rOOts 2005: Sample code and slides for *a New Discipline of Programming*

Trygve Reenskaug, Department of informatics, University of Oslo, Norway
http://heim.ifi.uio.no/~trygver

1 NetworkTester CODE

class NetworkTester

```
Object subclass: #NetworkTester
   instanceVariableNames: 'activitiesData simpleComponent maestroComponent
                                                declarativeComponent
```

public methods

test

```
   " NetworkTester new test. "
   self initialize.
   " Test SimpleComponent. "
   simpleComponent := SimplePort new initialize.
   self runPort: simpleComponent.
   self checkPort: simpleComponent.
   " Test MaestroComponent. "
   maestroComponent := MaestroPort new initialize.
   self runPort: maestroComponent.
   self checkPort: simpleComponent.
   " Test DeclarativeComponent. "
   declarativeComponent := DeclarativePort new initialize.
   self runPort: declarativeComponent.
   self checkPort: simpleComponent.
   " Inspect. "
   self inspect.
```

private methods

checkPort: port

```
   port activityNames = (activitiesData collect: [:dict | dict at: 'name'])
      ifFalse: [self error].
   activitiesData do:
      [:indict || outdict |
      outdict := port activityDescriptorFor: (indict at: 'name').
      indict keys = outdict keys ifFalse: [self error: ‘different activities’]
      indict keys do: [:key | (indict at: key) = (outdict at: key) ifFalse: [self error: key]].
```

createDictA

```
   | dict |
   dict := Dictionary new.
   dict at: 'name' put: 'actA'.
   dict at: 'duration' put: 2.
   dict at: 'earlyStart' put: 1.
   dict at: 'earlyFinish' put: 2.
   dict at: 'successors' put: #('actC').
   dict at: 'predecessors' put: #().
   activitiesData add: dict.
```

createDictB

```
    | dict |
    dict := Dictionary new.
    dict at: 'name' put: 'actB'.
    dict at: 'duration' put: 7.
    dict at: 'earlyStart' put: 1.
    dict at: 'earlyFinish' put: 7.
    dict at: 'successors' put: #('actD').
    dict at: 'predecessors' put: #().
    activitiesData add: dict.
```

createDictC

```
    | dict |
    dict := Dictionary new.
    dict at: 'name' put: 'actC'.
    dict at: 'duration' put: 3.
    dict at: 'earlyStart' put: 3.
    dict at: 'earlyFinish' put: 5.
    dict at: 'successors' put: #('actD').
    dict at: 'predecessors' put: #('actA').
    activitiesData add: dict.
```

createDictD

```
    | dict |
    dict := Dictionary new.
    dict at: 'name' put: 'actD'.
    dict at: 'duration' put: 2.
    dict at: 'successors' put: #().
    dict at: 'predecessors' put: #('actC' 'actB').
    dict at: 'earlyStart' put: 8.
    dict at: 'earlyFinish' put: 9.
activitiesData add: dict.
```

initialize

```
    super initialize.    activitiesData := Set new.
    self createDictA.
    self createDictB.
    self createDictC.
    self createDictD.
```

runPort: port

```
    " Define activities. "
    activitiesData do: [:dict | port newActivity: (dict at: 'name') duration: (dict at: 'duration')].
    " Define dependencies. "
    activitiesData do:
        [:dict |
        (dict at: #predecessors) do: [:predNam | port addDependencyFrom: predNam to: (dict at: 'name')].
    " Frontload. "
    port frontload: 1.
```

2 SimpleComponent CODE

class SimplePort

Object subclass: #SimplePort
    instanceVariableNames: 'activities'

SimplePort methodsFor: 'public'

activityDescriptorFor: actNam
```
^(self activityNamed: actNam) activityDescriptor.
```

activityNamesSorted
```
^(activities collect: [:act | act name]) asSortedCollection asArray
```

addDependencyFrom: predNam to: succNam
```
(self activityNamed: predNam) successor: (self activityNamed: succNam)
```

frontload: firstWeek
```
activities do: [:act | act resetFrontload].
(activities select: [:act | act predecessors isEmpty])
    do: [:act | act frontload: firstWeek]
```

newActivity: actNam duration: dur
```
| activity |
activity := SimpleActivity new initialize.
activity name: actNam.
activity duration: dur.
activities add: activity.
```

SimplePort methodsFor: 'private'

activityNamed: actNam
```
^activities detect: [:act | act name = actNam]
```

defaultSymbolColor
```
^Color lightBlue
```

initialize
```
super initialize.
activities := Set new.
```

class SimpleActivity

Object subclass: #SimpleActivity
**   instanceVariableNames: 'name duration earlyStart predecessors successors counter'**

SimpleActivity methodsFor: 'accessors'

duration: dur
```
duration := dur
```

name
```
^name
```

name: aString
```
name := aString
```

predecessor: pred
```
(predecessors includes: pred)
ifFalse:
    [predecessors add: pred.
    pred successor: self].
```

predecessors
```
^predecessors
```

successor: succ
```
(successors includes: succ)
ifFalse:
    [successors add: succ.
    succ predecessor: self].
```

SimpleActivity methodsFor: 'component'

activityDescriptor
```
| dict |
dict := Dictionary new.
dict at: 'name' put: name.
dict at: 'duration' put: duration.
dict at: 'earlyStart' put: earlyStart.
dict at: 'earlyFinish' put: self earlyFinish.
dict at: 'successors' put: (successors collect: [:act | act name]) asArray.
dict at: 'predecessors' put: (predecessors collect: [:act | act name]) asArray.
^dict
```

earlyFinish
```
^earlyStart + duration - 1.
```

frontload: start
```
earlyStart
    ifNil: [counter := 0. earlyStart := start]
    ifNotNil: [earlyStart := (start max: earlyStart)].
counter := counter+1.
counter >= predecessors size
    ifTrue: [successors do: [:succ | succ frontload: self earlyFinish + 1]].
```

resetFrontload
```
earlyStart := nil.
```

SimpleActivity methodsFor: 'private'

initialize
```
super initialize.
name := nil.
duration := nil.
earlyStart := nil.
predecessors := Set new.
successors := Set new.
counter := 0.
```

3 MasteredComponent CODE

MaestroPort class

Object subclass: #MaestroPort
instanceVariableNames: 'maestro publicSelectors'

MaestroPort methodsFor: 'public'

```
doesNotUnderstand: aMessage
    ^(publicSelectors includes: aMessage selector)
        ifTrue: [aMessage sendTo: maestro]
        ifFalse: [super doesNotUnderstand: aMessage]
```

MaestroPort methodsFor: 'private'

```
initialize
    super initialize.
    maestro := MaestroMaestro new initialize.
    publicSelectors := #(activityDescriptorFor: activityNamesSorted addDependencyFrom:to: frontload:
newActivity:duration:)
```

MaestroMaestro class

Object subclass: #MaestroMaestro
instanceVariableNames: 'activities'

MaestroMaestro methodsFor: 'port'

```
activityDescriptorFor: actNam
    ^(self activityNamed: actNam) activityDescriptor.

activityNamesSorted
    ^(activities collect: [:act | act name]) asSortedCollection asArray

addDependencyFrom: predNam to: succNam
    (self activityNamed: predNam) successor: (self activityNamed: succNam)

frontload: firstWeek
    | frontActs |
    activities do: [:act | act resetFrontload].
    [(frontActs := activities select:
        [:act |
            act earlyStart isNil
                and: [act predecessors allSatisfy: [:pred | pred earlyStart notNil]]])
        notEmpty]
    whileTrue:
        [frontActs do: [:act | act frontload: firstWeek] ]

newActivity: actNam duration: dur
    | activity |
    activity := MaestroActivity new initialize.
    activity name: actNam.
    activity duration: dur.
    activities add: activity.
```

MaestroMaestro methodsFor: 'private'

activityNamed: actNam
```
^activities detect: [:act | act name = actNam]
```

initialize
```
super initialize.
activities := Set new.
```

MaestroActivity class

Object subclass: #MaestroActivity
instanceVariableNames: 'name duration earlyStart predecessors successors'

MaestroActivity methodsFor: 'accessors'

duration: dur
```
duration := dur
```

earlyStart
```
^earlyStart
```

name
```
^name
```

name: aString
```
name := aString
```

predecessor: pred
```
(predecessors includes: pred)
ifFalse:
    [predecessors add: pred.
    pred successor: self].
```

predecessors
```
^predecessors
```

successor: succ
```
(successors includes: succ)
ifFalse:
    [successors add: succ.
    succ predecessor: self]
```

MaestroActivity methodsFor: 'component'

activityDescriptor
```
| dict |
dict := Dictionary new.
dict at: 'name' put: name.
dict at: 'duration' put: duration.
dict at: 'earlyStart' put: earlyStart.
dict at: 'earlyFinish' put: self earlyFinish.
dict at: 'successors' put: (successors collect: [:nod | nod name]) asArray.
dict at: 'predecessors' put: (predecessors collect: [:nod | nod name]) asArray.
^dict
```

earlyFinish
```
^earlyStart + duration - 1
```

frontload: start

```
earlyStart := start.
predecessors
    do:
        [:pred |
        earlyStart := earlyStart max: pred earlyFinish + 1
        ]
```

resetFrontload

```
earlyStart := nil.
```

MaestroActivity methodsFor: 'private'

initialize

```
super initialize.
name := nil.
duration := nil.
earlyStart := nil.
predecessors := Set new.
successors := Set new.
```

4 DeclarativeComponent CODE

DeclarativeActivity class

```
Object subclass: #DeclarativeActivity
    instanceVariableNames: 'name duration earlyStart'
```

DeclarativeActivity methodsFor: 'accessors'

duration
```
^duration
```

duration: dur
```
duration := dur
```

earlyFinish
```
^self earlyStart + self duration - 1
```

earlyStart
```
^earlyStart
```

name
```
^name
```

name: aString
```
name := aString
```

DeclarativeActivity methodsFor: 'component'

activityDescriptorInContext: context
```
| dict |
dict := Dictionary new.
dict at: 'name' put: self name.
dict at: 'duration' put: duration.
dict at: 'earlyStart' put: earlyStart.
dict at: 'earlyFinish' put: self earlyFinish.
dict at: 'successors' put: ((context at: 'successors') collect: [:succ | succ name]) asArray.
dict at: 'predecessors' put: ((context at: 'predecessors') collect: [:pred | pred name]) asArray.
^dict
```

frontload: firstWeek inContext: context
```
earlyStart := firstWeek.
(context at: 'frontPreds') do: [:pred | earlyStart := (earlyStart max: (pred earlyFinish + 1))].
```

resetFrontloadInContext: context
```
earlyStart := nil.
```

DeclarativeActivity methodsFor: 'private'

initialize
```
super initialize.
name := nil.
duration := nil.
earlyStart := nil.
```

DeclarativePort class

Object subclass: #DeclarativePort
instanceVariableNames: 'maestro publicSelectors'

DeclarativePort methodsFor: 'public'

doesNotUnderstand: aMessage
```
^(publicSelectors includes: aMessage selector)
    ifTrue: [aMessage sendTo: maestro]
    ifFalse: [super doesNotUnderstand: aMessage]
```

DeclarativePort methodsFor: 'private'

initialize
```
super initialize.
maestro := DeclarativeMaestro new initialize.
publicSelectors := #(activityDescriptorFor: activityNamesSorted addDependencyFrom:to: frontload:
newActivity:duration:).
```

DeclarativeMaestro class

Object subclass: #DeclarativeMaestro
instanceVariableNames: 'base'

DeclarativeMaestro methodsFor: 'network definition'

addDependencyFrom: predNam to: succNam
```
base addAssociationFrom: predNam to: succNam
```

newActivity: actNam duration: dur
```
base insertIntoActivities: #(#name #duration) values: {actNam. dur.
```

DeclarativeMaestro methodsFor: 'network operations'

activityDescriptorFor: actNam
```
| context |
context := base activityDescriptorContext: actNam.
^(context at: ‘activity’) activityDescriptorInContext: context.
```

activityNamesSorted
```
| context |
context := base activitiesContext.
^((context at: 'activities')
    collect: [:act | act name])
        asSortedCollection
```

frontload: firstWeek
```
| context |
context := base activitiesContext.
(context at: 'activities') do:
    [:act | act resetFrontloadInContext: context ].

[    context := base frontContext.
    (context at: 'frontAct') notNil ]
whileTrue:
    [ (context at: 'frontAct') frontload: firstWeek inContext: context. ].
```

DeclarativeMaestro methodsFor: 'private'

initialize

```
super initialize.
base := DeclarativeBase new initialize.
```

DeclarativeBase class

```
Object subclass: #DeclarativeBase
    instanceVariableNames: 'activities associations'
```

DeclarativeBase methodsFor: 'data definition'

addAssociationFrom: predNam to: succNam

```
(associations detect: [:assoc | assoc pred name = predNam and: [assoc succ name = succNam]] ifNone: [nil])
ifNil:
    [associations add:
        (DeclarativeBaseAssociation new initialize
            pred: (self activityNamed: predNam)
            succ: (self activityNamed: succNam)
        )]
```

insertIntoActivities: keyNameArray values: valueArray

```
| act |
act := DeclarativeActivity new initialize.
1 to: keyNameArray size do:
    [:ind |
    act
        perform: ((keyNameArray at: ind) , ':') asSymbol
        withArguments: {valueArray at: ind}
    ].
activities add: act.
```

DeclarativeBase methodsFor: 'external schemas'

activitiesContext

```
| dict |
dict := Dictionary new.
dict at: 'activities' put: activities.
^dict
```

activityDescriptorContext: actNam

```
| dict act |
act := self activityNamed: actNam.
dict := Dictionary new.
dict at: 'activity' put: act.
dict at: 'predecessors' put: (self predecessorsOf: act).
dict at: 'successors' put: (self successorsOf: act).
^dict
```

frontContext

```
| frontActs cntx thisAct |
frontActs := activities select:
   [    :act |
     act earlyStart isNil
     and: [(self predecessorsOf: act) noneSatisfy: [:pred | pred earlyStart isNil]].
   ].
cntx := Dictionary new.
frontActs
   ifEmpty:
      [cntx at: 'frontAct' put: nil.
      cntx at: 'frontPreds' put: Array new]
   ifNotEmpty:
      [thisAct := frontActs anyOne.
      cntx at: 'frontAct' put: thisAct.
      cntx at: 'frontPreds' put: (self predecessorsOf: thisAct)].
^cntx
```

DeclarativeBase methodsFor: 'private'

activityNamed: actNam

```
^activities detect: [:act | act name = actNam]
```

initialize

```
super initialize.
activities := Set new.
associations := Set new.
```

predecessorsOf: act

```
^associations select: [:assoc | assoc succ = act] thenCollect: [:assoc | assoc pred]
```

successorsOf: act

```
^associations select: [:assoc | assoc pred = act] thenCollect: [:assoc | assoc succ]
```

References

[Blue book] Goldberg and Robson: Smalltalk-80. The language and implementation. Addison-Wesley 1983. ISBN 0-201-11371-6

[Dijkstra] Edsger Dijkstra: *A Discipline of Programming*, 1976

[DOI] *The Digital Object Identifier System.* The International DOI Foundation http://www.doi.org OR UUID?????

[ECOOP-04] Trygve Reenskaug: *Empowering People with BabyUML: A sixth Generation Programming Language.* Opening talk, ECOOP 2004, Oslo. http://heim.ifi.uio.no/~trygver/2004/ECOOP-04/EcoopHandout.pdf

[Eng62] Douglas C. Engelbart: *Augmenting Human Intellect: A Conceptual Framework.* Summary Report AFOSR-3223 under Contract AF 49(638)-1024, SRI Project 3578 for Air Force Office of Scientific Research, Stanford Research Institute, Menlo Park, Ca., October 1962:

[GOF] Gamma et.al.: *Design Patterns.* Addison Wesley1995

[Hay-03] David Hay: *What Exactly IS a Data Model?* DM Review Magazine, February 2003 Issue

[Hy-60] T. Hysing: On the Use of Numerical Methods in the Design and Manufacture of Ships. Proceedings of the First International Congress of the International Federation of Automatic Control. Moscow, 1960. Butterworths, London.

[Norman88] Donald A. Norman: *The Design of Everyday Things.* Doubleday, 1988

[ODMG] Cattell, Barry: *The Object Data Standard: ODMG 3.0.* Academic Press, London, 2000. ISBN 1-55860-647-4

[OORAM] Trygve Reenskaug: *Working with objects. The OOram Software Engineering Method.* Manning/Prentice Hall 1996. ISBN 0-13-452930-8.
Out of print. Late draft may be downloaded here .PDF
also
Trygve Reenskaug et.al.: *OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems.* JOOP 27-41 (October 1992)

[Pawson04] Richard Pawson: *Naked Objects.* PhD thesis, Trinity College, Dublin, 2004.

[Ree-73] Trygve Reenskaug: *Administrative control in the shipyard.* ICCAS conference, Tokyo, 1973. Scanned by the author July 2003 to http://heim.ifi.uio.no/~trygver/1973/iccas/1973-08-ICCAS.pdf

[Schärli-03] Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz and Andrew Black, *Traits: Composable Units of Behavior*, Proceedings ECOOP 2003 (European Conference on Object-Oriented Programming), LNCS, vol. 2743, Springer Verlag, July 2003, pp. 248—274.

[UML] Unified Modeling Language: Superstructure. Version 2.0. Object Management Group. http://www.omg.org

[Web Services] Web Services Architecture http://www.w3.org/TR/ws-arch/

[Kay77] Kay, Alan, *Microelectronics and the Personal Computer*, Scientific American, September 1977, p. 244.

[Baby] Kulwinder S. Gill: The Manchester Small Scale Experimental Machine-"The Baby" http://www.computer50.org/mark1/new.baby.htm

[Facit] Facit EDB 3 computer and ECM 64 carousell. Some documents are archived at The Norwegian Museum of Science and Technology, Autokon archive, Box #1.

[HyRee-63] Thomas Hysing, Trygve Reenskaug: A system for computer plate preparation. Numerical methods applied to shipbuilding. A NATO Advanced Study Institute organized by Central Institute for Industrial Research and Det norske Veritas. Oslo-Bergen 1963. pp.324ff
A copy of the proceedings is archived at The Norwegian Museum of Science and Technology, Autokon archive, Box #2.

About the author

Trygve Reenskaug, prof.em.,
Department of informatics,
University of Oslo, Norway.

mailto: trygver <at> ifi.uio.no
http://heim.ifi.uio.no/~trygver

5 Slides

How to Implement the BabyUML *Discipline of Programming*

Workshop:
How to do it.

rOOts 2005

Trygve Reenskaug
University of Oslo
http://heim.ifi.uio.no/~trygver
trygver@ifi.uioi.no

Legal Notice

Summary-1: *Deal With Comprehensible Programs*

All Programs

All Meaningful Programs

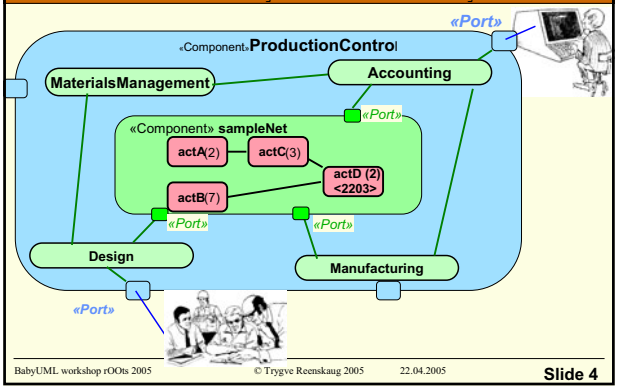
All Comprehensible Programs

Two ways of constructing asoftware design (C.A.R. Hoare)
Make it

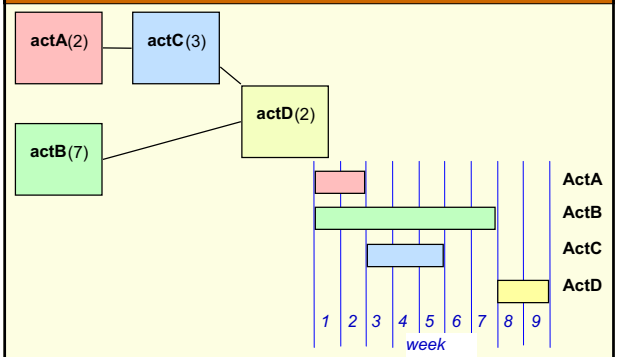
1) so simple that there are obviously no deficiencies

2) so complicated that there are no obvious deficiencies.

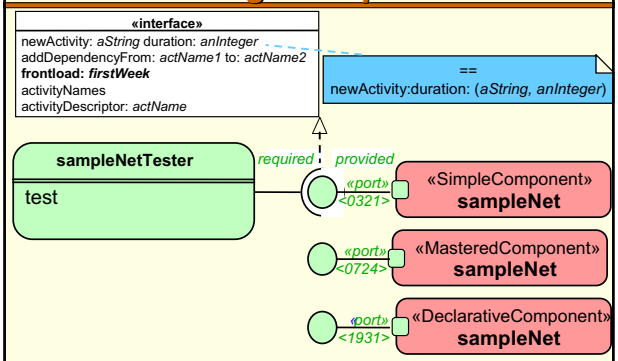
Summary-2: *Divide and Conquer with Components*



Example: *Activity Network Frontloading*



First Example: *Frontloading a simple network*



NetworkTester
class definition

```
Object
   subclass: #NetworkTester
   instanceVariableNames: 'activitiesData simpleComponent
                           maestroComponent declarativeComponent'
```


NetworkTester
initialize

```
initialize
   super initialize.
   activitiesData := Set new.
   self createDictA.
   self createDictB.
   self createDictC.
   self createDictD.
```


NetworkTester
createDictD

```
createDictD
   | dict |
   dict := Dictionary new.
   dict at: 'name' put: 'actD'.
   dict at: 'duration' put: 2.
   dict at: 'successors' put: #().
   dict at: 'predecessors' put: #('actC' 'actB').
   dict at: 'earlyStart' put: 8.
   dict at: 'earlyFinish' put: 9.
   activitiesData add: dict.
```

== at:put: ('name', 'actD')

NetworkTester
test

```
   self initialize.
" Test SimpleComponent. "
   simpleComponent := SimplePort new initialize.
   self runPort: simpleComponent.
   self checkPort: simpleComponent.
" Test MaestroComponent. "
   maestroComponent := MaestroPort new initialize.
   self runPort: maestroComponent.
   self checkPort: simpleComponent.
" Test DeclarativeComponent. "
   declarativeComponent := DeclarativePort new initialize.
   self runPort: declarativeComponent.
   self checkPort: simpleComponent.
" Inspect. "
   self inspect.
```

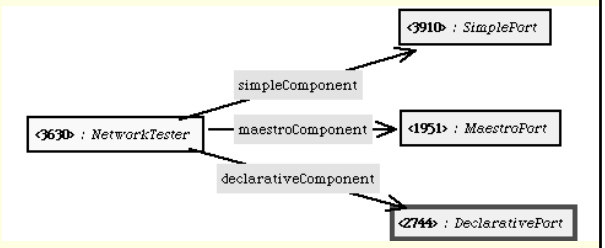

NetworkTester
runPort: *port*

```
" Define activities. "
   activitiesData do:
      [:dict | port newActivity: (dict at: 'name')
                   duration: (dict at: 'duration')].
" Define dependencies. "
   activitiesData do:
      [:actdict |
         (actdict at: #predecessors) do:
            [:predNam |
            port addDependencyFrom: predNam
                 to: (dict at: 'name')].
      ].
" Frontload. "
   port frontload: 1.
```

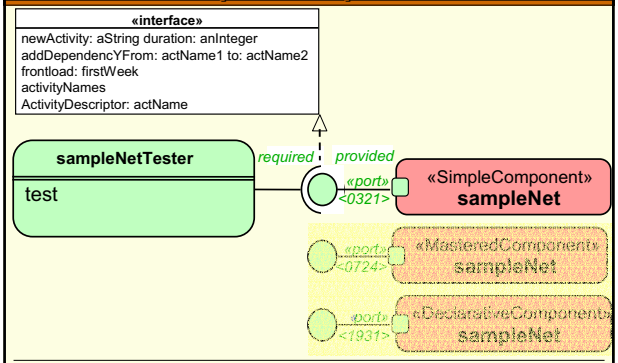

NetworkTester
checkPort: *port*

```
port activityNames
                = (activitiesData collect: [:dict | dict at: 'name'])
   ifFalse: [self error].
activitiesData do:
   [:indict || outdict |
   outdict := port activityDescriptorFor: (indict at: 'name').
   indict keys = outdict keys
            ifFalse: [self error: 'different activities']
   indict keys do:
      [:key | (indict at: key) = (outdict at: key)
               ifFalse: [self error: key]].
```

Object structure NetworkTester



Test SimpleComponent



SimplePort class + frontLoad: *firstWeek*

```
Object subclass: #SimplePort
        instanceVariableNames: 'activities'.

frontload: firstWeek
activities do: [:act | act resetFrontload].
(activities select: [:act | act predecessors isEmpty])
        do: [:act | act frontload: firstWeek]
```

- What are *activities*?
- What are their *predecessors*?
- How does it *frontload:* ?

SimpleActivity *frontload: start*

```
earlyStart
    ifNil: [counter := 0. earlyStart := start]
    ifNotNil: [earlyStart := (start max: earlyStart)].
counter := counter+1.
counter >= predecessors size
    ifTrue: [successors do: [:succ | succ frontload: self earlyFinish + 1]].
```

- What are *activities*?
- What are *predecessors* and *successors* ?
- Do all activities *frontload* the same way ?

SimpleActivity *class definition*

```
Object
        subclass: #SimpleActivity
        instanceVariableNames:
                'name duration earlyStart
                 predecessors successors
                 counter'
```

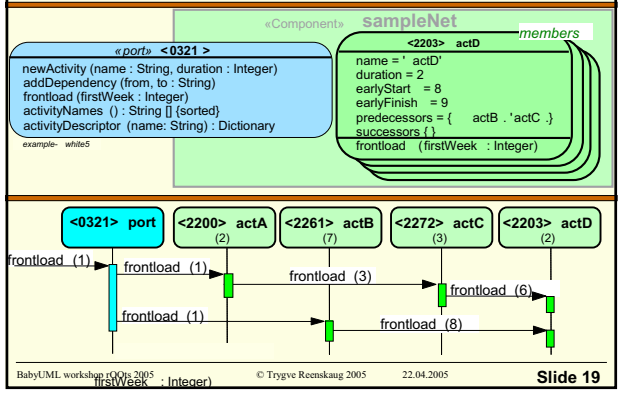
- *What are activities?*
- *What are predecessors and successors ?*
- *Do all activities frontload the same way ?*

SimpleActivity predecessor: *pred*

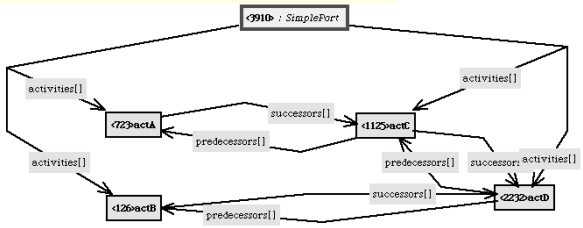
```
predecessor: pred
  (predecessors includes: pred)
    ifFalse:
      [predecessors add: pred.
       pred successor: self
      ].
```

- *What are activities? - OK*
- *What are predecessors and successors - OK*
- *Do all activities frontload the same way ?*
- *What about predecessors/successors integrity ?*

The Simple Component

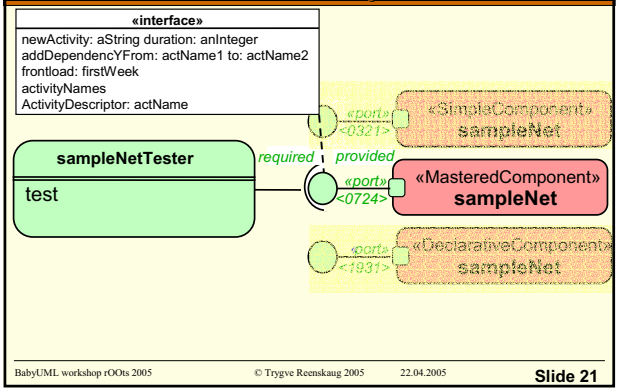


The objects of the SimpleComponent test



Bowl of noodles?

Test Mastered Component



The MasteredComponent

MaestroPort
class + essential methods

```
Object subclass: #MaestroPort
   instanceVariableNames: 'maestro publicSelectors'
```

```
initialize
   super initialize.
   maestro := MaestroMaestro new initialize.
   publicSelectors :=
      #(activityDescriptorFor: activityNamesSorted
        addDependencyFrom:to: frontload: newActivity:duration:
```

```
doesNotUnderstand: aMessage
   ^(publicSelectors includes: aMessage selector)
      ifTrue: [aMessage sendTo: maestro]
      ifFalse: [super doesNotUnderstand: aMessage]
```

- *An implementation of the facade pattern*

MaestroMaestro
frontload: firstWeek

```
| frontActs |
activities do: [:act | act resetFrontload].
[ (frontActs := activities select:
        [:act |
          act earlyStart isNil
          and: [act predecessors
                    allSatisfy: [:pred | pred earlyStart notNil]]
        ] notEmpty
] whileTrue:
        [frontActs do: [:act | act frontload: firstWeek] ]
```

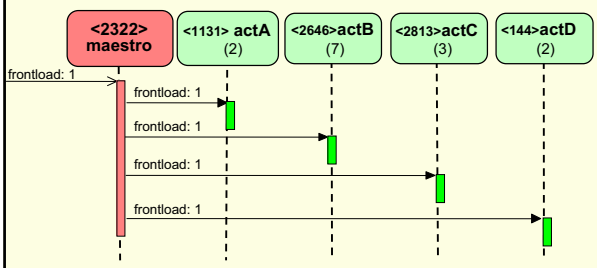
- *xxx?*

MaestroActivity
frontload: start

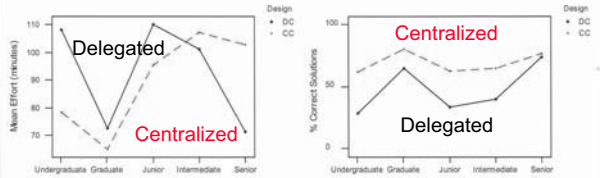
```
frontload: start
   earlyStart := start.
   predecessors
      do:
         [:pred |
          earlyStart := earlyStart max: pred earlyFinish + 1
         ]
```

- *xxx?*

The Mastered Component
Interaction



Resultater fra et kontrollert eksperiment (*)



DC = Delegated Control Style
CC = Centralized Control Style

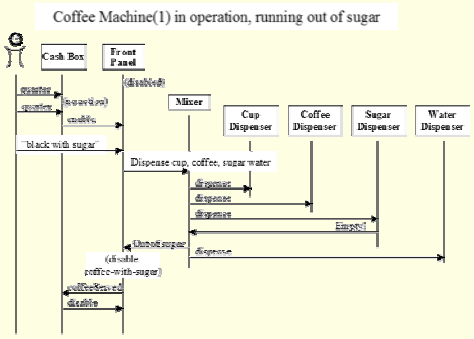
Totalt 158 Java-utviklere deltok, og skulle gjøre endringer på enten et *DC* eller et *CC* design alternativ for det samme systemet.

Vi målte tid ("Mean Effort") og kvalitet ("% correct solutions")

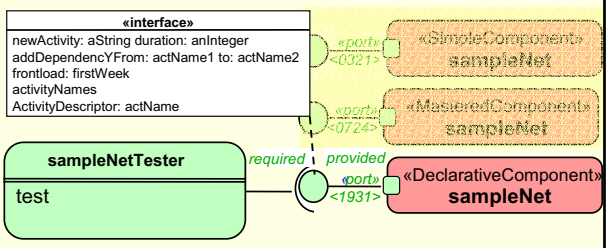
Kun seniorkonsulentene ser ut til å gjøre oppgavene bedre med et *DC* design

* Erik Arisholm and Dag Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions on Software Engineering*, 2004

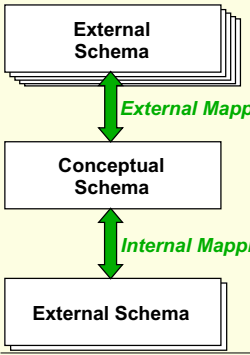
The Coffee Machine Design Problem:
version 1



Test
Declarative Component



Separation of concern
Three Schema Architecture



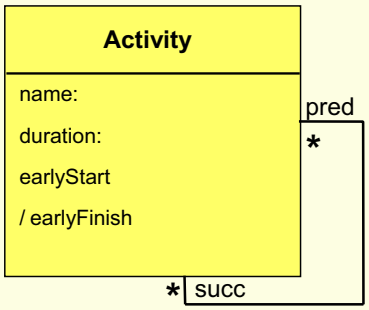
How are they used?

What are the component members?

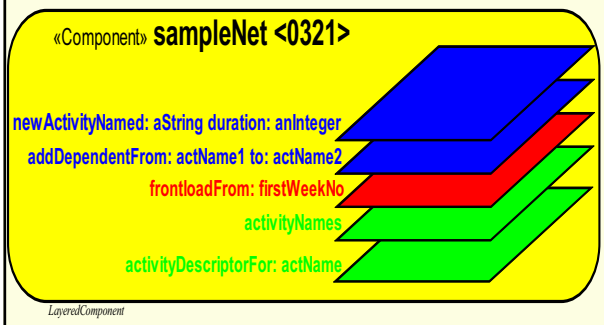
How are they implemented?

Thanks to Ragnar Normann

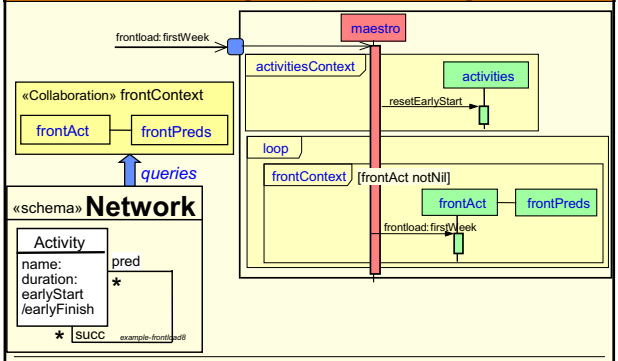
The Declarative Component
Conceptual Schema



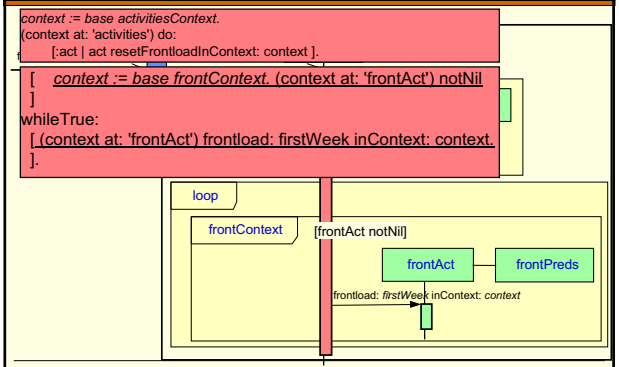
The Declarative Component
Layered – External Schemas



The Declarative Component
frontloading sequence diagram



DeclarativeMaestro
frontload: *firstWeek*



DeclarativeActivity
frontload: *firstWeek* inContext: *context*

```
frontload: firstWeek inContext: context
   earlyStart := firstWeek.
   (context at: 'frontPreds') do:
      [:pred |
      earlyStart := (earlyStart max: (pred earlyFinish + 1))].
```

- *xxx?*

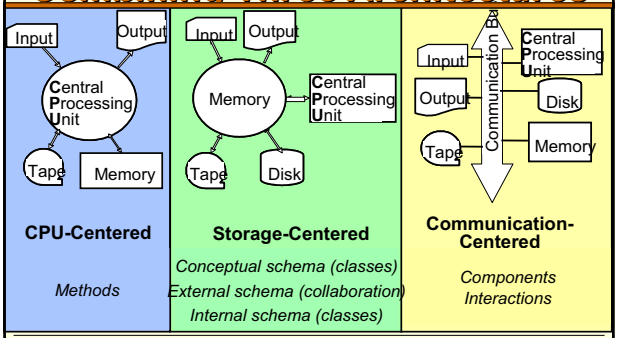
DeclarativeBase
frontContext (ODMG)

```
define query frontActivities as
  select act
  from activities
  where act.earlyStart.isNil
  and
        ( select succ
          from dependencies
          where succ = act
          and pred.earlyStart.isNil)
        ) isEmpty
```

DeclarativeBase
frontContext (Smalltalk)

```
frontActs := activities select:
   [:act |
     act earlyStart isNil
     and: [(self predecessorsOf: act) noneSatisfy: [:pred | pred earlyStart
isNil]].
   ].
cntx := Dictionary new.
frontActs
   ifEmpty:
      [cntx at: 'frontAct' put: nil.
      cntx at: 'frontPreds' put: Array new]
   ifNotEmpty:
      [thisAct := frontActs anyOne.
      cntx at: 'frontAct' put: thisAct.
      cntx at: 'frontPreds' put: (self predecessorsOf: thisAct)].
^cntx
```


Summary:
A Discipline of Programming Combining Three Architectures



CPU-Centered

Methods

Storage-Centered

Conceptual schema (classes)
External schema (collaboration)
Internal schema (classes)

Communication-Centered

Components
Interactions

Thank You

More info at

heim.ifi.uio.no/~trygver

More details ....

http://www.ifi.uio.no/~trygver mailto: **trygver 'at' ifi.uio.no**

Trygve Reenskaug:
Towards a New Discipline of Programming with the BabyUML Language Laboratory.
http://heim.ifi.uio.no/~trygver/2005/babyuml/BabyUML.book.pdf

[UML] *Unified Modeling Language: Superstructure, Version 2.0.*
Object Management Group. Document ptc/04-10-02 or later
http://www.omg.org

Edsger Dijkstra: *A Discipline of Programming*, 1976

Trygve Reenskaug: Empowering People with BabyUML:
A sixth GenerationProgramming Language. Opening talk, ECOOP 2004, Oslo.
http://heim.ifi.uio.no/~trygver/2004/ECOOP-04/EcoopHandout.pdf

Model Driven Architecture
See: *http://www.omg.org/mda/*

Cattell, Barry: *The Object Data Standard: ODMG 3.0.* Academic Press,
London, 2000. ISBN 1-55860-647-4

Erik Arisholm and Dag Sjøberg, *A Controlled Experiment with Professionals to Evaluate the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software,* Simula Research Laboratory Technical Report 2003-6
http://www.simula.no/publication_one.php?publication_id=601

Alistair Cockburn: *The Coffee Machine Design Problem: Part 1,*
http://alistair.cockburn.us/crystal/articles/cmdp1/coffeemachineproblem1.htm