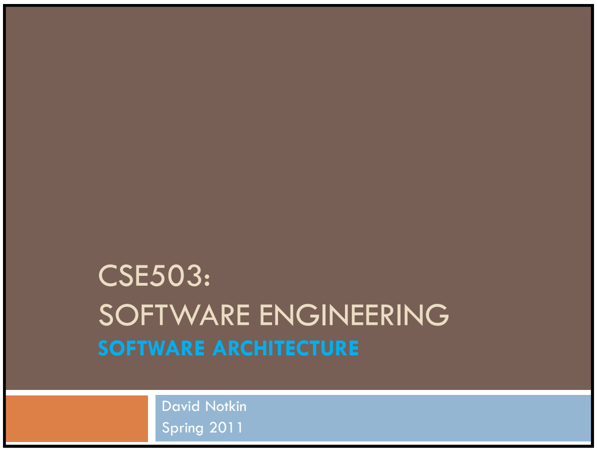
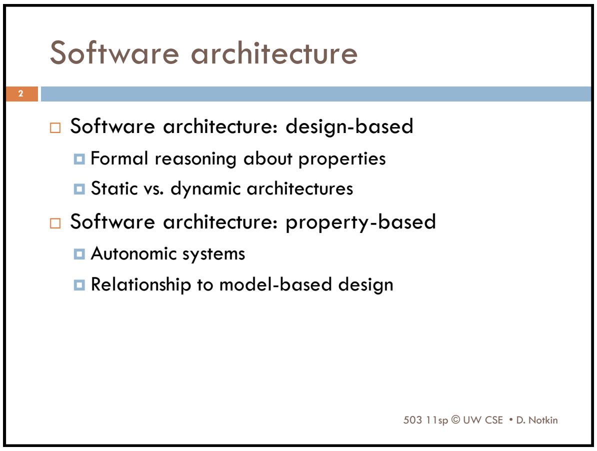


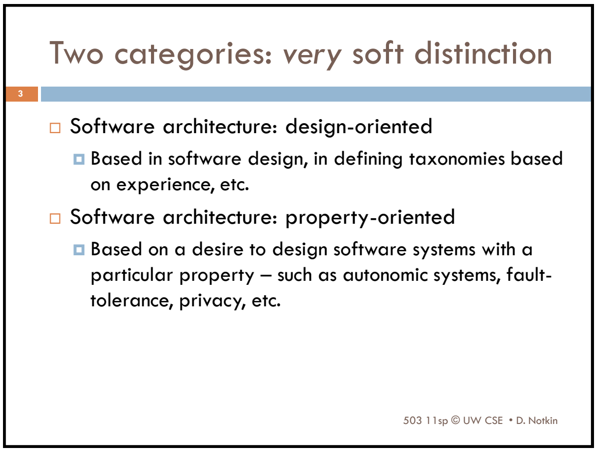
CSE503:
SOFTWARE ENGINEERING
SOFTWARE ARCHITECTURE

David Notkin
Spring 2011



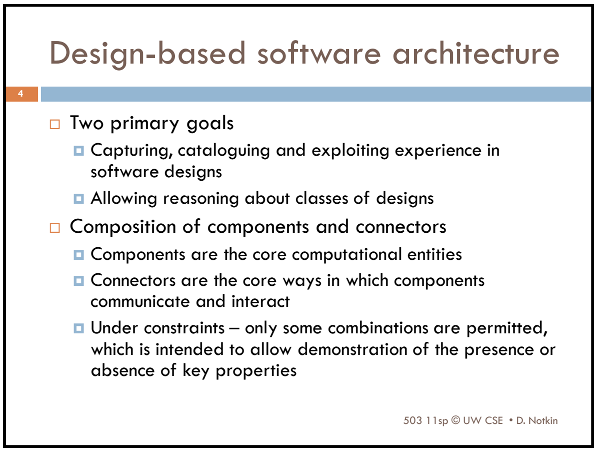
Software architecture

- Software architecture: design-based
  - Formal reasoning about properties
  - Static vs. dynamic architectures
- Software architecture: property-based
  - Autonomic systems
  - Relationship to model-based design



Two categories: *very* soft distinction

- Software architecture: design-oriented
  - Based in software design, in defining taxonomies based on experience, etc.
- Software architecture: property-oriented
  - Based on a desire to design software systems with a particular property – such as autonomic systems, fault-tolerance, privacy, etc.



Design-based software architecture

- Two primary goals
  - Capturing, cataloguing and exploiting experience in software designs
  - Allowing reasoning about classes of designs
- Composition of components and connectors
  - Components are the core computational entities
  - Connectors are the core ways in which components communicate and interact
  - Under constraints – only some combinations are permitted, which is intended to allow demonstration of the presence or absence of key properties

Describing architectures

5

- There are, roughly, two approaches to describing software architectures
- The first – and the most heavily explored – is to define an ADL – architecture description language
- The second is to extend a programming language with architectural constructs

Partial Comparison

ADL

✓ Can focus on architectural issues
✓ Can allow architecture-related analysis
✓ Separates architectural activities from lower-level activities
✗ Separates architecture from software, allowing drift
✗ Requires additional learning and experience by developers, testers, etc.

Extend PL

✓ Provides transition to adopt architecture for existing systems
✓ Connects architecture with program, reducing drift
✓ Incremental cost to train developers, testers, etc.
✗ Fuzzier distinction between architecture and program
✗ May constrain possible analyses

First generation ADLs

- ACME (CMU/USC)
- Rapide (Stanford)
- Wright (CMU)
- Unicon (CMU)
- Aesop (CMU)
- MetaH (Honeywell)
- C2 SADL (UCI)
- SADL (SRI)
- Lileanna
- UML
- Modechart

- From 1999 MCC report
- Much of the following material is adapted from that report

Second generation ADLs

8

- Changes from MCC list with respect to Wikipedia's list (1/9/2010)
- Added
  - LePUS3 and Class-Z (University of Essex)
  - ABACUS (UTS)
  - AADL (SAE) - Architecture Analysis & Design Language
- Removed:
  - UML
  - Modechart

ADL +/-'s

Positives

- Formal representation of architecture
- Higher level system description than previously possible
- Permit analysis of architectures – completeness, consistency, ambiguity, and performance
- Can support automatic generation of software systems

Negatives

- No universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture
- Tend to be very vertically optimized toward a particular kind of analysis

Architecture definition

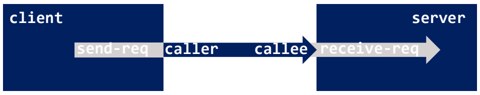
- Components
- Connectors
- Configurations (topologies)
- Constraints (restrictions)

MCC 1999 Report: ACME

- Developed jointly by Monroe, Garlan (CMU), Wile (ISI/USC)
- A general purpose ADL originally designed to be a lowest common denominator interchange language
- Simple, consistent with interchange objective, allowing only syntactic linguistic analysis

MCC report 1999
simple ACME example (client-server)

```
System simple_cs = {
   Component client = {Port send-req}
   Component server = {Port receive-req}
   Connector rpc = {Roles {caller, callee}}
   Attachments :
     {client.send-req to rpc.caller;
      server.receive-req to rpc.callee}
}
```



Very much in the flavor of module interconnection languages

1999 MCC: Rapide

- By Luckham at Stanford
- A general purpose ADL designed with an emphasis on simulation yielding partially ordered sets of events
- Fairly sophisticated, including data types and operations
- Analysis tools focus on posets
  - matching simulation results against patterns of allowed/prohibited behaviors
  - some support for timing analysis
  - focus on causality

Rapide

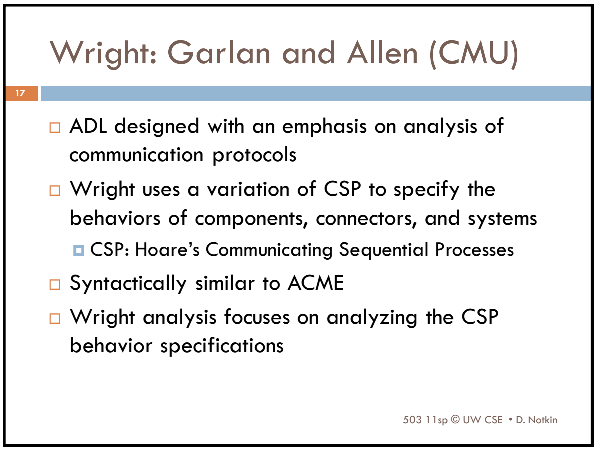
- Components
  - Interface objects
  - Architecture that implements an interface
  - Module that implements an interface
- Connections
  - Connects "sending interfaces" to "receiving interfaces"
  - Components communicate through connections by calling actions or functions in its own interface
  - Events generated by components trigger event pattern connections between their interfaces – basic, pipe, agent

Rapide constraints

- Pattern
  - Bound execution in terms of event patterns
  - Appear in an interface and/or architecture definition
  - `[label] filter_part constraint_body`
  - Filter creates context
  - Constraint body constrains computation in context
- Sequential
  - Bound execution in terms of boolean expressions
  - Normally appear in module level behavior
  - Applied to parameters, types, objects and statements

Rapide example

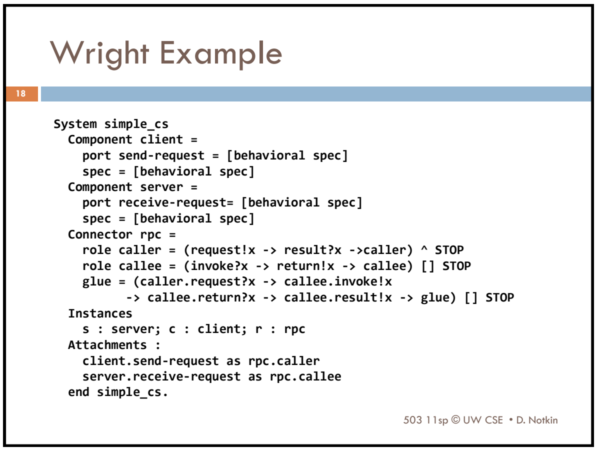
```
type Producer (Max : Positive) is interface
   action out Send (N: Integer);
   action in Reply(N : Integer);
behavior
   Start => send(0);
   (?X in Integer) Reply(?X) where ?X<Max => Send(?X+1);
end Producer;
type Consumer is interface
    action in Receive(N: Integer);
    action out Ack(N : Integer);
behavior
    (?X in Integer) Receive(?X) => Ack(?X);
end Consumer
architecture ProdCon() return SomeType is
     Prod : Producer(100); Cons : Consumer;
connect
    (?n in Integer) Prod.Send(?n) => Cons.Receive(?n);
    Cons.Ack(?n) => Prod.Reply(?n);
end architecture ProdCon;
```



Wright: Garlan and Allen (CMU)

17

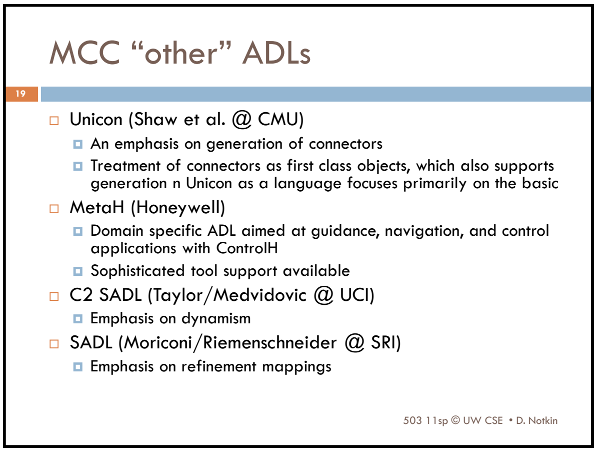
- ADL designed with an emphasis on analysis of communication protocols
- Wright uses a variation of CSP to specify the behaviors of components, connectors, and systems
  - CSP: Hoare's Communicating Sequential Processes
- Syntactically similar to ACME
- Wright analysis focuses on analyzing the CSP behavior specifications



Wright Example

18

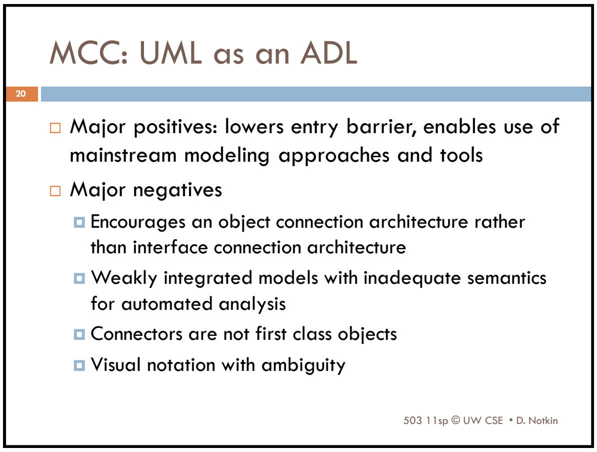
```
System simple_cs
  Component client =
    port send-request = [behavioral spec]
    spec = [behavioral spec]
  Component server =
    port receive-request= [behavioral spec]
    spec = [behavioral spec]
  Connector rpc =
    role caller = (request!x -> result?x ->caller) ^ STOP
    role callee = (invoke?x -> return!x -> callee) [] STOP
    glue = (caller.request?x -> callee.invoke!x
            -> callee.return?x -> callee.result!x -> glue) [] STOP
  Instances
    s : server; c : client; r : rpc
  Attachments :
    client.send-request as rpc.caller
    server.receive-request as rpc.callee
  end simple_cs.
```



MCC "other" ADLs

19

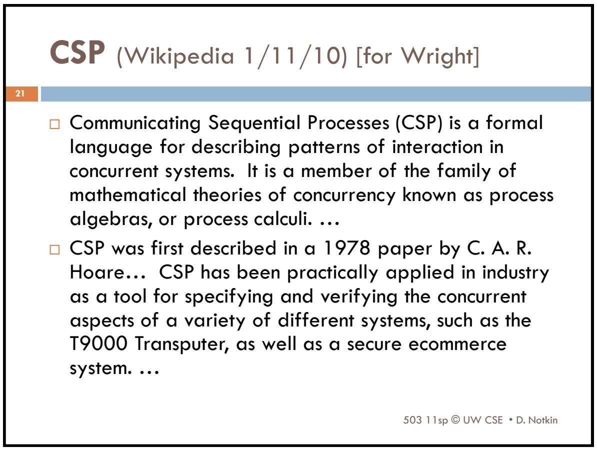
- Unicon (Shaw et al. @ CMU)
  - An emphasis on generation of connectors
  - Treatment of connectors as first class objects, which also supports generation n Unicon as a language focuses primarily on the basic
- MetaH (Honeywell)
  - Domain specific ADL aimed at guidance, navigation, and control applications with ControlH
  - Sophisticated tool support available
- C2 SADL (Taylor/Medvidovic @ UCI)
  - Emphasis on dynamism
- SADL (Moriconi/Riemenschneider @ SRI)
  - Emphasis on refinement mappings



MCC: UML as an ADL

20

- Major positives: lowers entry barrier, enables use of mainstream modeling approaches and tools
- Major negatives
  - Encourages an object connection architecture rather than interface connection architecture
  - Weakly integrated models with inadequate semantics for automated analysis
  - Connectors are not first class objects
  - Visual notation with ambiguity

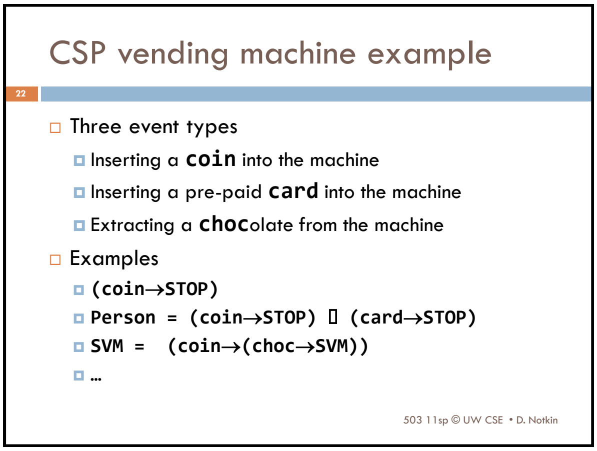


CSP (Wikipedia 1/11/10) [for Wright]

21

- Communicating Sequential Processes (CSP) is a formal language for describing patterns of interaction in concurrent systems. It is a member of the family of mathematical theories of concurrency known as process algebras, or process calculi. …
- CSP was first described in a 1978 paper by C. A. R. Hoare… CSP has been practically applied in industry as a tool for specifying and verifying the concurrent aspects of a variety of different systems, such as the T9000 Transputer, as well as a secure ecommerce system. …

503 11sp © UW CSE • D. Notkin

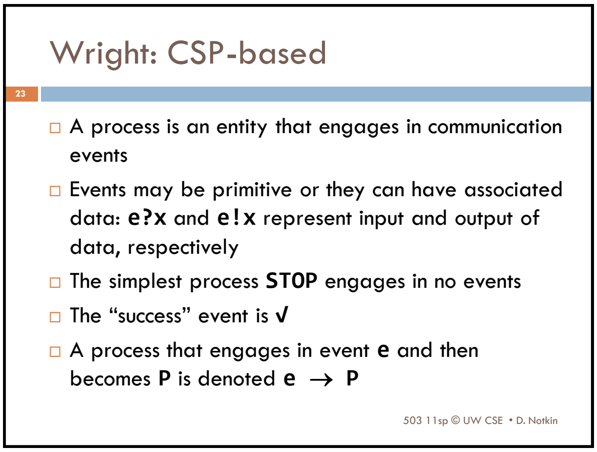


CSP vending machine example

22

- Three event types
  - Inserting a **coin** into the machine
  - Inserting a pre-paid **card** into the machine
  - Extracting a **choc**olate from the machine
- Examples
  - (coin→STOP)
  - Person = (coin→STOP) ▯ (card→STOP)
  - SVM = (coin→(choc→SVM))
  - …

503 11sp © UW CSE • D. Notkin

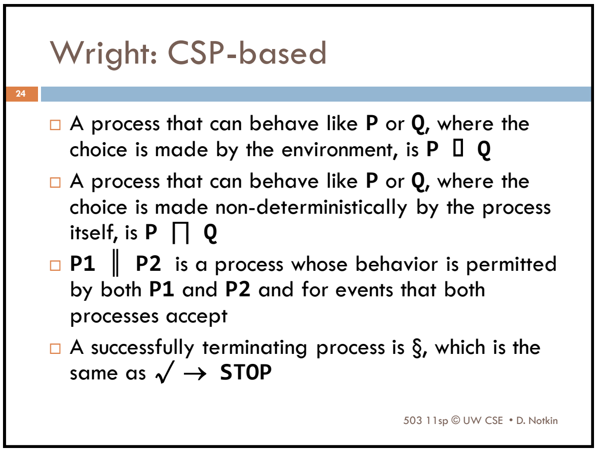


Wright: CSP-based

23

- A process is an entity that engages in communication events
- Events may be primitive or they can have associated data: **e?x** and **e!x** represent input and output of data, respectively
- The simplest process **STOP** engages in no events
- The "success" event is **√**
- A process that engages in event **e** and then becomes **P** is denoted **e** → **P**

503 11sp © UW CSE • D. Notkin

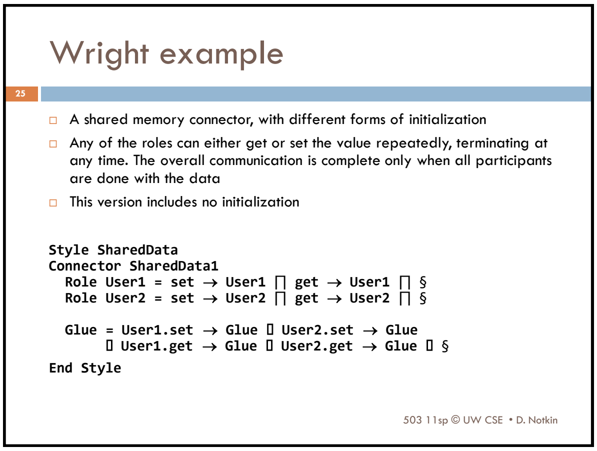


Wright: CSP-based

24

- A process that can behave like **P** or **Q**, where the choice is made by the environment, is **P ▯ Q**
- A process that can behave like **P** or **Q**, where the choice is made non-deterministically by the process itself, is **P** ∏ **Q**
- **P1** ‖ **P2** is a process whose behavior is permitted by both **P1** and **P2** and for events that both processes accept
- A successfully terminating process is §, which is the same as √ → **STOP**

503 11sp © UW CSE • D. Notkin

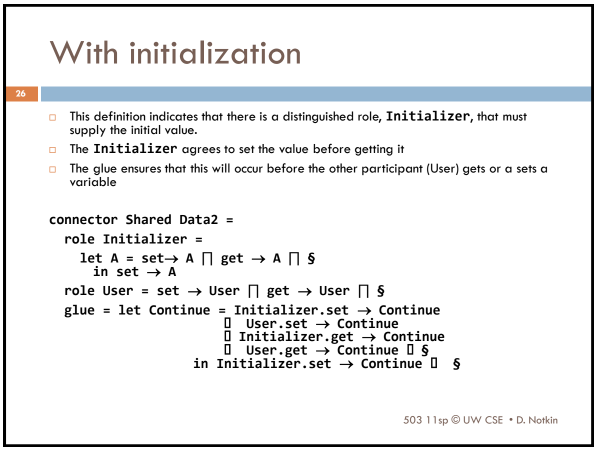


Wright example

25

- A shared memory connector, with different forms of initialization
- Any of the roles can either get or set the value repeatedly, terminating at any time. The overall communication is complete only when all participants are done with the data
- This version includes no initialization

```
Style SharedData
Connector SharedData1
  Role User1 = set → User1 ∏ get → User1 ∏ §
  Role User2 = set → User2 ∏ get → User2 ∏ §

  Glue = User1.set → Glue ▯ User2.set → Glue
       ▯ User1.get → Glue ▯ User2.get → Glue ▯ §
End Style
```

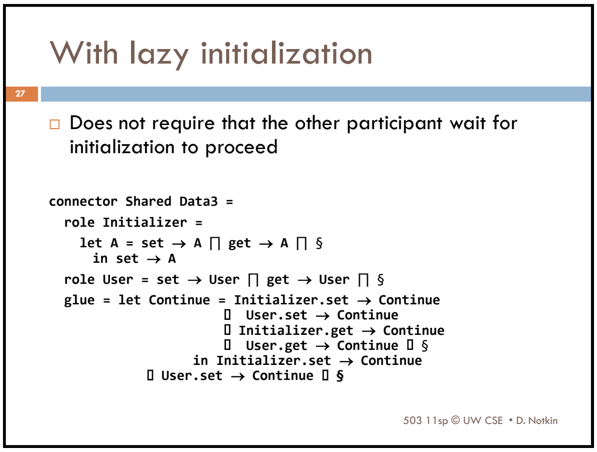


With initialization

26

- This definition indicates that there is a distinguished role, **`Initializer`**, that must supply the initial value.
- The **`Initializer`** agrees to set the value before getting it
- The glue ensures that this will occur before the other participant (User) gets or a sets a variable

```
connector Shared Data2 =
  role Initializer =
    let A = set→ A ∏ get → A ∏ §
      in set → A
  role User = set → User ∏ get → User ∏ §
  glue = let Continue = Initializer.set → Continue
                        ▯  User.set → Continue
                        ▯ Initializer.get → Continue
                        ▯  User.get → Continue ▯ §
                   in Initializer.set → Continue ▯  §
```

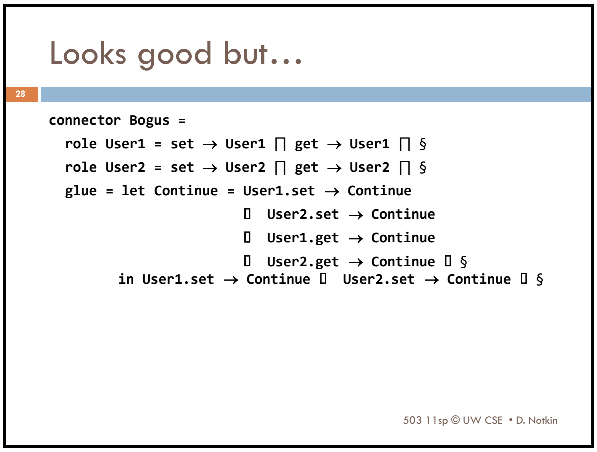


With lazy initialization

27

- Does not require that the other participant wait for initialization to proceed

```
connector Shared Data3 =
  role Initializer =
    let A = set → A ∏ get → A ∏ §
      in set → A
  role User = set → User ∏ get → User ∏ §
  glue = let Continue = Initializer.set → Continue
                      ▯  User.set → Continue
                      ▯ Initializer.get → Continue
                      ▯  User.get → Continue ▯ §
                 in Initializer.set → Continue
             ▯ User.set → Continue ▯ §
```



Looks good but…

28

```
connector Bogus =
  role User1 = set → User1 ∏ get → User1 ∏ §
  role User2 = set → User2 ∏ get → User2 ∏ §
  glue = let Continue = User1.set → Continue
                      ▯  User2.set → Continue
                      ▯  User1.get → Continue
                      ▯  User2.get → Continue ▯ §
        in User1.set → Continue ▯  User2.set → Continue ▯ §
```

Analysis

- □ An analysis of a well-formed system should be able to show that it is deadlock-free
- □ For architectural connectors, the means avoiding the situation in which two components can wait in the middle of an interaction, each port expecting the other to take some action that will never happen
- □ A connector process is free from deadlock if whenever it cannot make progress, then the last event to have taken place must have been √
- □ In other words, the roles and glue work in such a way that if the overall connector process stops, it will be in a situation that is a success state for all the parties

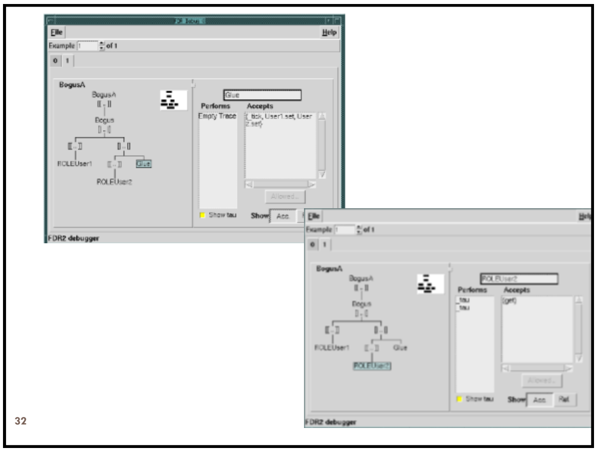
Wright tools

- □ Allow you to assert deadlock-freedom and to have it automatically checked
  - ◘ It converts Wright descriptions into FDR, a commercial model-checker that offers the choice of verification using CSP Traces Refinement, Failures Refinement, and Failures-Divergences Refinement models
- □ Asserts might be, for the shared data example:
  - `? DFA [FD=User1`
  - `? DFA [FD=User2`
  - `? DFA [FD=SharedData1`
- **DFA** means DeadlockFree Process
- **FD** means Failures-Divergences Refinement model
- Returns true if proven, false with counterexample otherwise

Counterexample example

- □ `√ DFA [FD=User1`
- □ `√ DFA [FD=User2`
- □ `X DFA [FD=Bogus`

- □ The connector glue requires that **`User1`** or **`User2`** initialize the variable, but does not specify which one
- □ If either begins with a **`set`**, then that event will occur first and all is OK
- □ But if **`User1`** and **`User2`** both attempt to perform an initial **`get`** – which is entirely legal – then the connector will deadlock
- □ The tool identifies a counterexample
  - ◘ The **`Glue`** process is ready to accept – **`√,User1.set,User2.set`** while both the **`User1`** and **`User2`** processes will only accept **`get`**

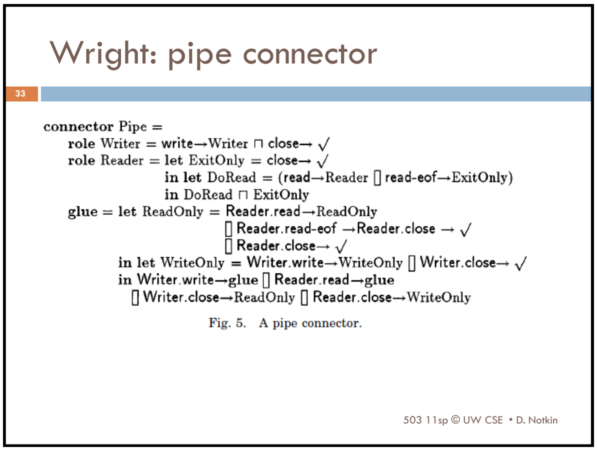


Performs: Empty Trace

Accepts: {_tick, User1.set, User2.set}

Performs: _tau _tau

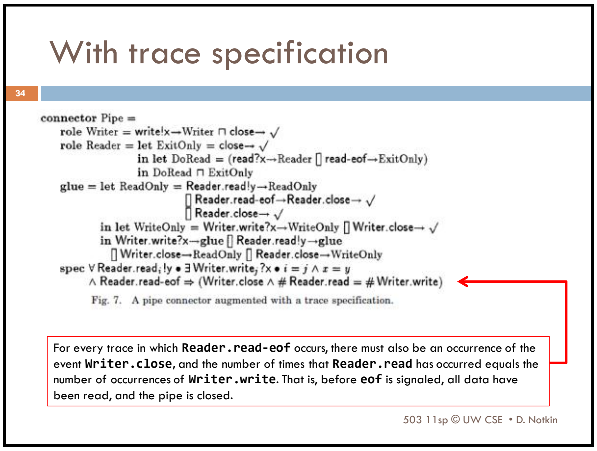
Accepts: {get}



Wright: pipe connector

```
connector Pipe =
    role Writer = write→Writer ⊓ close→ √
    role Reader = let ExitOnly = close→ √
                  in let DoRead = (read→Reader □ read-eof→ExitOnly)
                  in DoRead ⊓ ExitOnly
    glue = let ReadOnly = Reader.read→ReadOnly
                        □ Reader.read-eof →Reader.close → √
                        □ Reader.close→ √
          in let WriteOnly = Writer.write→WriteOnly □ Writer.close→ √
          in Writer.write→glue □ Reader.read→glue
            □ Writer.close→ReadOnly □ Reader.close→WriteOnly
```

Fig. 5. A pipe connector.

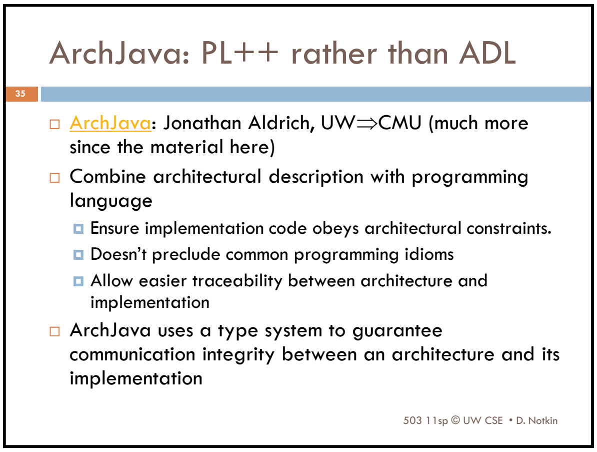


With trace specification

```
connector Pipe =
    role Writer = write!x→Writer ⊓ close→ √
    role Reader = let ExitOnly = close→ √
                  in let DoRead = (read?x→Reader □ read-eof→ExitOnly)
                  in DoRead ⊓ ExitOnly
    glue = let ReadOnly = Reader.read!y→ReadOnly
                        □ Reader.read-eof→Reader.close→ √
                        □ Reader.close→ √
          in let WriteOnly = Writer.write?x→WriteOnly □ Writer.close→ √
          in Writer.write?x→glue □ Reader.read!y→glue
            □ Writer.close→ReadOnly □ Reader.close→WriteOnly
    spec ∀ Reader.read_i!y • ∃ Writer.write_j?x • i = j ∧ x = y
        ∧ Reader.read-eof ⇒ (Writer.close ∧ # Reader.read = # Writer.write)
```

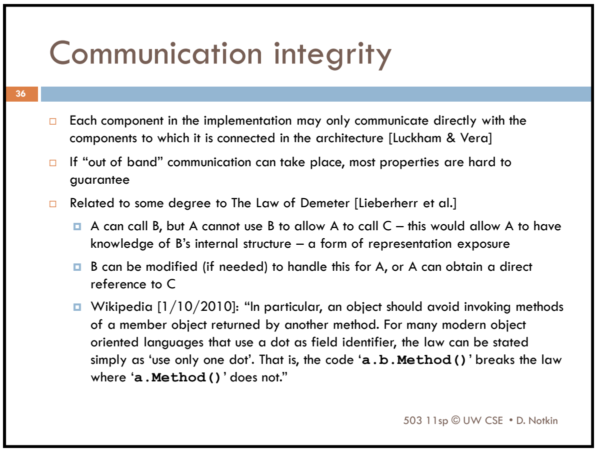
Fig. 7. A pipe connector augmented with a trace specification.

For every trace in which `Reader.read-eof` occurs, there must also be an occurrence of the event `Writer.close`, and the number of times that `Reader.read` has occurred equals the number of occurrences of `Writer.write`. That is, before `eof` is signaled, all data have been read, and the pipe is closed.



ArchJava: PL++ rather than ADL

- ArchJava: Jonathan Aldrich, UW⇒CMU (much more since the material here)
- Combine architectural description with programming language
  - Ensure implementation code obeys architectural constraints.
  - Doesn't preclude common programming idioms
  - Allow easier traceability between architecture and implementation
- ArchJava uses a type system to guarantee communication integrity between an architecture and its implementation



Communication integrity

- Each component in the implementation may only communicate directly with the components to which it is connected in the architecture [Luckham & Vera]
- If "out of band" communication can take place, most properties are hard to guarantee
- Related to some degree to The Law of Demeter [Lieberherr et al.]
  - A can call B, but A cannot use B to allow A to call C – this would allow A to have knowledge of B's internal structure – a form of representation exposure
  - B can be modified (if needed) to handle this for A, or A can obtain a direct reference to C
  - Wikipedia [1/10/2010]: "In particular, an object should avoid invoking methods of a member object returned by another method. For many modern object oriented languages that use a dot as field identifier, the law can be stated simply as 'use only one dot'. That is, the code '`a.b.Method()`' breaks the law where '`a.Method()`' does not."

Component example

37

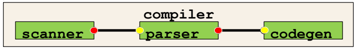
```
public component class Parser {
  public port in {
    provides void setInfo(Token symbol, SymTabEntry e);
    requires Token nextToken() throws ScanException;
  }
  public port out {
    provides SymTabEntry getInfo(Token t);
    requires void compile(AST ast);
  }
  public void parse() {
    Token tok = in.nextToken();
    AST ast = parseFile(tok);
    out.compile(ast);
  }
  AST parseFile(Token lookahead) { ... }
  void setInfo(Token t, SymTabEntry e) {...}
  SymTabEntry getInfo(Token t) { ... }
  ...
}
```

- A component can only communicate with other components at its level in the architecture through explicitly declared ports—regular method calls between components are not allowed
- A port represents a logical communication channel between a component and one or more components that it is connected to

Component example

38

```
public component class Compiler {
  private final Scanner scanner = ...;
  private final Parser parser = ...;
  private final CodeGen codegen = ...;

  connect scanner.out, parser.in;
  connect parser.out, codegen.in;

  public static void main(String args[]) {
    new Compiler().compile(args);
  }
  public void compile(String args[]) {
  // for each file in args do:
   ...parser.parse();...
  }
}
```



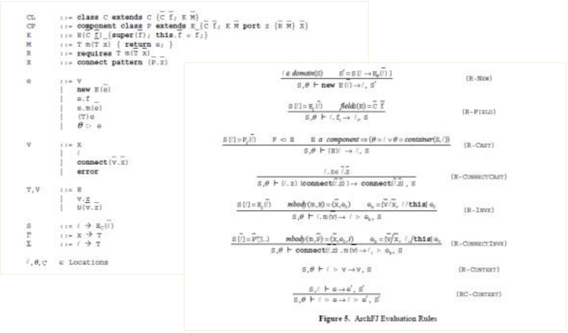
Communication integrity

39

- Ensures that the implementation does not communicate in ways that could violate reasoning about control flow in the architecture
- A component instance A may not call the methods of another component instance B unless
  - B is A's subcomponent, or
  - A and B are sibling subcomponents of a common component instance that declares a connection or connection pattern between them

How does ArchJava work?

40



```
CL      ::= class C extends C {C̄ f̄; K M̄}
CP      ::= component class P extends E {C̄ f̄; K M̄ port z {R̄ M̄} X̄}
K       ::= E(C̄ f̄) {super(f̄); this.f̄ = f̄;}
M       ::= T m(T̄ x̄) { return e; }
R       ::= requires T m(T̄ x̄)
X       ::= connect pattern (P.z̄)

e       ::= v
          | new E(ē)
          | e.f
          | e.m(ē)
          | (T)e
          | θ ▷ e

v       ::= ℓ
          | connect(v̄.z̄)
          | error

T,V     ::= E
          | v.z̄
          | U(v̄.z̄)

S       ::= ℓ → E_C(ℓ̄)
Γ       ::= x → T
Σ       ::= ℓ → T

ℓ,θ,ς   ∈ Locations
```

$$\frac{\ell \notin domain(S) \qquad S' = S[\ell \mapsto E_\theta(\bar{\ell})]}{S,\theta \vdash \mathbf{new}\ E(\bar{\ell}) \to \ell,\ S'} \qquad \text{(R-New)}$$

$$\frac{S[\ell] = E_\varsigma(\bar{\ell}) \qquad fields(E) = \bar{C}\ \bar{f}}{S,\theta \vdash \ell.f_i \to \ell_i,\ S} \qquad \text{(R-Field)}$$

$$\frac{S[\ell] = F_\varsigma(\bar{\ell}) \qquad F <: E \qquad E\ a\ component \Rightarrow (\theta = \ell \vee \theta = container(S,\ell))}{S,\theta \vdash (E)\ell \to \ell,\ S} \qquad \text{(R-Cast)}$$

$$\frac{\ell.z \in \bar{\ell}.\bar{z}}{S,\theta \vdash (\ell.z)\ (\mathbf{connect}(\bar{\ell}.\bar{z})) \to \mathbf{connect}(\bar{\ell}.\bar{z}),\ S} \qquad \text{(R-ConnectCast)}$$

$$\frac{S[\ell] = E_\varsigma(\bar{\ell}) \qquad mbody(m,E) = (\bar{x},e_0) \qquad e_b = [\bar{v}/\bar{x},\ \ell/\mathbf{this}]e_0}{S,\theta \vdash \ell.m(\bar{v}) \to \ell \rhd e_b,\ S} \qquad \text{(R-Invk)}$$

$$\frac{S[\bar{\ell}] = \bar{P}_\varsigma(\ldots) \qquad mbody(m,\bar{P}) = (\bar{x},e_0,\ell) \qquad e_b = [\bar{v}/\bar{x},\ \ell_i/\mathbf{this}]e_0}{S,\theta \vdash \mathbf{connect}(\bar{\ell}.\bar{z}).m(\bar{v}) \to \ell_i \rhd e_b,\ S} \qquad \text{(R-ConnectInvk)}$$

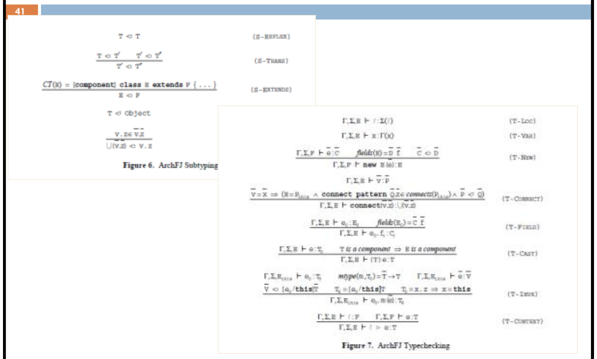
$$S,\theta \vdash \ell \rhd v \to v,\ S \qquad \text{(R-Context)}$$

$$\frac{S,\ell \vdash e \to e',\ S'}{S,\theta \vdash \ell \rhd e \to \ell \rhd e',\ S'} \qquad \text{(RC-Context)}$$

Figure 5. ArchFJ Evaluation Rules

More…

41



$$T <: T \qquad \text{(S-Reflex)}$$

$$\frac{T <: T' \quad T' <: T''}{T <: T''} \qquad \text{(S-Trans)}$$

$$\frac{CT(E) = [\texttt{component}]\ \texttt{class}\ E\ \texttt{extends}\ F\ \{\ \ldots\ \}}{E <: F} \qquad \text{(S-Extends)}$$

$$T <: \texttt{Object}$$

$$\frac{v.z \in \overline{v.z}}{\sqcup(\overline{v.z}) <: v.z}$$

Figure 6. ArchFJ Subtyping

$$\Gamma,\Sigma,E \vdash \ell : \Sigma(\ell) \qquad \text{(T-Loc)}$$

$$\Gamma,\Sigma,E \vdash x : \Gamma(x) \qquad \text{(T-Var)}$$

$$\frac{\Gamma,\Sigma,F \vdash \overline{e} : \overline{C} \quad fields(E) = \overline{D}\ \overline{f} \quad \overline{C} <: \overline{D}}{\Gamma,\Sigma,F \vdash \texttt{new}\ E(\overline{e}) : E} \qquad \text{(T-New)}$$

$$\frac{\begin{array}{c}\Gamma,\Sigma,E \vdash \overline{v} : \overline{P} \\ \overline{v} = \overline{x} \Rightarrow (E = P_{this} \wedge \texttt{connect pattern}\ \overline{Q.z} \in connects(P_{this}) \wedge \overline{P} <: \overline{Q})\end{array}}{\Gamma,\Sigma,E \vdash \texttt{connect}(\overline{v.z}) : \sqcup(\overline{v.z})} \qquad \text{(T-Connect)}$$

$$\frac{\Gamma,\Sigma,E \vdash e_0 : E_0 \quad fields(E_0) = \overline{C}\ \overline{f}}{\Gamma,\Sigma,E \vdash e_0.f_i : C_i} \qquad \text{(T-Field)}$$

$$\frac{\Gamma,\Sigma,E \vdash e : T_0 \quad T\ \textit{is a component} \Rightarrow E\ \textit{is a component}}{\Gamma,\Sigma,E \vdash (T)\,e : T} \qquad \text{(T-Cast)}$$

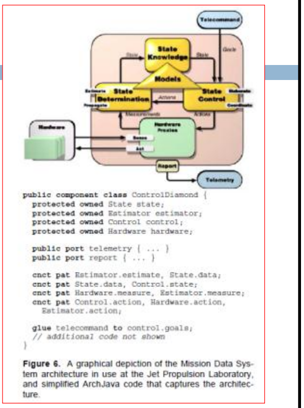
$$\frac{\begin{array}{c}\Gamma,\Sigma,E_{this} \vdash e_0 : T_0 \quad mtype(m,T_0) = \overline{T} \rightarrow T \quad \Gamma,\Sigma,E_{this} \vdash \overline{e} : \overline{V} \\ \overline{V} <: [e_0/\texttt{this}]\overline{T} \quad T_r = [e_0/\texttt{this}]T \quad T_r = x.z \Rightarrow x = \texttt{this}\end{array}}{\Gamma,\Sigma,E_{this} \vdash e_0.m(\overline{e}) : T_r} \qquad \text{(T-Invk)}$$

$$\frac{\Gamma,\Sigma,E \vdash \ell : F \quad \Gamma,\Sigma,F \vdash e : T}{\Gamma,\Sigma,E \vdash \ell \rhd e : T} \qquad \text{(T-Context)}$$

Figure 7. ArchFJ Typechecking

Data sharing

- ArchJava extensions to describe architectural constraints on data sharing (using alias control analysis)
- Can describe data that is confined within a component, passed linearly from one component to another, or shared temporarily or persistently between components.
- Careful use of sophisticated language/type constructs like uniqueness, lending, mutability, etc.



```
public component class ControlDiamond {
  protected owned State state;
  protected owned Estimator estimator;
  protected owned Control control;
  protected owned Hardware hardware;

  public port telemetry { ... }
  public port report { ... }

  cnct pat Estimator.estimate, State.data;
  cnct pat State.data, Control.state;
  cnct pat Hardware.measure, Estimator.measure;
  cnct pat Control.action, Hardware.action,
    Estimator.action;

  glue telecommand to control.goals;
  // additional code not shown
}
```

Figure 6. A graphical depiction of the Mission Data System architecture in use at the Jet Propulsion Laboratory, and simplified ArchJava code that captures the architecture.

503 11sp © UW CSE • D. Notkin

Quick recap

43

- Architectural description via specially designed languages (ADLs) and via programming language extensions
- Reasoning about architectural properties
  - ADLs (like Wright) allow the definition of properties to check within the scope of the language and analysis tools
  - PL++ (like ArchJava) define the properties to always be checked
- Other tradeoffs with respect to adoption, to implementation issues, etc.

503 11sp © UW CSE • D. Notkin

Static vs. dynamic architectures

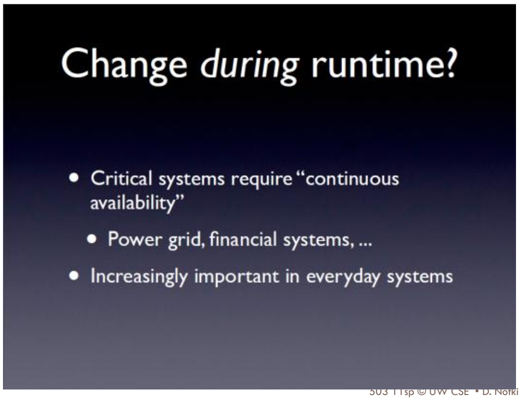
44

- ACME, WRIGHT, etc. define static architectures – essentially, all processes need to be known statically (and thus not created during execution of the implementation)
- The need for dynamic architectures – creation and management of processes at run-time – has been a hot topic for at least a decade (in the software architecture research area, that is)
- Lots of work on this, but I'll focus on
  - Peyman Oreizy, Nenad Medvidovic, Richard N. Taylor. "Architecture-Based Runtime Software Evolution". International Conference on Software Engineering (April 1998)

503 11sp © UW CSE • D. Notkin

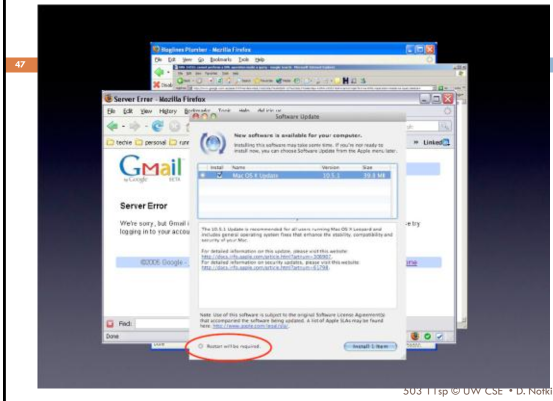
ICSE N-10 award paper

- This paper received the ICSE 2008 Most Influential Paper Award, which recognizes the paper with the most influence on theory or practice during the 10 years since its publication
- The following (partial set of) slides are stolen from the retrospective talk at ICSE 2008 by Peyman, Neno and Dick (http://www.ics.uci.edu/~peymano/dynamic-arch/)



Change *during* runtime?

- Critical systems require "continuous availability"
  - Power grid, financial systems, ...
- Increasingly important in everyday systems


Bloglines Plumber - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Server Error - Mozilla Firefox
File Edit View History Bookmarks Tools Help
techie personal
Linked
GMail by Google BETA
Find:
Done
Software Update
Install
Install 1 Item

Server Error

We're sorry, but Gmail i... logging in to your accou...

©2006 Google -

New software is available for your computer.

Installing this software may take some time. If you're not ready to install now, you can choose Software Update from the Apple menu later.

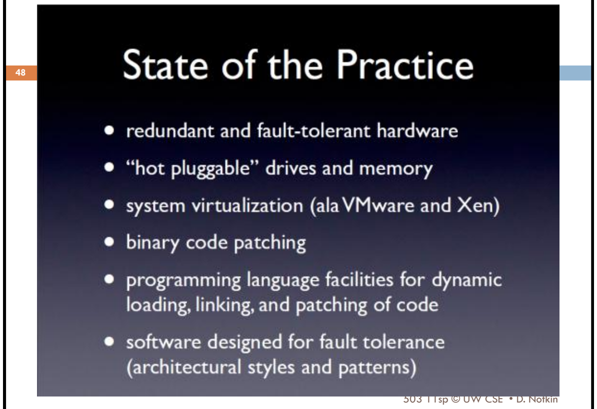
Install	Name	Version	Size
☑	Mac OS X Update	10.5.1	39.3 MB

The 10.5.1 Update is recommended for all users running Mac OS X Leopard and includes general operating system fixes that enhance the stability, compatibility and security of your Mac.

For detailed information on this update, please visit this website: http://docs.info.apple.com/article.html?artnum=306907.
For detailed information on security updates, please visit this website: http://docs.info.apple.com/article.html?artnum=61798.

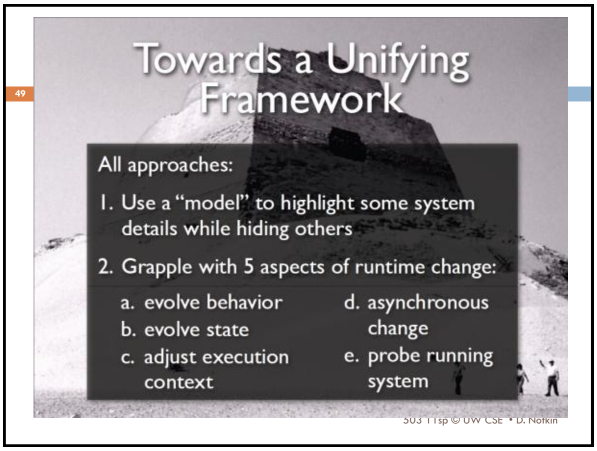
Note: Use of this software is subject to the original Software License Agreement(s) that accompanied the software being updated. A list of Apple SLAs may be found here: http://www.apple.com/legal/sla/.

Restart will be required.



State of the Practice

- redundant and fault-tolerant hardware
- "hot pluggable" drives and memory
- system virtualization (ala VMware and Xen)
- binary code patching
- programming language facilities for dynamic loading, linking, and patching of code
- software designed for fault tolerance (architectural styles and patterns)



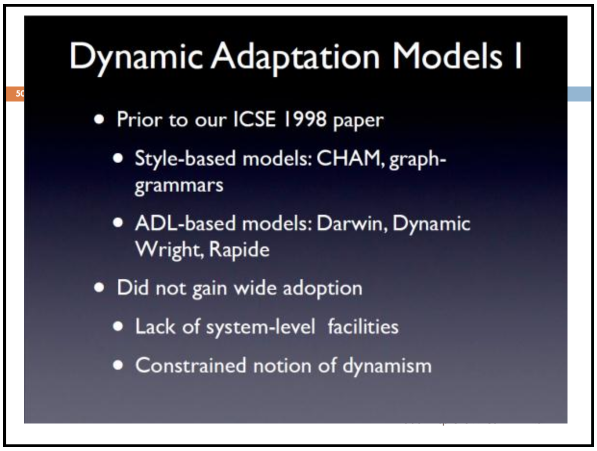
49

Towards a Unifying Framework

All approaches:

1. Use a "model" to highlight some system details while hiding others
2. Grapple with 5 aspects of runtime change:

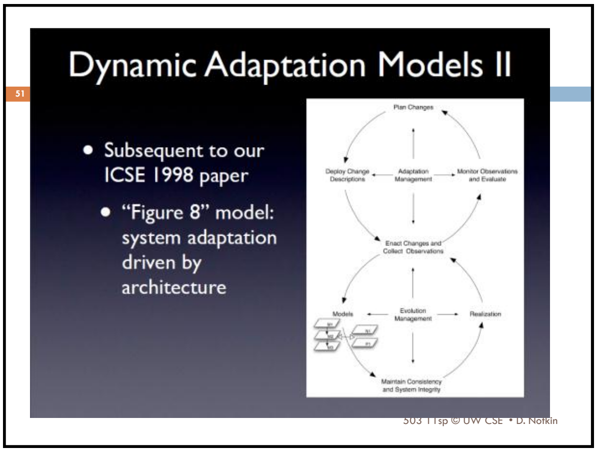
   a. evolve behavior
   b. evolve state
   c. adjust execution context
   d. asynchronous change
   e. probe running system



50

Dynamic Adaptation Models I

- Prior to our ICSE 1998 paper
  - Style-based models: CHAM, graph-grammars
  - ADL-based models: Darwin, Dynamic Wright, Rapide
- Did not gain wide adoption
  - Lack of system-level facilities
  - Constrained notion of dynamism

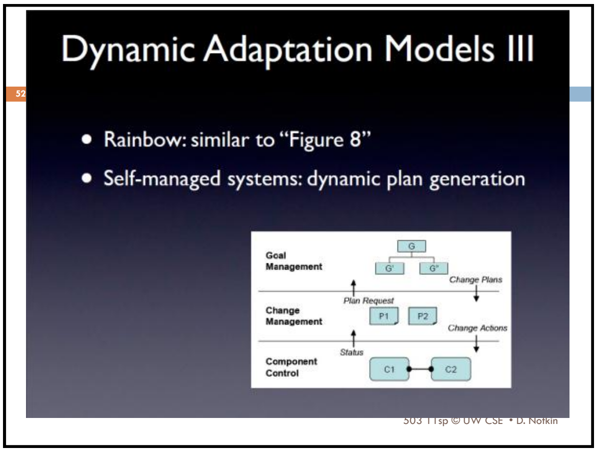


51

Dynamic Adaptation Models II

- Subsequent to our ICSE 1998 paper
  - "Figure 8" model: system adaptation driven by architecture

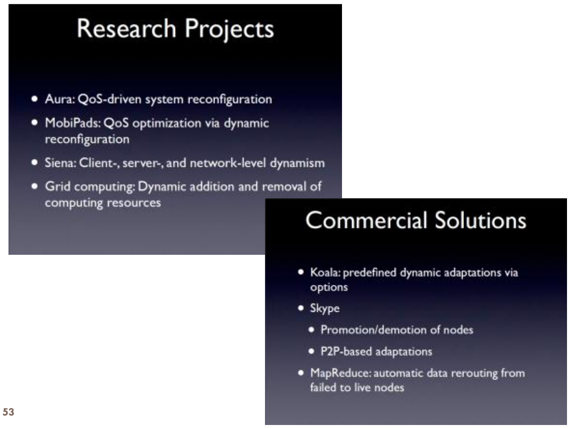
Plan Changes
Deploy Change Descriptions
Adaptation Management
Monitor Observations and Evaluate
Enact Changes and Collect Observations
Models
Evolution Management
Realization
Maintain Consistency and System Integrity



52

Dynamic Adaptation Models III

- Rainbow: similar to "Figure 8"
- Self-managed systems: dynamic plan generation



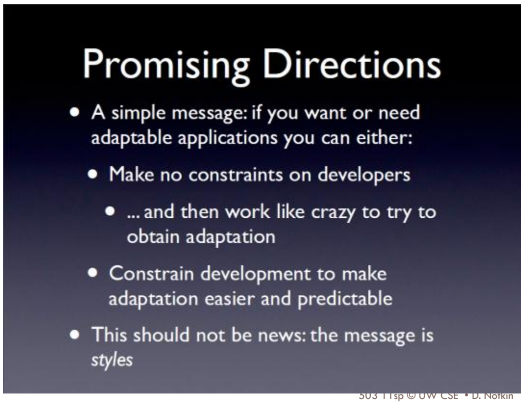
Research Projects

- Aura: QoS-driven system reconfiguration
- MobiPads: QoS optimization via dynamic reconfiguration
- Siena: Client-, server-, and network-level dynamism
- Grid computing: Dynamic addition and removal of computing resources

Commercial Solutions

- Koala: predefined dynamic adaptations via options
- Skype
  - Promotion/demotion of nodes
  - P2P-based adaptations
- MapReduce: automatic data rerouting from failed to live nodes

53

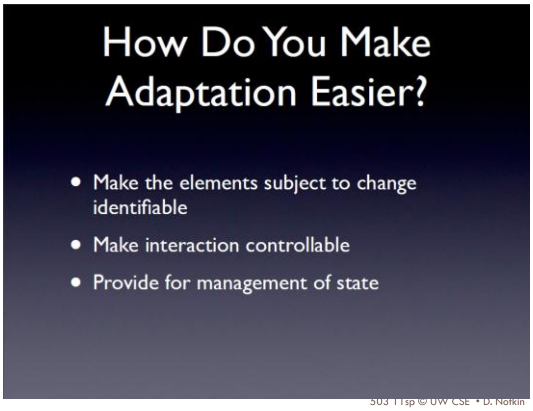


Promising Directions

- A simple message: if you want or need adaptable applications you can either:
  - Make no constraints on developers
    - ... and then work like crazy to try to obtain adaptation
  - Constrain development to make adaptation easier and predictable
- This should not be news: the message is *styles*

503 11sp © UW CSE • D. Notkin

54

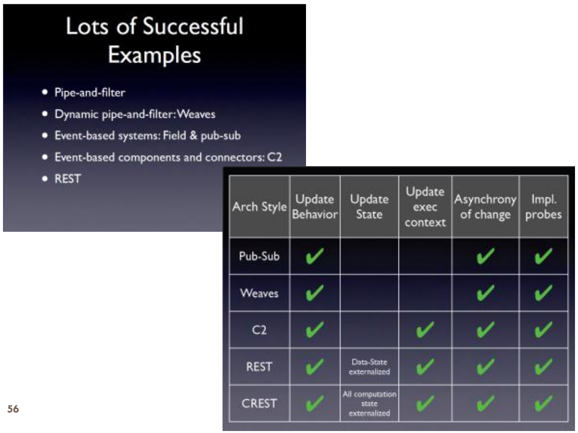


How Do You Make Adaptation Easier?

- Make the elements subject to change identifiable
- Make interaction controllable
- Provide for management of state

503 11sp © UW CSE • D. Notkin

55

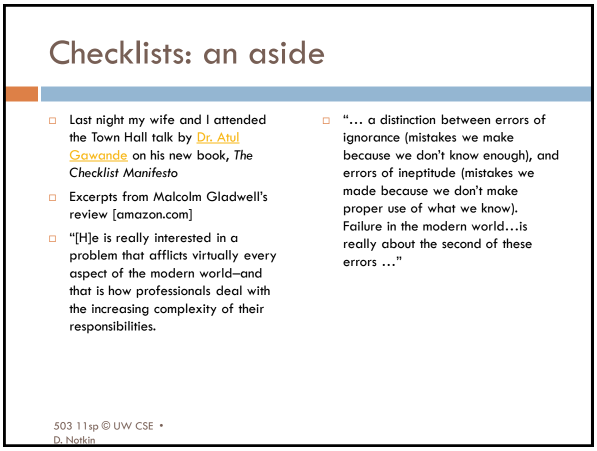


Lots of Successful Examples

- Pipe-and-filter
- Dynamic pipe-and-filter: Weaves
- Event-based systems: Field & pub-sub
- Event-based components and connectors: C2
- REST

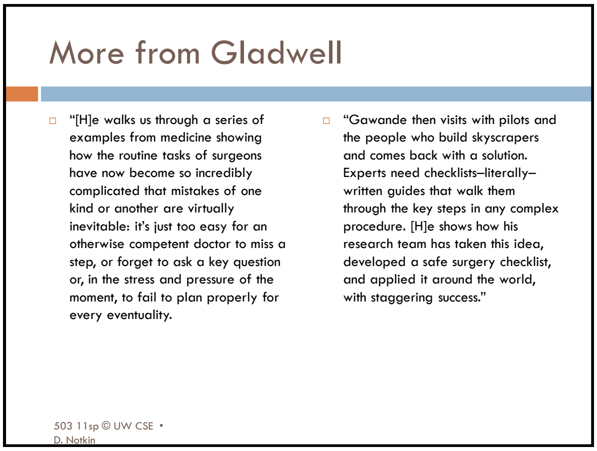
Arch Style	Update Behavior	Update State	Update exec context	Asynchrony of change	Impl. probes
Pub-Sub	✓			✓	✓
Weaves	✓			✓	✓
C2	✓		✓	✓	✓
REST	✓	Data-State externalized	✓	✓	✓
CREST	✓	All computation state externalized	✓	✓	✓

56



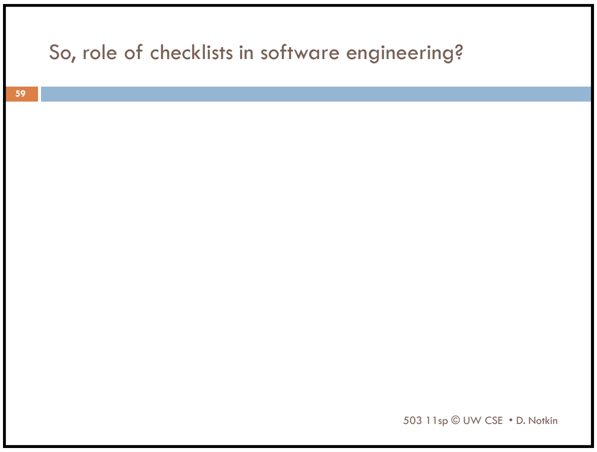
Checklists: an aside

- Last night my wife and I attended the Town Hall talk by Dr. Atul Gawande on his new book, *The Checklist Manifesto*
- Excerpts from Malcolm Gladwell's review [amazon.com]
- "[H]e is really interested in a problem that afflicts virtually every aspect of the modern world–and that is how professionals deal with the increasing complexity of their responsibilities.
- "… a distinction between errors of ignorance (mistakes we make because we don't know enough), and errors of ineptitude (mistakes we made because we don't make proper use of what we know). Failure in the modern world…is really about the second of these errors …"

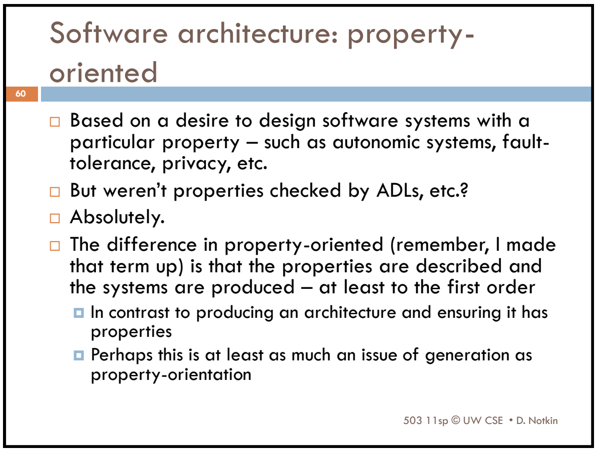


More from Gladwell

- "[H]e walks us through a series of examples from medicine showing how the routine tasks of surgeons have now become so incredibly complicated that mistakes of one kind or another are virtually inevitable: it's just too easy for an otherwise competent doctor to miss a step, or forget to ask a key question or, in the stress and pressure of the moment, to fail to plan properly for every eventuality.
- "Gawande then visits with pilots and the people who build skyscrapers and comes back with a solution. Experts need checklists–literally–written guides that walk them through the key steps in any complex procedure. [H]e shows how his research team has taken this idea, developed a safe surgery checklist, and applied it around the world, with staggering success."



So, role of checklists in software engineering?



Software architecture: property-oriented

- Based on a desire to design software systems with a particular property – such as autonomic systems, fault-tolerance, privacy, etc.
- But weren't properties checked by ADLs, etc.?
- Absolutely.
- The difference in property-oriented (remember, I made that term up) is that the properties are described and the systems are produced – at least to the first order
  - In contrast to producing an architecture and ensuring it has properties
  - Perhaps this is at least as much an issue of generation as property-orientation

Model-based design

- [Wikipedia 1/11/10]: "… is a mathematical and visual method of addressing problems associated with designing complex control systems. … Model-based design is a methodology applied in designing embedded software."
- "The automotive industry has embraced model-based approaches mainly for the following reasons: (i) These graphical functional models visualize both the underlying mathematics (i.e. the differential equations) and the software that later on implements the functions on a specific processor. … (ii) The models can be simulated … very early in the development process. … (iii) The models can be used as a basis for automatic code generation. This not only saves the efforts for the manual coding of the algorithms but also prevents transcription errors from the models to the code."
- Also related to domain specific modeling

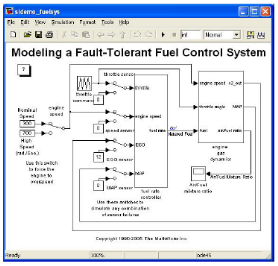
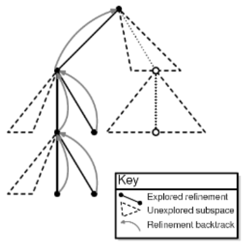


Figure 1: A Simulink model representing a controller. The basic blocks represent atomic mathematical operations like multiplications or differential equations. The model always shows the whole control behavior for a function, thus C code can be generated automatically, e.g. using dSPACE's TargetLink.

Niggemann, O. and Stroop, J. Models for model's sake: why explicit system models are also an end to themselves. *30th international Conference on Software Engineering* (May 2008).

Principle of Alternatives [via E. Jackson]

- A high-level specification defines a design space
- The design space is complex, so some refinements are dead-ends and require backtracking through the design space
- Model-based design (or model integrated computing), provides tool support to simultaneously explore multiple alternatives



http://research.microsoft.com/en-us/um/people/ejackson/publications/asm07_pres.pptx

How to describe the design space?

- UML, UML variants (e.g., Executable UML), etc.
- MatLab/Simulink
- Design-time approximations of embedded system models
- Abstract state machines (E. Jackson et al.)
- Security policies as complex data + invariants
- Model transformations for semantic anchoring and code generation.
- Many more!

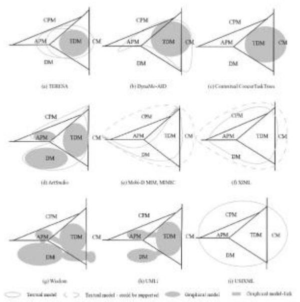
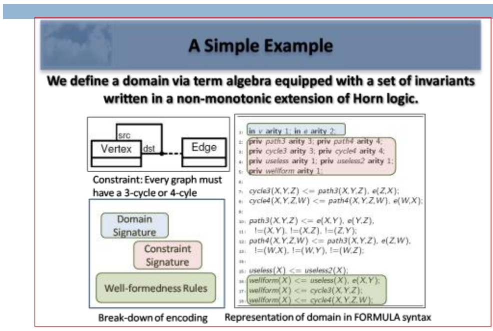


Figure 2 Models used in various model based design approaches

Van den Bergh, J. and Coninx, K. 2004. Model-based design of context-sensitive interactive applications: a discussion of notations. In *Proc.3rd Annual Conference on Task Models and Diagrams* (2004).

E. Jackson: FORMULA (sketch!)



A Simple Example

We define a domain via term algebra equipped with a set of invariants written in a non-monotonic extension of Horn logic.

src
Vertex
dst
Edge
Domain Signature
Constraint Signature
Well-formedness Rules

Constraint: Every graph must have a 3-cycle or 4-cyle

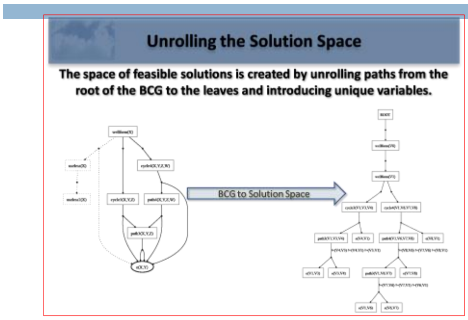
```
in v arity 1; in e arity 2;
priv path3 arity 3; priv path4 arity 4;
priv cycle3 arity 3; priv cycle4 arity 4;
priv useless arity 1; priv useless2 arity 1;
priv wellform arity 1;

cycle3(X,Y,Z) <= path3(X,Y,Z), e(Z,X);
cycle4(X,Y,Z,W) <= path4(X,Y,Z,W), e(W,X);

path3(X,Y,Z) <= e(X,Y), e(Y,Z),
  !=(X,Y), !=(X,Z), !=(Z,Y);
path4(X,Y,Z,W) <= path3(X,Y,Z), e(Z,W),
  !=(W,X), !=(W,Y), !=(W,Z);

useless(X) <= useless2(X);
wellform(X) <= useless(X), e(X,Y);
wellform(X) <= cycle3(X,Y,Z);
wellform(X) <= cycle4(X,Y,Z,W);
```

Break-down of encoding

Representation of domain in FORMULA syntax

E. Jackson con’t

65

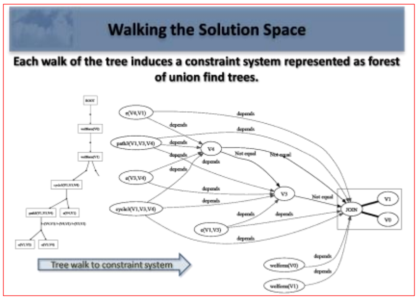


Unrolling the Solution Space

The space of feasible solutions is created by unrolling paths from the root of the BCG to the leaves and introducing unique variables.

E. Jackson con’t

66

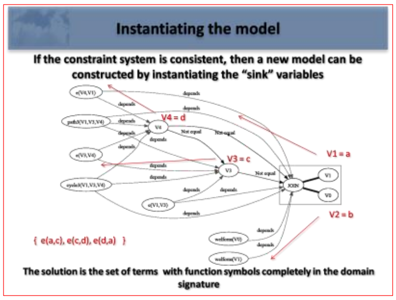


Walking the Solution Space

Each walk of the tree induces a constraint system represented as forest of union find trees.

E. Jackson con’t

67



Instantiating the model

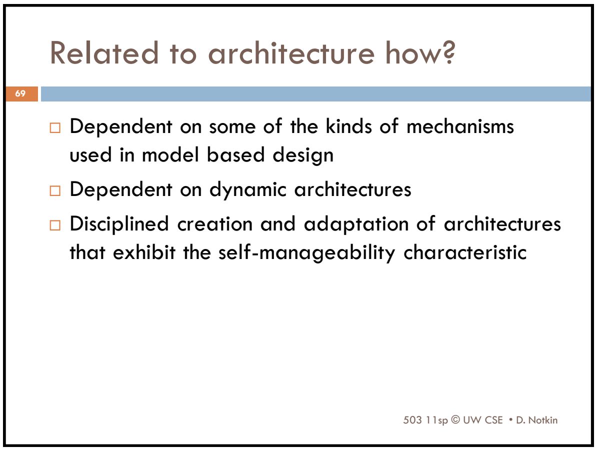
If the constraint system is consistent, then a new model can be constructed by instantiating the “sink” variables

The solution is the set of terms with function symbols completely in the domain signature

Autonomic computing

68

- □ IBM’s term for self-managing and self-adaptive software systems
- □ Systems get more complex, increasing the difficulty and cost of building larger systems in new domains, etc.
- □ Autonomic computing systems are intended to adapt to unpredictable changes in the environment to remove the need for explicit adaption from the users and developers

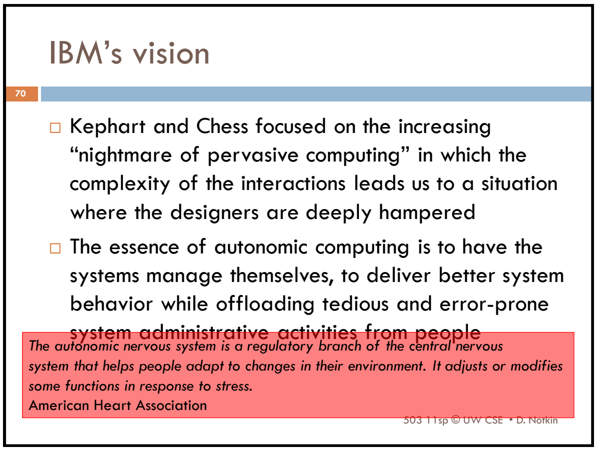


Related to architecture how?

69

- □ Dependent on some of the kinds of mechanisms used in model based design
- □ Dependent on dynamic architectures
- □ Disciplined creation and adaptation of architectures that exhibit the self-manageability characteristic

503 11sp © UW CSE • D. Notkin



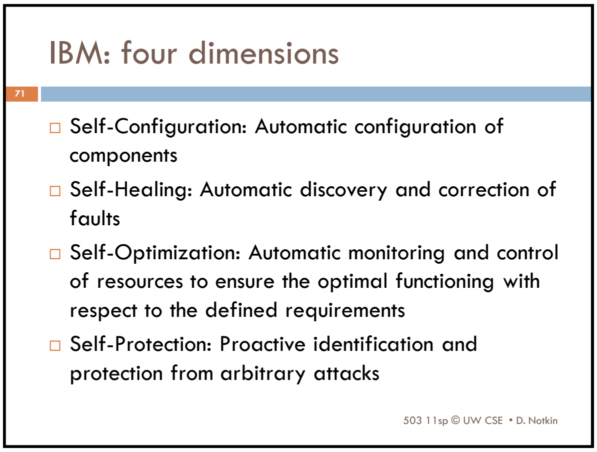
IBM's vision

70

- □ Kephart and Chess focused on the increasing "nightmare of pervasive computing" in which the complexity of the interactions leads us to a situation where the designers are deeply hampered
- □ The essence of autonomic computing is to have the systems manage themselves, to deliver better system behavior while offloading tedious and error-prone system administrative activities from people

The autonomic nervous system is a regulatory branch of the central nervous system that helps people adapt to changes in their environment. It adjusts or modifies some functions in response to stress.
American Heart Association

503 11sp © UW CSE • D. Notkin

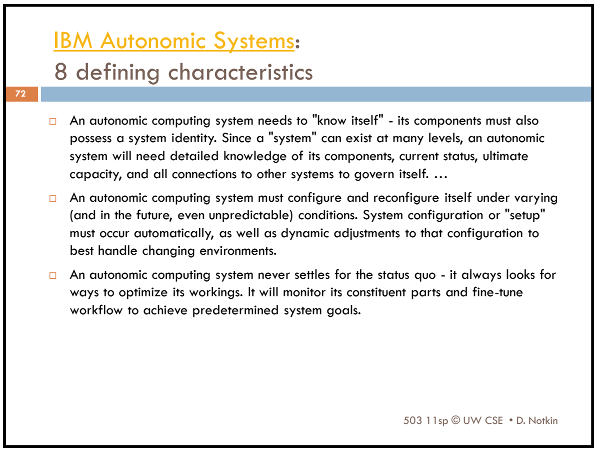


IBM: four dimensions

71

- □ Self-Configuration: Automatic configuration of components
- □ Self-Healing: Automatic discovery and correction of faults
- □ Self-Optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements
- □ Self-Protection: Proactive identification and protection from arbitrary attacks

503 11sp © UW CSE • D. Notkin

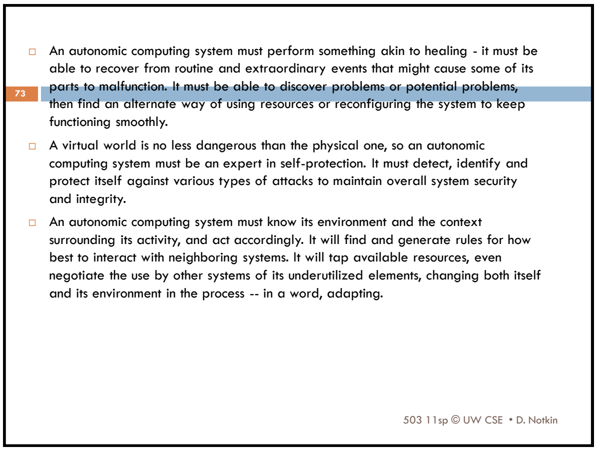


IBM Autonomic Systems: 8 defining characteristics

72

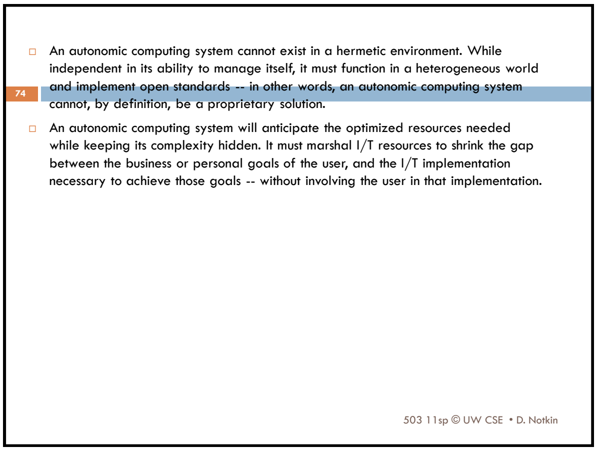
- □ An autonomic computing system needs to "know itself" - its components must also possess a system identity. Since a "system" can exist at many levels, an autonomic system will need detailed knowledge of its components, current status, ultimate capacity, and all connections to other systems to govern itself. …
- □ An autonomic computing system must configure and reconfigure itself under varying (and in the future, even unpredictable) conditions. System configuration or "setup" must occur automatically, as well as dynamic adjustments to that configuration to best handle changing environments.
- □ An autonomic computing system never settles for the status quo - it always looks for ways to optimize its workings. It will monitor its constituent parts and fine-tune workflow to achieve predetermined system goals.

503 11sp © UW CSE • D. Notkin



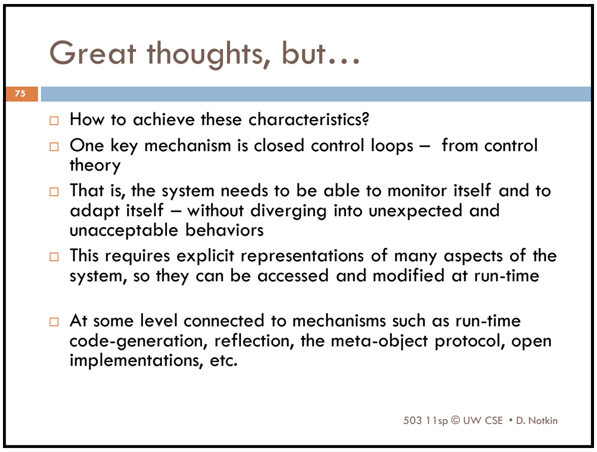
73

- An autonomic computing system must perform something akin to healing - it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction. It must be able to discover problems or potential problems, then find an alternate way of using resources or reconfiguring the system to keep functioning smoothly.
- A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection. It must detect, identify and protect itself against various types of attacks to maintain overall system security and integrity.
- An autonomic computing system must know its environment and the context surrounding its activity, and act accordingly. It will find and generate rules for how best to interact with neighboring systems. It will tap available resources, even negotiate the use by other systems of its underutilized elements, changing both itself and its environment in the process -- in a word, adapting.



74

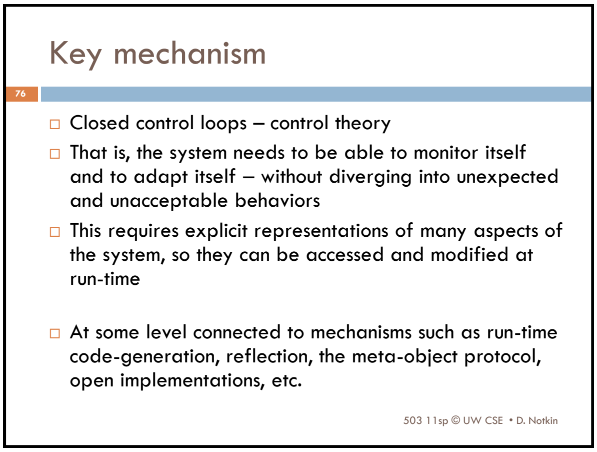
- An autonomic computing system cannot exist in a hermetic environment. While independent in its ability to manage itself, it must function in a heterogeneous world and implement open standards -- in other words, an autonomic computing system cannot, by definition, be a proprietary solution.
- An autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden. It must marshal I/T resources to shrink the gap between the business or personal goals of the user, and the I/T implementation necessary to achieve those goals -- without involving the user in that implementation.



Great thoughts, but…

75

- How to achieve these characteristics?
- One key mechanism is closed control loops – from control theory
- That is, the system needs to be able to monitor itself and to adapt itself – without diverging into unexpected and unacceptable behaviors
- This requires explicit representations of many aspects of the system, so they can be accessed and modified at run-time
- At some level connected to mechanisms such as run-time code-generation, reflection, the meta-object protocol, open implementations, etc.



Key mechanism

76

- Closed control loops – control theory
- That is, the system needs to be able to monitor itself and to adapt itself – without diverging into unexpected and unacceptable behaviors
- This requires explicit representations of many aspects of the system, so they can be accessed and modified at run-time
- At some level connected to mechanisms such as run-time code-generation, reflection, the meta-object protocol, open implementations, etc.

Cheng et al. 2009: Roadmap

77

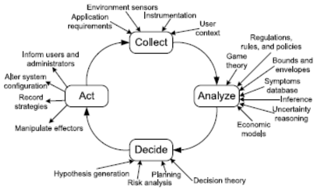


Fig. 1. Activities of the control loop

Alternative mechanisms

78

- Biologically-inspired… stay tuned

Suggestions for third topic…

79

- …after architecture and tools?

Questions?

80