Dynamic Updates of Components in the .NET Framework

Andreas Rasche and Wolfgang Schult

Operating Systems & Middleware Group

Prof. Dr. Andreas Polze

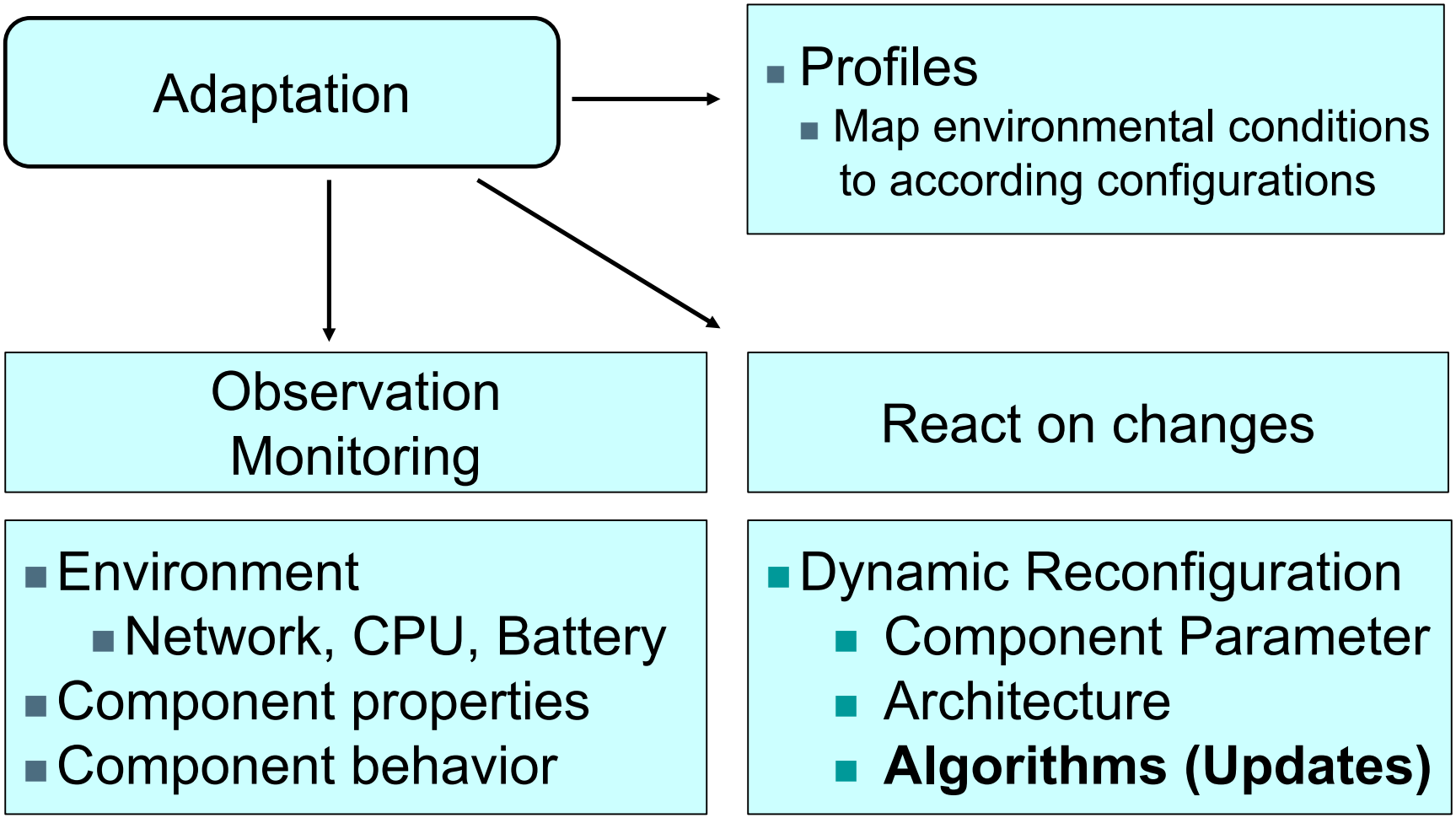
Hasso-Plattner-Institute

University of Potsdam

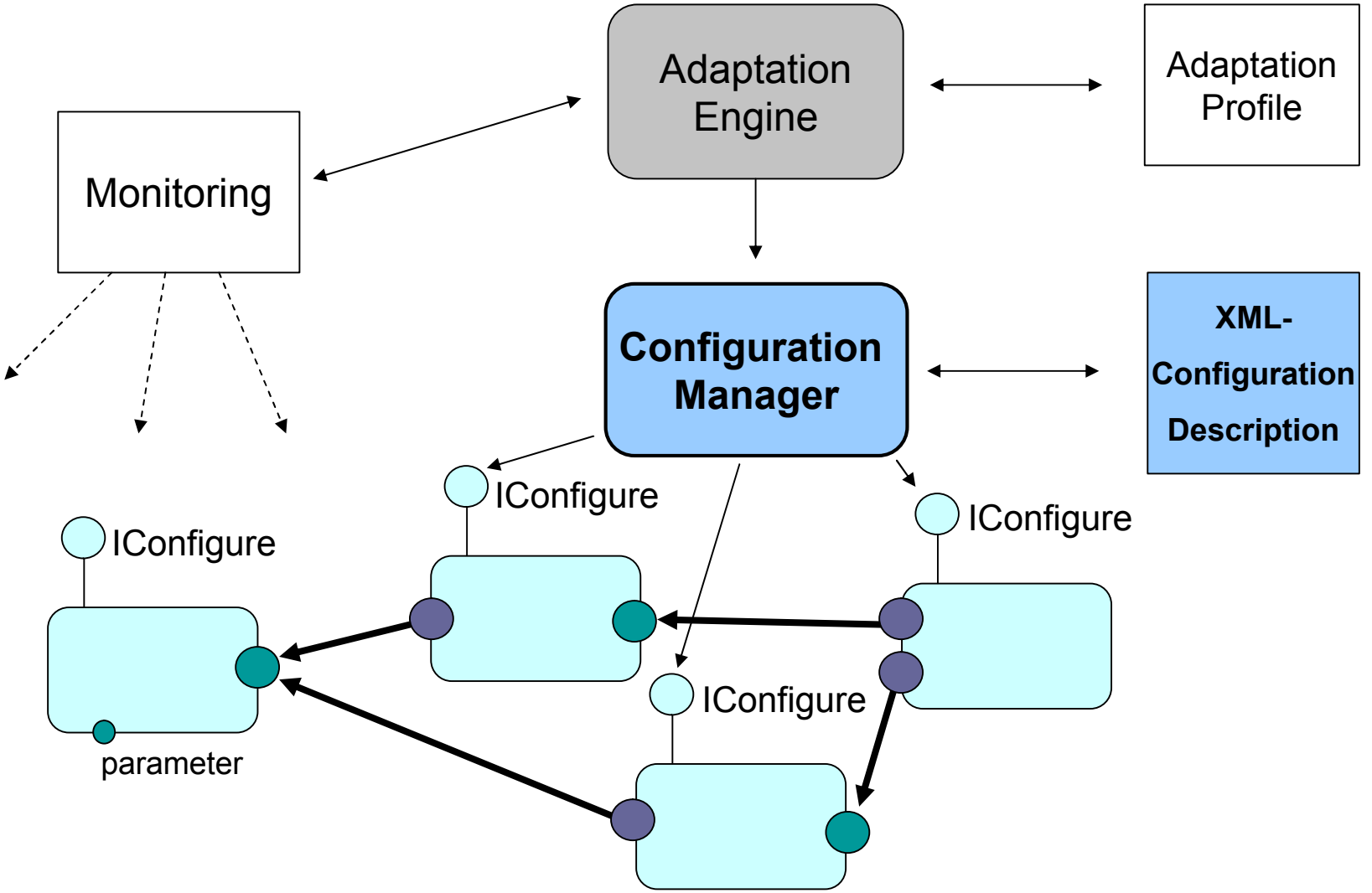
Outline

- Motivation
- Adaptive Applications: Adapt.Net Framework
- Dynamic Component Updates
  - Reaching a reconfigurable state
  - Updating object graphs
- Updating graphical components
- Performance Evaluation
- Concluding remarks

Adaptation: Building Blocks



The Adapt.Net Configuration Infrastructure

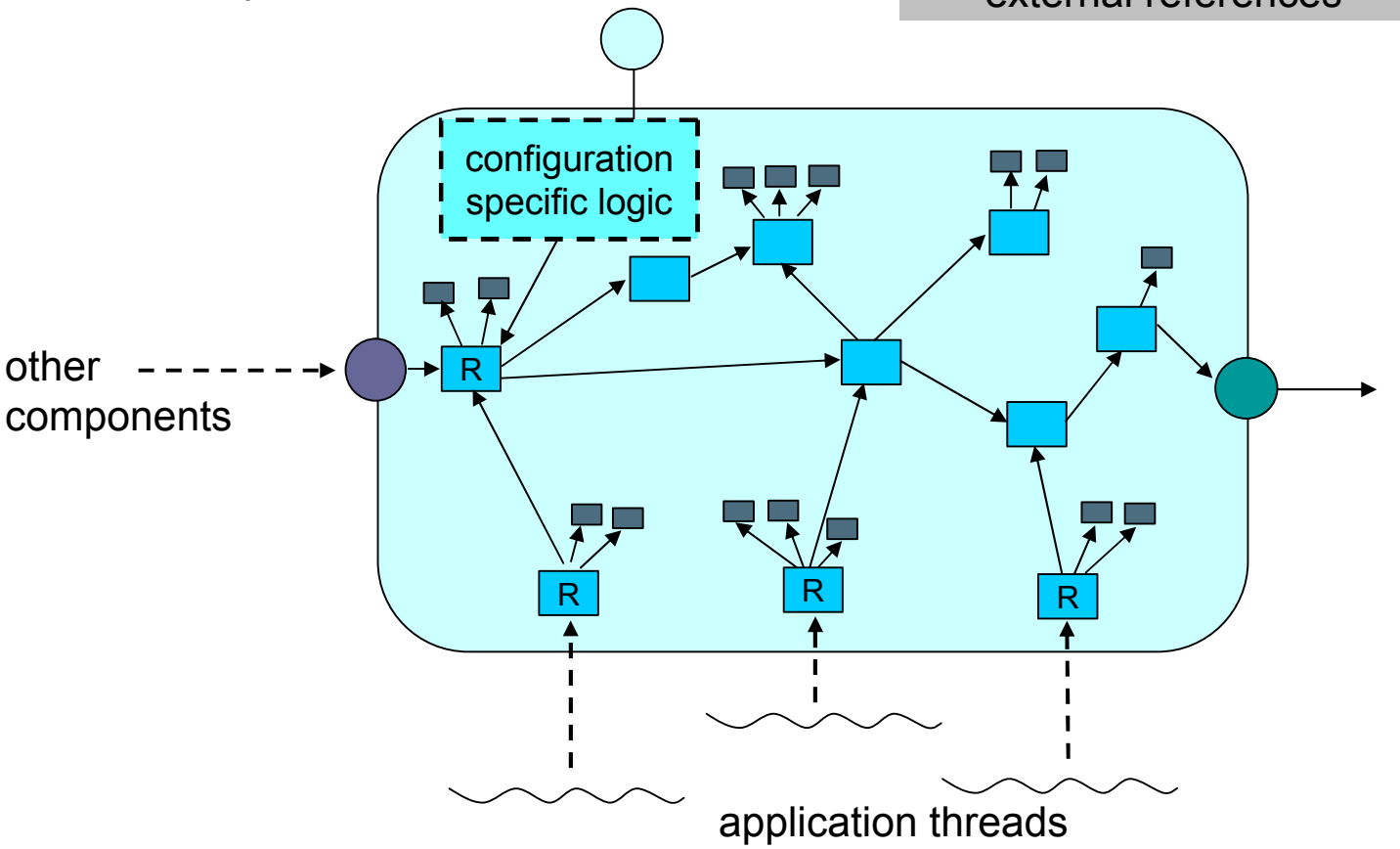


Dynamic Component Updates

- Components have to be updated dynamically to:
  - Activate more appropriate algorithms at runtime
  - Change graphical representation of adapted architecture
- Adaptivity demands short black-out times:
  - Updates must be performed at runtime
  - Component restart sometimes not possible
- State must be transferred from old to new version
- Update must be atomic in order to maintain:
  - Structural consistency
  - Consistency of application invariants

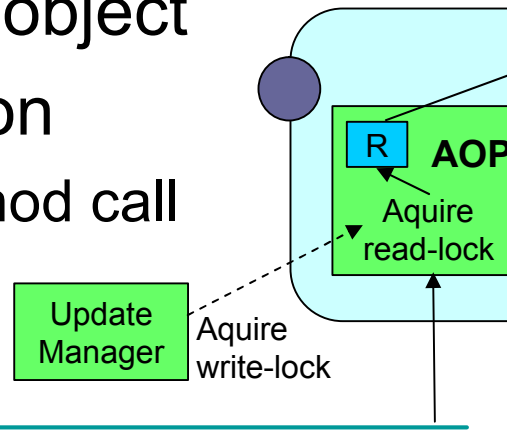
Components & Root Objects

- A component is a set of objects
- Each object has a type
- Each type is defined in an assembly
- Each assembly has a version

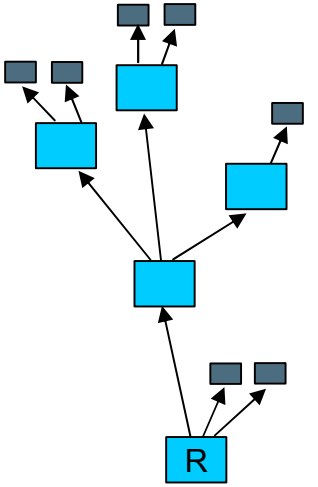


Reaching a reconfigurable state

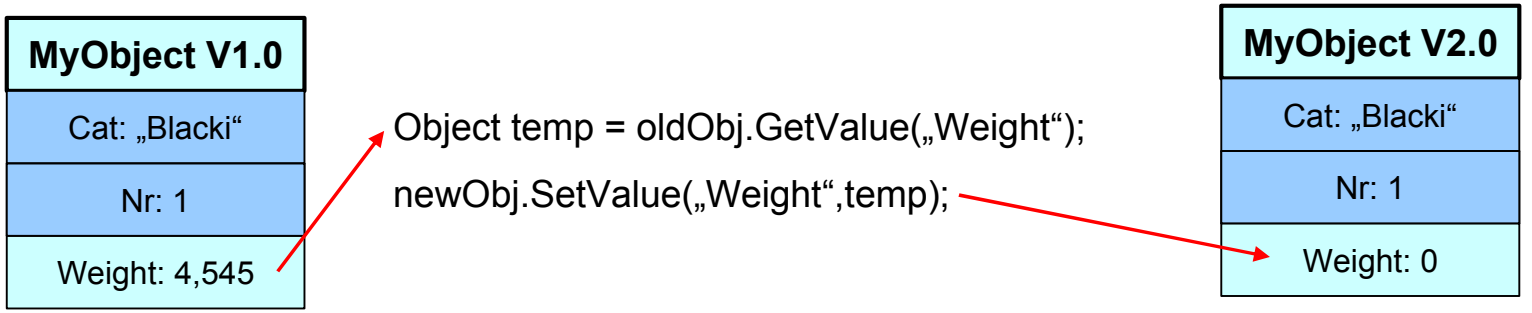
- A component is reconfigurable if there is **no on-going method execution of component‘s objects on any threads‘ stack**!
- A reconfigurable state can be reached by:
  - Blocking new method calls
  - Waiting for all ongoing method calls to complete
- Aspect-Oriented Programming (AOP) can be used to add synchronization logic to each root-object
- Reader-Writer-Locks for synchronization
  - Read-Lock is aquired for each normal method call
  - Write-Lock is aquired by the update logic
  - Usage of recursive locks for recursive calls



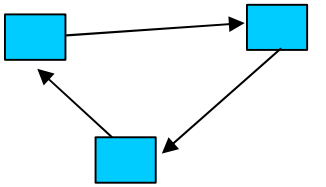
Traversing the Object Graph



- Start from all root objects
- For each field of all objects traverse all references
- In case of an update:
  - Create an instance of the new version
  - Copy the state by transferring all fields from the old to the new instance
  - For reference fields: traverse target first an install potential new version afterwards
- Usage of .NET Reflection (GetFields, Set-/GetValue)

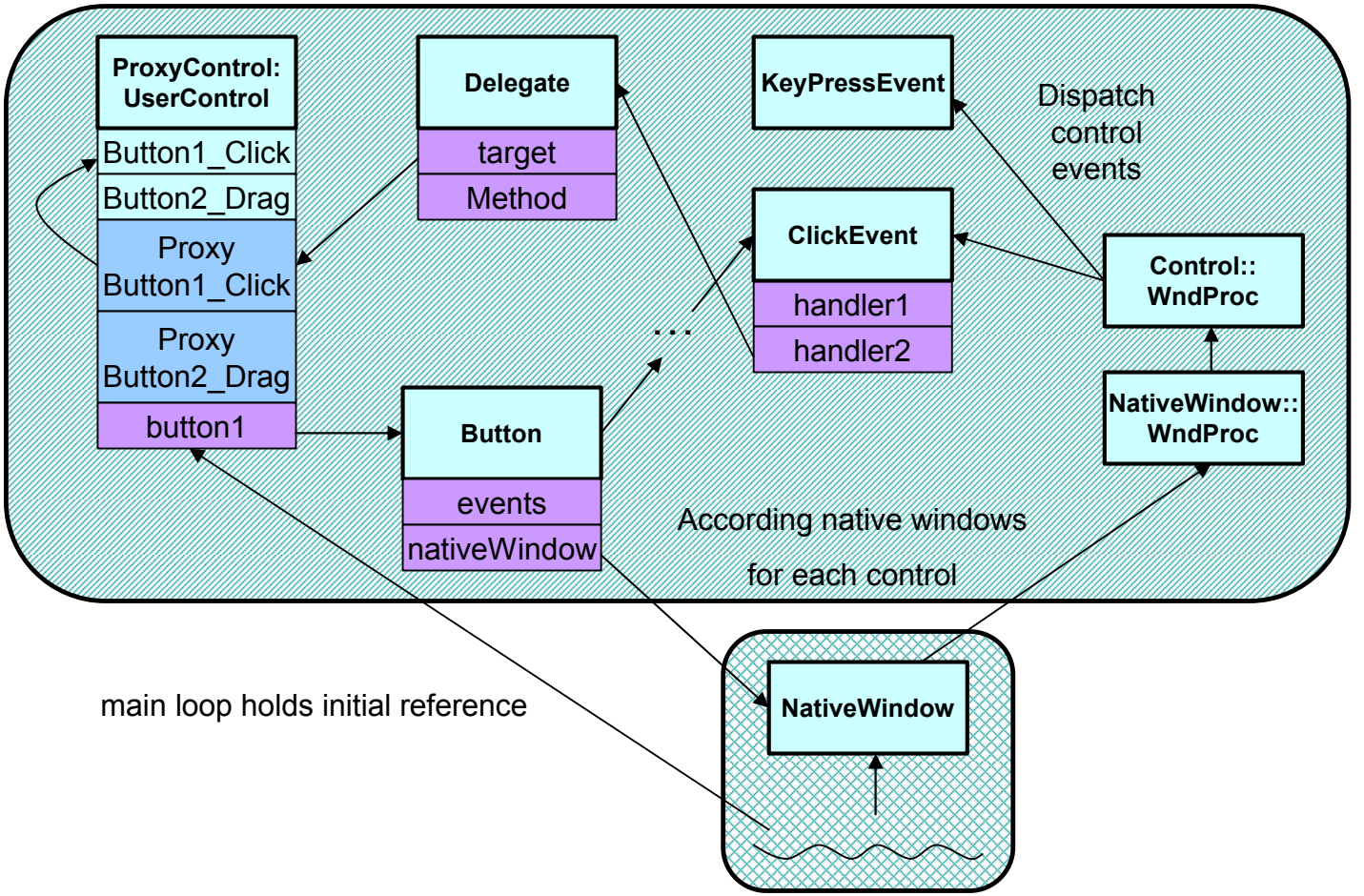


Traversing the Object Graph II



- Cycle recognition (visited nodes)
- Creation of new types (no constructor execution)
- Dynamic assembly loading (shadow copies)
- Arrays (update type and content)
- Delegates (update target and method)
- Generics (update bound types)
- Type and assembly objects
- Activation/deactivation/update of aspects

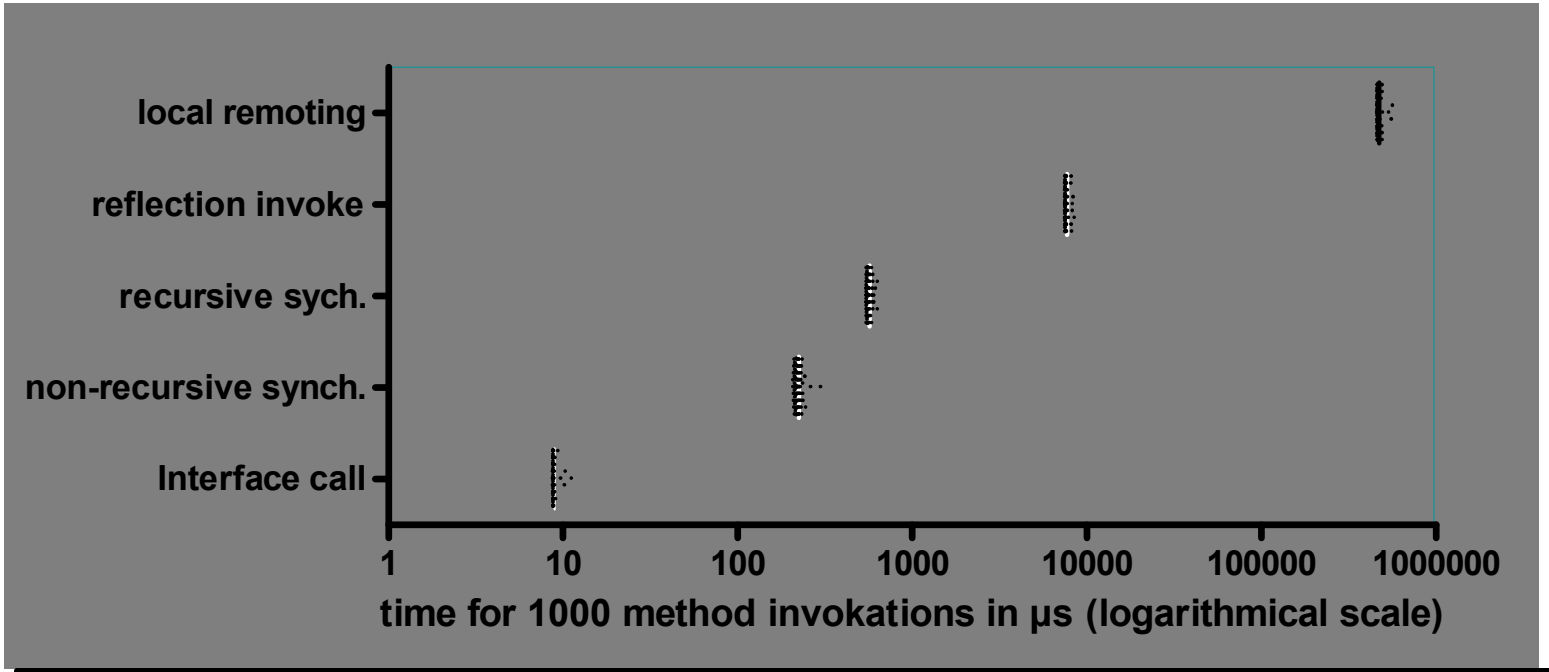
Graphical Components in .NET



Updating Graphical Components

- Deferred Update: UserControls/Forms can only be manipulated in the thread that created it
- Creation of reference control with constructor run (Initialization of GUI layout)
  - Addition/removal of sub-controls
  - Addition/removal of event handler
  - Update of event handler (delegates)

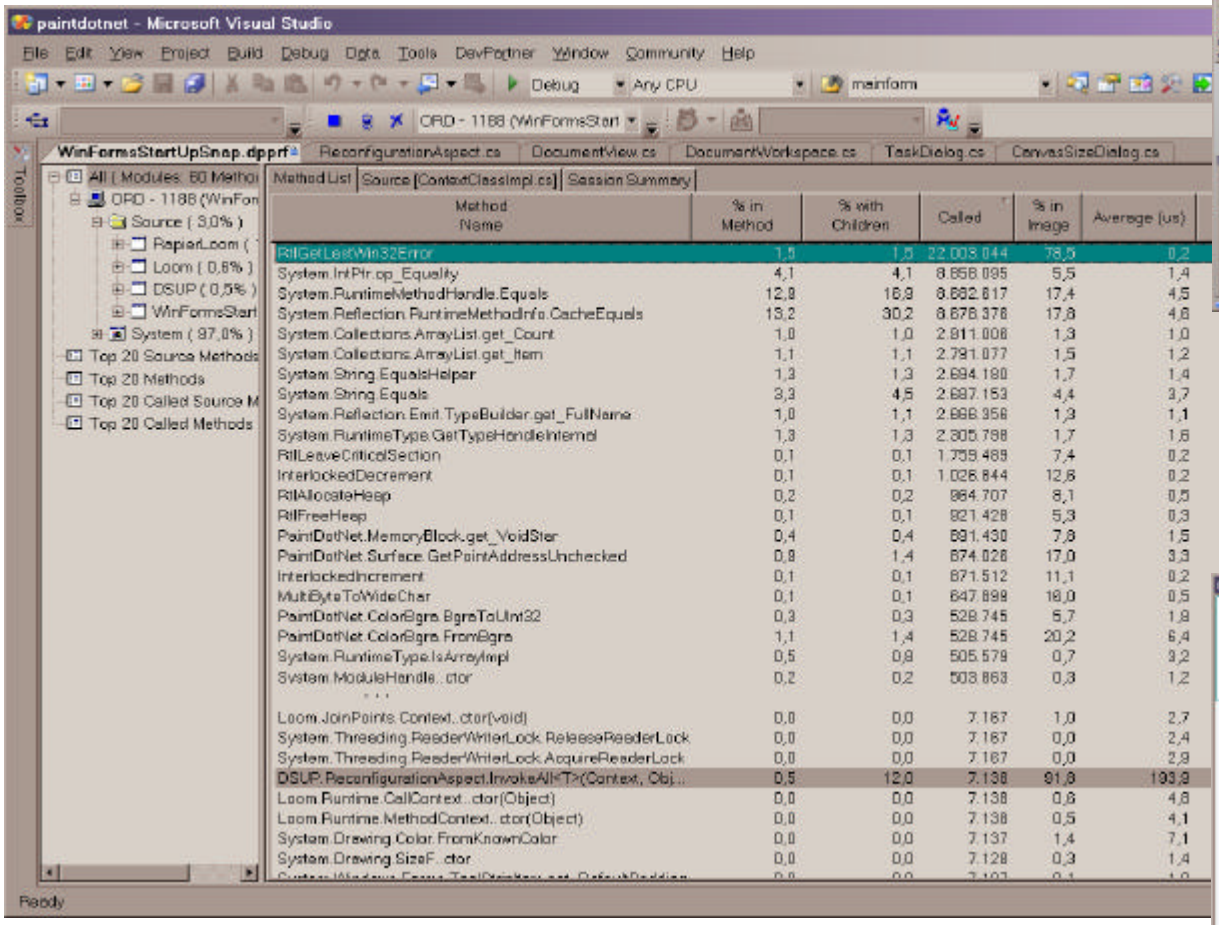
Evaluation: Method call overhead

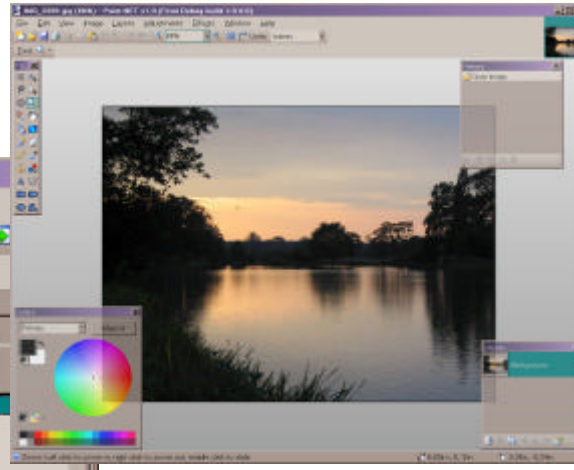


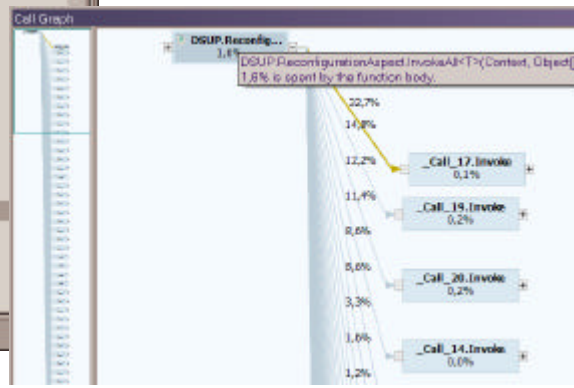
in-process .NET **remoting call**	474610,7±5872,8µs
normal .NET **reflection invoke** call	7709,7±1996 µs
Rapier-Loom aspect for synch. of **recursive comps**. (rw-lock)	578,8±24,3µs
Rapier-Loom aspect for synch. of **non-recursive** components	226,3±7,9µs
normal **interface call**	9,0±0,1µs

Xeon 2,8 GHz 1 CPU, 2 GB RAM, Windows XP Sp2, .NET 2.0,
time for 1000 method invokations of int Count(int c), 100 measurements, 1st. skipped

It's not that bad ...

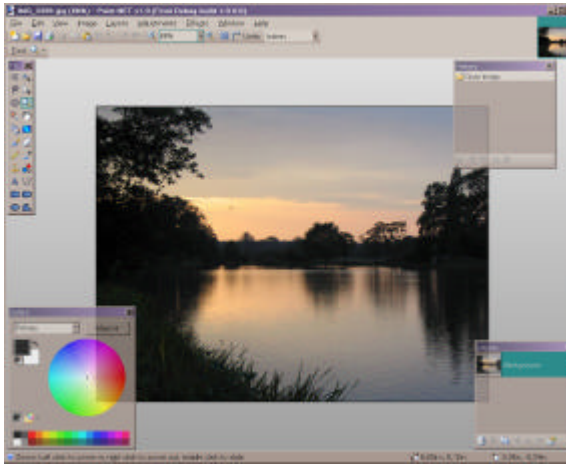






- Only a few method invokations are synchronized / on interwoven (root-) objects

Evaluation: PaintDotNet



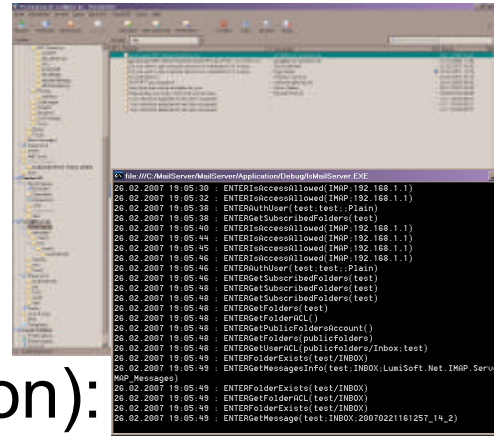
- Free open source C# image editor
- >133.000 lines of code
- Update statistics (fixing a small bug):

Update time	3,52 ± 0,20 sec.
Traversed Nodes	ca. 28.000
Updated objects	ca. 200
Handled leaf nodes	ca. 700.000

- Changed lines of code: ca. 30
- Synchronization overhead: not noticable

Xeon 2,8 GHz 2 CPUs, 2 GB RAM, Windows XP Sp2, Microsoft .NET 2.0, PaintDotNet 3.0

Evaluation: Lumisoft Mail-Server



- Freeware implemented in C#
- Ca. 61.000 lines of code
- Update statistics (trace aspect activation):

Update time	213 ± 7,5 ms
Traversed Nodes	ca. 1.326
Updated objects	ca. 3

- Overhead Evaluation: IMAP-Throughput (Fetchmail:10kByte)
  - Without synch. aspect: 45,1 ± 3,69 fetches in 10 seconds
  - With synch. aspect:45,24 ± 4,49 fetches in 10 seconds

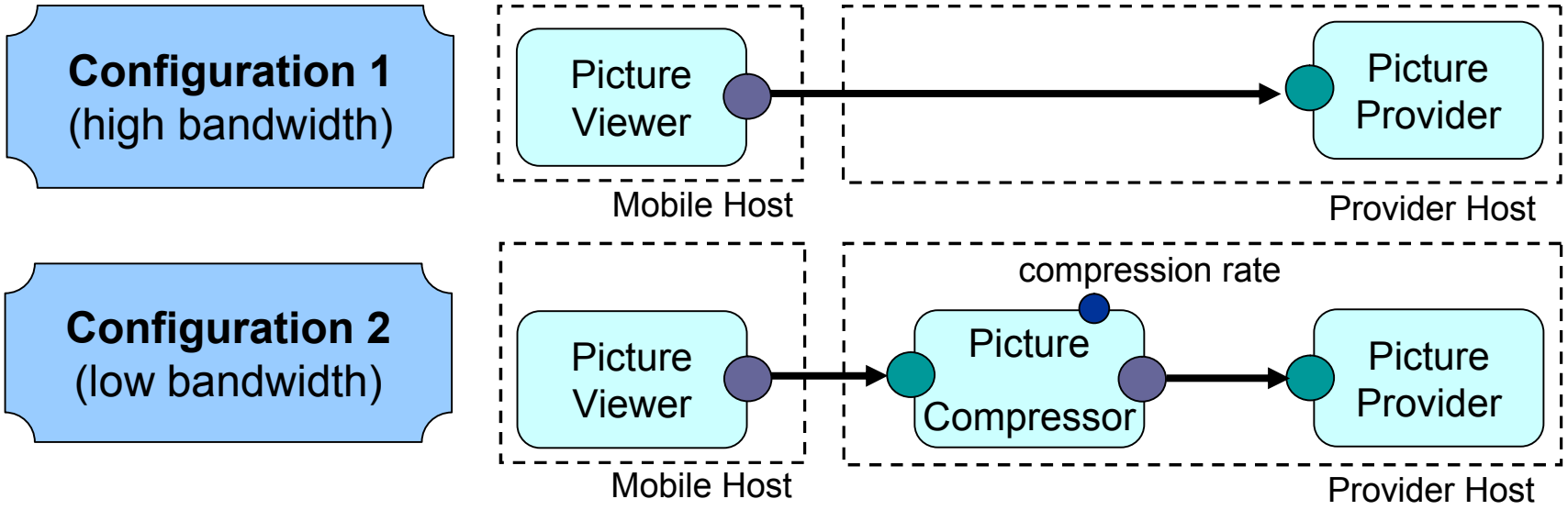
Xeon 2,8 GHz 2 CPUs, 2 GB RAM, Windows XP Sp2, Microsoft .NET 2.0, MailServer Version 0.88

http://www.lumisoft.ee/lswww/download/downloads/MailServer/Devel/

Evaluation: PictureShow

- Small picture viewer implemented in C#
- Update statistics (trace aspect activation):

Update time	110 ± 1 ms
Traversed Nodes	ca. 596
Updated objects	1



Conclusions

- Dynamic software updates on unchanged .NET execution environment
  - Applicable to complex third-party software components
  - Almost no noticeable synchronization overhead
  - Short blackout times for component updates
- Usage of aspect-oriented programming for adding configuration-specific concerns promising
- Adapt.Net supports runtime updates of graphical components

http://www.dcl.hpi.uni-potsdam.de

BackUp

Erlaubte Änderungen in neuer Version

- Hinzufügen von Methoden und Properties in benutzte Klassen
- Implementierung neuer Interfaces in existierenden Klassen
- Hinzufügen neuer Klassen in die Assemblies
- Definition neuer Interfaces, Structs, Enums in existierenden Assemblies
- Ändern/Hinzufügen/Entfernen von Methodenparametern in existierenden Methoden, wenn alle Aufrufe angepasst werden.
- Änderung der Implementierung existierender Methoden/Properties (Änderung/Hinzufügen/Entfernen beliebiger il-Codes innerhalb der Methoden/Properties)
- Hinzuzufügen / Entfernen / Ändern von Attributen
- Hinzufügen von Aspekten
- Hinzufügen neuer Assemblies
- Entfernen von Assemblies

Zunächst nicht erlaubt

- Ändern von Memberdatentypen
- Ändern von Statischen Variablen
- Hinzufügen von Membern
- Hinzufügen statischer Variablen
- Entfernen von Membern
- Entfernen Statischer Variablen
- Vererbungshierarchie von Klassen zu verändern

Einschränkung an die Komponenten

- Sie verwenden keinen unsafe Code.
- Sie verwenden kein P/Invoke.
- Sie enthalten .NET 2.0 konforme Programme
- Die Hauptklasse einer Komponente darf nicht sealed oder internal sein sein
- Die Property System.Reflection.Assembly.Location wird nicht benutzt (da sie durch DSUP verändert wird)
- Versionierung der Assemblies muss aktiviert sein
- Objekt.GetHashCode darf nicht überschrieben sein