Tool-Support for the Development of Adaptive Applications

Dipl.-Inf. Andreas Rasche

Operating Systems & Middleware Group

Hasso-Plattner-Institute
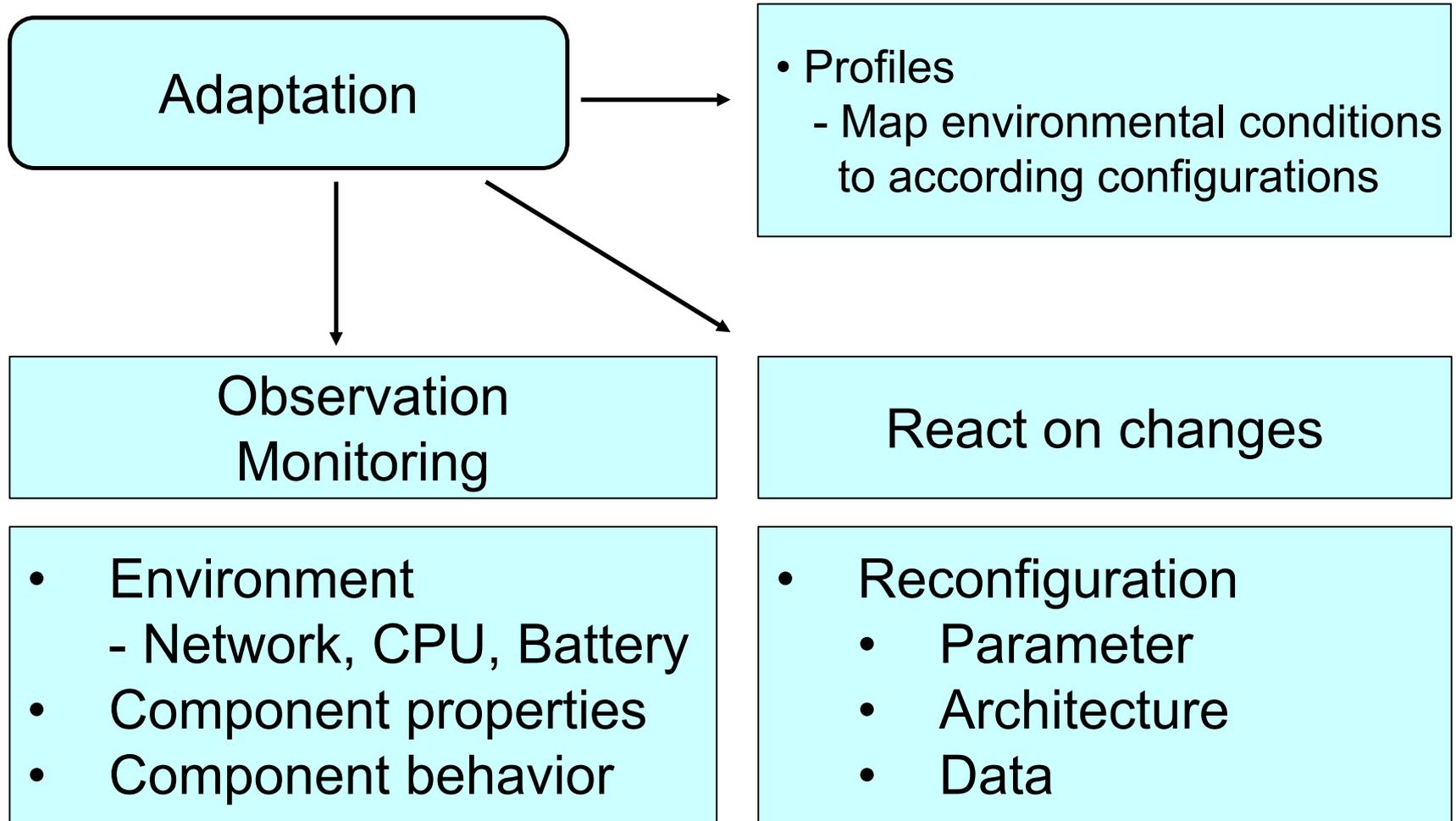
University of Potsdam

Outline

- **Motivation**
- **Adaptive Application with the Adapt.Net Framework**
    - ❑ Dynamic reconfiguration of component-based applications
    - ❑ Synchronization patterns
    - ❑ Creating configurable components
    - ❑ Runtime Infrastructure
- **Building alternative configurations:**
    - ❑ Integrating architectural patterns
- **Concluding remarks**

New Patterns for Developing Adaptive Applications

- **Dynamic environments of modern software demand adaptation of software**

- **Self-configuration: change must be programmed**

- **State-of-the-art development realize adaptation application-specific**

- **Improving development of adaptive software by:**

  - Identification of architectural patterns
  - Tool-support for automatic integration of adaptation-specific concerns
 - Support for developing alternative configurations

Adaptation: Building Blocks

Adaptation

- Profiles
  - Map environmental conditions to according configurations

Observation Monitoring

React on changes

- Environment
  - Network, CPU, Battery
- Component properties
- Component behavior

- Reconfiguration
  - Parameter
  - Architecture
  - Data

The Adapt.Net Framework

- Selection among alternative application configurations enables adaptive behavior

- Dynamic reconfiguration / runtime deployment

- Tool-support for creating reconfigurable applications

- Building alternative configurations: architectural patterns

- Runtime updates and migration

- Monitoring infrastructure: environmental properties, component parameter, component state

- Choosing configurations: Adaptation Profiles

Application Model
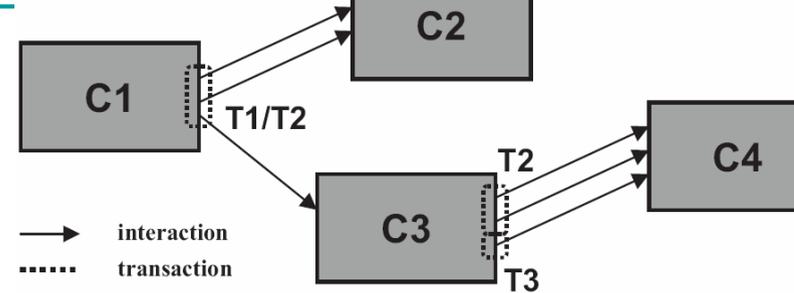
- Based on a work of J. Magee, J. Kramer, M. Wermelinger
- Components: Interconnected computational entities
  - ❑ Provide interfaces : in-ports
  - ❑ Require other components: out-ports
- Components connected by connectors
- No cycles in application topology graph
- A **transaction** virtually combines a number of bidirectional interactions between components
  - ❑ completes in bounded time
  - ❑ Initiator is informed about completion
- A transaction T1 is dependent on a subsequent transaction T2 (T1/T2) if its completion depends on the completion of this transaction

XML-Based Configuration Description

- "A **Configuration** of a component-based application denotes the set of its parameterized components and the connections among them."

- Configuration description:
  - List of components
  - Component attributes
  - Component ports
  - Connectors

```
<configuration configurationname="c1">
    <component name="Viewer" args="" loadtype="OBJECT" assembly="MessageView
        .dll" assemblyVersion="..." access="" type="AdaptNet.
        ConfiguredObjectProxy.MessageView" location="localhost">
        <port name="m_source" type="OUT" vartype="sample.proxys.
            IMessageSource" />
        <port name="default" type="IN" vartype="System.Object" />
    </component>
    <component name="Source" args="" loadtype="CorbaComponent" assembly="
        MessageView.dll" assemblyVersion="..." access="" type="sample.
        proxys.IMessageSource" location="localhost">
        <port name="default" type="IN" vartype="System.Object" />
        <property name="encoding" value="UTF-8" type="System.String" />
    </component>
    <connector sourcecomponent="Viewer" sourceport="m_source" sinkcomponent=
        "Source" sinkport="default" type="IIOP" />
</configuration>
```

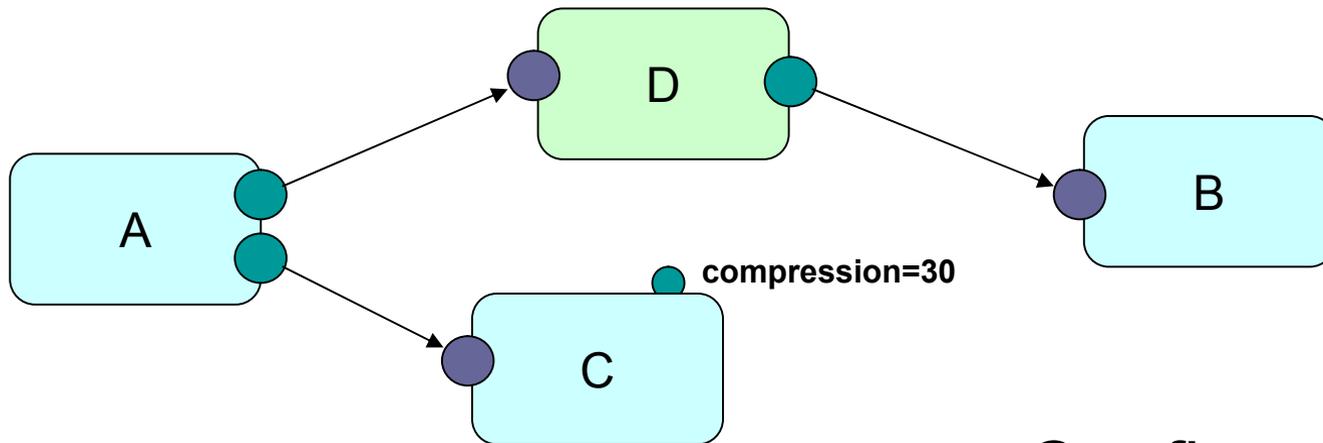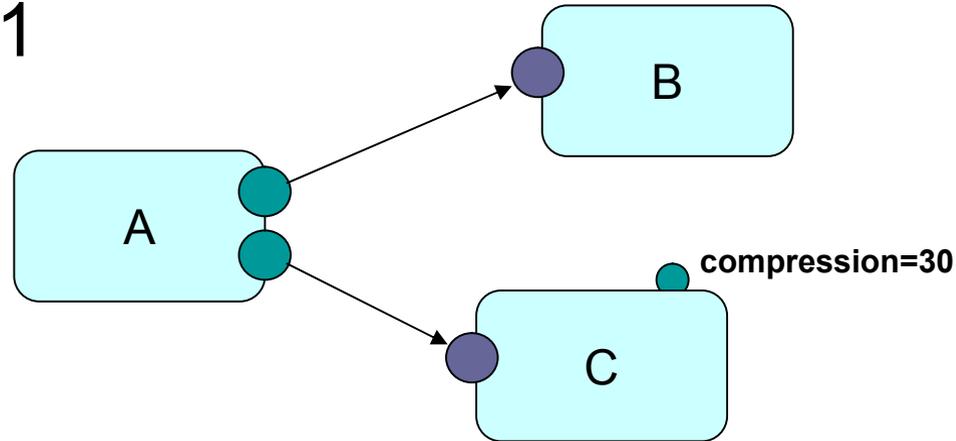# Configuration Infrastructure
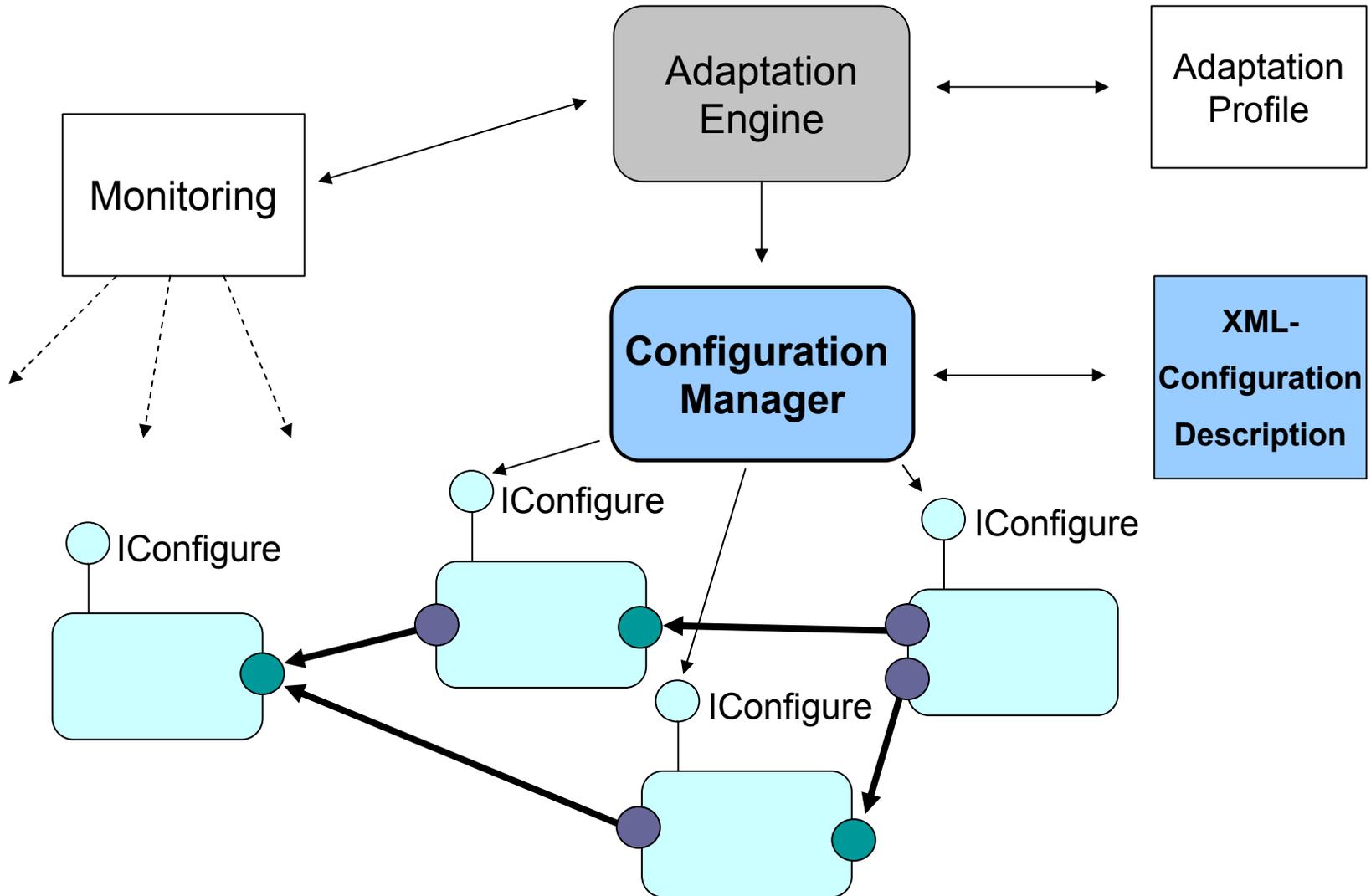
Reconfiguration Algorithm

1. Loading of new components
2. Bringing application into reconfigurable state
   - Application consistency must be preserved during reconfiguration
   - Blocking all connections: Wait for all on-going transactions to complete; don't allow initialization of new ones
   - Blocking must be ordered due to dependent transactions
3. Transferring state of migrated/updated components
4. Setting changed component parameters
5. Reconnecting components (create connectors)
6. Restarting component processing
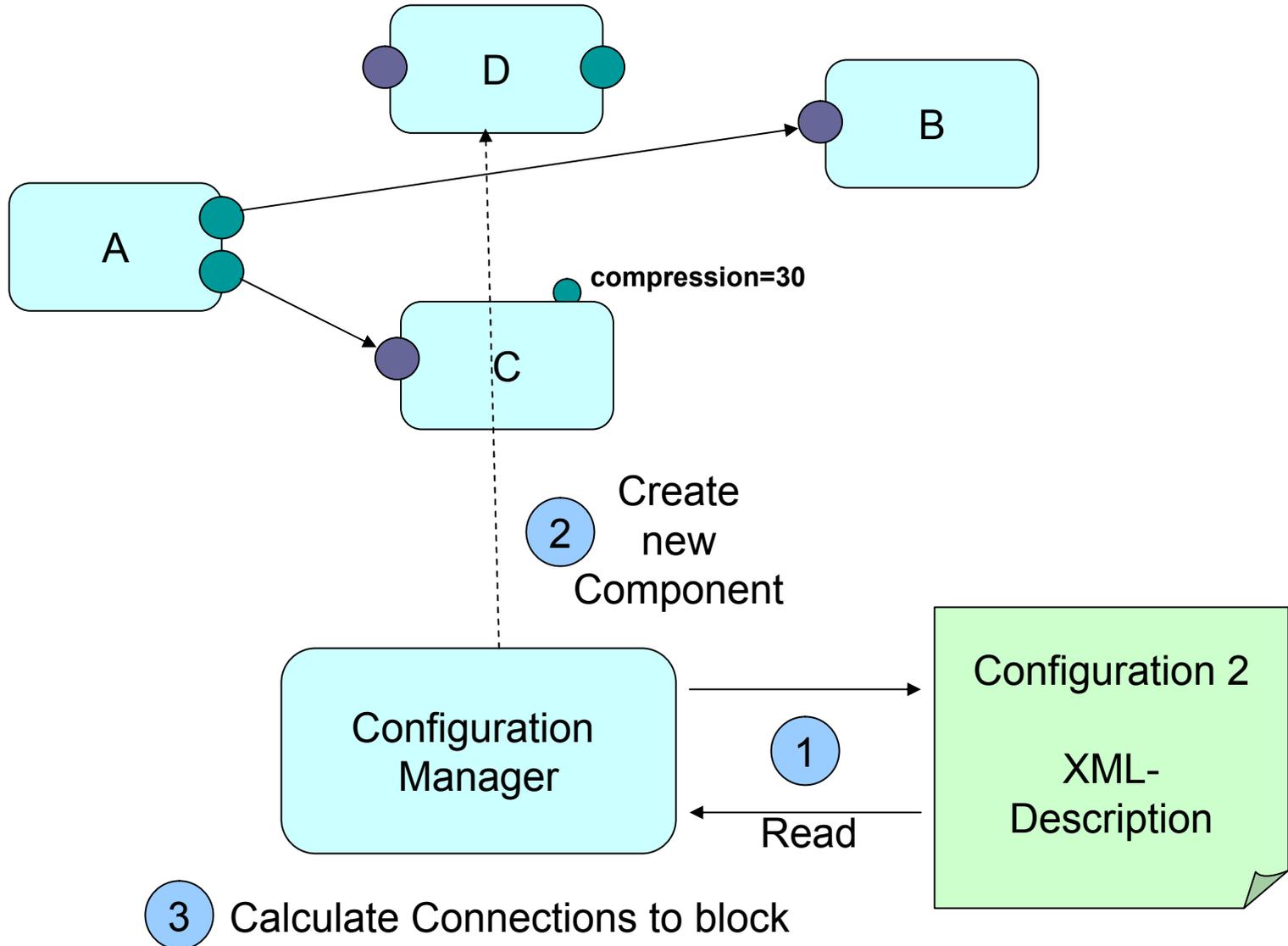7. Removing old components

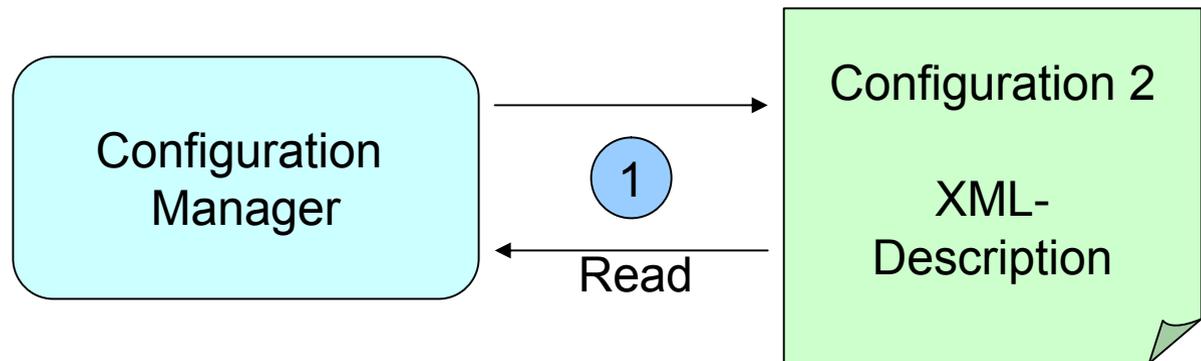# Dynamic Reconfiguration

Configuration 1

compression=30

compression=30

Configuration 2

Dynamic Reconfiguration

compression=30

2 Create new Component

Configuration Manager

1

Read

Configuration 2

XML-Description

3 Calculate Connections to block

Dynamic Reconfiguration

compression=30

Configuration Manager

Configuration 2

XML-Description

1

Read

Dynamic Reconfiguration

Block new connections
Wait for on-going to complete

5

D

B

A

compression=30

C

Reconfigurable
State

6

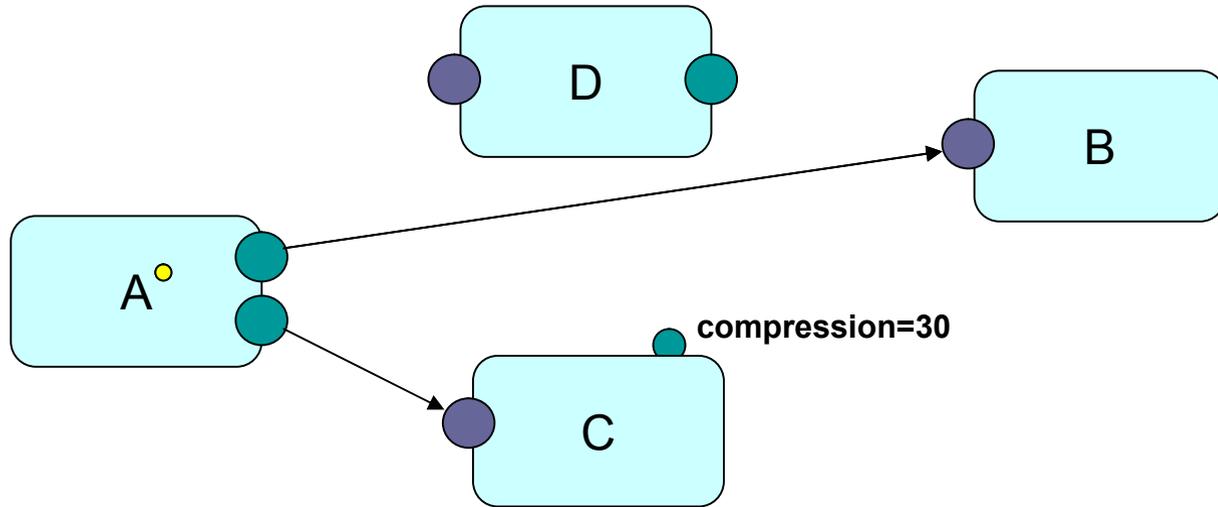Block Connection

4

Configuration
Manager

1

Read
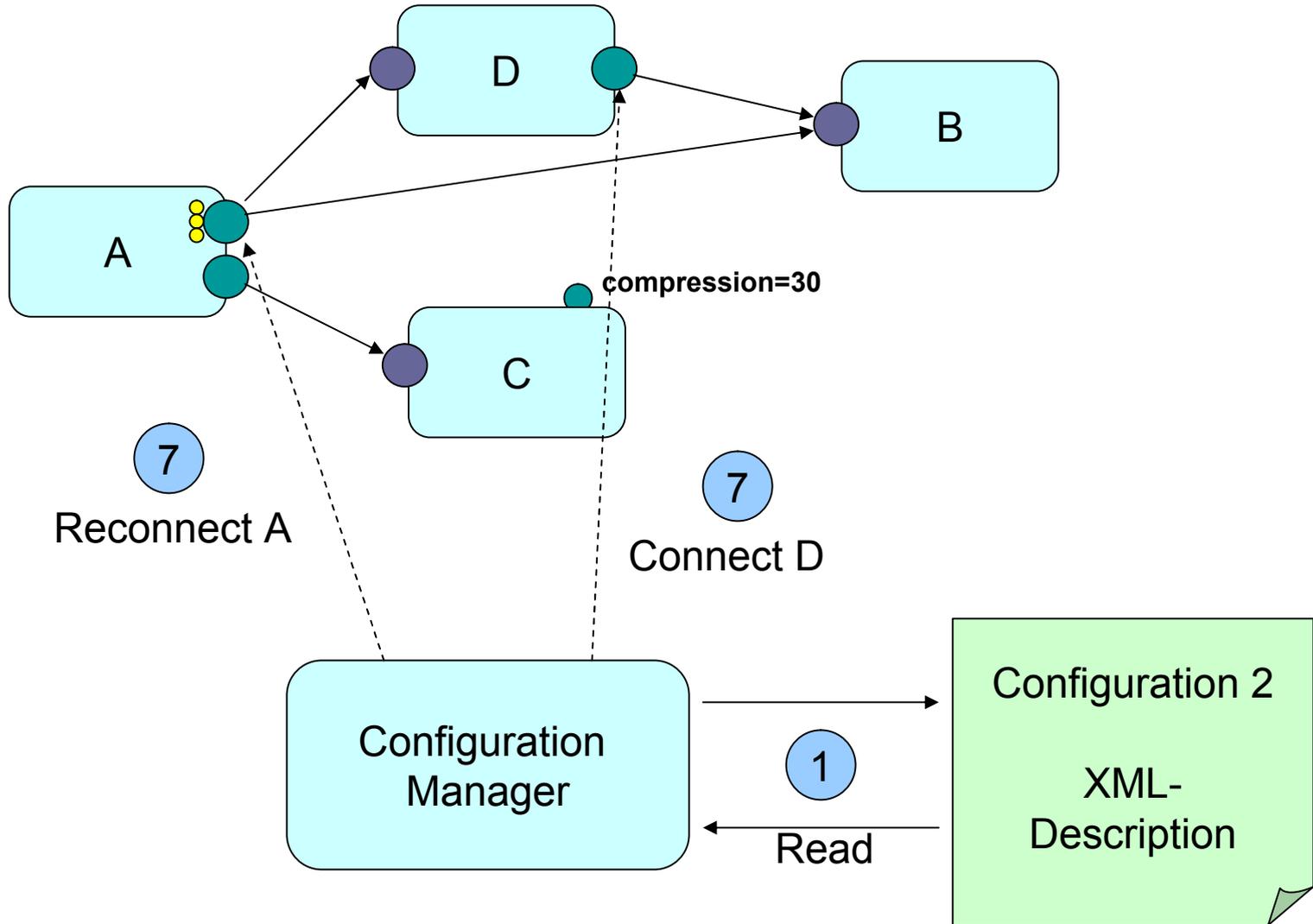
Configuration 2

XML-
Description

Dynamic Reconfiguration

Dynamic Reconfiguration

D

B

A

compression=30

C

8 Restart processing

Configuration Manager

1

Read

Configuration 2

XML-Description
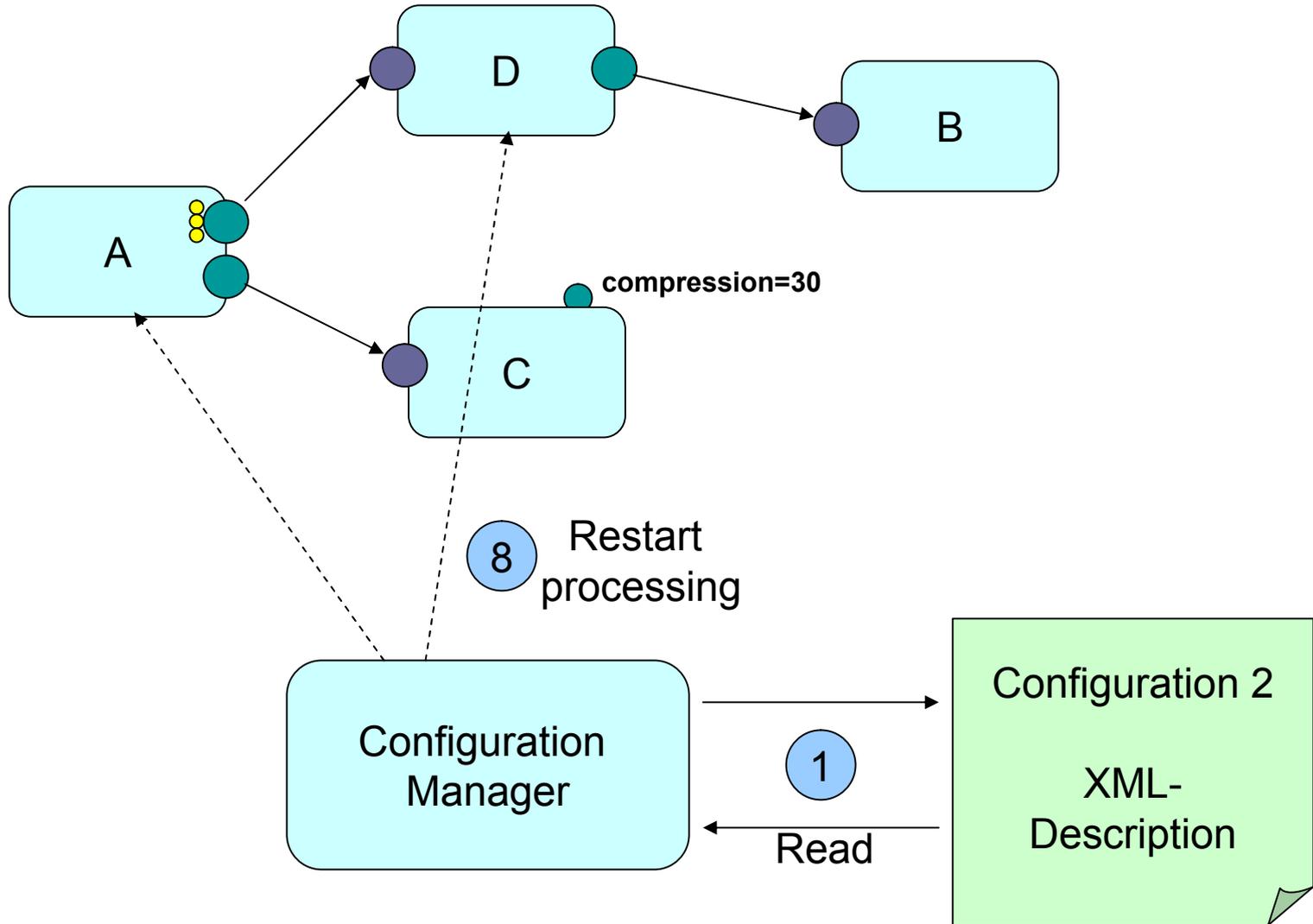
Blocking a Connection

- **Component Code calls**
  - **TransactionBegin:**
    If(blocked) Wait(block_semaphore);
    processing=true;
    Wait(proc_semaphore);
  - **TransactionEnd:**
    processing=false;
    Release(proc_semaphore)

- **Configuration interface**
  - **BlockConnection**
    blocked=true;
    Wait(block_semaphore)
    if(processing) Wait(proc_semaphore)
    Release(proc_semaphore)
  - **Start** of Configuration interface
    blocked=false;
    Release(block_semaphore);

```
int a = 3;

int b = 5;

    Transaction.Begin();

        b = calc.Add(1,b);

        a = calc.Sub(a,b);

    Transaction.End();
```

Multithreading ?
Recursive Function ?

Reader-Writer Lock for Synchronization

- Synchronizes multiple read and concurrent write requests
- On write request: wait for all acquired read locks to complete
- New read requests are queued
- No synchronization needed for a read request if there is no write request
- Here we synchronize methods calls (reads) and reconfiguration requests (writes)
- Recursive read locks: threads already owning a read lock can acquire new read locks (for on-going method calls) despite pending write request

Configuration : A crosscutting concern
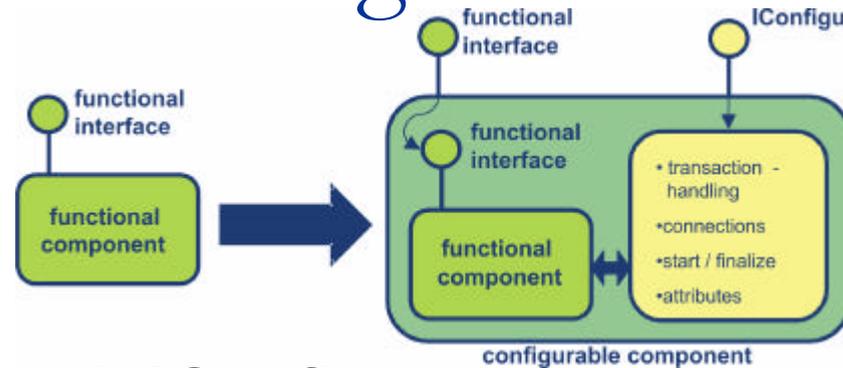
- **Each component has to implement a configuration hook IConfigure:**
  - ❏ Start component processing
  - ❏ Block connections
  - ❏ Set properties
  - ❏ Connect/disconnect out-going ports
  - ❏ Initialization / Finalization
- **Implementation for IConfigure can be generated**
  - ❏ Integrated into graphical development tool of Adapt.Net
  - ❏ Usage of Aspect-Oriented Development (AOP) tools

Configurable Components

```
public class ConfiguredFilter:Filter,IConfigure
{
    void SetProperty(string name, string val){
        cval = Convert(val);
        this.GetField(name).SetValue(cval)
    }
    void Connect(string port, obj target){
        this.GetField(port).SetValue(obj);
    }
    void BlockConnection(string conn){
        mutex.Aquire();
    }
    void StartProcessing(){
        mutex.Release();
    }
}
```
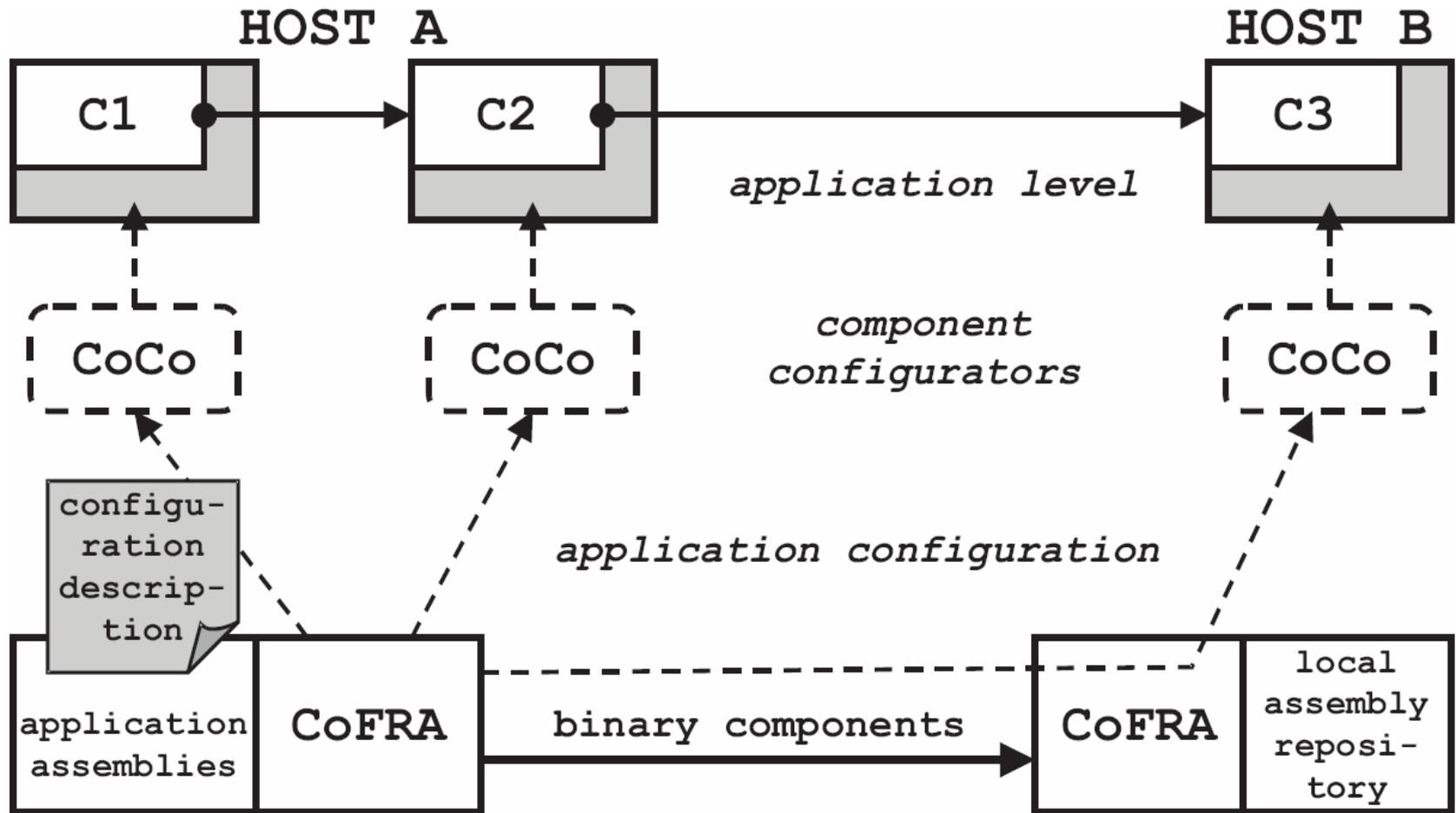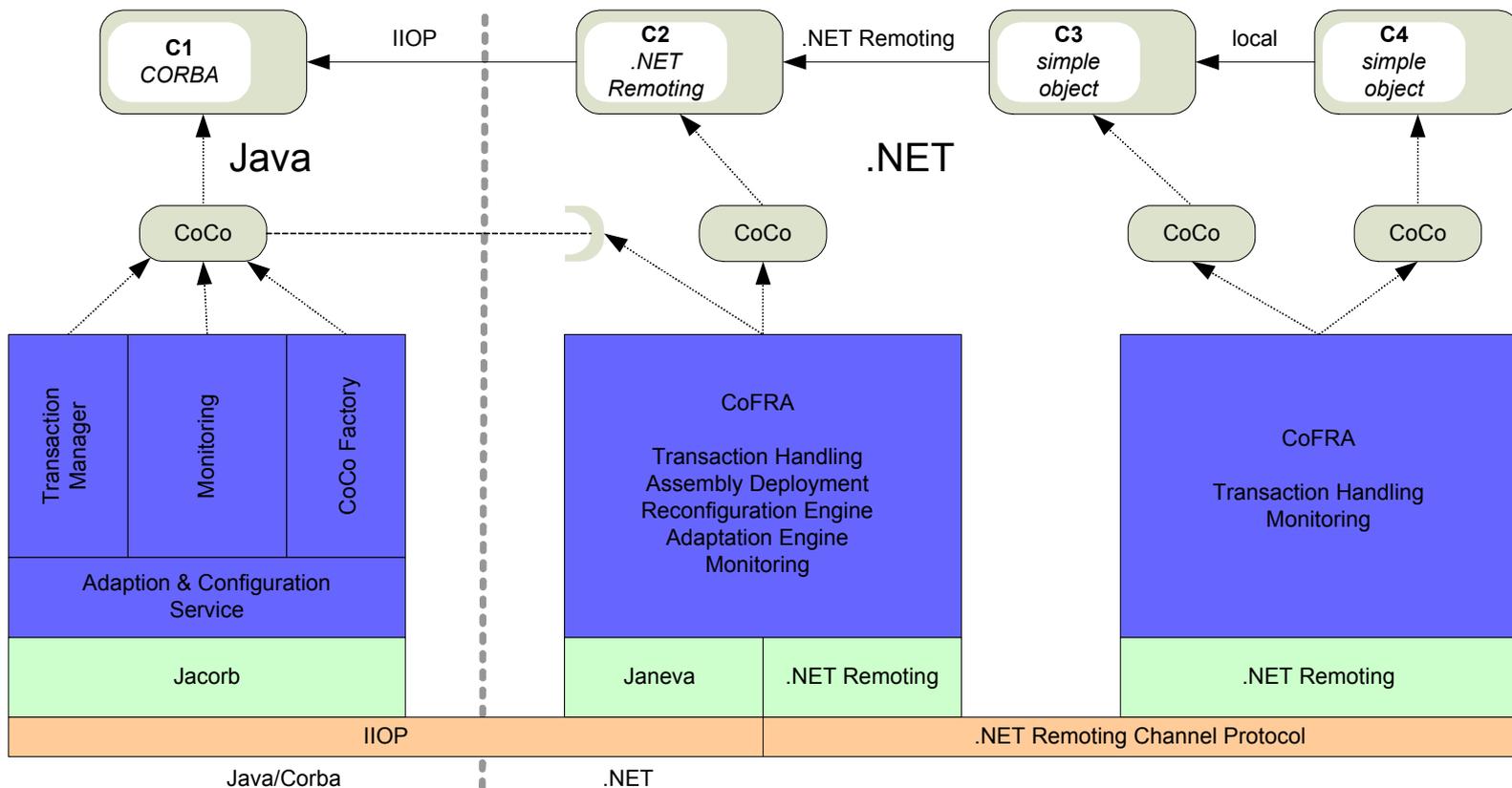
```
public class Filter{
    [AdaptNet.Property]
    int compression;
    [AdaptNet.Connection]
    IViewer viewer;

    public void Send(byte[] data)
    {
        Transaction.Begin(viewer);
        viewer.Send(data);
        Transaction.End(viewer);
    }
}
```

CoFRA Infrastructure / Deployment

The Component Configurator - abstracting away CORBA, Java, and .NET

C1 CORBA — IIOP → C2 .NET Remoting — .NET Remoting → C3 simple object — local → C4 simple object

Java

.NET

CoCo

CoCo

CoCo

CoCo

Transaction Manager | Monitoring | CoCo Factory

Adaption & Configuration Service

Jacorb

CoFRA

Transaction Handling
Assembly Deployment
Reconfiguration Engine
Adaptation Engine
Monitoring

Janeva | .NET Remoting

CoFRA

Transaction Handling
Monitoring

.NET Remoting

IIOP | .NET Remoting Channel Protocol

Java/Corba | .NET

Component Type Loaders

- **Component Configurators** hide implementation details of component instances during runtime
- Available **Component Type Loaders** manage components life cycle : loading, initialization, deletion, finalization
- Existing Component Type Loaders and component start
  - **Simple Object**
    - Created using **new** language construct
  - **.NET Remoting Object**
    - Registration of Remote Service
  - **Process Component Type**
    - CreateProcess and Registration of Management Service Interface
  - **Java-based CORBA Object**
    - Initialization of CORBA runtime, object creation and registration

Connector Architecture

- Connector between components exchangeable during runtime
- Connector establishment implemented in IConfigure.Connect method
- .NET Remoting Connector
  - .NET-.NET
- Local Call Connector
  - Between simple Objects
- IIOP Connector
  - CORBA -.NET and CORBA-CORBA
- Planned connector patterns:
  - Tupelspace connector for online/offline configurations
  - Webservice connector to integrate available functionality
  - Gridservice connector to exploit available computing grids

```
public bool Connect(string portname, string conntype, object opti
{
  ...
  if(conntype == "LOCAL")
  {
    FieldInfo conn = this.GetFieldofThis(portname);
    conn.SetValue(this,options);
    return true;
  }
  if(conntype == "REMOTING")
  {
    object[] params = (object[]) options;
    FieldInfo conn = this.GetFieldofThis(portname);
    string connStr = "tcp://"+params[1]+"/"+params[0];
    conn.SetValue(this,Activator.GetObject(conn.FieldType,connStr
    return true;
  }
  if(conntype == "IIOP")
  {
    FieldInfo conn = this.GetFieldofThis(portname);
    object corbaRef = Narrow(options);
    conn.SetValue(this,corbaRef);
    return true;
  }
  ...
}
```

Architectural Patterns for Alternative Configurations

- **Smart Connector pattern**
- **Voter Pattern**
  - Compare output of 3 objects supporting the same interface
- **Safe update pattern**
  - Update a running object to a new version
- **Load balancer pattern**
- **Migration pattern**
  - Migrate an object from one host to another
- **Filter, Compression, Encryption Pattern**
- **Simplex Pattern**
  - Used in Foucault's Pendulum

Grafical Support for Pattern Integration
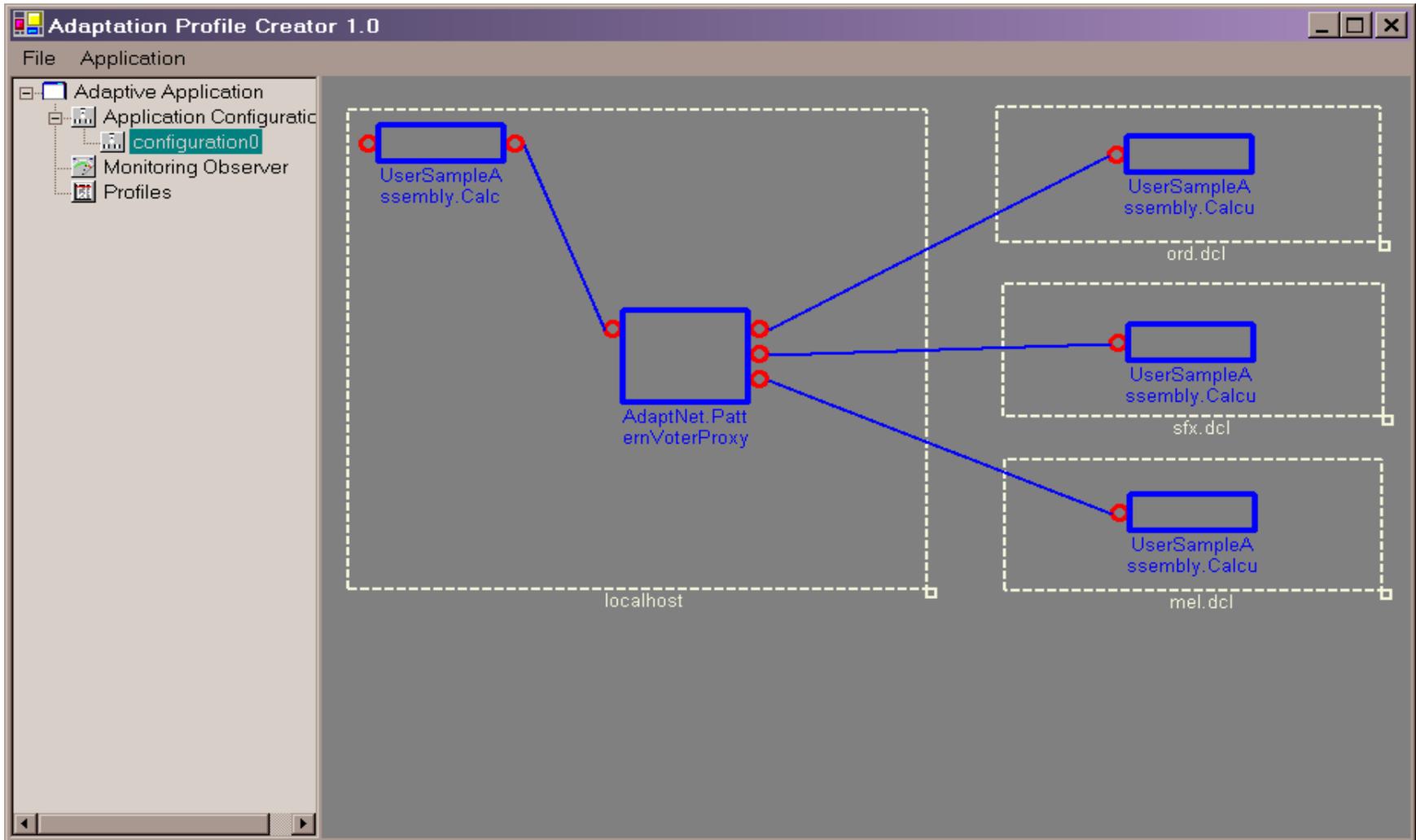
The Simplex Pattern Applied
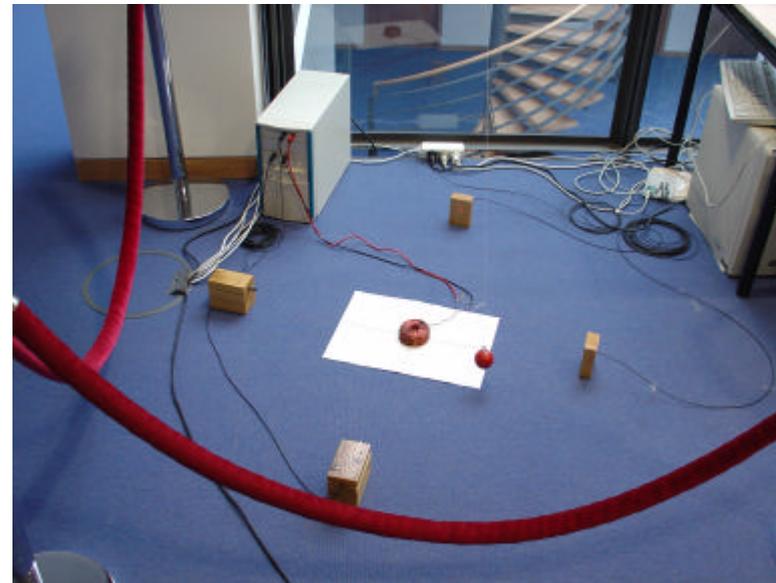## Foucault's Pendulum in the Distributed Control Lab

Configuration 1 : safety controller

Safety Controller

Configuration 2 : user program (cold standby)

Safety Controller

User Program

Configuration 3 : user program (warm standby)

Safety Controller

Event Duplicator

User Program

USB-Driver

Event Queuing

USB

Adaptation Profiles
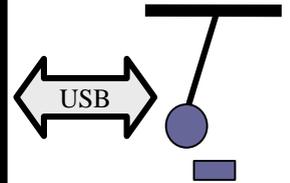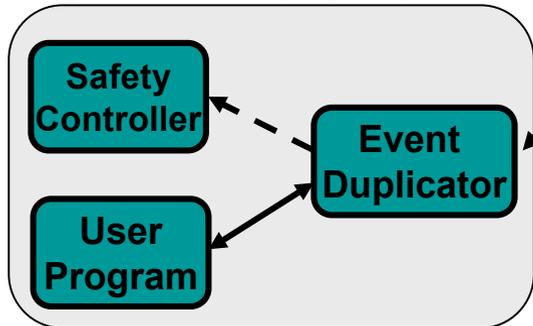
Dynamic Reconfiguration & State

- Realizing component updates
- Component migration for mobility

Component Migration: C from H1 to H2

Host 1

Host 2

B

A

C

C

Transfer Files

3

5

Block Connections

c.dll

c1.dll

c2.dll

2

Analyze File
Dependencies

4

Instantiate C

Configuration
XML-
Description

Configuration
Manager

Configuration
Manager

1

Read Configuration

Component Migration: C from H1 to H2

Andreas Rasche - SAKS Kassel 2006
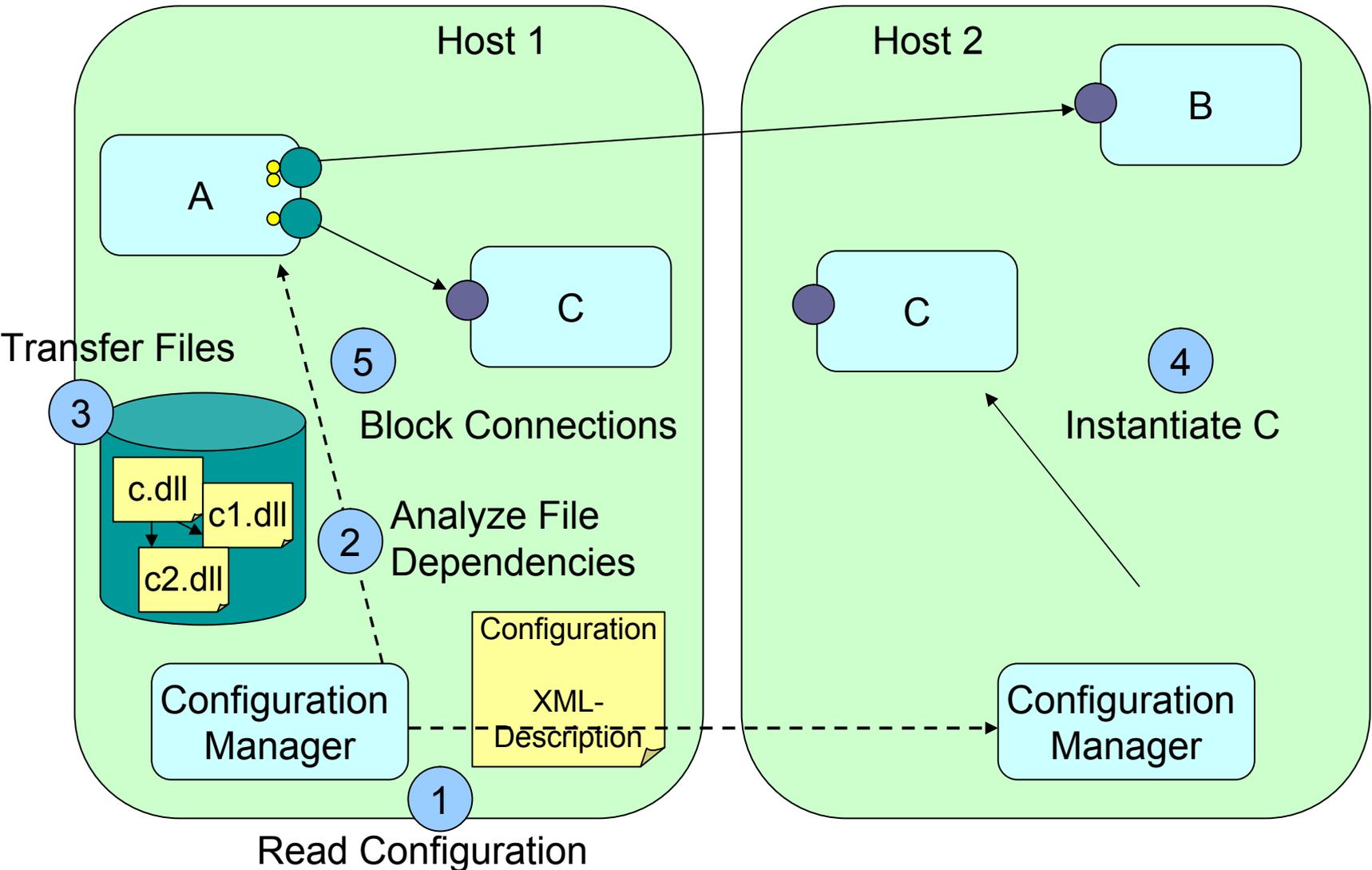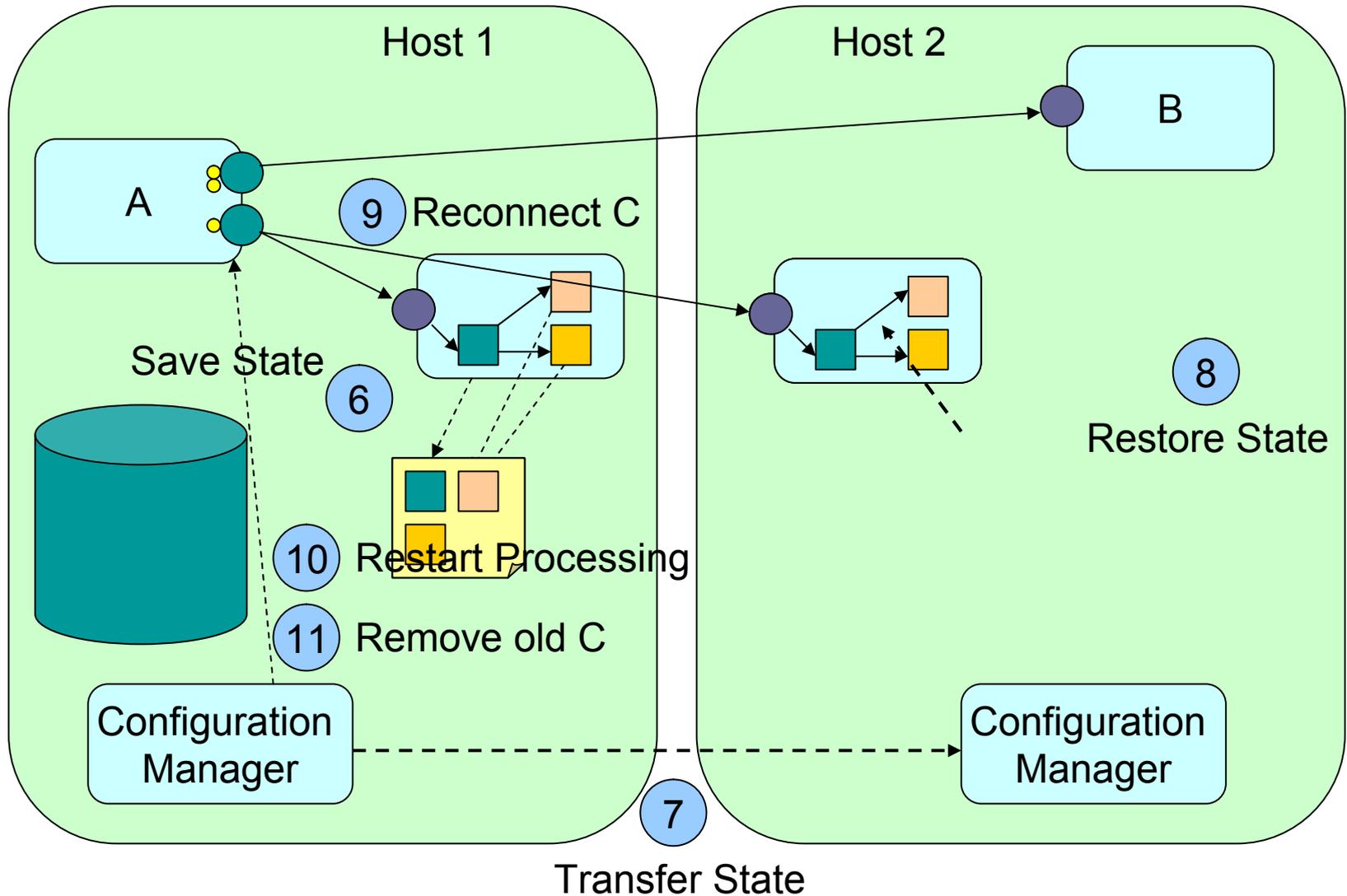
Component Updates: An Algorithm

- **Bring components into reconfigurable state**
- **Traversal of whole object graph**
- **Member-wise investigation of nodes using Reflection**
- **Investigation of each node for update**
- **Update specified via assembly mapping:**
  - Specification of updated assemblies
- **State transfer via member-wise close**

Component Update State Transfer II

- ## Cycle recognition – visited nodes:

  ```
  bool firstTime;
  long id = idGenerator.GetId(node,out firstTime);
  if(!firstTime)
              return visitedNodes[id];
  ```

- ## Save updated nodes in visited nodes list

- ## Investigation of all arrays

  - Traverse all reference type members
  - Create new version in case of base-type update
  - Copy content of updated arrays

Component Update: Delegates

```
public abstract class Delegate
{
    private System.Type target_type;
    private object m_target;
    private string method_name;
    private int method_ptr;
}
```

- **Funtion pointers in .NET**

- **Traverse all targets of delegates**

- **Multicast delegates contain multiple delegates**

- **Updating multicast delegates:**
  - ❑ Extract all delegates from multicast delegate
  - ❑ Create new versions: `Delegate.CreateDelegate`
  - ❑ Combine to new multicast delegate

Finding Optimal Configurations

- **R: Available Ressources**
  - CPU
  - Battery Power
  - Network Bandwidth
- **Q: Non-funktional Properties (QoS)**
  - Video Resolution
  - Framerate
  - Audio Bandwidth
- **User Utility: UU: Q -> [0..1]**
- **Ressource Profile: RP: R -> Q**
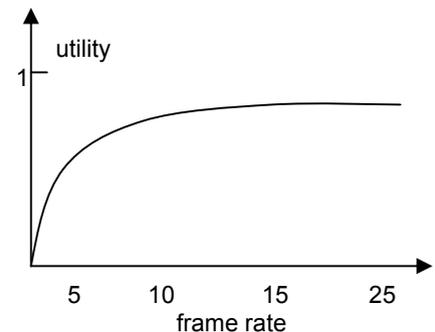
Solving the Configuration Problem

Andreas Rasche - SAKS Kassel 2006

Ongoing Work

- Dynamic reconfiguration / adaptation in real-time systems
- Service Integration in ad-hoc networks
  - Finding best service available
- Beyond adaptation profiles : Closed looped control of application parameter
- Resource manager for handling concurrent applications
- Integrated solution of the configuration problem
  - Creation of a optimal configuration out of a set of given software components
- Dynamic Aspect Activation

Conclusions

- **Development of adaptive applications can be improved by:**
  - Tool-support for building alternative application configurations
  - Reusable Runtime Infrastructures
  - Tool-based generation of configuration-specific code
  - Identification of patterns for dynamic reconfiguration, migration, monitoring, deployment improves development productivity

Tool-Support for the Development of (Self-)Adaptive Applications

Dipl.-Inf. Andreas Rasche

Operating Systems & Middleware Group

Hasso-Plattner-Institute

University of Potsdam

Pattern: Dynamic Aspect Activation

- Loading of new configurations containing interwoven aspect code

- Allows for activation / deactivation of aspects with very small runtime overhead

- Integration Security aspects in unsafe environments

- Dynamic Trace Code Activation

The Server-side approach: dynamic proxies

Client Components

Reconfigurable Component

```
pu
{
....
IS
b.
...
}
```

```
pu
{
....
IS
b.
...
}
```

```
public class A
{
    ....
    IService b = ...
    b.method1();
    ...
}
```

```
class Proxy:IService{
  public method1()
  {
    //track call start
    inst.method1();
    //track call stop
  }...
}
```

```
class B:IService
{
  public method1()
    {...}
    ....
}
```

Service Configuration

Multithreaded Application

dynamic aspect code