The Role of Event Description in Architecting Dependable Systems

Marcio S. Dias
mdias@ics.uci.edu

Debra J. Richardson
djr@ics.uci.edu

Information and Computer Science
University of California, Irvine

ICSE 2002 – Workshop on Architecting Dependable Systems

The Context:
Architecting Dependable Systems

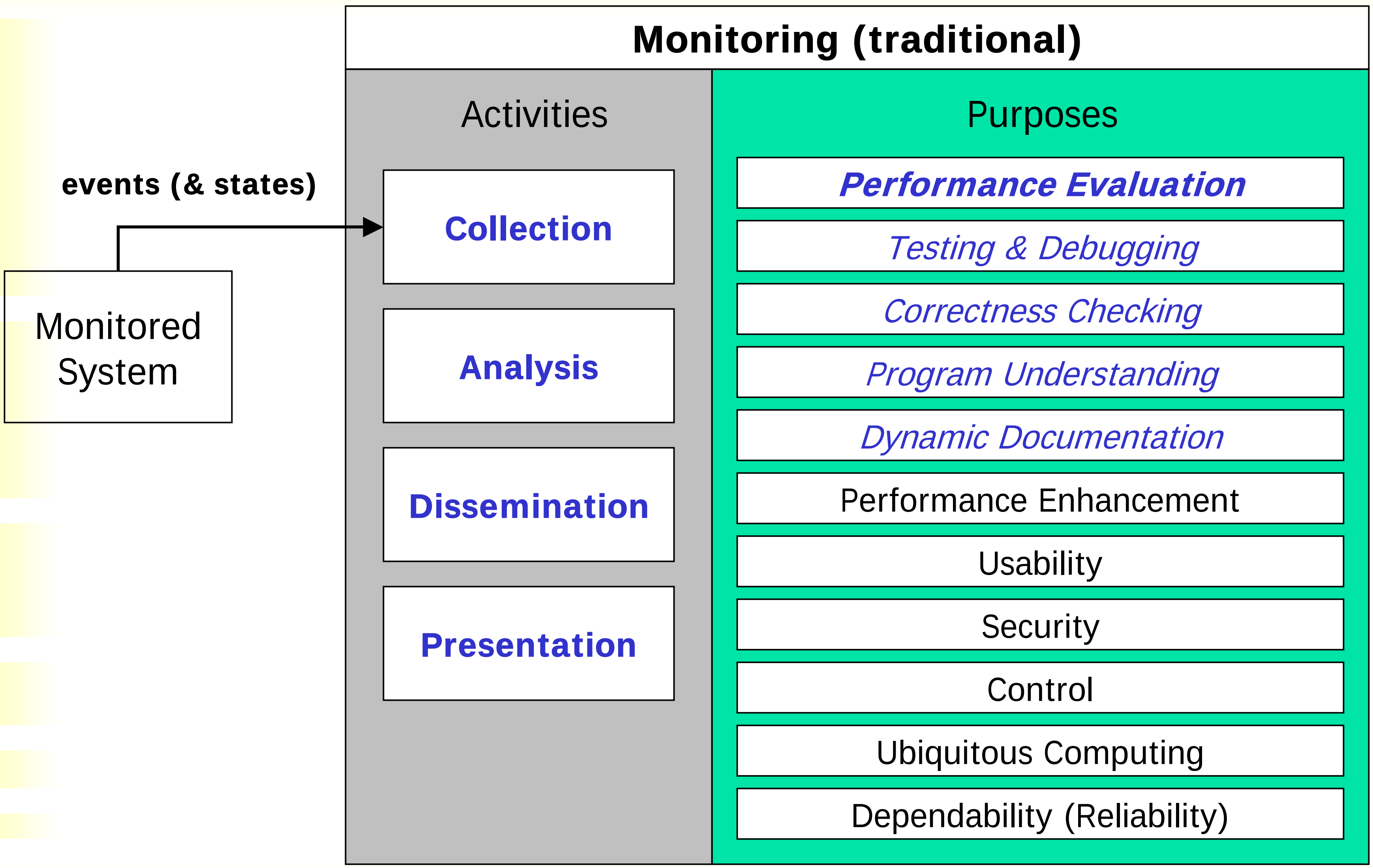
- Software architecture level of abstraction:
  - assists the understanding of broader system concerns
  - helps the developer in dealing with system complexity

- Building dependable systems:
  - higher complexity
  - additional management services required:
    - fault-tolerance and safety
    - as well as: security, resource management, etc

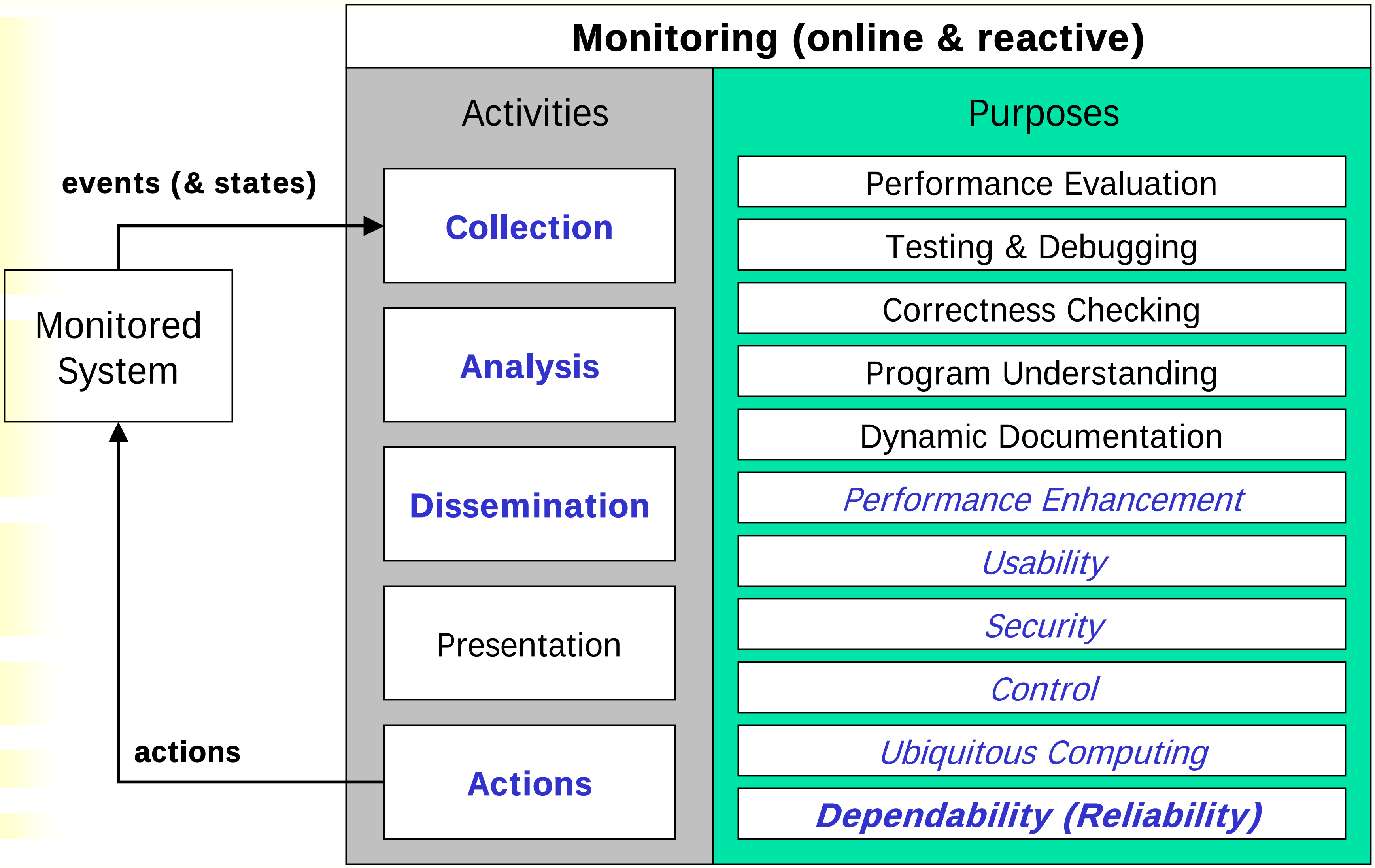
Software Monitoring:
Important underlying support technique

Monitoring - Multi-purpose Technique

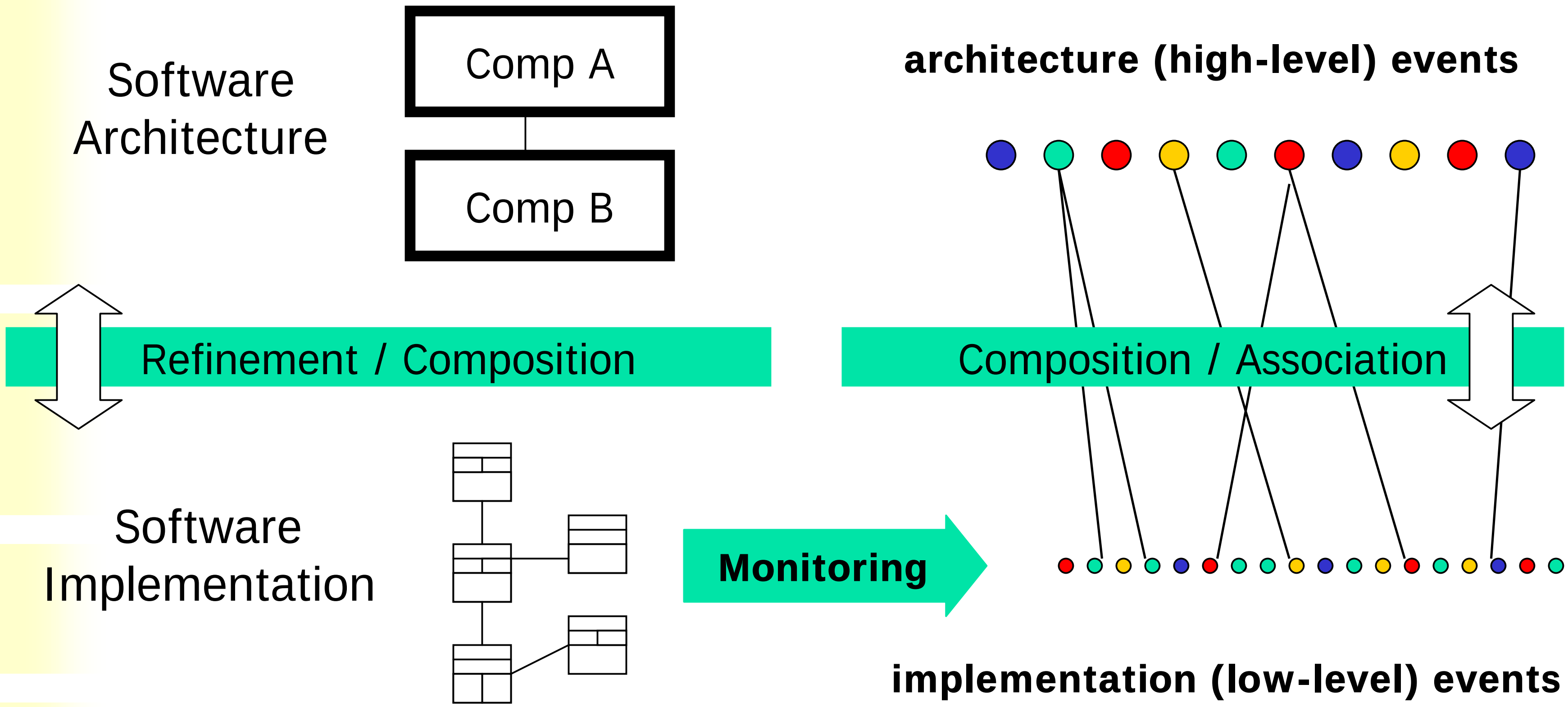
Traditional Monitoring



Monitoring - Multi-purpose Technique
Online (& Reactive) Monitoring

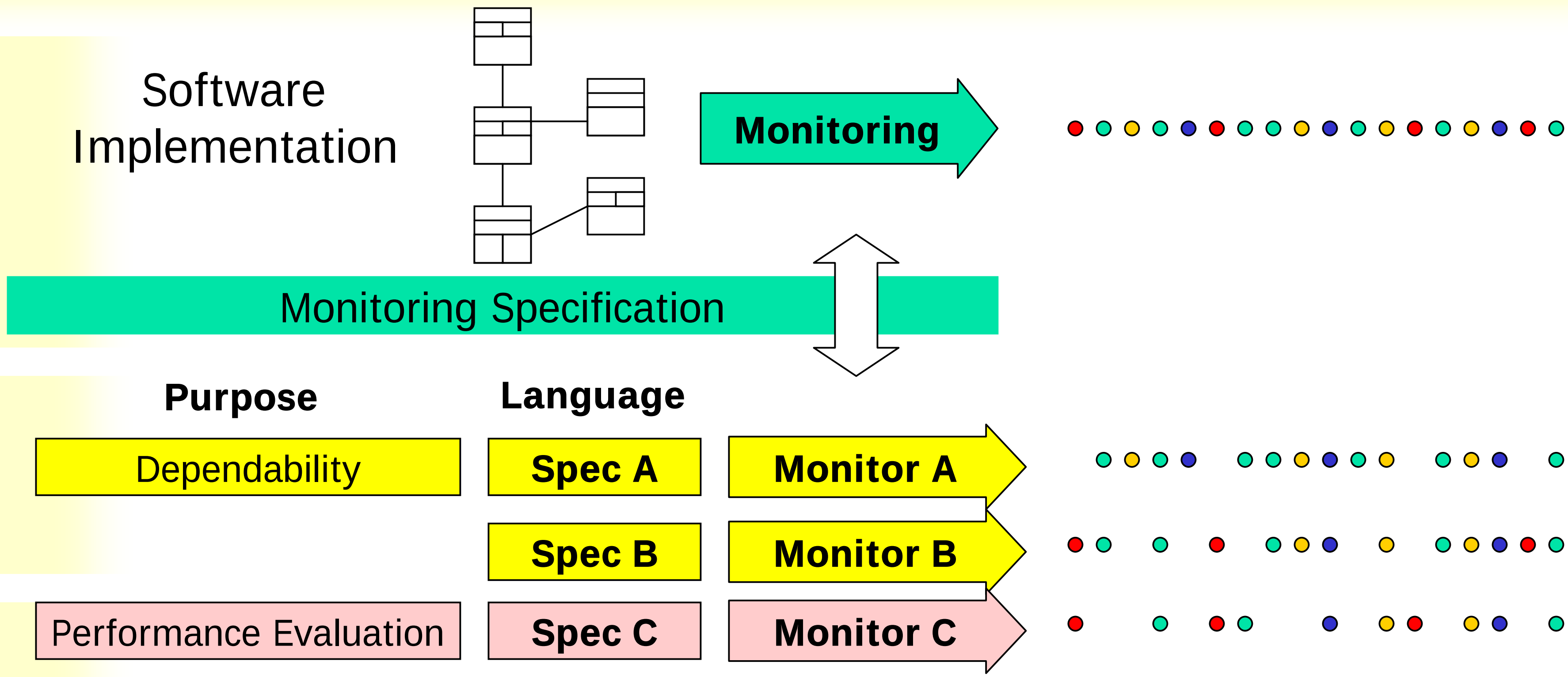


Inherent Gap between Software Architecture and Monitoring



Need to describe how low-level events are related to high-level events

Monitoring Specification Languages



-restricted to monitor system and purpose(s)
-not only events, but also analysis/actions ...
-biased to the analysis performed by monitor
-do not associate monitored events to architecture
-replication of event description

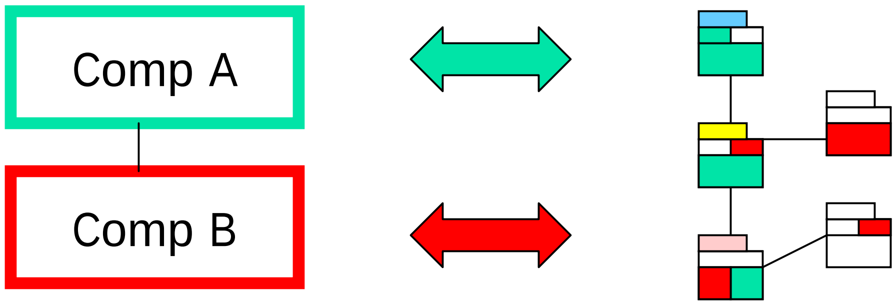
This Paper in a Nutshell

- Software monitoring:
  - supports the development of dependable systems
  - has been widely applied for this purpose
  - does not associate collected data to software architecture
  - provides specification language limited to its purpose
- In the paper we:
  - Discuss the importance of event description:
    - monitoring at the architectural level to support dependability
    - bridging different levels of abstraction
  - Describe requirements for event description languages
  - Present our ongoing work on xMonEve
    - XML-based language for describing monitoring events
    - not to replace, but to integrate monitoring specifications

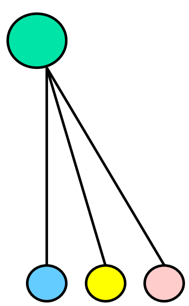
Importance of Event Description

Mapping between Architecture to Implementation

- *Structures* may not correspond (*)



- *Functional* instead of *structural* mapping



Event X from Comp A to Comp B =

Event R from Object1 to Class2

(*Object1 calls Class2.Received*) +

Event S from Object1 to Object3

(*Object1 calls Object3.Send*) +

Event T from Object3 to Object4

(*Object3 calls Object4.Transfer*)

xMonEve
Event Description Language

- Extensible language
- Describe “what” the events are
  - Levels of abstraction:
    - Primitive and Composed events
    - Designer defined “abstraction”
  - Common features:
    - Name / Type / ID ; **Abstraction** ; **Attributes**

```
<event name=open type=primitive ID=#>
  <abstraction>File</abstraction>
  <description>opening file</description>
  <attributes>
    <field name=filename ...>
  </attributes>
  <...>
</event>
```

xMonEve
Primitive vs. Composed Events

- Primitive Events:
  - <mapping>
    - Association of event to implementation
- Composed Events:
  - <composition>
    - Events that compound higher-level event
  - <correlation>
    - Relation between events
      - Boolean expressions; regular expressions; LTL; ...
  - <condition>
    - Restrictions in relation to events attributes

xMonEve
Primitive Event – Example

```
<event name="open" ID=#>
<abstraction>File</abstraction>
<primitive>
  ...
  <mapping>
    <system ref="java_library"/>
    <language name="java"/>
    <class name="java.io.File"/>
    <type name="operation">File(String pathname)</type>
    <when type="method_exit"/>
    <assignments>
      <set>
         <field>filename</field>
         <parameter>pathname</parameter>
      </set>
    </assignments>
  </mapping>
  <...>
</primitive>
</event>
```

Primitive Event

`File.Open [ filename = "test.xml" ]`

on return of constructor call

`... = new java.io.File( "test.xml" );`

xMonEve
Composed Event – Example

```
<event name=AccountTranfer ID=#>
<abstraction>Client</abstraction>
<composite>
  <composition>
    <alias name=before event=Bank.TransferRequest/>
    <alias name=withdraw event=Account.Withdraw/>  ...
  </composition>
  <attributes> ... </attributes>
 <correlation>
    <regexp>
      <sequence min=1 max=1>
        <event alias=before min=1 max=1/>
        <parallel min=1 max=1>
          <event alias=withdraw/>
          <event alias=deposit/>
        </parallel>
        <event alias=after min=1 max=1/>
      </sequence>
    </regexp>
  </correlation>
  <conditions>
    <and> <exp> withdraw.amount = deposit.amount </exp> ...
    </and>
  </condition>
</composite>
</event>
```

Composition

b = Bank.TransferRequest

w = Account.Withdraw

d = Account.Deposit

a = Bank.TransferCommit

Correlation

Regular Expression

b • (w • d | d • w) • a

Conditions

w.amount = d.amount

w.account <> d.account

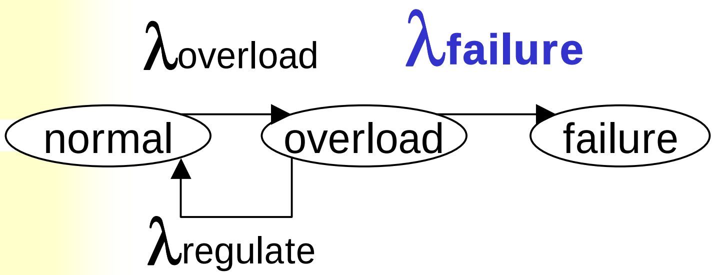
...

Architecting Dependable Systems with xMonEve

- xMonEve
  - independent of the development process
  - events described in *top-down* and *bottom-up* approaches
- Top-Down Example – Component Failure
  - Extension for Markov model
  - Decompose events
- Bottom-Up Example – Component Availability
  - Compose component event from primitive events
  - Associate reliability actions at architecture level

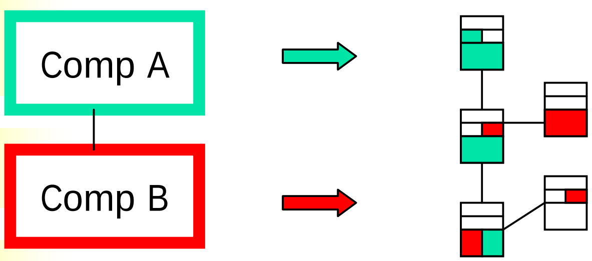
Architecting Dependable Systems
xMonEve – Top-Down Example

- Failure in Component A:
  - Markov Model (for component failure)




```
<event name=failure type=composite ...>
  <abstraction>ComponentA</abstraction>
  <markov_model>
    <transition from="overload" to="failure"/>
    <distribution type="normal" ... />
    <...>
  </markov_model>
</event>
```

  - Architecture to Implementation (classes)

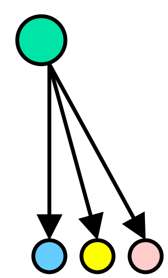


CompA.failure =

any calls Class1.Transmit +

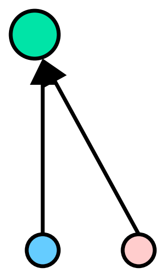
Class1 calls Object2.Flush +

Object2 throws Exception

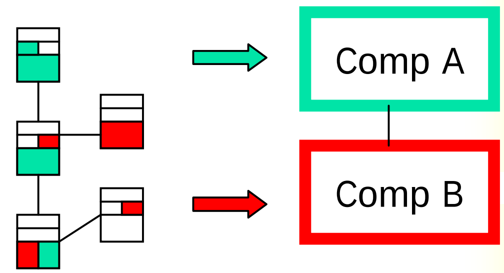


Architecting Dependable Systems
xMonEve – Bottom-Up Example

- Availability of Component B
  - Implementation to Architecture

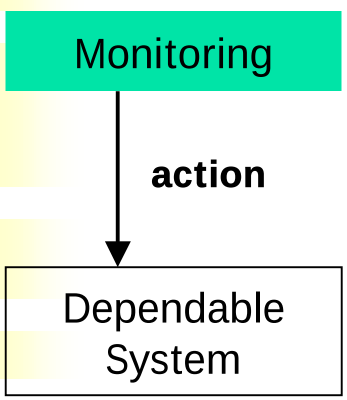


    - Class1 calls Class2.SendHeartbeat +
    - Class2 throws TimeoutException =
      - CompB.NotAvailable

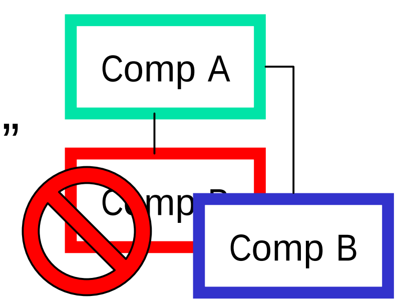


  - Possible Monitoring Action

    (when CompB.NotAvailable event detected)



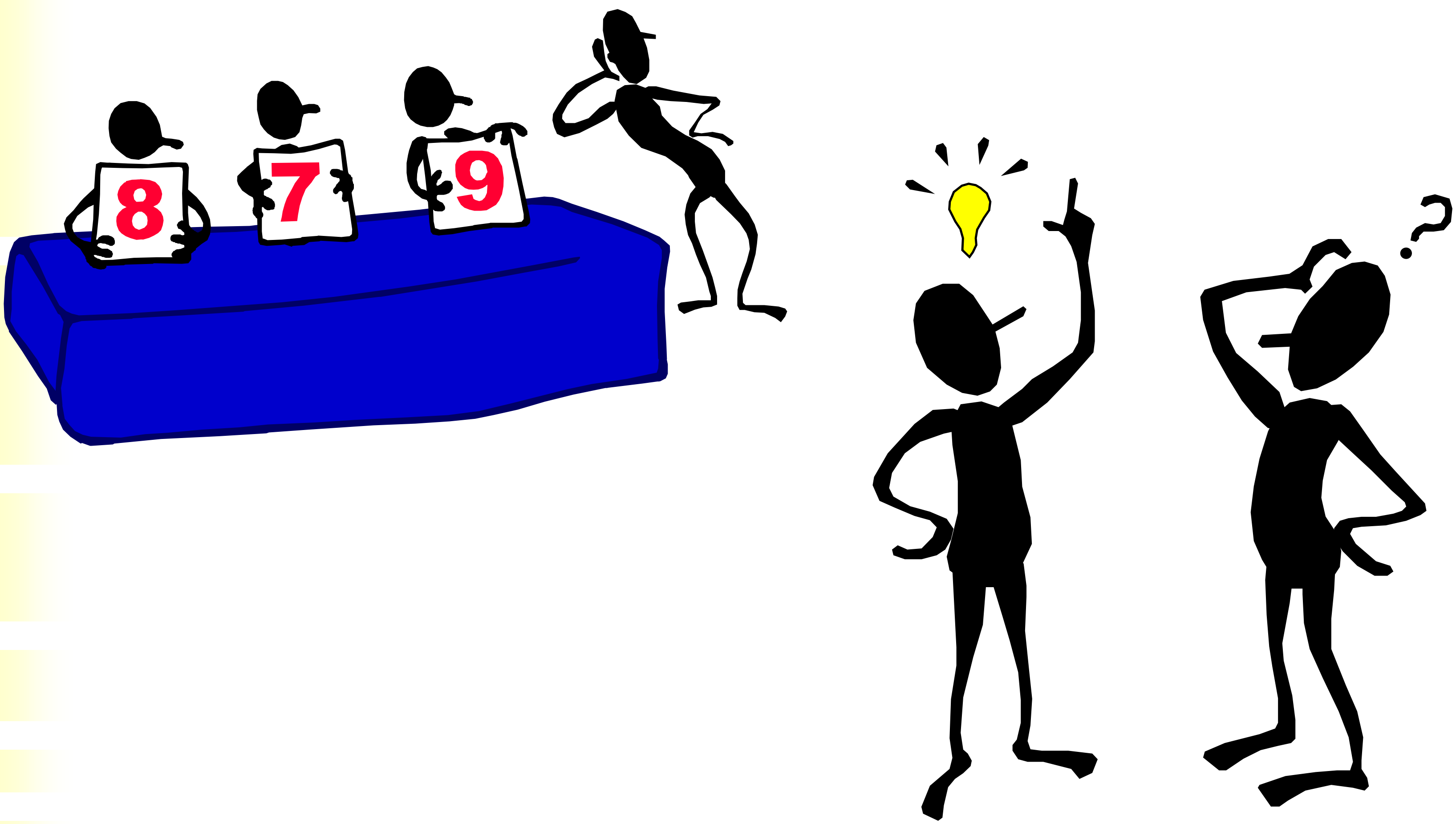
    - Wait and resend heartbeat
    - Restart process / thread:
      - CompA restart “process/thread B”
    - Load new component B
    - Call management functions
    - …



Conclusion

- Events (and their definition) play a major role:
  - As “common abstraction” for development techniques
  - However, a “common description” is required
  - xMonEve description language
    - Integration purpose – interchangeable description for events
- Current status:
  - Definition and refinement of xMonEve
  - Identifying how different purposes affect monitoring systems
    - same monitoring functionality in many occasions
    - family of monitoring systems with customizable components
  - Development of tools to support definition of events

Questions, Comments & Discussion



Extracting Event Description from Software Documents and Process Level Events

From:

Software Specification Documents

Sequence diagram	CSP	Scenario	Petri-nets
Activity diagram	Posets	LTL	Assertion
Markov model	Statechart	FSM	. . .

Process Level Events

UI events	OS events	Network messages	...

To:

Event Description

xMonEve

For:

Monitoring
(multi-purpose)

Requirements for Event Description Languages

- *general purpose*
  - need to be flexible enough to accommodate event description for multiple monitoring purposes (i.e. independent of the analysis to be performed);
- *independence of monitoring system*
  - must allow generic description of events, both primitive and composed, not restricted to a specific monitoring system (or environment);
- *implementation independence*
  - need to provide mechanisms that separate the conceptual event to the implementation mapping;
- *reusable*
  - event description should be reusable independently of the program implementation and monitoring system
- *extensible*
  - extension of event description should be supported, so more specific information can be associated to the events. For instance, one extension can be the association of monitoring events to software architectural elements

Inherent Gap between Software Architecture and Monitoring

- Level of Abstraction:
  - Software Architecture (higher level)
    - Components, connectors, configuration, style
  - Software Monitoring (lower level)
    - Gather and analyze data from implementation (code) execution
- Different levels of abstraction:
  - collected events vs. software architecture
  - need to describe how (primitive) events are related to higher-level (composed) events
- Monitoring specification languages:
  - Restricted to a single monitoring system
  - Not generic for multiple purposes
  - Cannot associate events to the software architecture

Event Monitoring Background

- Tracing (event-driven) [vs. Sampling (time-driven)]:
  - better understanding and reasoning of the system behavior
  - much larger volume of data
- Reducing the complexity of the monitoring task:
  - integrating *sampling* and *tracing* monitoring
  - collecting the state information through events
- Monitoring system needs to know:
  - what events should be collected
  - what kind of analysis should be performed
    - correct behavior; conditions of interest; behavior characterization
- Monitoring specification languages:
  - describe not only the events, but also the analysis
  - are biased to the kind of analysis performed by the monitoring system

Motivation

- Complexity in Dynamic Software Behavior
  - understanding and reasoning about the dynamic system behavior are complex tasks for humans
  - static analysis techniques are not adequate to check dynamic properties, such as timing, performance and system load
- Dynamic Analysis Techniques and Automated Tools Required
- Software Monitoring as:
  - Intermediate Technique
    - core task for many other dynamic techniques (multiple purposes)
  - Complimentary Technique
    - may (should) be used together with static analysis techniques

What Is Monitoring?
(Dictionary Definitions)

- Meaning of “to monitor”:
  - 1 - make continuous observation of (sth); record or test the operation of (sth). 2 – listen to and report on [Oxford Dictionary]
  - to watch, keep track of, or check usually for a *special purpose* [Merriam Webster’s online]
- Related Verbs:
  - Observe, listen, watch => Collect
  - Record, keep track => Record
  - Test, check => Analyze
  - Report => Display

What Is Software Monitoring?
(Some Selected Definitions)

- Joyce et al. [1987]:
  - *"The monitoring of distributed systems involves the **collection**, **interpretation**, and **display** of information concerning the interactions among concurrently executing processes."*
- Snodgrass [1988]:
  - *"Monitoring is the **extraction** of dynamic information concerning a computational process, as that process executes. This definition encompasses aspects of **observing**, **measurement**, and **testing**."*
- Shim and Ramamoorthy [1990,1991]:
  - *"Monitoring consists of **collecting** information from system and **detecting** particular events and states using the collected information, (which) are subject to further **analysis**."*
- Al-Shaer [1998]:
  - *"Monitoring is defined as the process of dynamic **collection**, **interpretation** and **presentation** of information concerning objects or software processes under scrutiny."*

What Are the Problems of Monitoring?

- Generic Monitoring Systems:
  - Volume = large amount of data to be processed
  - Intrusion = execution slowdown
  - Dimensionality = dimensions to be analyzed (stack/position/in-out/...)
  - Access = restrictions to access program variables and structures
  - "Overheads" = performance/data/programming overhead
- Monitoring Distributed and Parallel Systems:
  - Many foci of control = sequential techniques not sufficient
  - Communication delays = global state (synchronization)
  - Nondeterminism = difficult to reproduce or test
  - Interference = alters behavior (different from slowing down sequential systems)
  - User Interaction = more complex interaction to developer
- Embedded Systems:
  - Target vs. Development environment = different behavior
- Real-Time:
  - Performance (acceptable?)