Calculating Requirements: an Approach Based on Architecture Style

Dave Wile
Teknowledge Corp.
Dwile@teknowledge.com

RE Calculi Candidates

- Performance
- Reliability
- Security
- Robustness

but, over what atoms?

The “Requirements”

- Integrate the functionality of:
  - Cell Phone (CP)
  - PDA
  - Digital Camera (DC)
  - GPS
  - Watch
  - Compass
  - Voice recorder
  - Email machine
  - Internet node
- Size, weight, resiliency, security, performance are important

Too Implementation Oriented?

Functional spec instead:

- Take and store pictures: time, date, position, and orientation-stamped.
- Call people: dial, voice activated, from address book, and from GPS DB.
- Receive calls from people: live, callback, recorded message.
- Take memos: voice, text, positions.
- Keep track of addresses and phone numbers

More Functions

- Manipulate to-do list, with alarms.
- Keep track of date book with appointment alarms
- Send and receive email.
- Internet access.
- Find businesses and friends close to where I am.
- Provide driving instructions.
- Check time easily.

Pretty Brittle!

Engineering

- Incremental evolution from known solutions
- Error analysis
- Sensitivity analysis – performance envelope
- Non-functional requirements

Use Software Architectures for Non-functional Requirement Specification

- Basics
  - Components
  - Connectors
  - Types
  - Attachments of connectors to components
  - Subarchitectures
  - Properties
- Associate structure with the connectors
- Analyzers for non-functional properties
- But, calculation too hard in general (requires catamorphism over cyclic structures - Gibbons)

BUT…

(Ta da) Architecture Styles

- May be much more constrained in their use of connectors
- Example: Model / View / Controller
- MVC idea is to separate
  - Information (model)
  - From the way it is displayed (view)
  - From the GUI for selecting it (controller)
- Our specifications will represent components of these three types (as sets)
- Connected by 5 operators
- Normally, we would have a graphical representation, but sometimes 1000 words produces prodigiously large pictures!

“Models:” Operators Intuitions

- + - Each model is kept separate:
  e.g. *addressList + memos*.
- | - Only one model will be chosen in the specification:
  e.g. *addresses | phones*.
- < - The right model overrides the left one:
  e.g. *PDA.time < GPS.time*.
- # - An integration or synchronization activity is necessary to put the two models together:
  e.g. *CP.phoneNums # PDA.addresses*

Views:
Operator Intuitions

- + - Each view is kept separate:
  e.g. *calendar + time*.
- | - Only one view will be chosen in the specification:
  e.g. *addressList | todoList*.
- < - The right view overwrites the left one:
  e.g. *normal < alarm*.
- # - An integration activity is necessary to put the two views together:
  e.g. *map # currentPosition*

Controllers: Operator Intuitions

- + -Each event is possible in parallel:
  e.g. ctlKey + |/{Akey, … ,Zkey}
- | - Only one event will occur:
  e.g. scrollUp | scrollDown.
- < - The right event overrides the left one:
  e.g. onOff < reset.
- # - An integration activity is necessary to put the two controls together:
  e.g. placeCall # selectAddress

Conventions

- + / {a,…,z} = a + … + z
- {a, b} instead of {a} +{b}
- {a: x} means a has the structure of x
- {Type …} means a set of elements of that type.
- || x means x | |/ x. E.g. a directory (x) is shown or one of the elements of x.
- x.y is used to refer to some attribute y of x
- operator precedence is strictly left-to-right
- nesting is indicated by parentheses
- Singleton confusion: a # S = {a} # S (when obvious)

Graphical Representation

- Would allow properties to be attached to each application of an operator
- Would also connect
  - the views with the models viewed
  - the views with the controllers
  - Here, connection indicated as additional properties – e.g. ModelFor('showAddressList')
- (Would eliminate parentheses)

Example PDA MVC Spec

- **Models:** {addresses:{Address…},
  toDos: {ToDo …},
  appointments: {Appointment …},
  memos: {Memo…},
  time, preferences}
- **Views:** { ||toDos, ||addresses, ||memos, preferences,
  |/{appointments.day, appointments.month,
  appointments.year, appointments.current}}
- **Controls:** |/{grafitti, keyboard, find, viewAppointments,
  viewToDos, viewAddresses,
  viewMemos, viewFind, viewPreferences,
  onOff}

Example CP MVC Spec

- **Models:** {phones: {Phone …},
  missedCall: {Call…},
  dialingNumber, time, preferences}
- **Views:** |/{ ||phones, ||missedCall,
  ||outgoingCalls, ||incomingCalls,
  dialingNumber}
  + (time < preferences)
- **Controls:** |/{enterDigit, answer, hangUp,
  selectPhones, selectMissed,
  selectOutgoing, SelectIncoming,
  viewPreferences, onOff}

A Composite Specification

- Electronic Swiss Army Knife (ESAK)
- May want to write:
  DC #GPS #PDA #CP
  (leaving all implementation decisions to the implementer!)
- More controlled spec:
  - ***M*** ESAK = ***M*** DC +***M*** GPS +***M*** PDA +***M*** CP
  - ***V*** ESAK = ***V*** DC |***V*** GPS |***V*** PDA |***V*** CP
  - ***C*** ESAK = ***C*** DC +***C*** GPS +***C*** PDA +***C*** CP

Introduce “-” Operator

- ***C*** ESAK = onOff +
  (***C*** DC- DC.onOff) +
  (***C*** GPS- GPS.onOff) +
  (***C*** PDA- PDA.onOff) +
  (***C*** CP- CP.onOff)

(note singleton confusion)

- ESAK.onOff=
  DC.onOff # GPS.onOff # PDA.onOff # CP.onOff
- ***M*** ESAK =
  …+ (CP.phones <PDA.addresses) +…

A Calculus (almost)

- Positive Selection Operators in a formula
- Number of Selects Value

NS(a+b)	NS(a) + NS(b)
NS(a-b)	NS(a) – NS(b)
NS(a|b)	NS(a) + NS(b) + 1
NS(a<b)	NS(a) + NS(b)
NS(a#b)	NS(a) + NS(b)
NS(a)	*0*

A Calculus: Tables

- Built in homomorphism

  $F(a\ op\ b) = F_{op}(F(a), F(b))$

- Sometimes need information from node itself
- Built in paramorphism

  $F(a\ op\ b) = F_{op}(node, F(a), F(b))$

  Where node = (op,a,b)

- E.g. factor of memory used, *ifactor*

(Maximum) Resources Used

- *Resources Used* *Apply to R(a), R(b)*

  R *where node = (op,a,b)*

a+b	+
a-b	-
a|b	max
a<b	λ ra,rb . (ra+rb) * node.ifactor
a#b	λ ra,rb . (ra+rb) * node.ifactor
a	*node.PR*

Across-type Effects

- Need to reference Model from Controller, for example
  - Might use connectors
  - Here, use property MCB – model controlled by
- Example, interested in resiliency:
  - compute Number of Available Controls as memory components degrade
  - Apply NAC to Controllers

Resilience: Number of Available Controls

- *NAC* *Apply to NAC(a), NAC(b) w/ node free*

a+b	+
a-b	-
a|b	+
a<b	λ ra,rb . **if** AV(node.b.MCB) **then** rb **else** ra
a#b	λ ra,rb . **if** AV(node.MCB) **then** rb + ra **else** 0
a	***if*** *AV(node.MCB)* ***then*** *1* ***else*** *0*

Availability Predicate

- *AV* *Apply to AV(a), AV(b), node free*

a+b	and
a-b	λ(ra,rb) ra
a|b	and
a<b	or
a#b	and
a	Exists *r: resource* |
	node.AssignedTo=r and
	LiveResource(r)

Samsung SPH-i300 (CP+PDA)

- **Models:**

(***M*** PDA - PDA.addresses - PDA.time) +
(***M*** CP + outgoingCalls +
incomingCalls - CP.phones - CP.time) +
((CP.phones + voiceData)
PDA.addresses.phones) + [[**1**]]
(PDA.addresses -
PDA.addresses.phones)) + [[**2**]]
(PDA.time < CP.time)) + [[**3**]]
{CP.thisNumber, CP.serviceNumber,
CP.speedDial}

Notes

1. Voice data added to phone list and then integrated with address book phones
2. Remainder of the address book information
3. Separate memories used

(All this is known because PDA/CP died)

Problems

- Equational reasoning:
  - interfered with by using node-specific properties. E.g.
    $|/a \# |/b = |/(a\#b)$
    may not hold if properties are attached to #.
  - property equivalence classes ~ comments
  - may need to consider other relationships between architectural elements that are not described by connectors, MCB, similarly
- Normal architectures have more complex (cyclic) structures – graph paramorphism as limit on tree paramorphisms

More Work

- Formalize – imprecise semantics of operators leads to sloppiness
- Abstraction wo / MVC
- Flesh out MVC style
- Calculus for another style with less hierarchy
- Promotion theorems