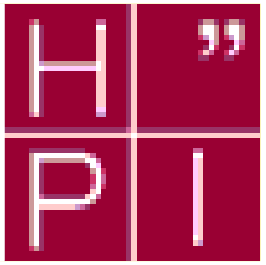


Self-Adaptive Multithreaded Applications - A Case for Dynamic Aspect Weaving

Andreas Rasche, Wolfgang Schult
and Andreas Polze

Operating Systems & Middleware
Hasso-Plattner-Institut for Software Engineering
at University Potsdam
Potsdam, Germany

{andreas.rasche | wolfgang.schult}@hpi.uni-potsdam.de

www.dcl.hpi.uni.potsdam.de

Extending the Reach of Middleware

- Patterns for predictable systems
  - Composite Objects - OO + Real Time
  - Analytic Redundancy and online replacement - DCL / iLab
  - Dynamic (Re-) Configuration of component-based systems
  - Object and Process Migration
- Aspect-Oriented Programming
  - Rapier-Loom.NET - C# and .NET
- Here we present our solution for dynamically updating running software using AOP

DP-ITS: Carbon# & KISS

- Deutsche Post IT-Solutions is using Rapier-LOOM.NET
  - Tracing/Debugging in the development process
  - Dynamic reconfiguration of adaptive counter software (13,000 post offices throughout Germany)
- Long running applications require software updates at runtime to reduce downtime
  - React to cyber attacks
  - Adapt to new environmental situations
  - Provide continuous service during updates

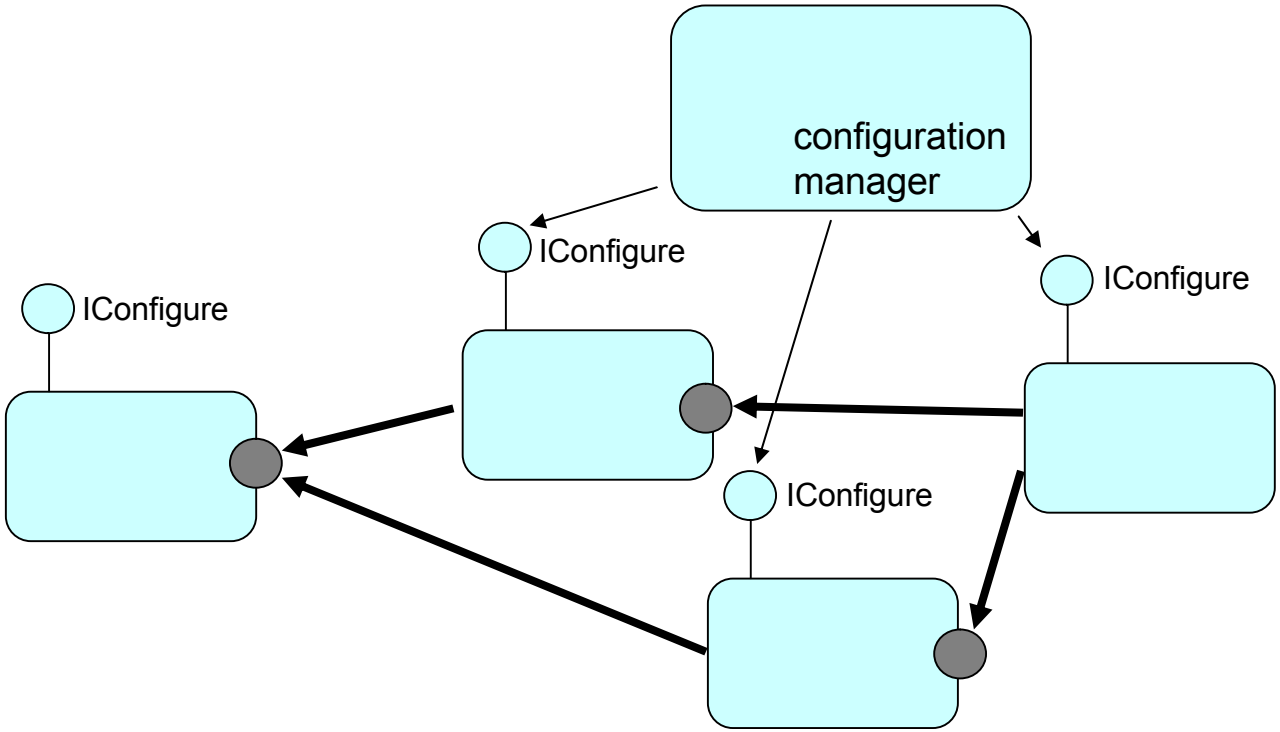
Dynamic Reconfiguration State of the Art

- Blocking of running applications in consistent state
- Reconfiguration through
  - addition/removal/update of components
  - change of component attributes
  - reconnection of components
- Valid reconfiguration points must be marked by application programmer
- Framework hooks at client-side mark save reconfiguration points
- Often no way to force reaching a reconfiguration point
- Limited support for multi-threaded applications

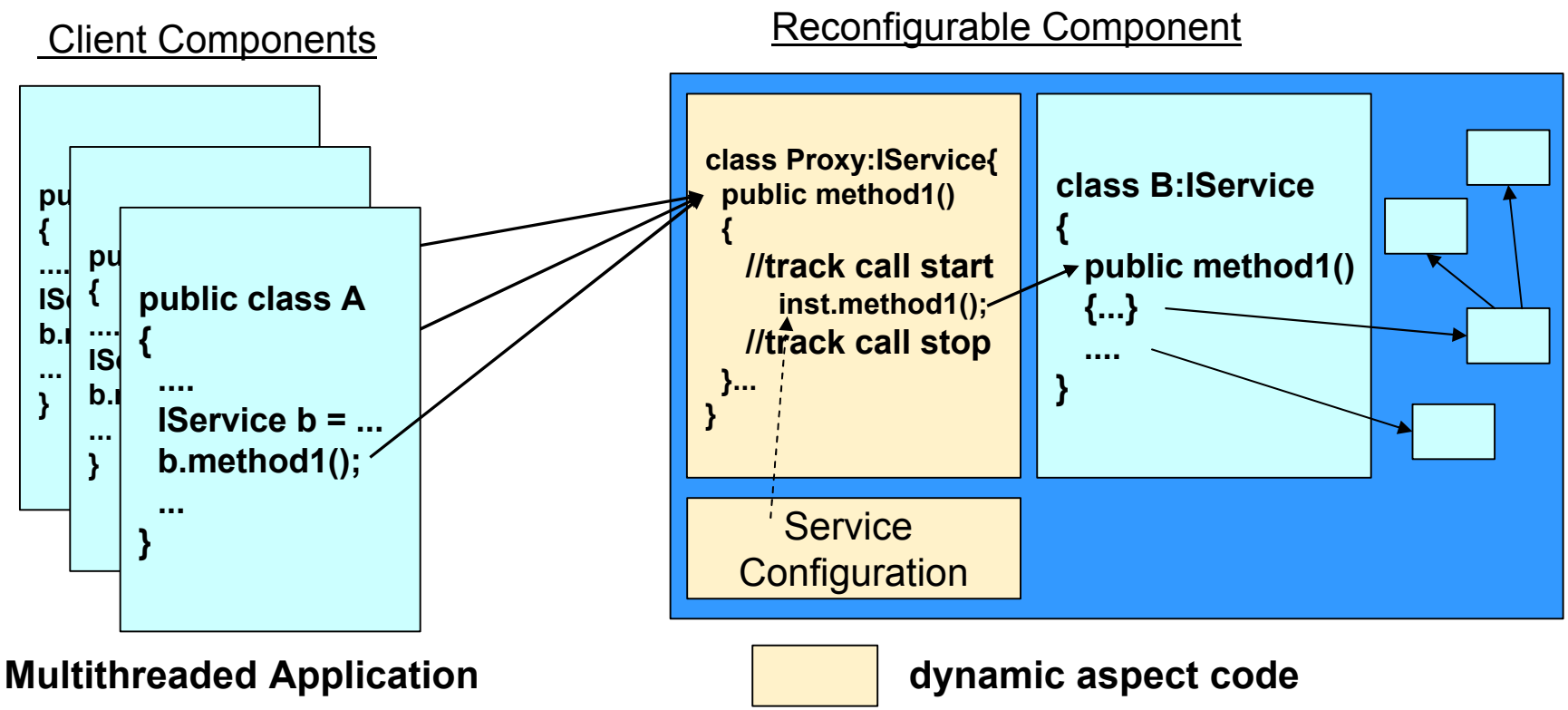
Our Application Model

- Applications are graphs of inter-connected components
- Component interact via public interfaces
- Running components contain a number of objects
  - Instantiation through instantiation of main-class
- Components accessed by a number of clients/threads
- Update of a component includes
  - the addition, removal and change of member variables
  - the addition and removal of methods (if no public interface method)
  - change of method signatures (if no public interface method)
  - code changes (addition, removal, change of instructions)

Architecture for Adaptive Systems



Track Inter-component references through dynamic proxies

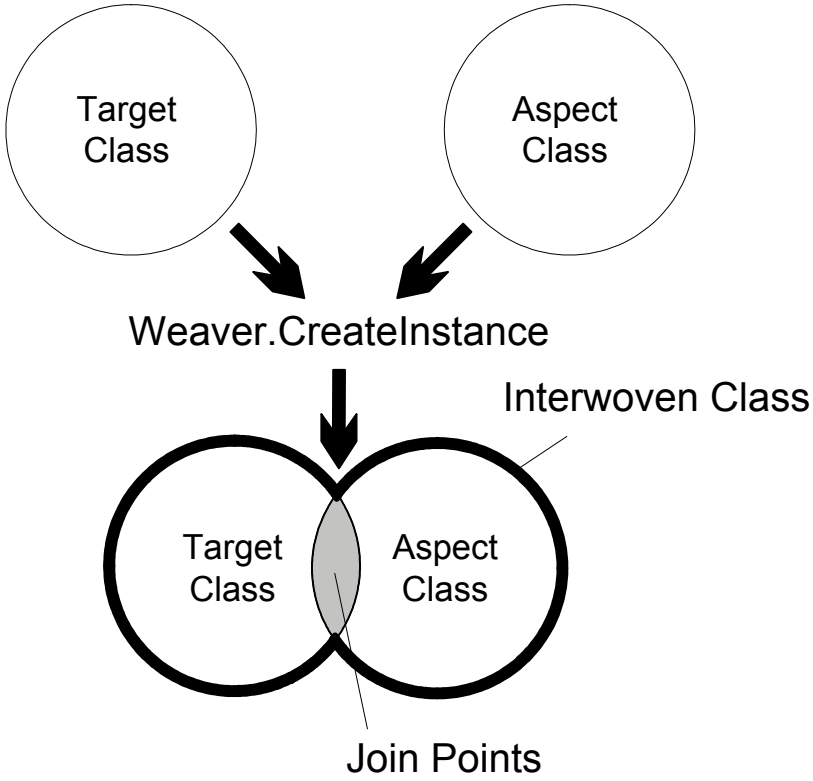


Reader-Writer Lock for Synchronization

- 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

Rapier-Loom.NET in a nutshell

- A *aspect class* is interwoven with a *target class* at defined *join points* by using the *CreateInstance* factory method.
- The result is a single instance of an *interwoven class.*



The Reconfiguration Aspect…

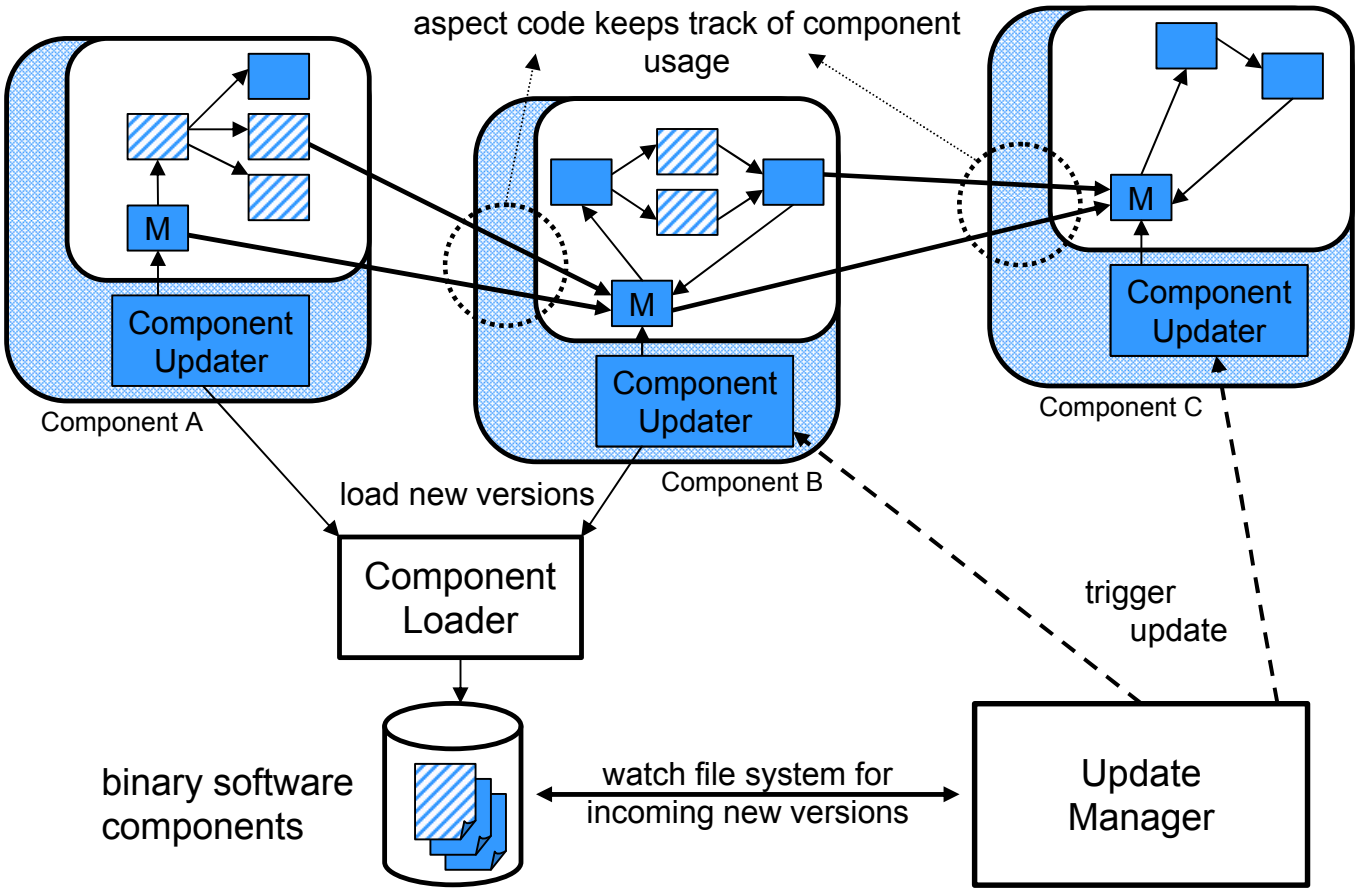
```
[Introduces(typeof(IConfigure))]
public class ReconfigurationAspect:Aspect,
IConfigure
{
        …
  [Create(Invoke.Instead)]
  public object RegisterComponent(Type t,
        object[] args)
  { … }

  [Call(Invoke.Instead)]
  [IncludeAll]
  public object InvokeAll(object[] args)
  { … }
  …
  // IConfigure implementation
}
```

… introduces a new interface IConfigure

…interweaves the object's creation

…interweaves every method call to the component's main class

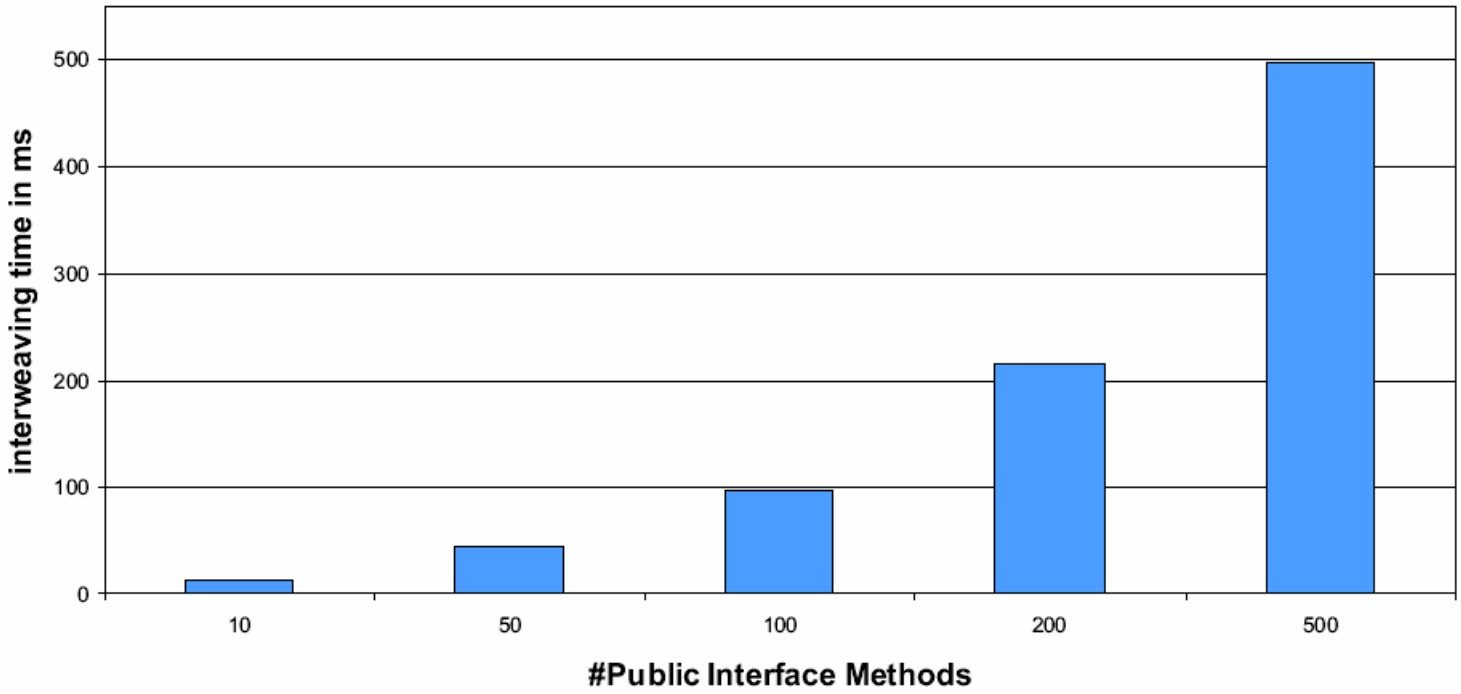
Dynamic Update Infrastructure



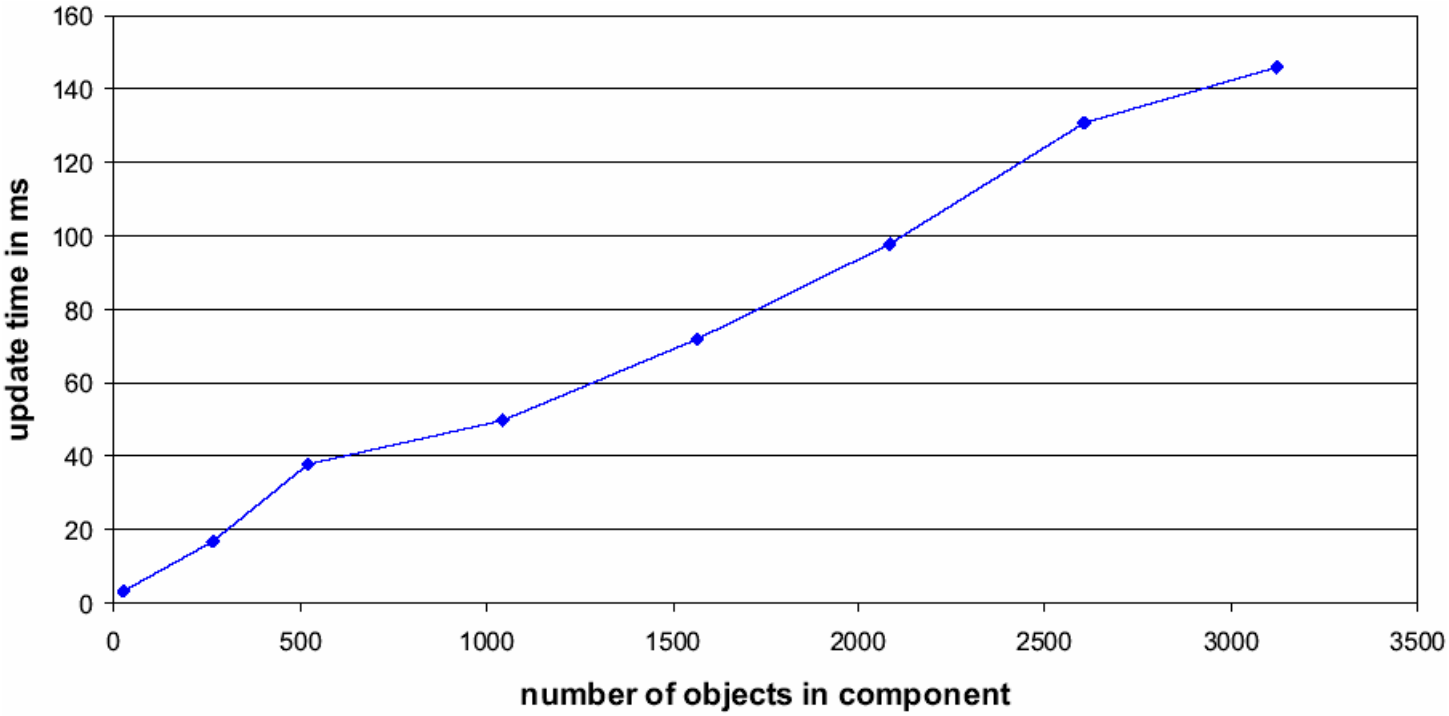
Transfer of state

- Aspect code ensures for no on-going method call on the components interface
- Automatic state transfer through member-wise clone
  - Object-graph traversed for objects to update
  - Recursive algorithm
    - First checks reference target for update
    - Copies state to objects running in new version
    - Values of primitive types are copied to new instances
  - Traversal of all object references, delegates (function pointer), arrays
- User-defined copy-constructors for manual transfer of state
  - Old state passed as construtor argument to new instance

Component Interweaving Time



Component Update Time



Conclusions

- We presented our solution for dynamically updating running software using AOP
- Dynamic aspect weaving for tracking of inter-component references
  - Applicable for off-the-shelf components
  - No explicit knowledge of component‘s clients
  - Only minor changes to existing component-based applications
  - No additional complex rules for application developers
  - No additional compile steps in software development process
- Short update times even for complex applications
- Dynamic Activation/Update of Aspects