Application Frameworks an Experience Report

Arden Thomas, Cincom Systems

World Headquarters
Cincinnati, Ohio

Cincom Smalltalk

– It makes hard things easy, and the impossible, possible

SIMPLIFICATION THROUGH INNOVATION™

Outline

- How it all started
- What Should a Framework do?
- Two Application Frameworks
- ValueModels
- ValueInterface
- Examples & Details
- Conclusion

How it Started

- New job
- Tasked to
  - Correct current problems
  - Set standards for development
    - A guide of how to do things
    - a consistent way of doing typical things
    - Framework assistance

Opportunity knocks …

- The opportunity to build a new application framework!
  - This is a fantastic opportunity for a strategic minded developer, who enjoys the challenge of building new infrastructure

….Reality Sets in

- You can build a new framework ……

….. If it can be done by tomorrow….

- Small shops who expect regular tangible results will often not allow longer term infrastructure work to occur with little to show for it in the meantime. Hard sell.

So ….

- Can’t develop in house, but can pick one

What should a framework do?

- Make things easier, simpler, clearer
- This enhances
  - Productivity
  - Understandability
  - Maintainability

What should a framework do?

- How to make things easier, simpler, clearer?
  - Simplify common tasks by creating methods that do many lines of work with a clear name
  - Simplify difficult tasks
  - Enable / facilitate larger scale reuse and integration

What should a framework do?

- Make things easier, simpler, clearer
  - Suggest conventions
    - Naming
    - Where, when and how to build standard things

What should a framework do?

- *** ***Not impede you, when you need to go beyond what it makes easy*** ***
- This is a key factor that can make or break a framework

Two Frameworks I knew of

- Tim Howard's DomainAdaptor
- Steve Abell's ValueInterface

DomainModel

- Tim Howard's DomainAdaptor
  - Described in his book
    - "The Smalltalk Developer's Guide to VisualWorks"
  - Introduced a lot of great ideas for improving ApplicationModel
  - Showed reader insights into how many things worked
  - Introduced (to many) the notion of utility methods to simplify common tasks

ValueInterface

- Steve Abell's ValueInterface
  - ParcPlace employee
  - built LearningWorks w/ Adele Goldberg
  - Trainer
  - VI Inspired by *slamdunk* architecture

What do the frameworks have in common?

- “One” domain, kept in a ValueModel
  - domainChannel , broker
- Provide assistance building ValueModels to access the domain’s values
- Hides valueModels so Application is not cluttered with instance variables
  - (no instance vars supporting them – hidden in builder)
- Provide for default domain

ValueInterface

- Extends/raises the concept of ***ValueModels*** to the application level
- A simple concept, consistently applied
- Very simple and robust idea
  - Similar to the attractiveness of Smalltalk
  - Consistently applied
  - More than it seems

So - *What is* the concept of ValueModels???

- Represent a model you can get information from
- Allows a clear simple interface (#value), to a potentially complex, sophisticated means of providing that value
- Leverages the dependency mechanism to make many things happen “automagically”
- ValueModels are a capable and heavily used framework component
- ValueModels are the tinkertoy pieces in VisualWorks

Usefulness of ValueModels?

- *simplicity* where wanted or needed
- *complexity* where wanted or better supported

Usefulness of ValueModels?

- “Locality of reference" (in human factors context)
  - It means it is a good idea to put things in one place or near each other, for better, easier understanding

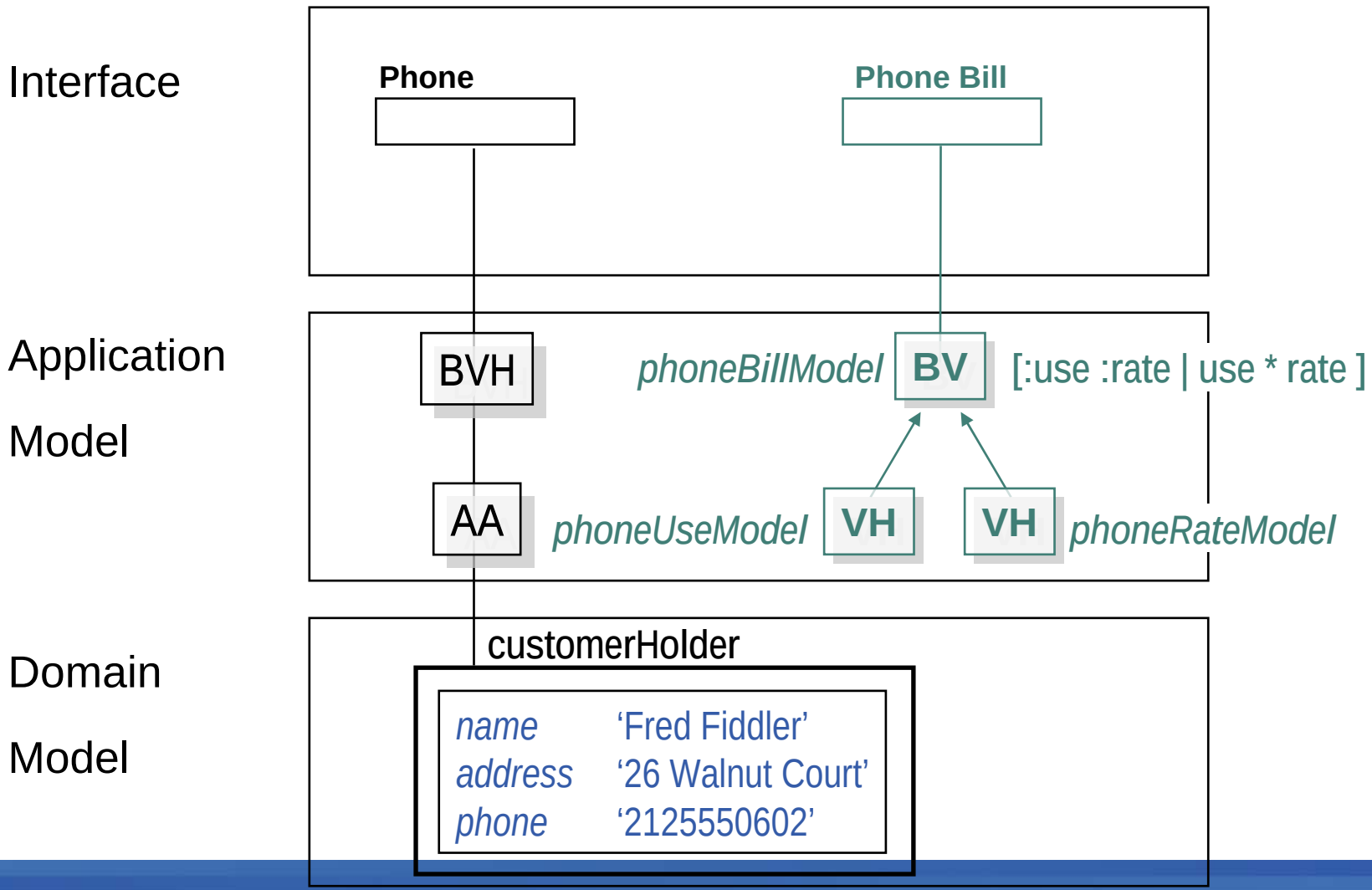
Tinkertoy pieces, ValueModels can:

- Simply hold a value (and report changes)
- “Buffer” a value until triggered
- Retrieve some aspect of a subject object
- Compute a value (and re-compute when needed)
- Trigger other things to happen through fundamental use of dependency

Why Tinkertoy then?

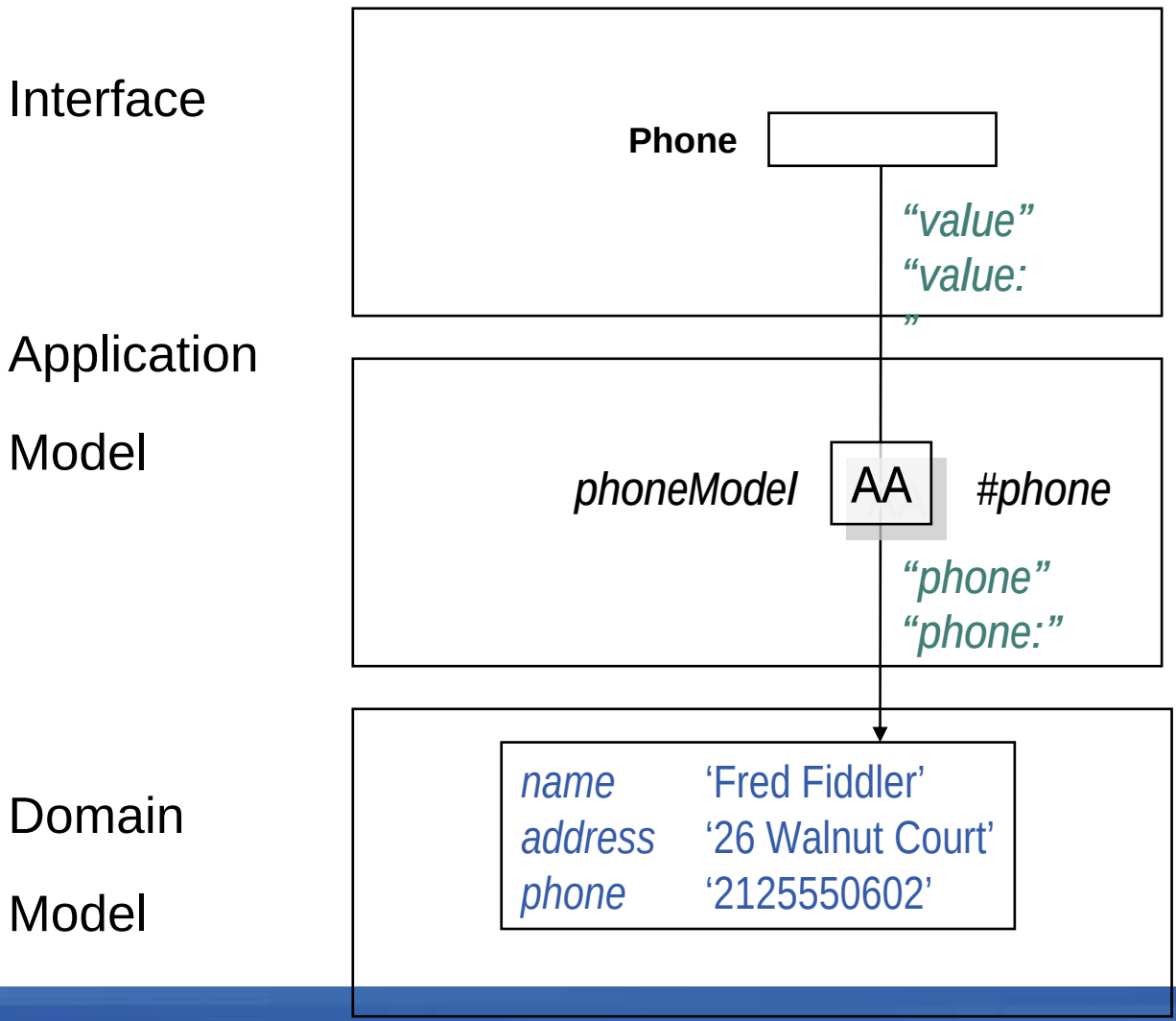
- You can stack or plug valueModels together to get the desired behavior

Stacking BVH on AA, BlockValue example

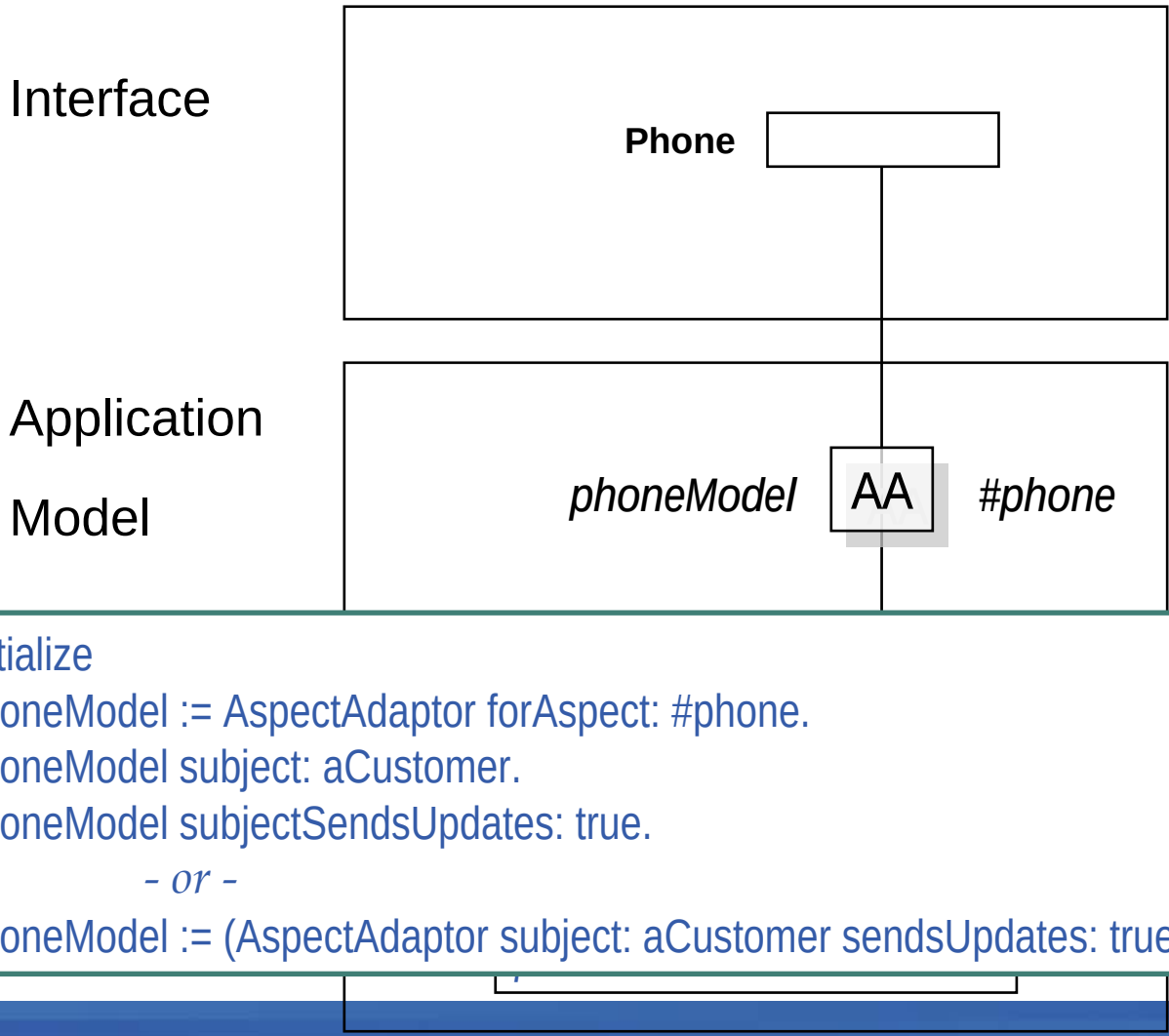


Cincom

ValueModel: AspectAdaptor

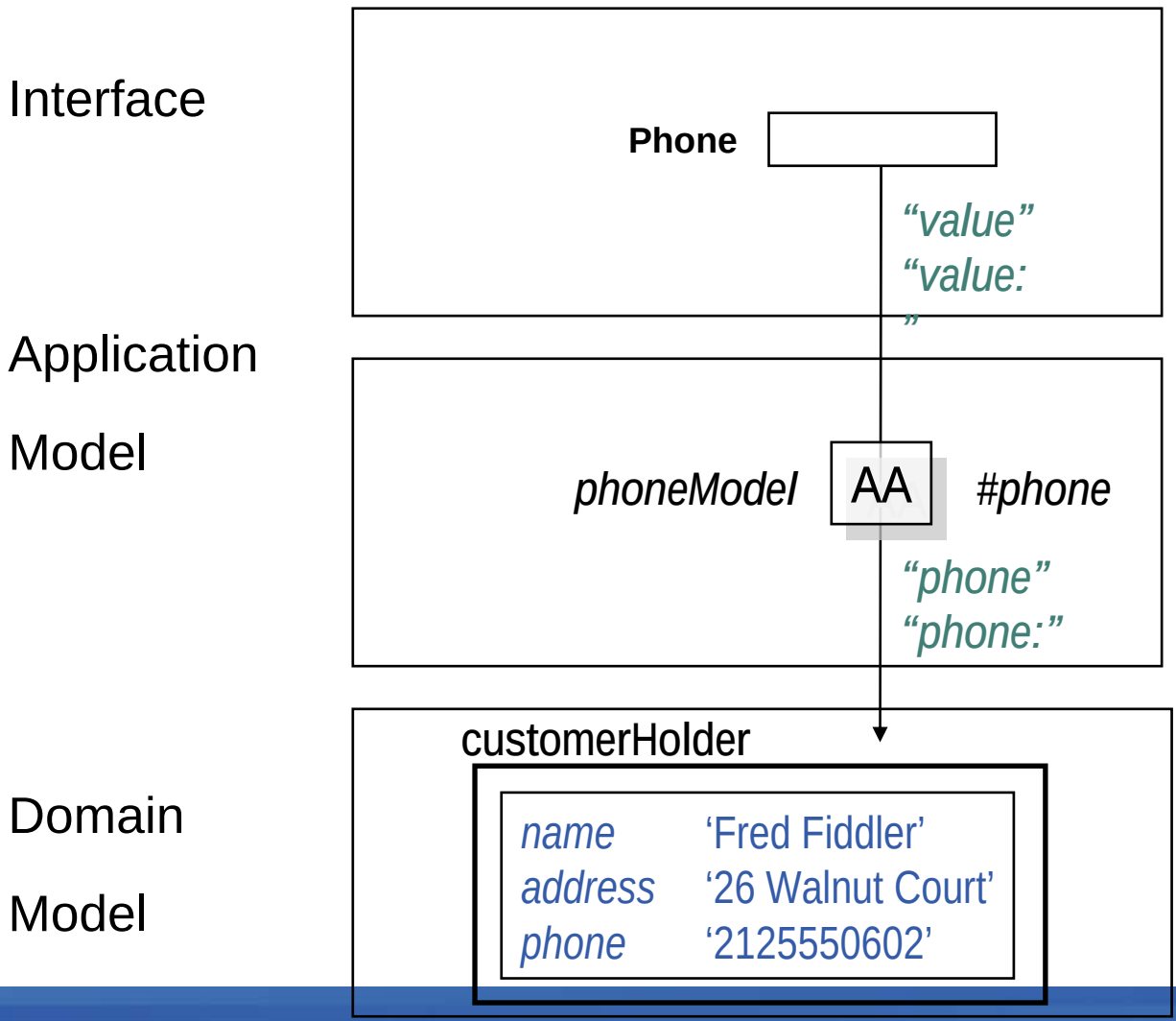


ValueModel: No Framework



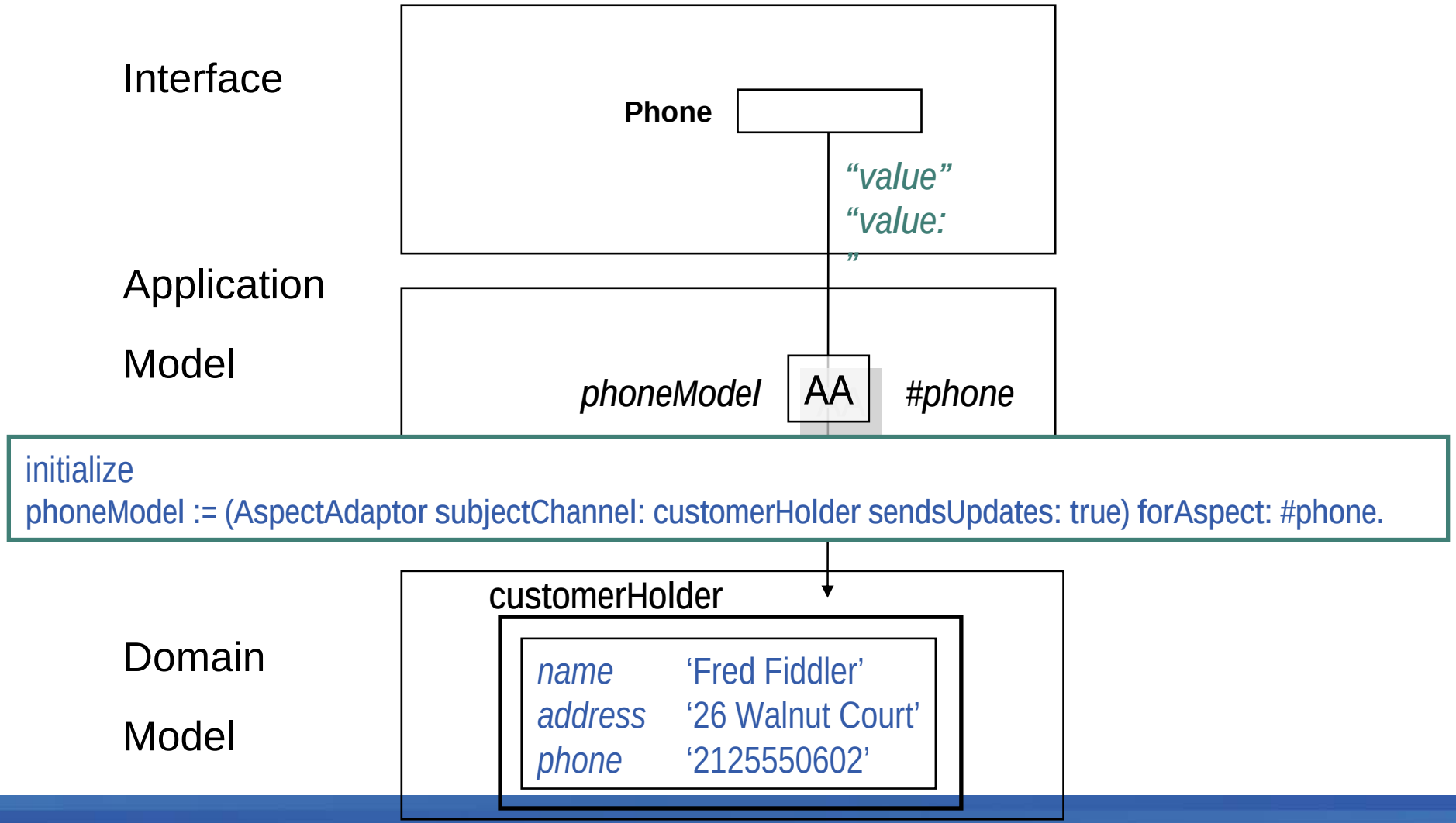

```
initialize
phoneModel := AspectAdaptor forAspect: #phone.
phoneModel subject: aCustomer.
phoneModel subjectSendsUpdates: true.
        - or -
phoneModel := (AspectAdaptor subject: aCustomer sendsUpdates: true) forAspect: #phone.
```

ValueModel: AspectAdaptor with Channel



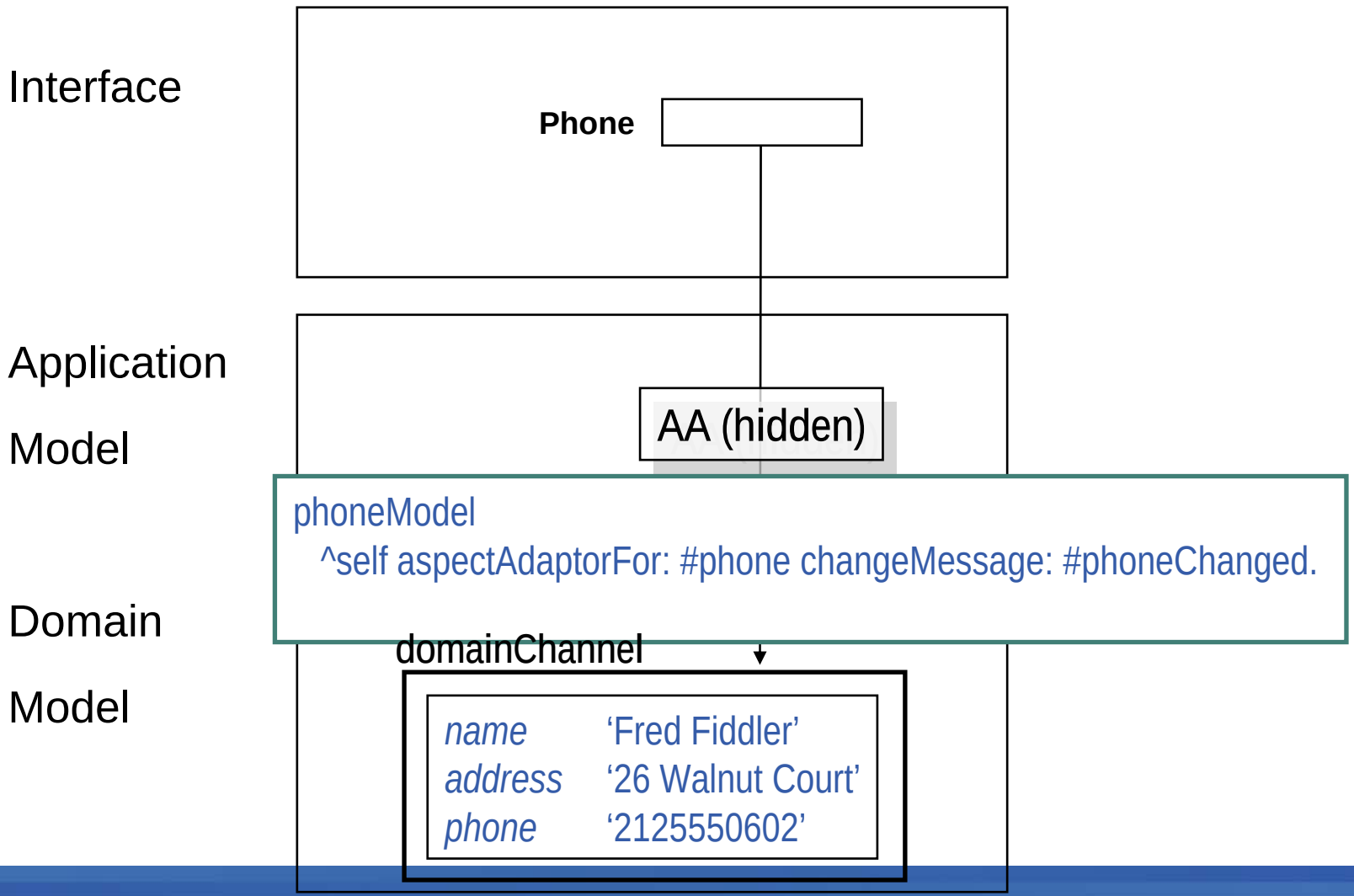
Cincom

ValueModel: AspectAdaptor



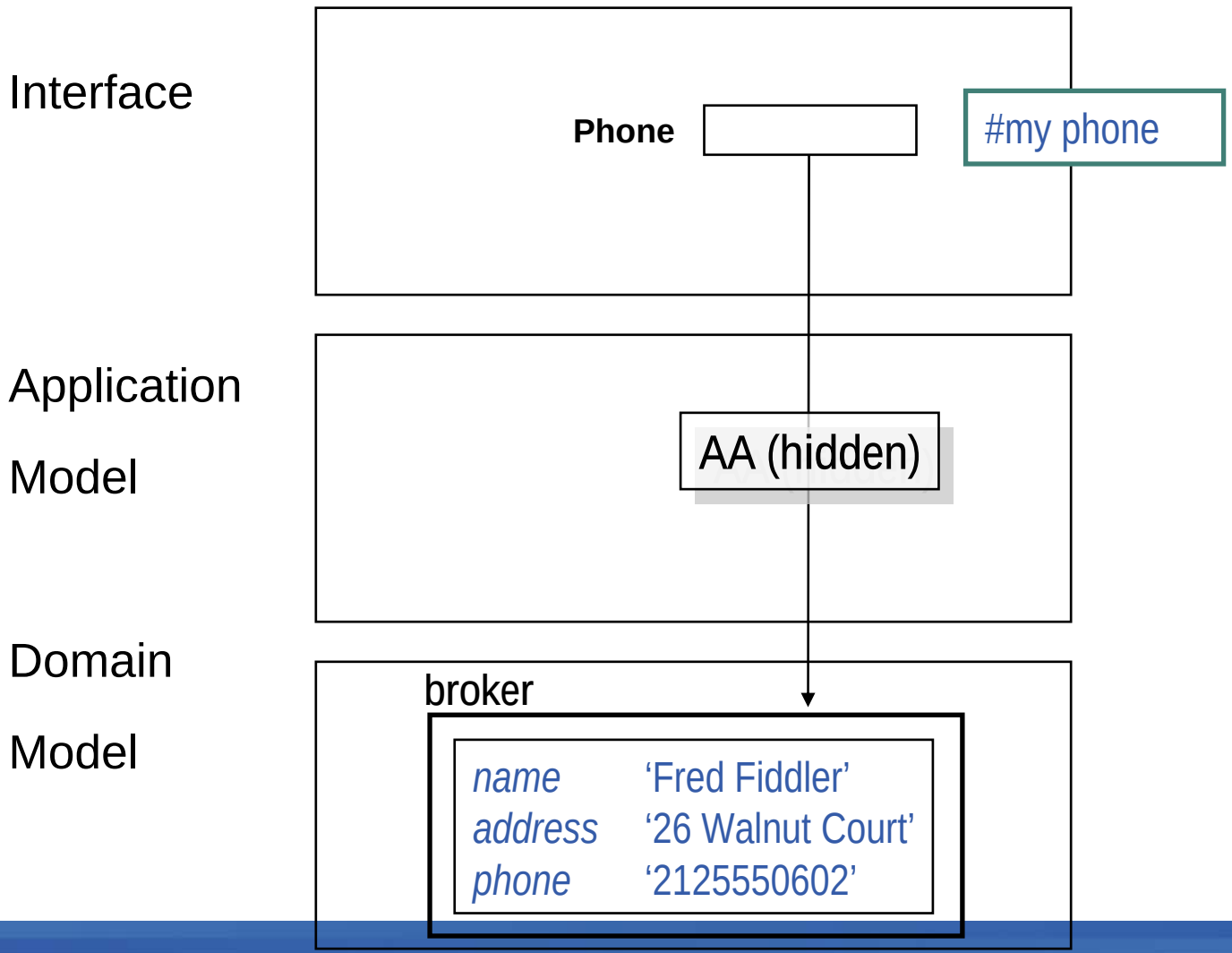
Cincom

ValueModel: DomainAdaptor Framework



Cincom

ValueModel: ValueInterface Framework



ValueInterface

"You can solve any computer science problem by adding a layer of indirection"

- Perhaps, but you want to do it carefully

ValueInterface

- Extends the concept of ValueModels to ***Applications***
  - By behaving like a valueModel (value, value:)
  - *** By allowing an *application* to have the same *tinkertoy like reusability* as a *valueModel*
  - Yields exceptional reuse and modularity
  - It also provides some code saving convenience facilities

What was added

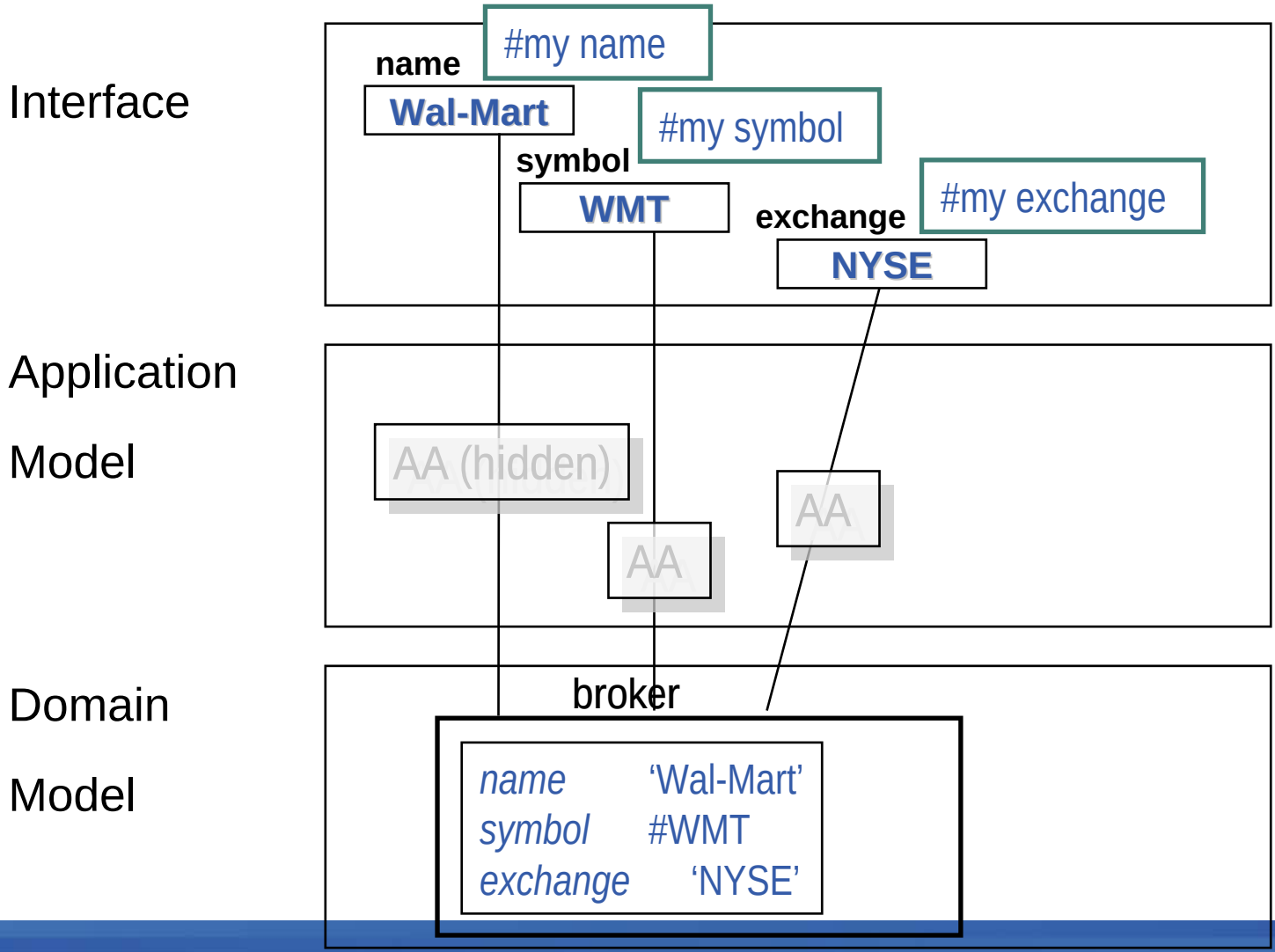
- Added in a subclass
- More options in the aspect property
  - “my”, so you could write “my name”, “my address” as properties to retrieve that value from the domain
  - Math, allowed +,-,*,/ for models
  - Filter can be specified
    - Allows a filter to be provided which translates the object into an appropriate display. i.e. upperCaseFilter

What was added

- Lots of utility methods
  - hide:, show: , enable: , disable: , widget: , component: ….
- Keyboard Navigation
- suppressChangesWhile:[ ]
- batchUpdates

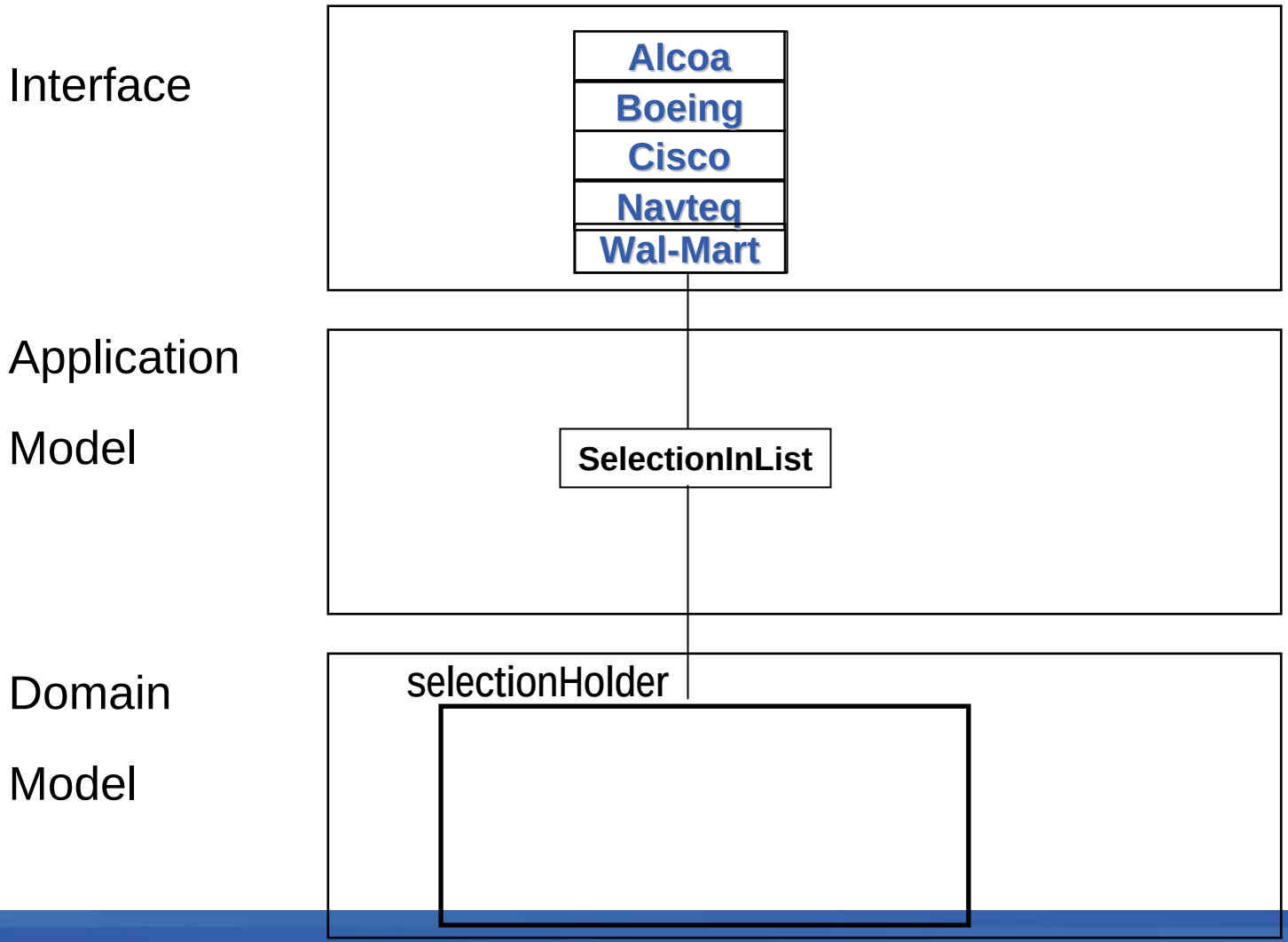
Examples

ValueInterface - TradingSecurityApp

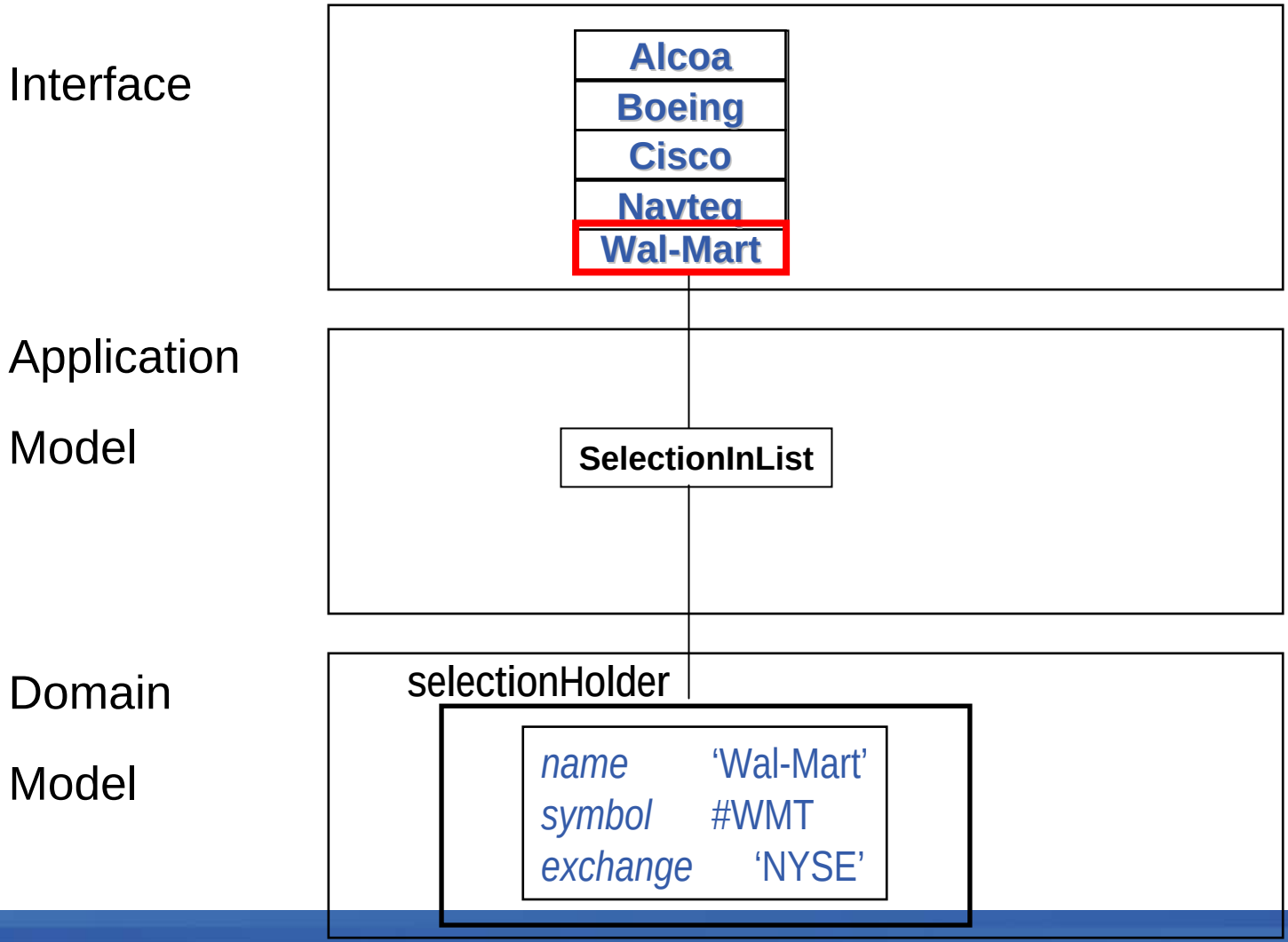


Cincom

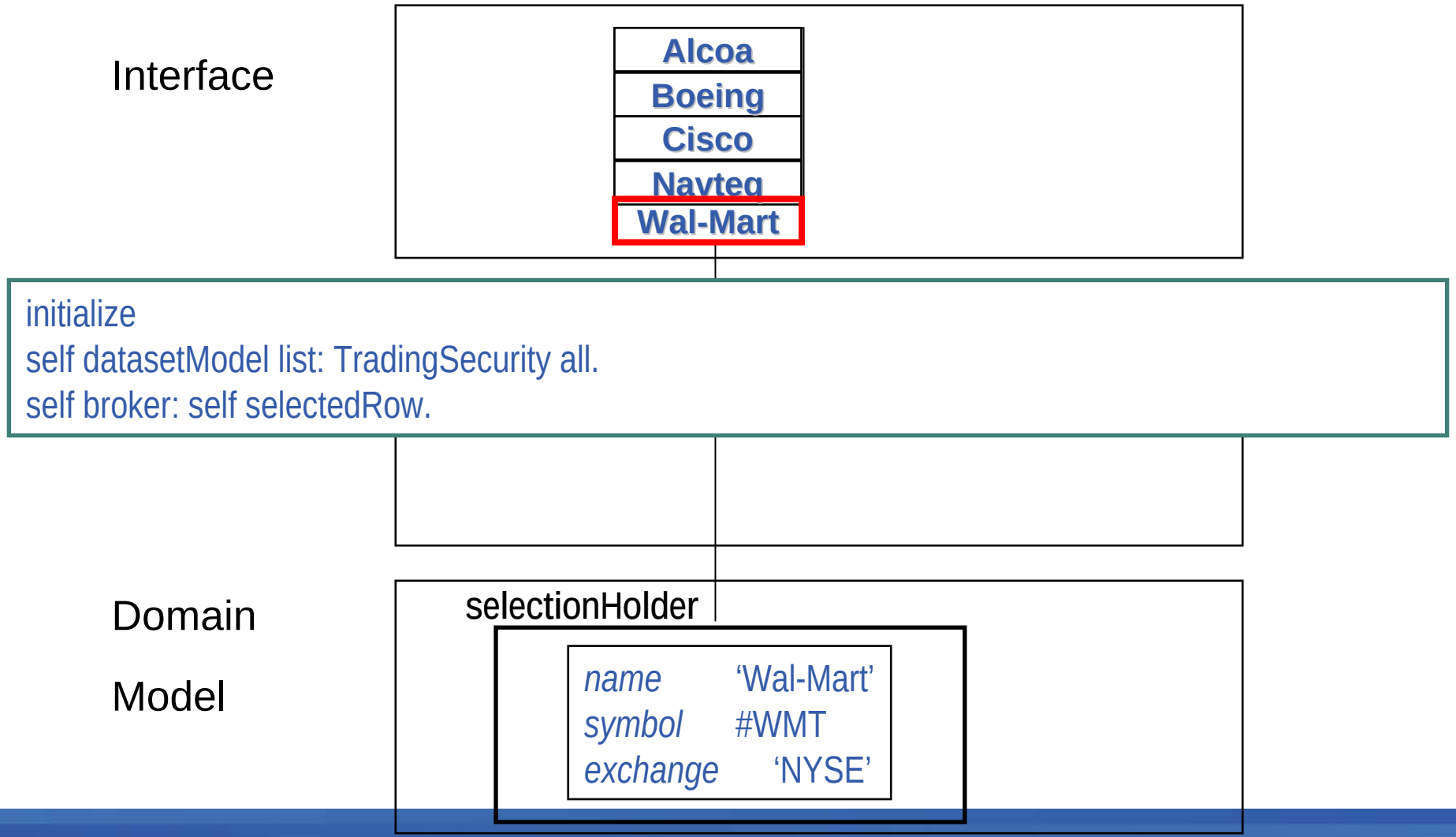
ValueInterface - SecurityListApp



ValueInterface - SecurityListApp

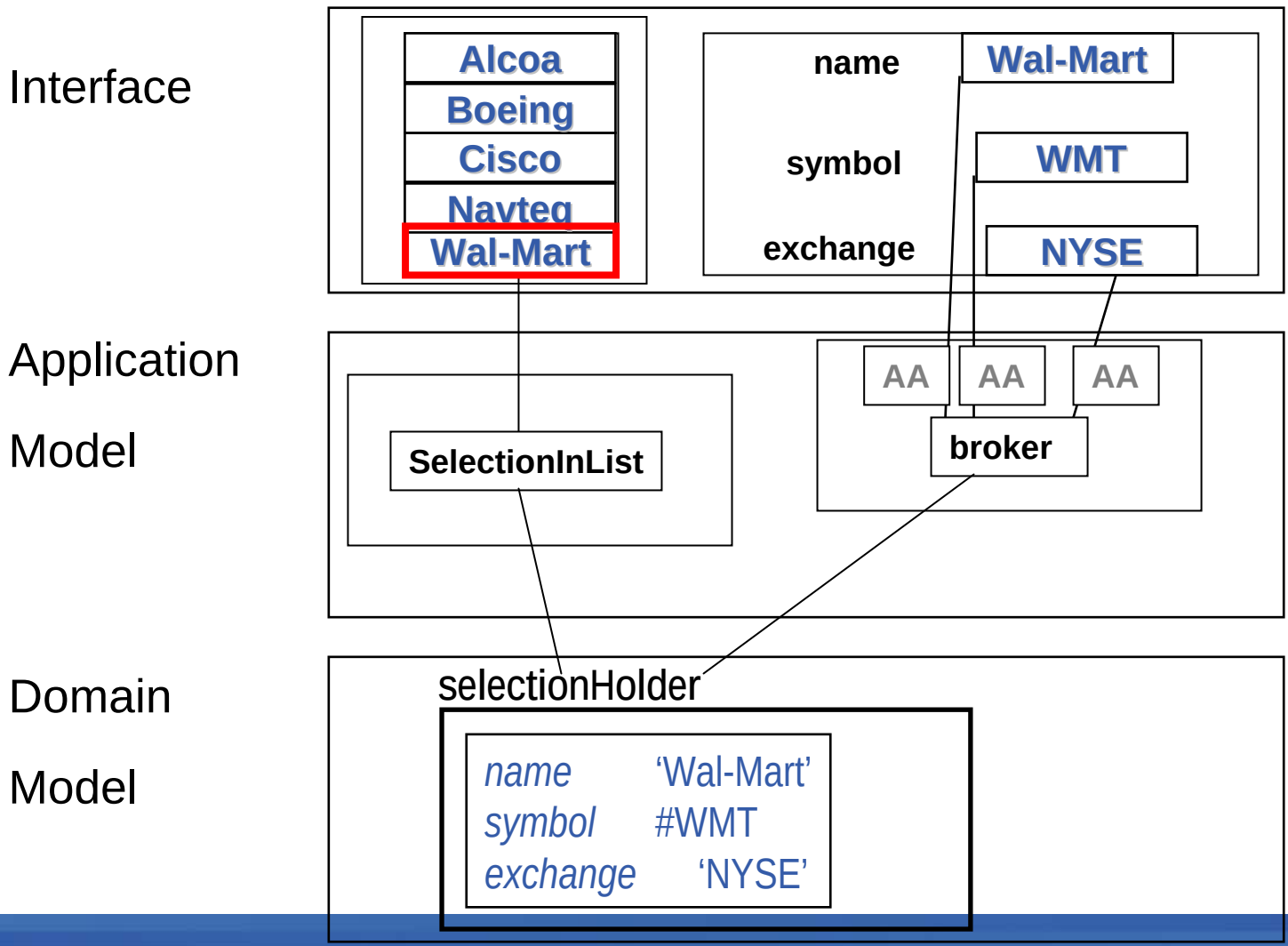


ValueInterface - SecurityListApp



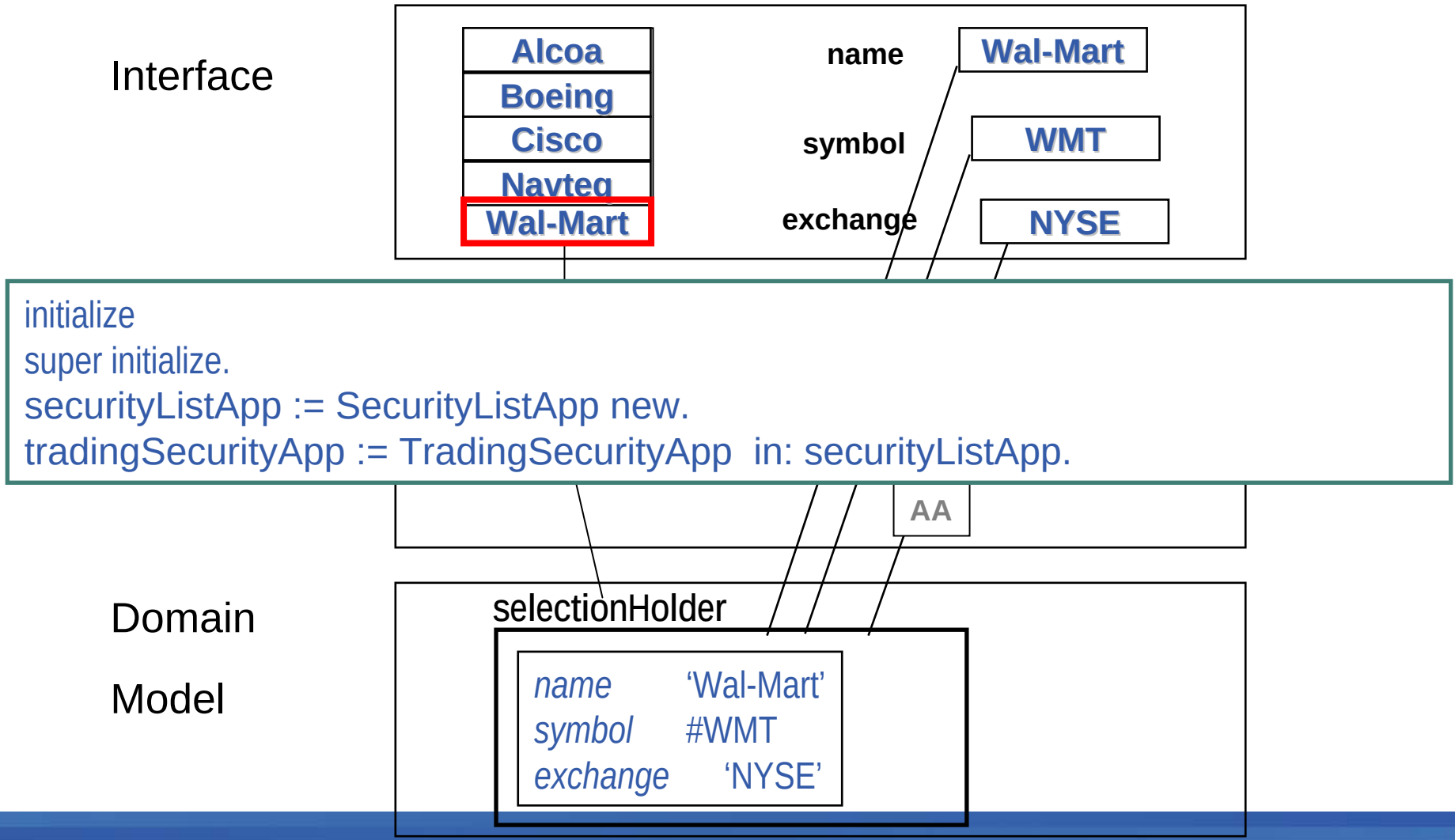
Cincom

ValueInterface



Cincom

ValueInterface



Cincom

Updates
– Application reacting to domain updates

- The standard to use in your domains, when their instVar is changing is: self changed: #aspectChanged

  name: aString

  ```
      name := aString
      self changed: #nameChanged
  ```

- You then simply implement the method #nameChanged in your application; it will be called whenever the domain changes its name

- You no longer need #update: with: from: and its logic to handle this

Updates II
– Application reacting to domain updates

- What if you need the parameters that were sent in #update:with:from: ????

nameChanged

```
^[:aReceiver :anAspect :aValue :aModel |

        ….your update code here ….  ].
```

Note: Use aReceiver instead of self in the block

Hooks

- broker:
  - Use this to hook up your own valueModel as the domain holder
- attachValue:
  - Override to have behavior before new object becomes domain
- detachValue:
  - Override to have behavior with the old object before the new object becomes domain

More hooks

- MyApp value: myDomain
- MyApp withValueHolder
- MyApp in: anotherApp
  - myApp shares anotherApp's domain
- MyApp forAspect: #anAspect in: anotherApp
  - myApp's domain is anAspect of anotherApp's domain

Other neat stuff

- void
  - Like nil, but silently disregards messages it does not understand
  - Use it sparingly, but nice to have in your toolkit
  - Use it where you have an option that is not there

Widgetry

- Applicability of this framework to the new Gui framework
- ObservedValue
  - The new ValueHolder
- UserInterface
  - The new ApplicationModel
- ObservedUserInterface
  - The new ValueInterface inspired framework

Conclusion

- Frameworks should make things easier, but not get in the way
- ValueInterface
  - A simple, robust concept, consistently applied
- More than it seems
- “One of the best decisions we made”

Contacts

- Arden Thomas, Cincom Systems Inc.
  - athomas@cincom.com

- Steven T. Abell (Author of ValueInterface)
  - info@brising.com