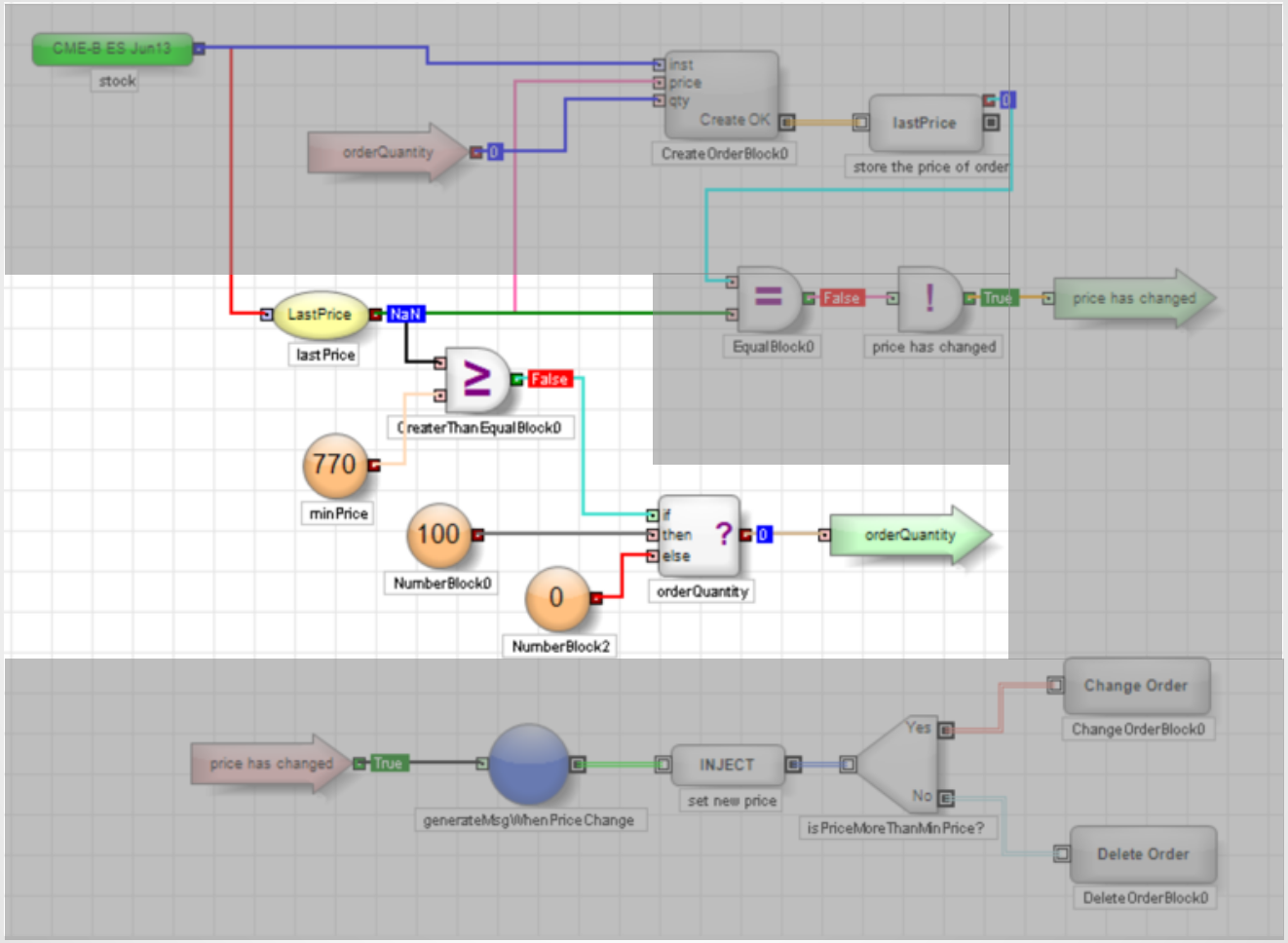
stock=GOOG;volume=5433728;LTP=782.03;

goto; conference

Domain Specific

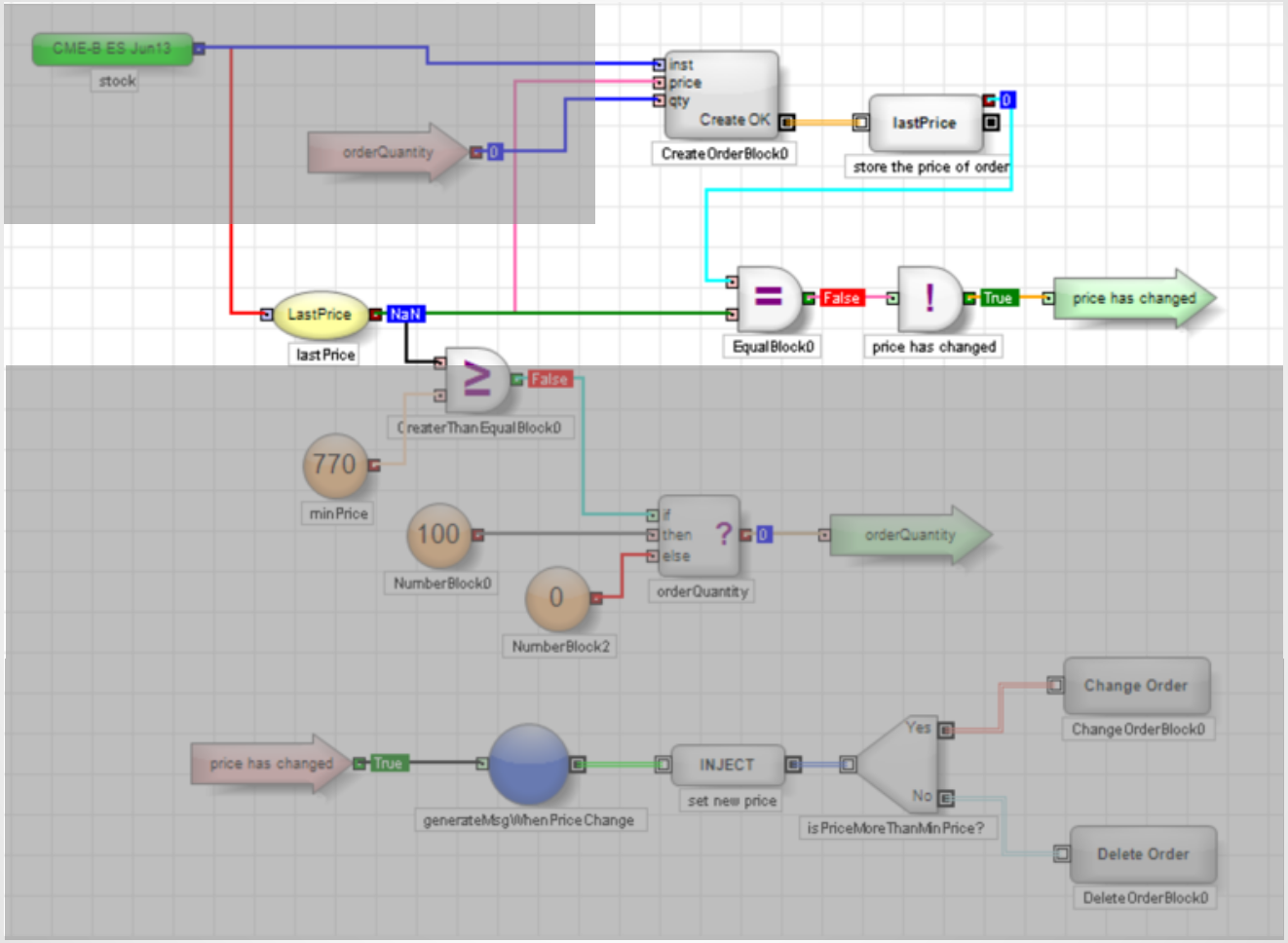


Domain Specific



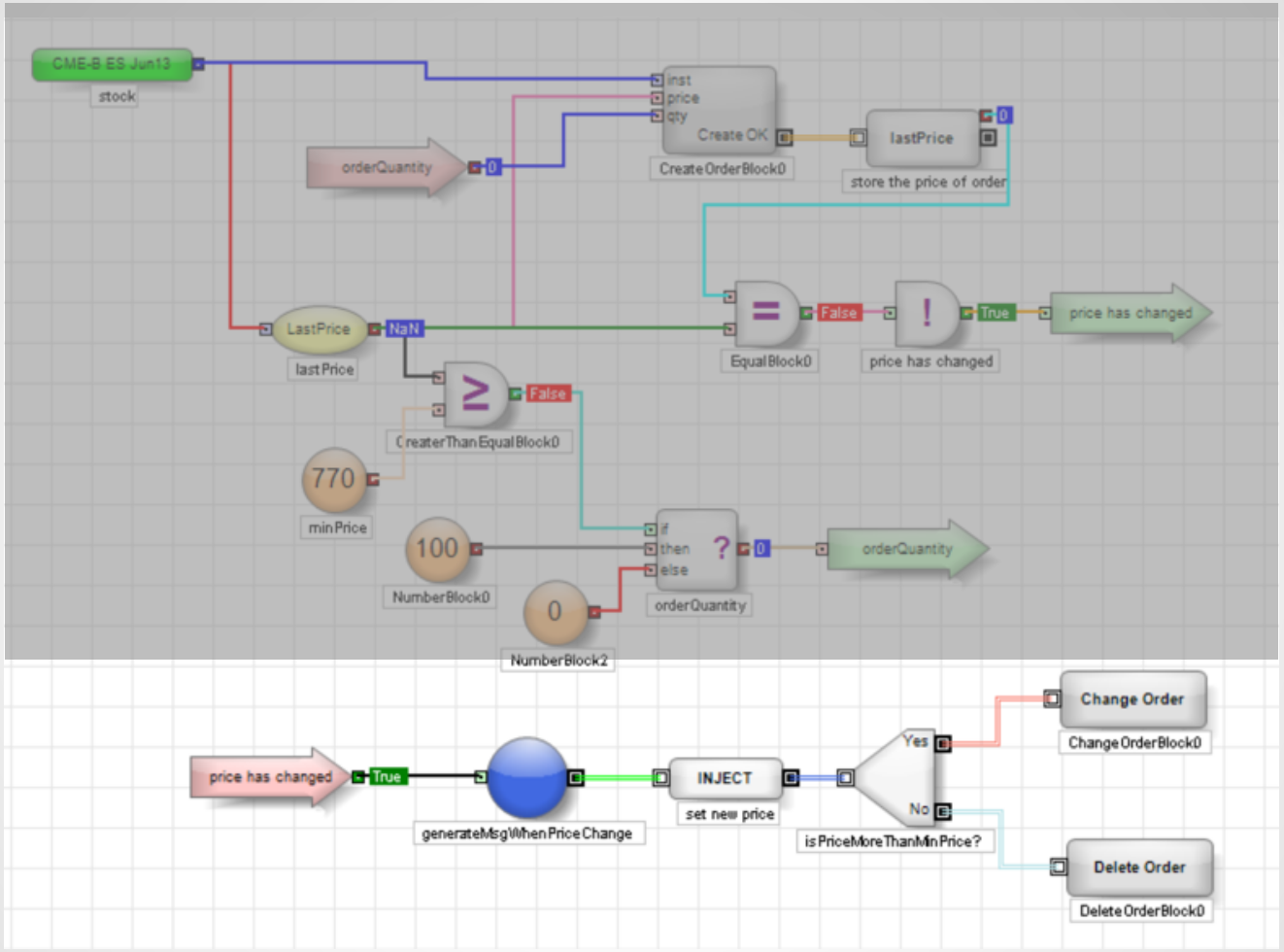
goto; conference

Domain Specific

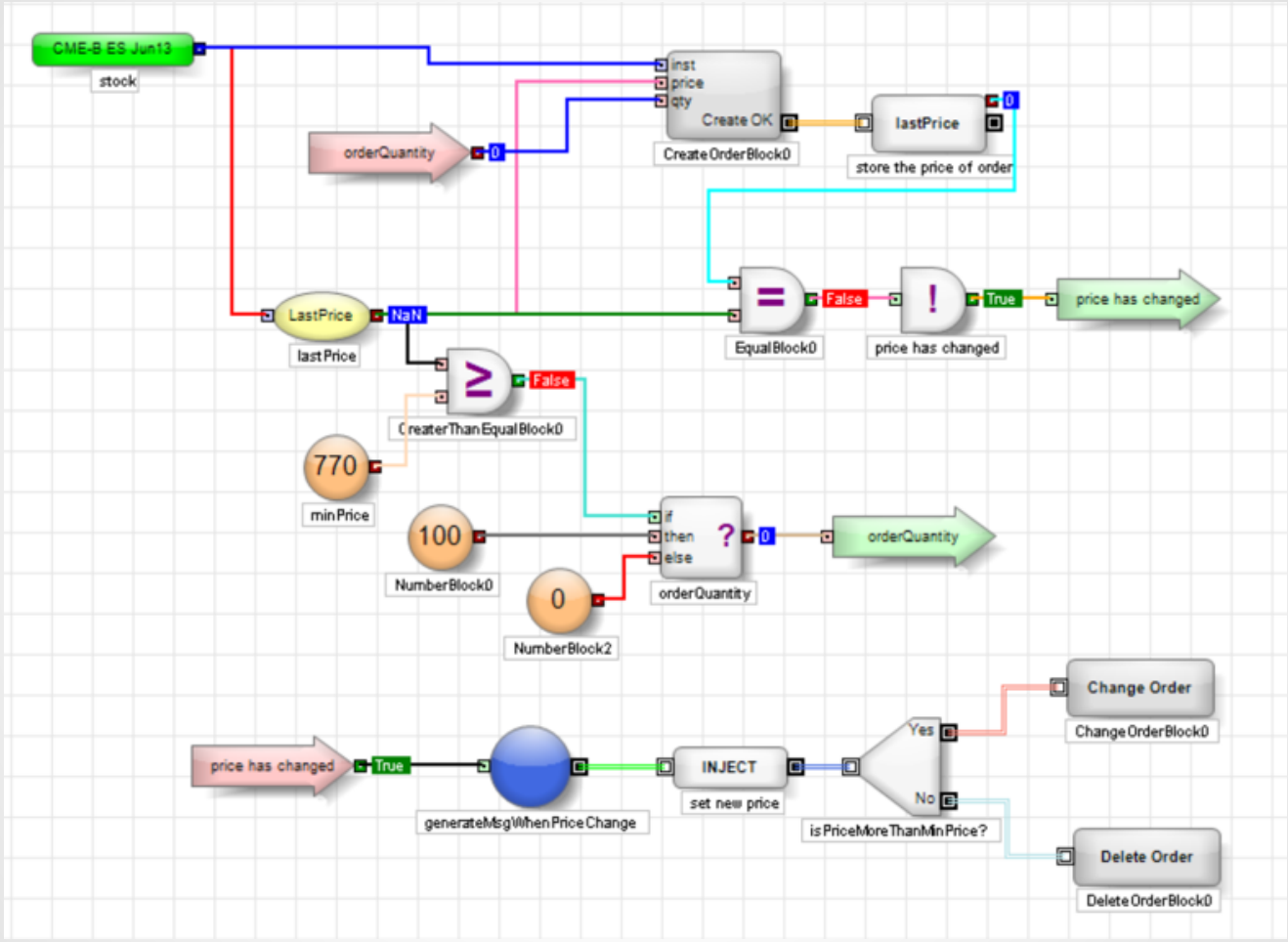


goto; conference

Domain Specific

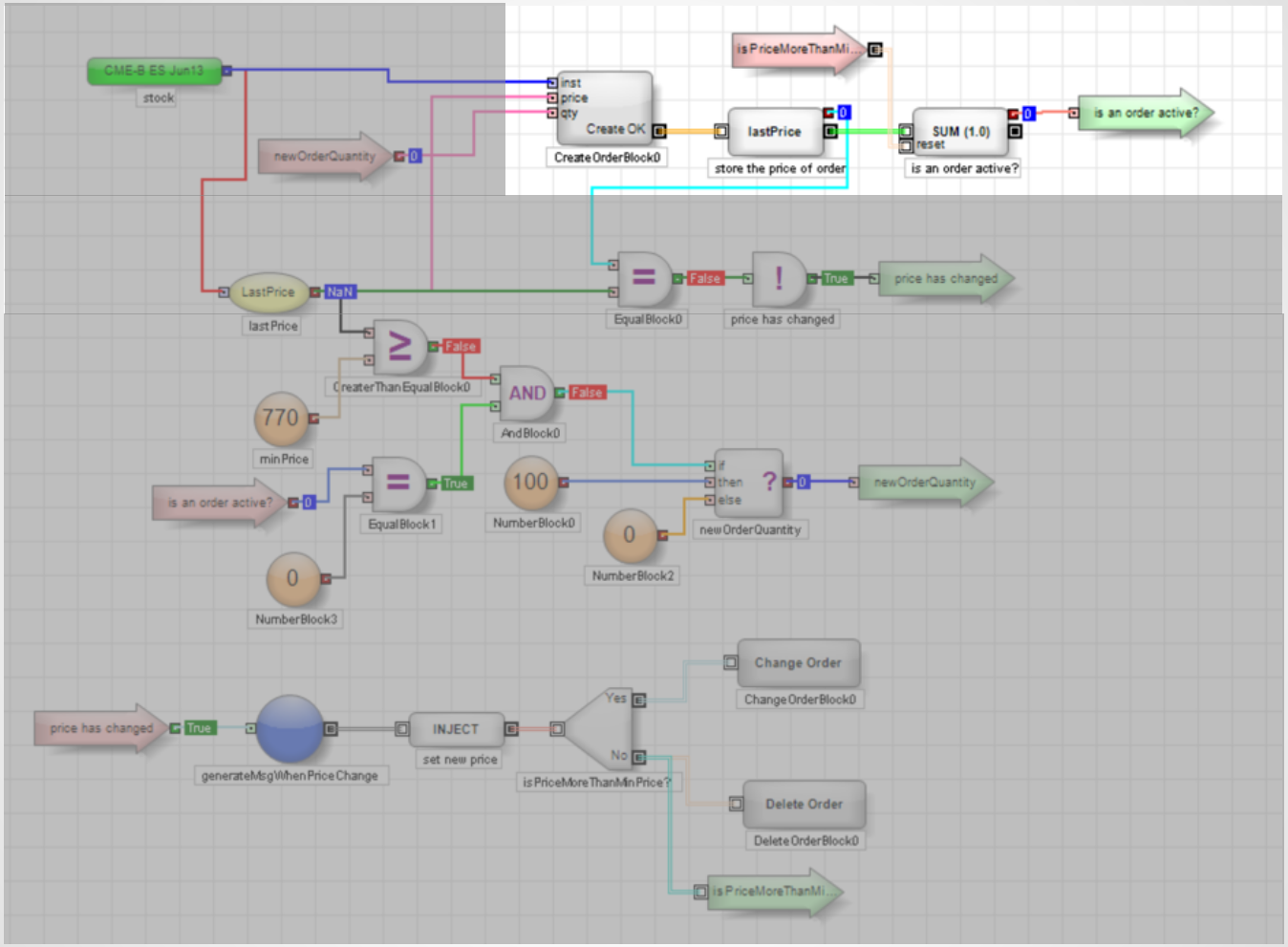


Domain Specific

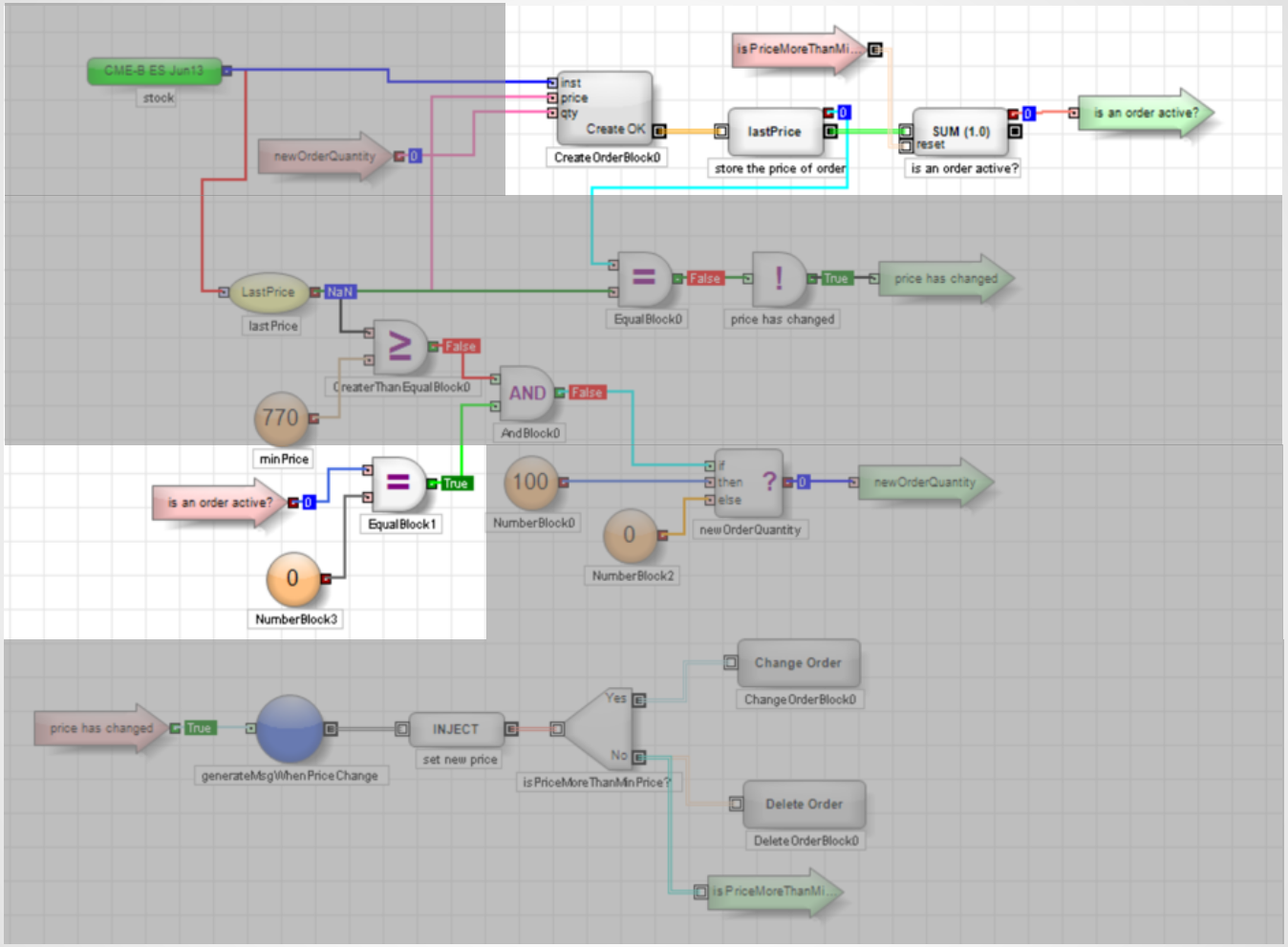


goto; conference

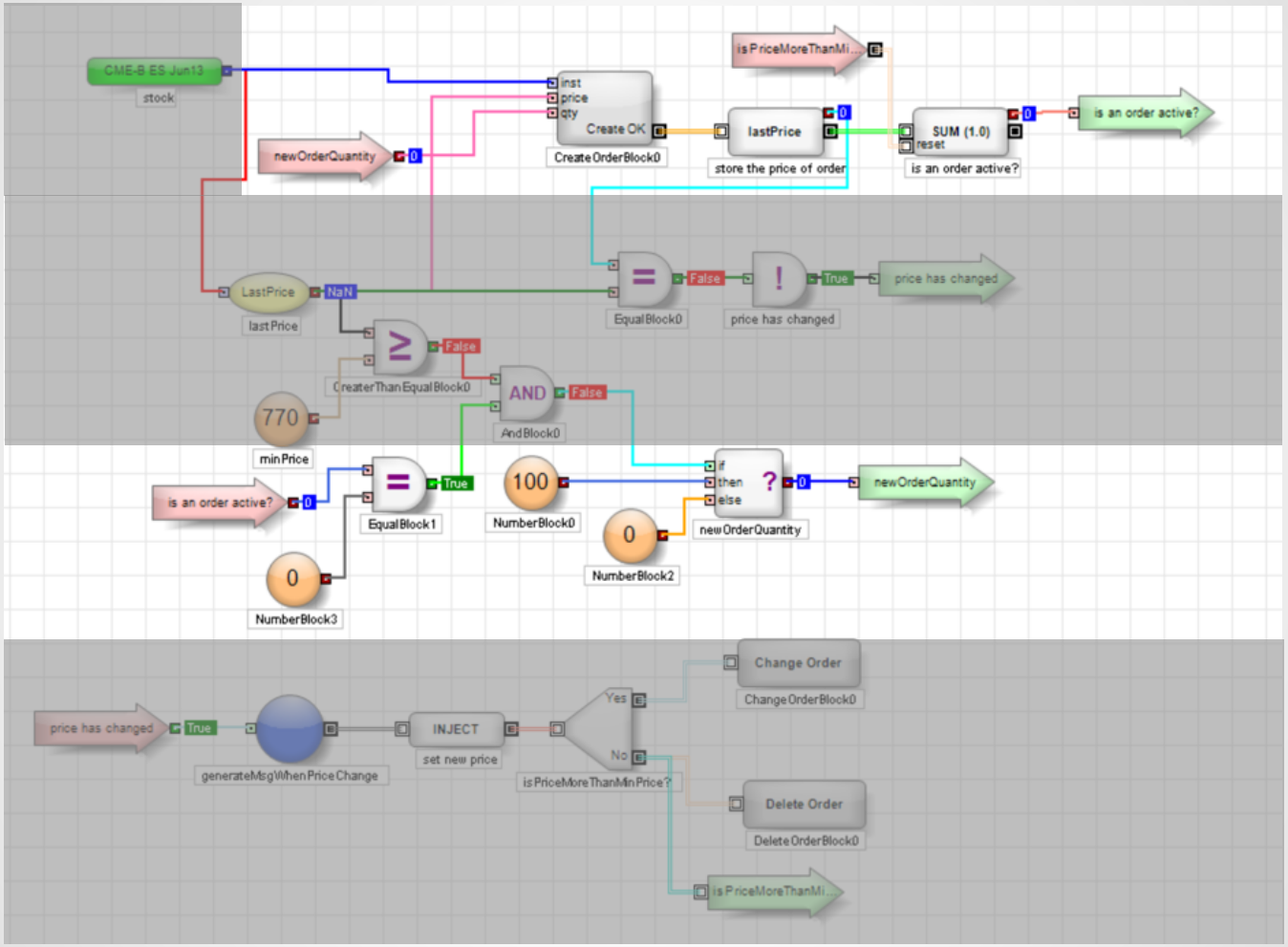
Domain Specific



Domain Specific

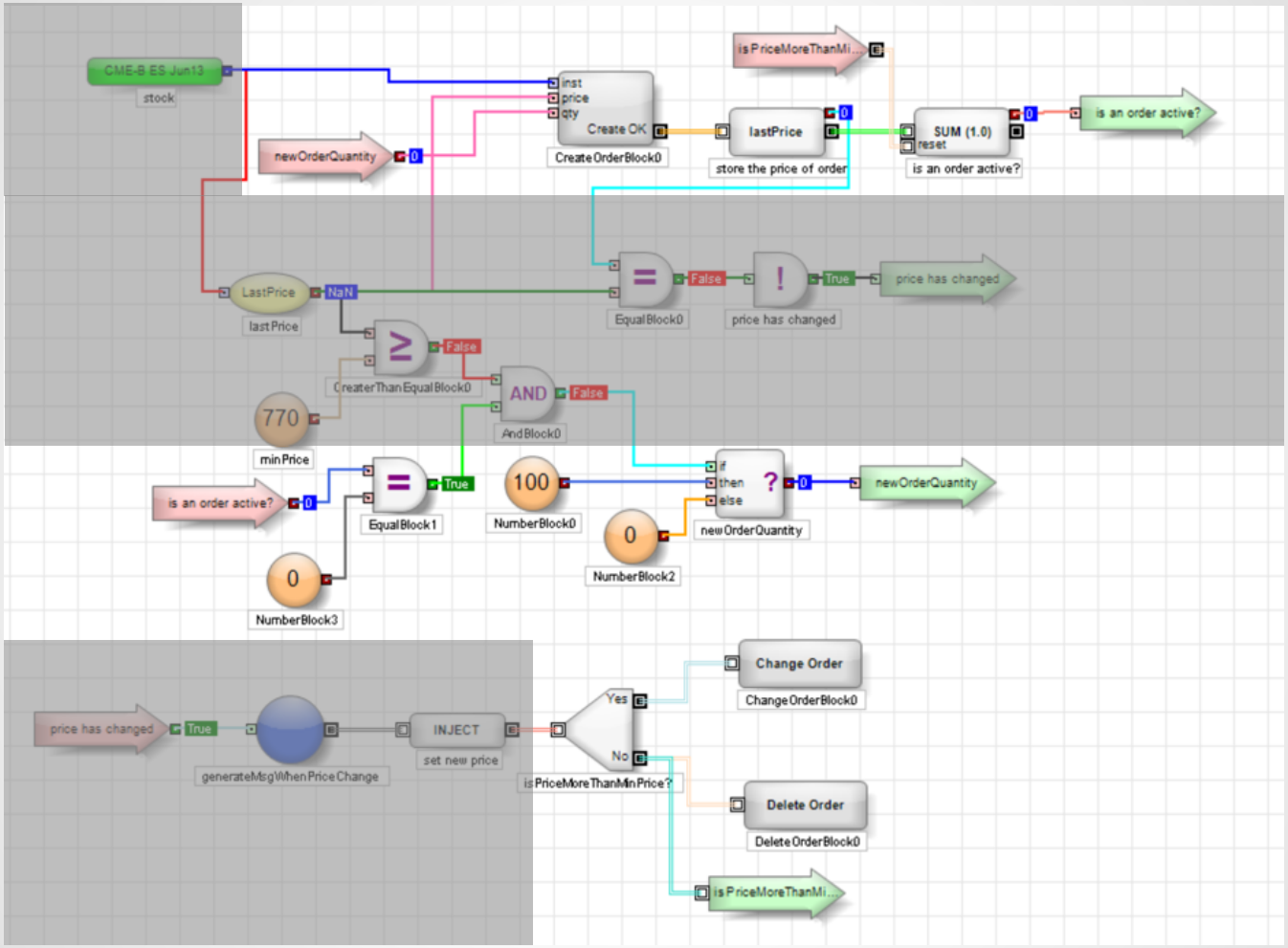


Domain Specific



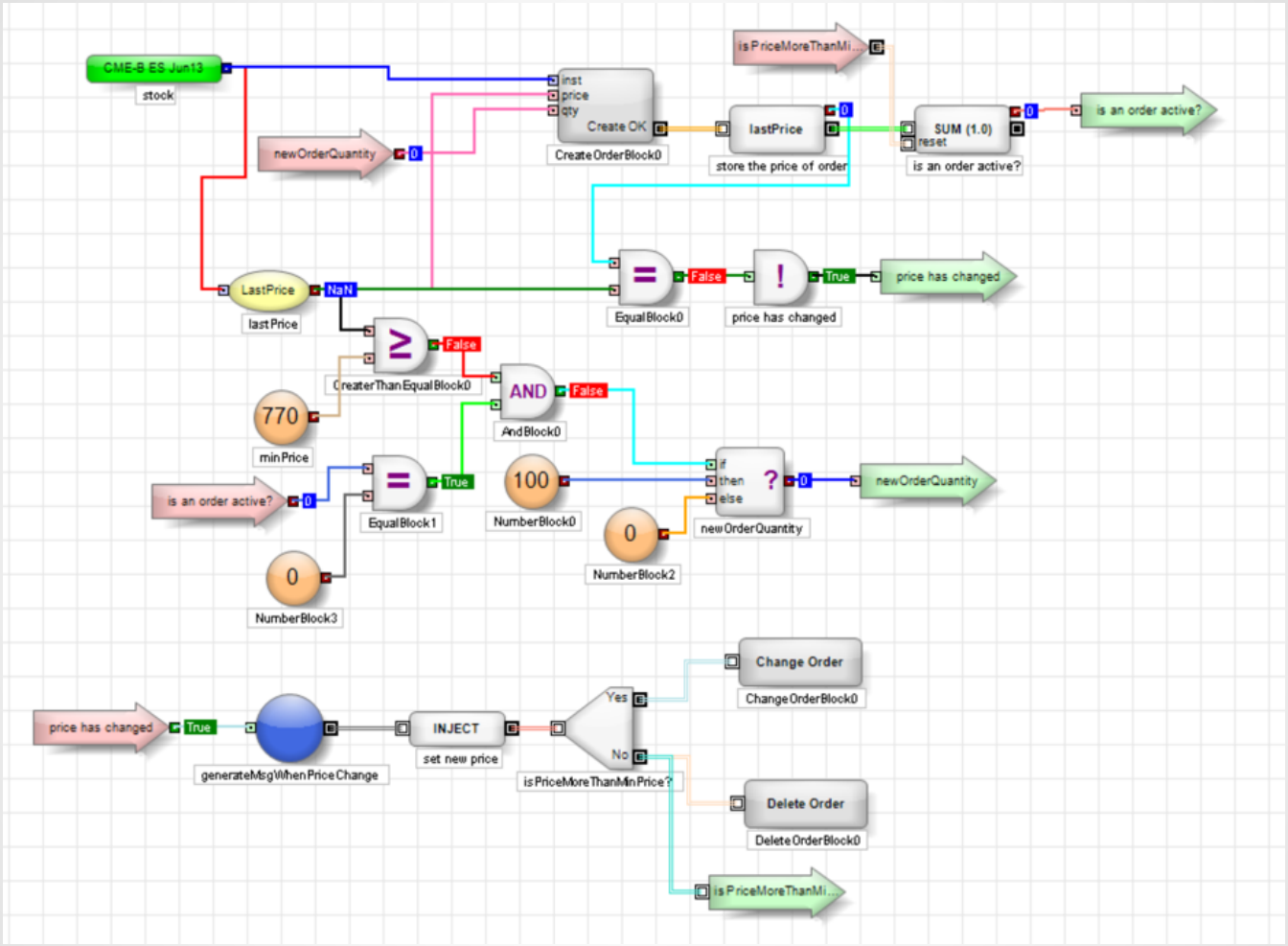
goto; conference

Domain Specific



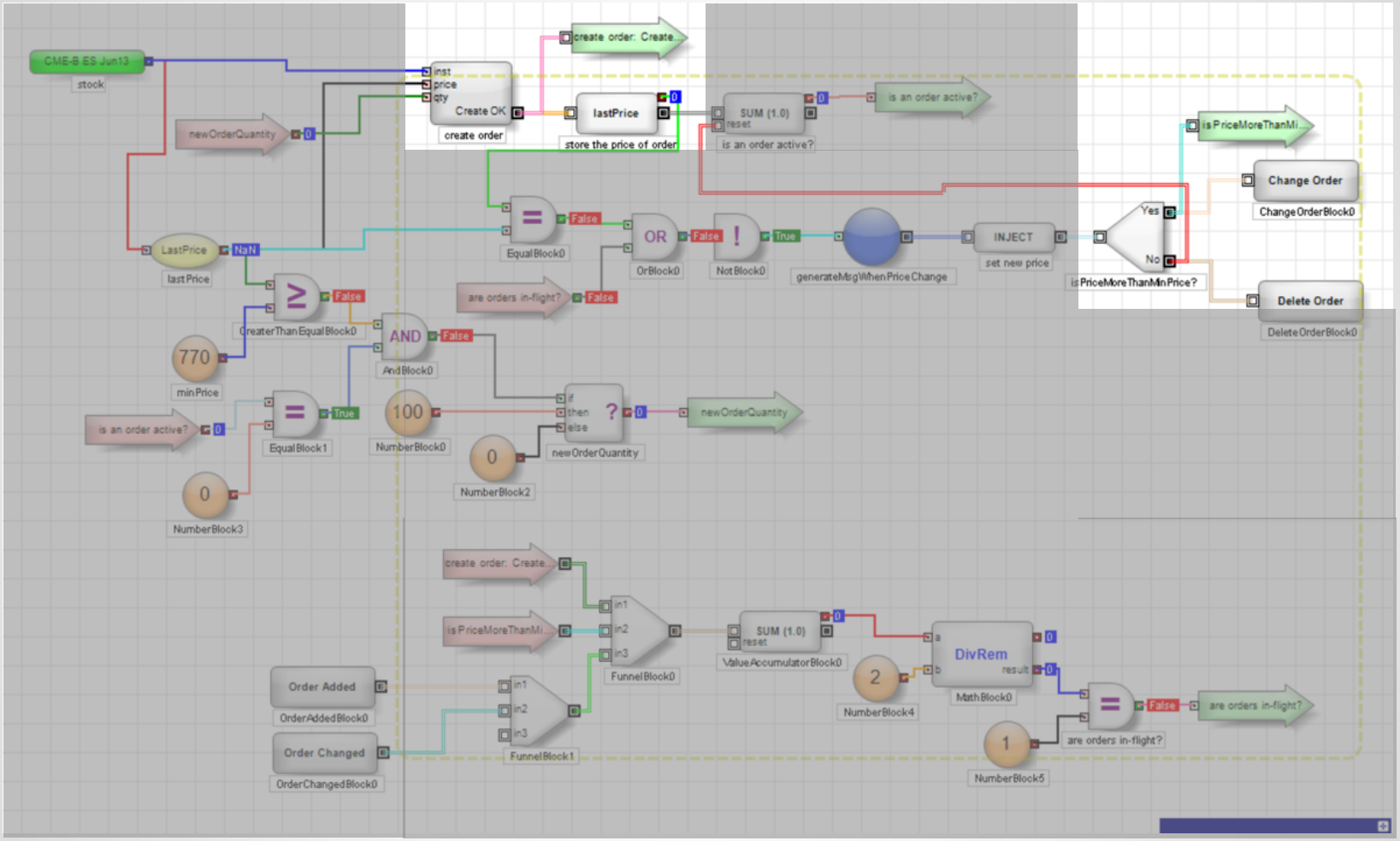
goto; conference

Domain Specific

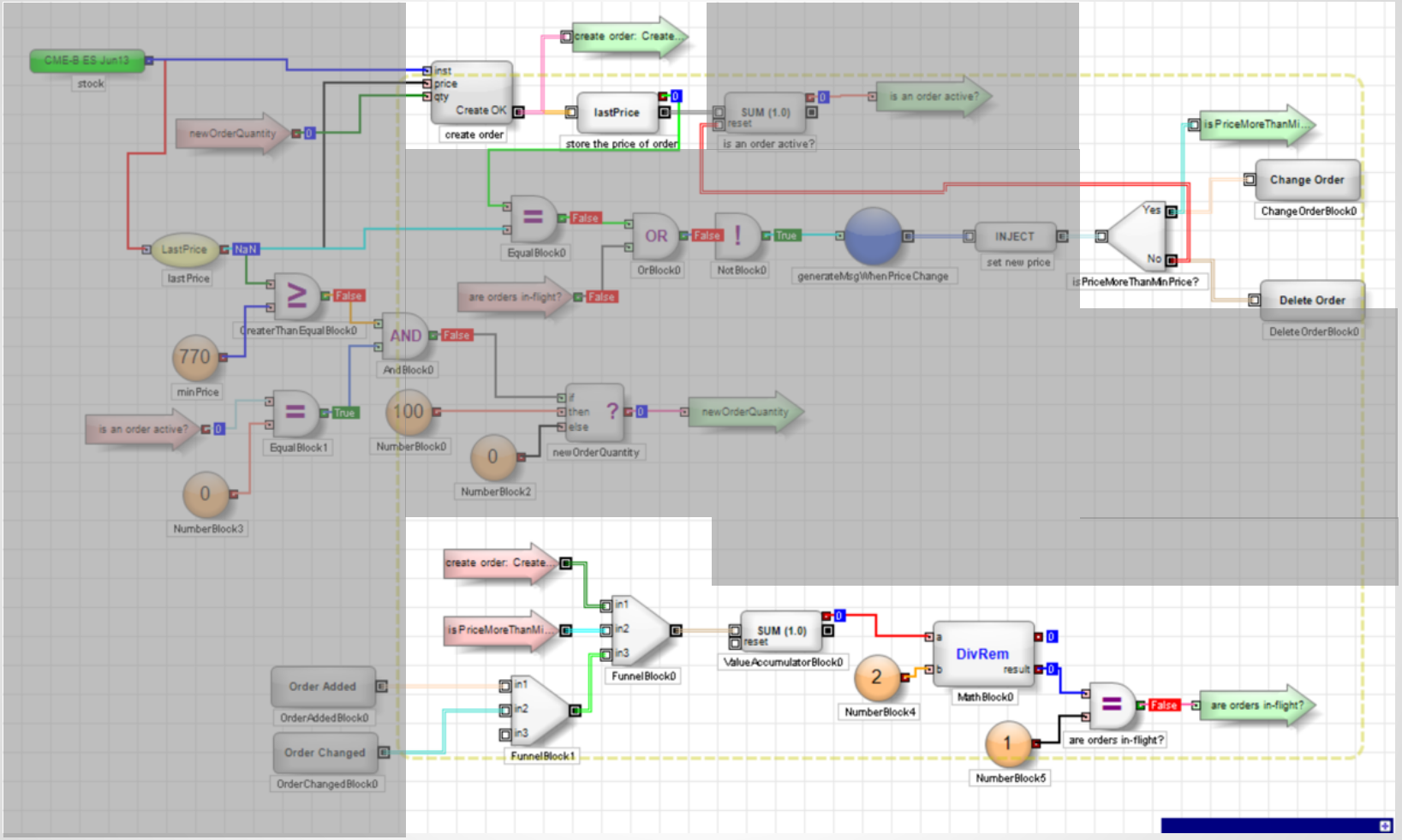


goto; conference

Domain Specific

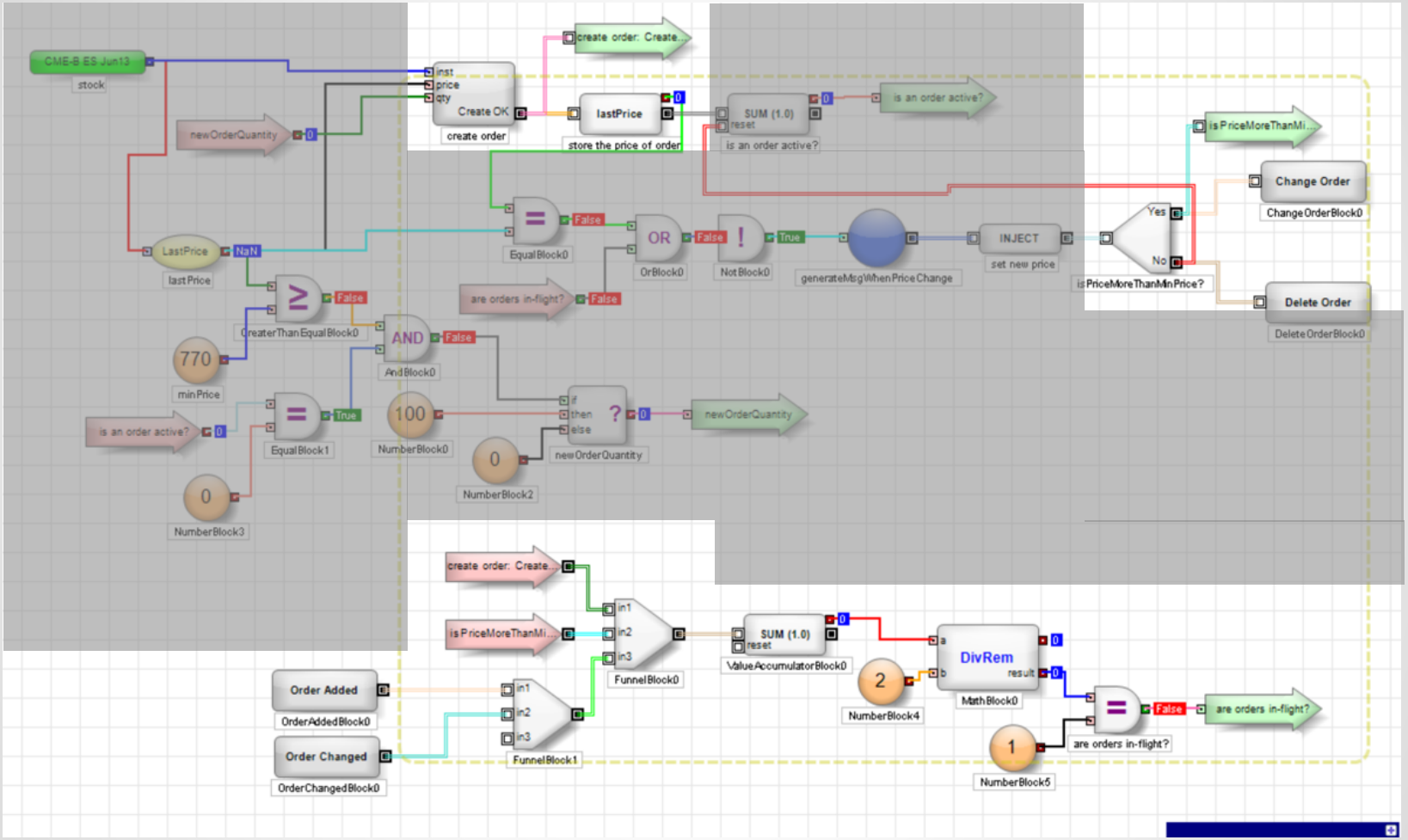


Domain Specific



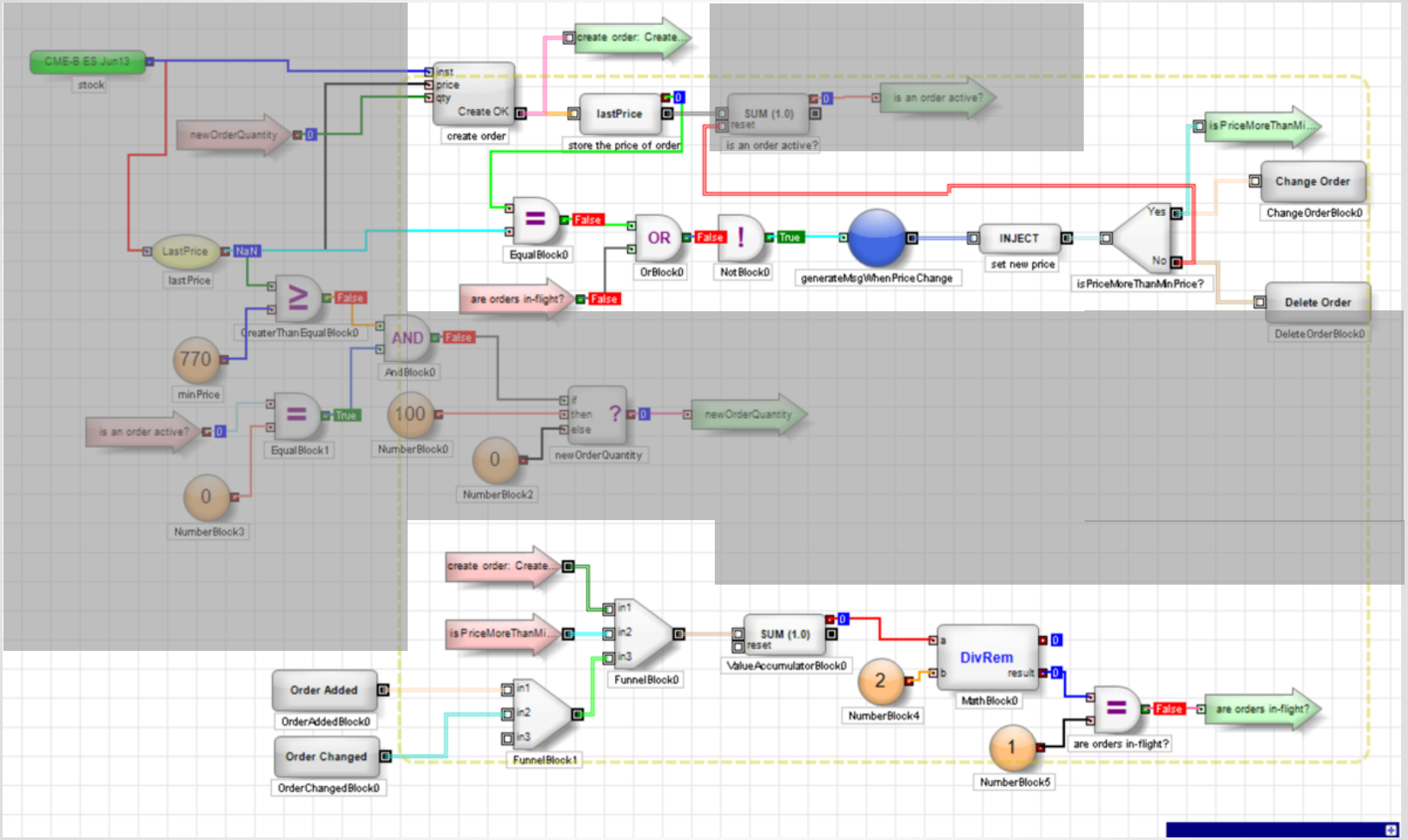
goto; conference

Domain Specific

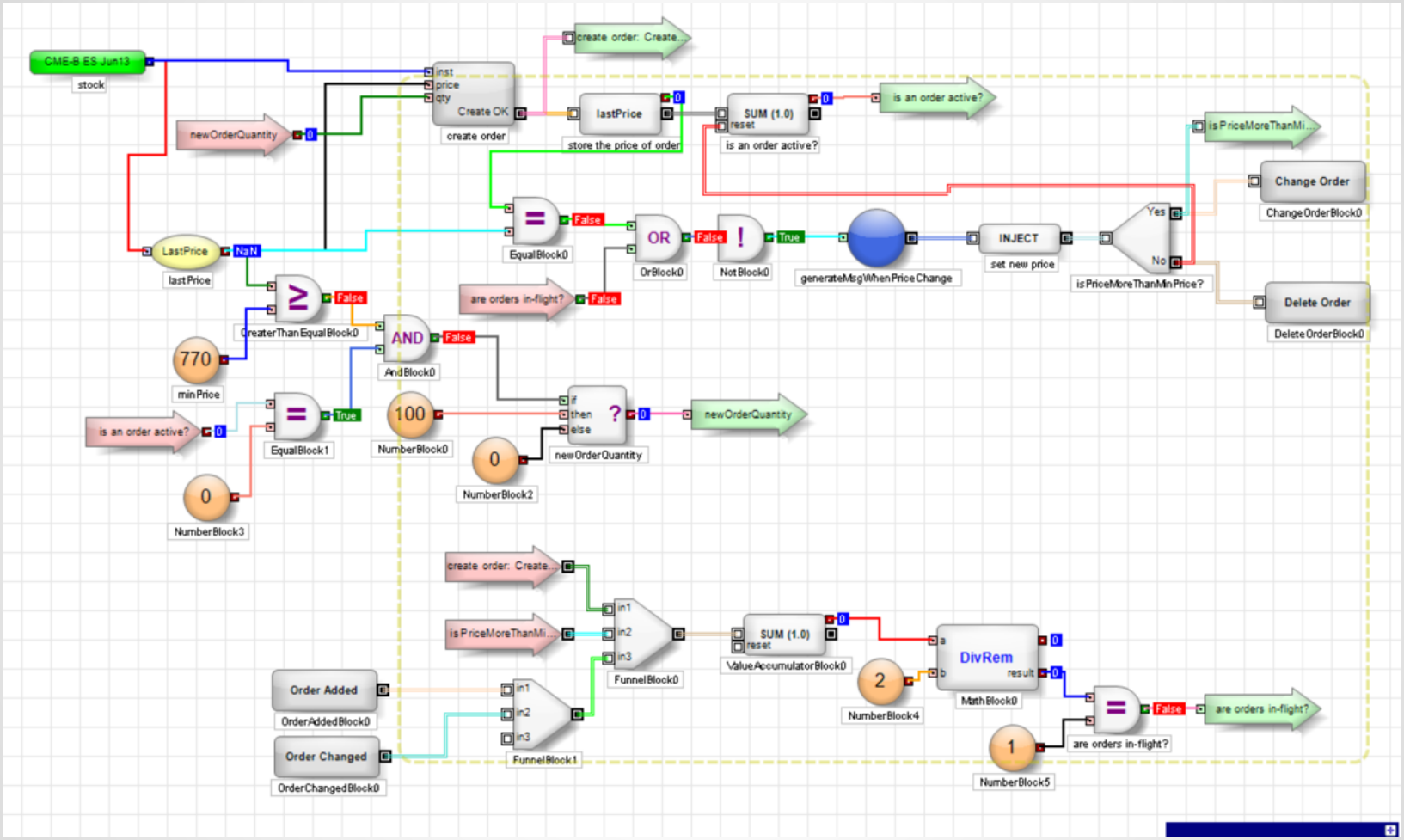


goto; conference

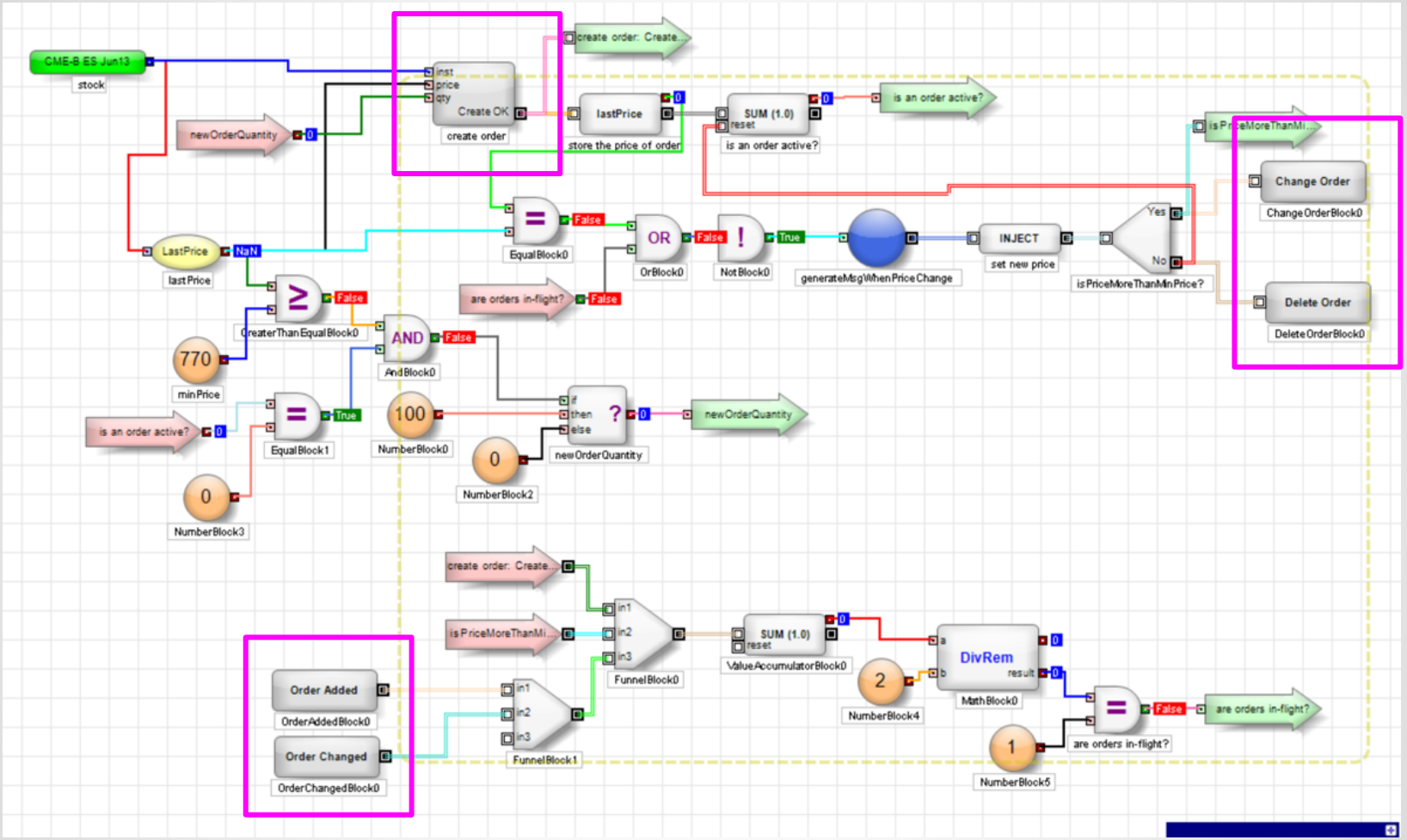
Domain Specific



Domain Specific

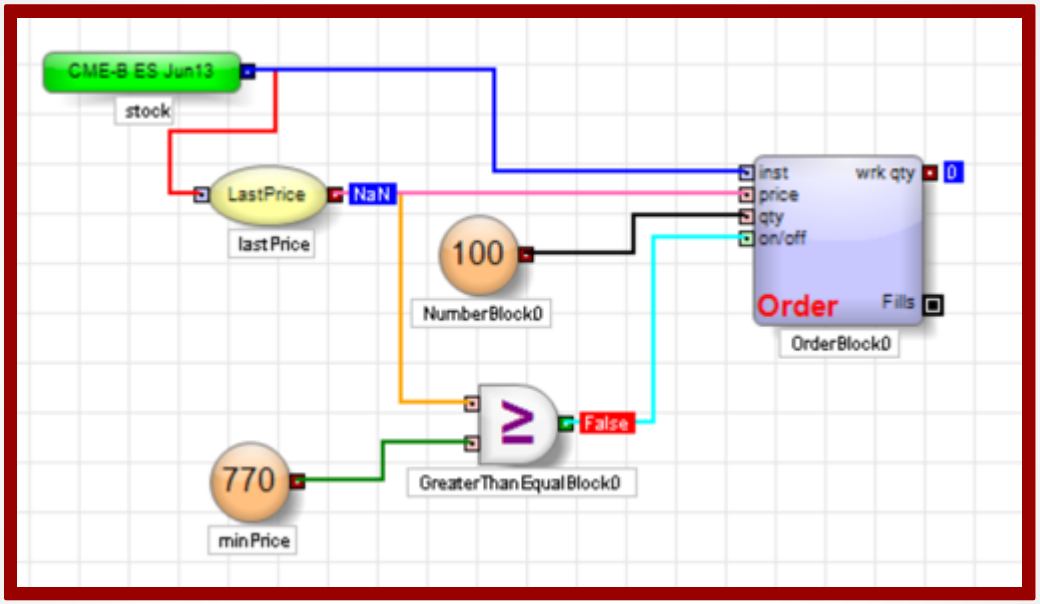


Domain Specific



goto; conference

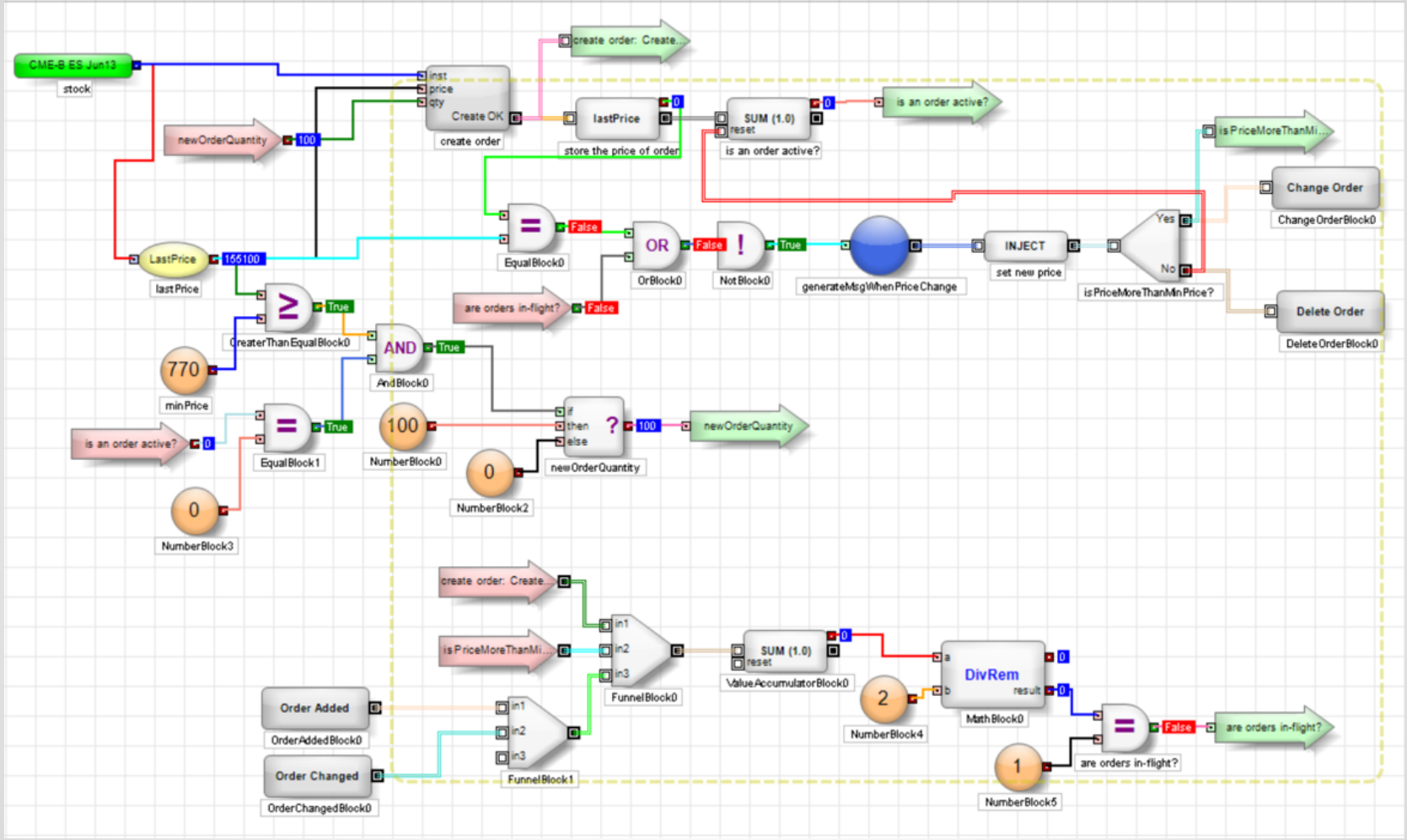
Algo Design Lab (ADL)



(that's it.)

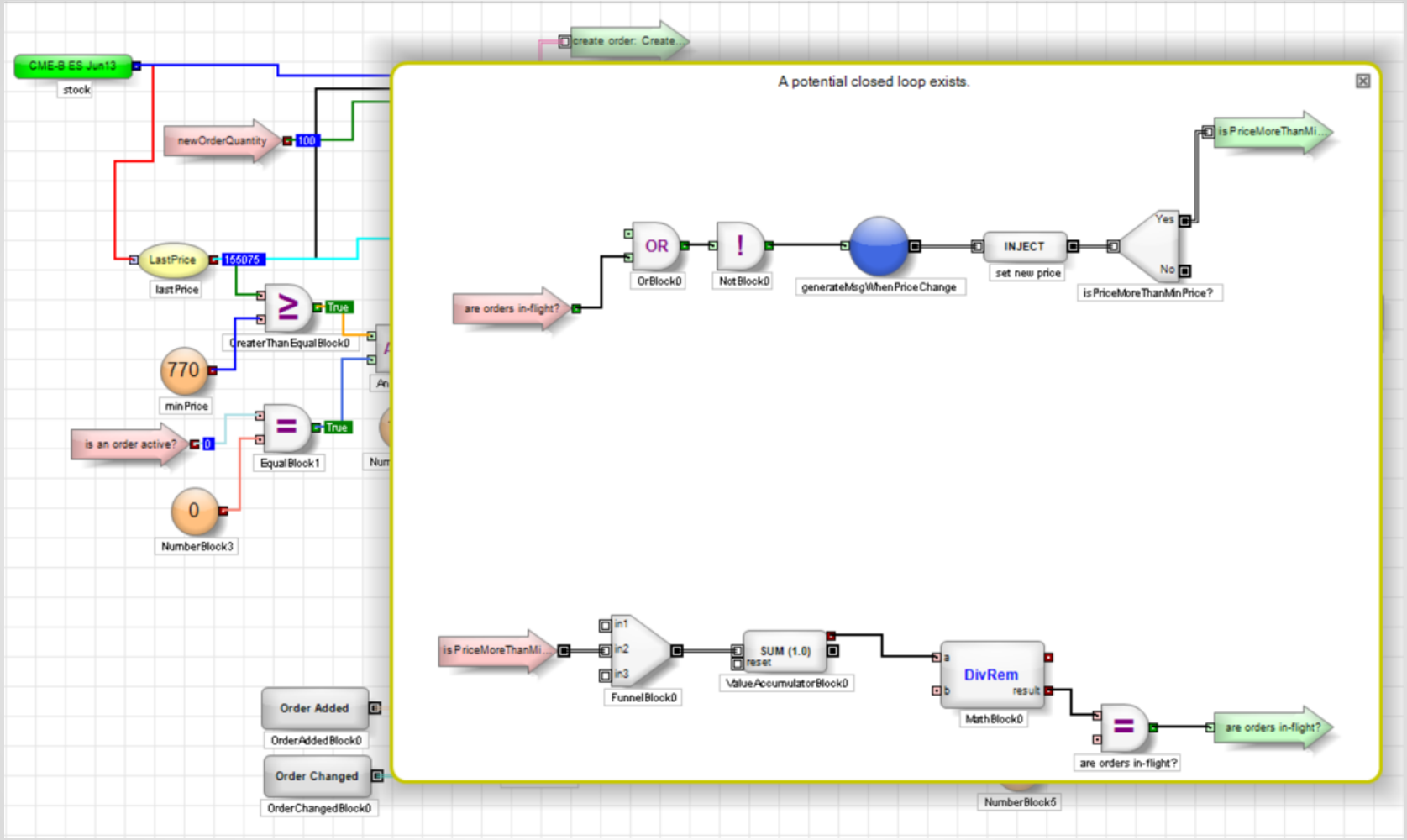
goto; conference

Domain Specific



goto; conference

Domain Specific



goto; conference

Wrap-up*

- VPLs can work for a very narrow set of industries and applications
- Domain specificity
- Hiding the dirty stuff
- Mitigating the "Deutsch Limit"
- **Are our users fruit or cake eaters?**

*** Please exit through the back for processing**

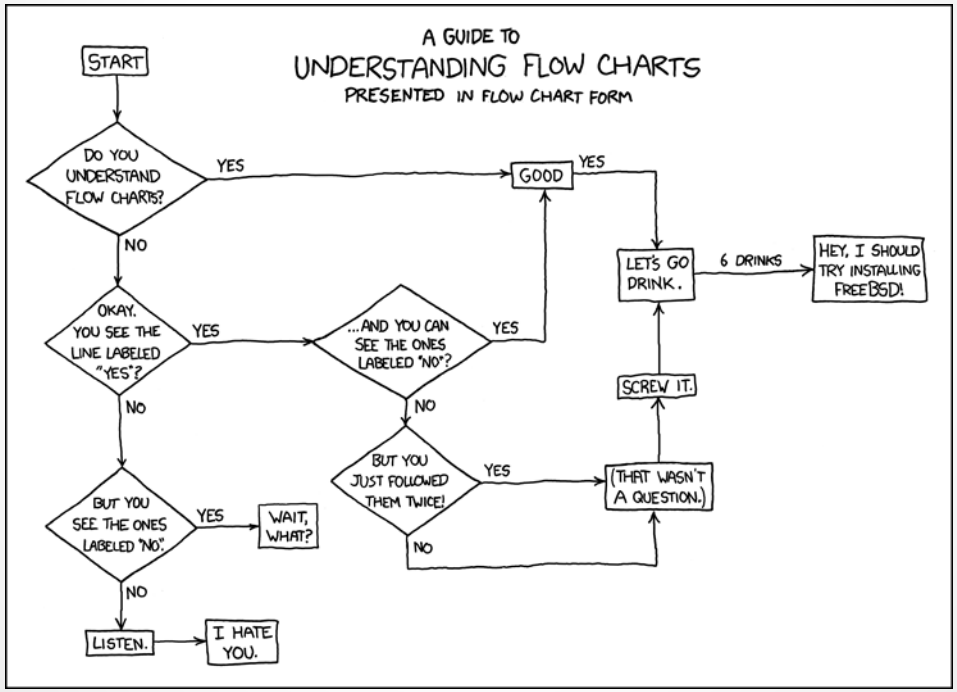
Questions?

Appendix

Flowcharts are evil

"The flowchart is a very poor abstraction of software structure."

-Fred Brooks



Flowcharts are evil

- Poor representation of the actual effort needed to realize the intended goal
- Flowcharts tend to be written *after* you write code, not before
- Too many assumptions; too abstract