
Dynamic Updates of Components in the .NET Framework

Andreas Rasche and Wolfgang Schult

Operating Systems & Middleware Group

Prof. Dr. Andreas Polze

Hasso-Plattner-Institute

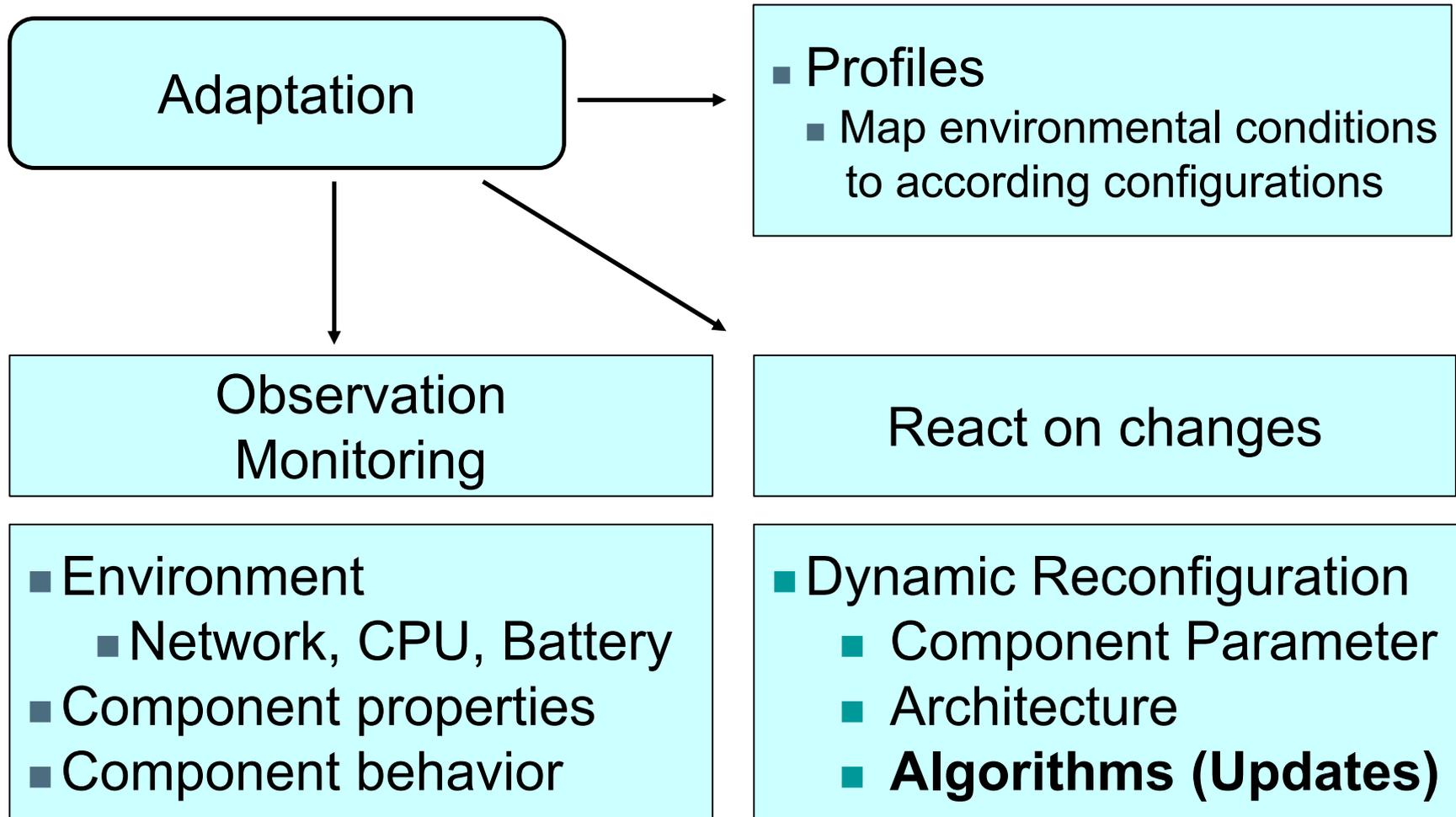
University of Potsdam



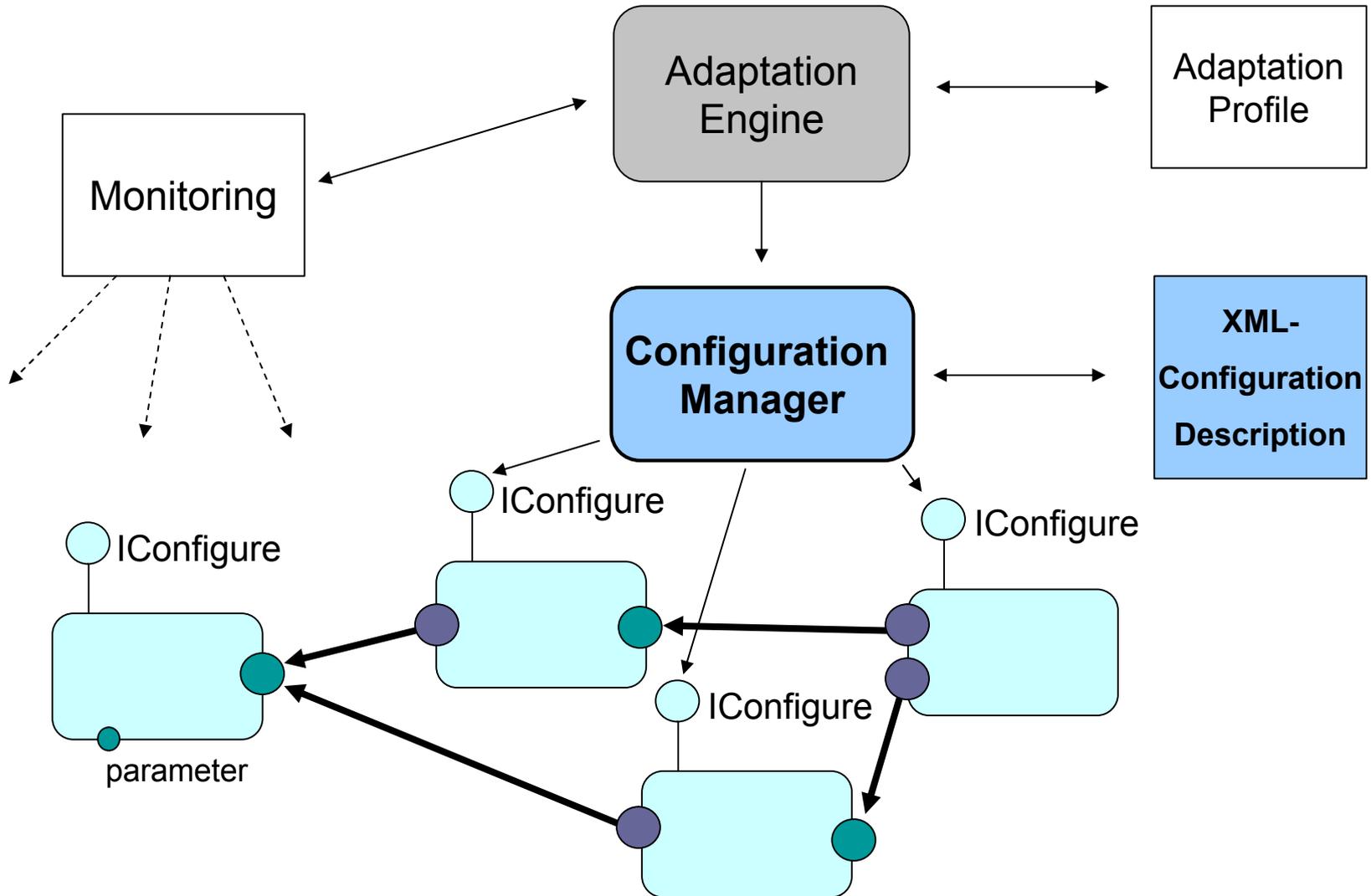
Outline

- 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

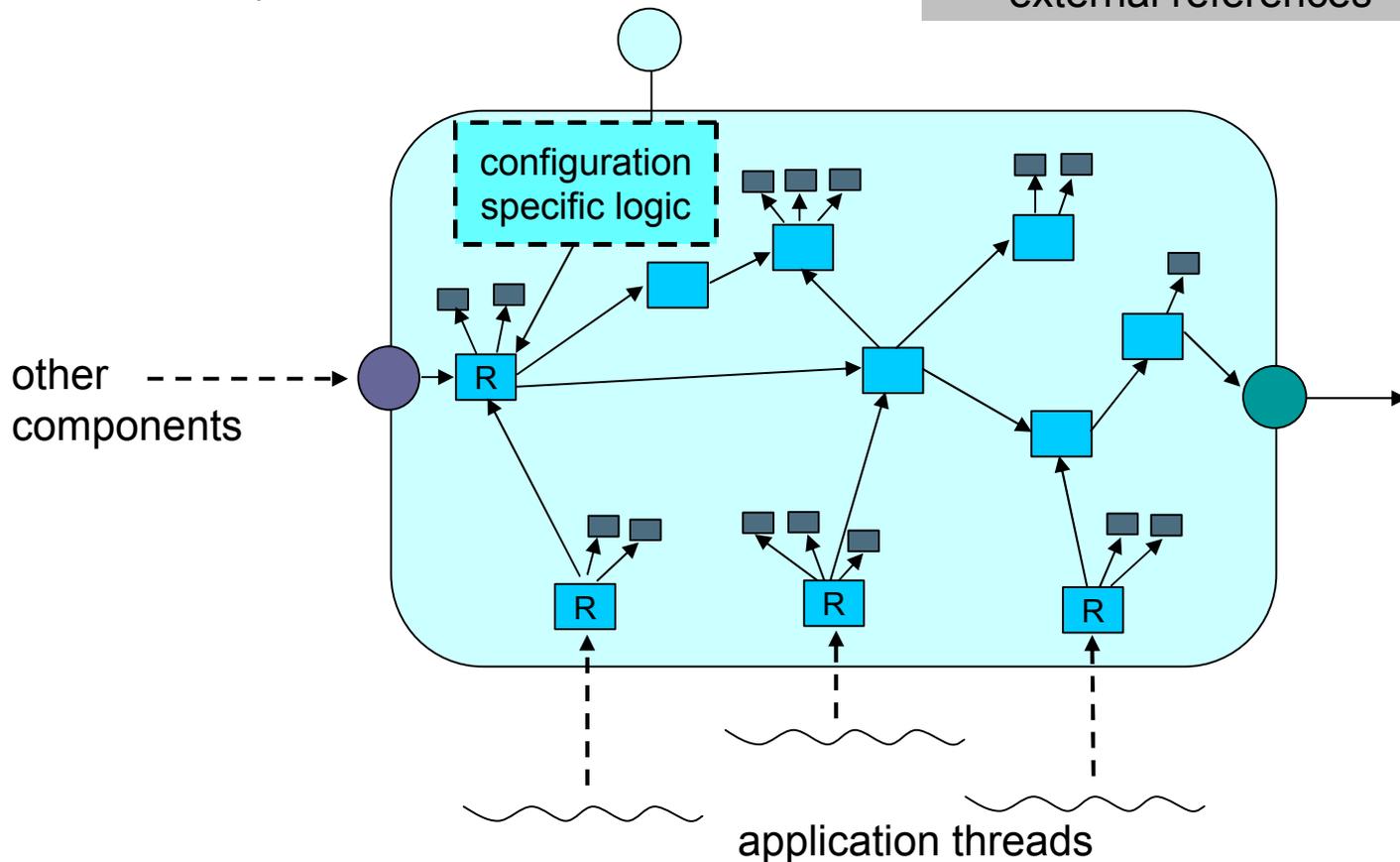
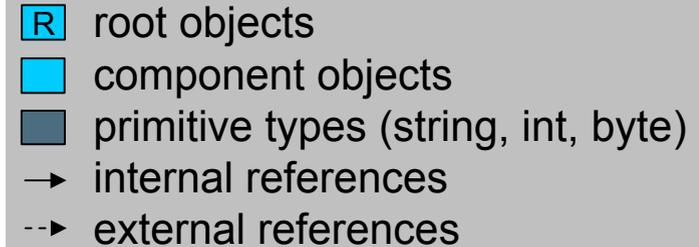


Dynamic Component Updates

- 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

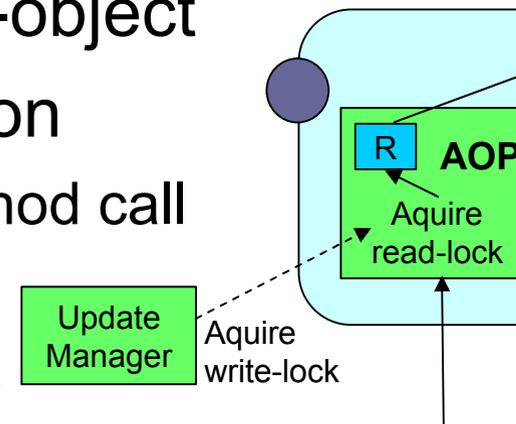
Components & Root Objects

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

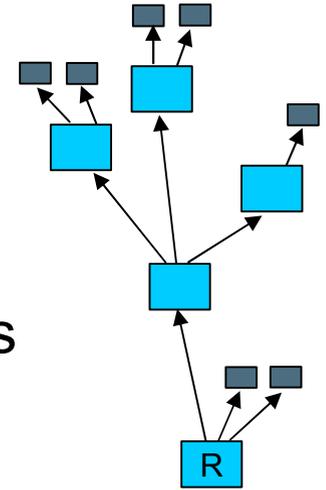


Reaching a reconfigurable state

- 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



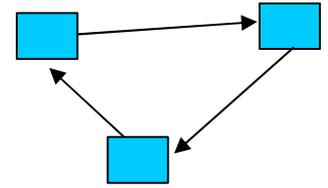
- Start from all root objects
- For each field of all objects traverse all references
- In case of an update:
 - Create an instance of the new version
 - Copy the state by transferring all fields from the old to the new instance
 - For reference fields: traverse target first and install potential new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

MyObject V1.0
Cat: „Blacki“
Nr: 1
Weight: 4,545

```
Object temp = oldObj.GetValue(„Weight“);  
newObj.SetValue(„Weight“,temp);
```

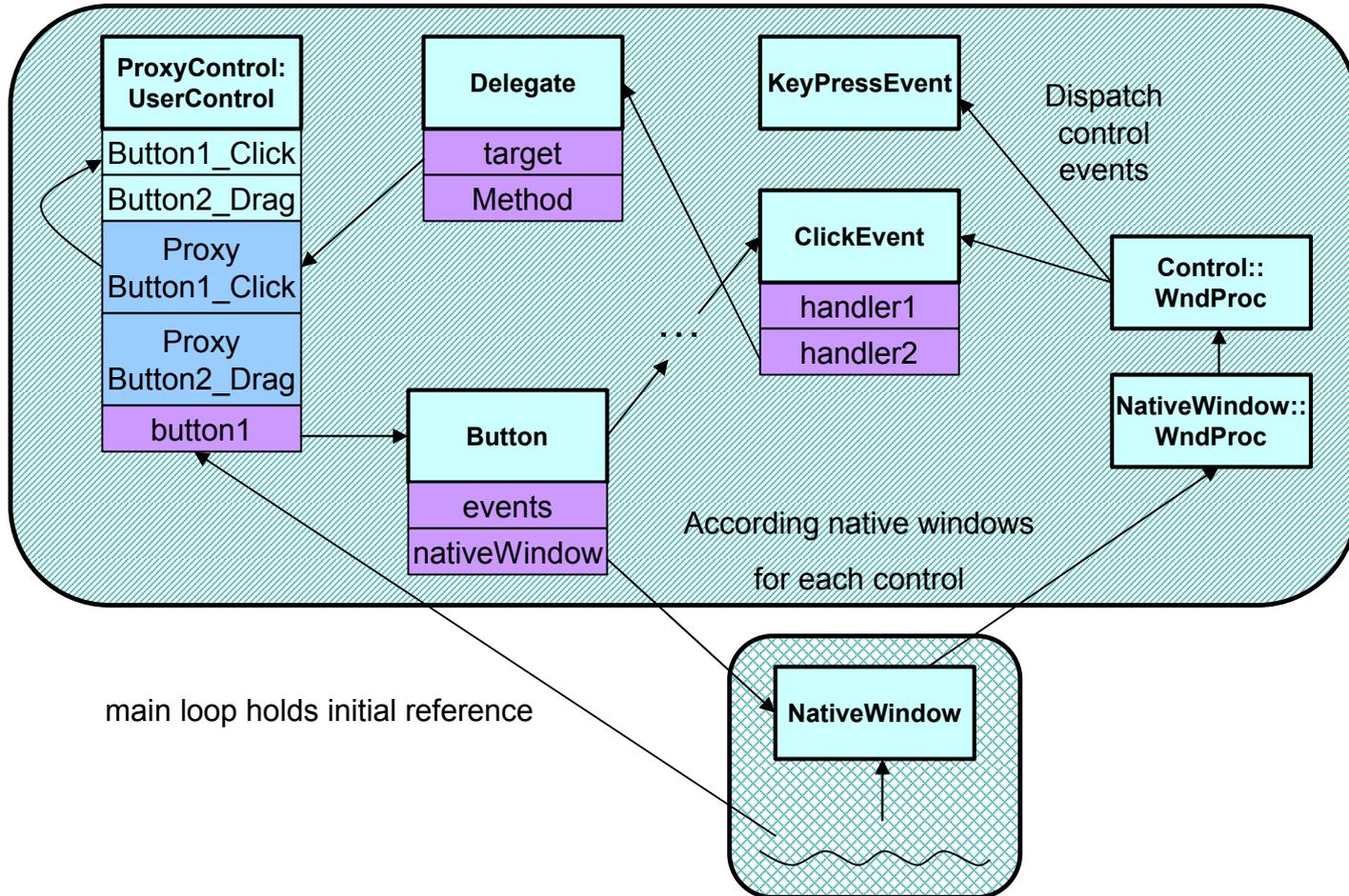
MyObject V2.0
Cat: „Blacki“
Nr: 1
Weight: 0

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

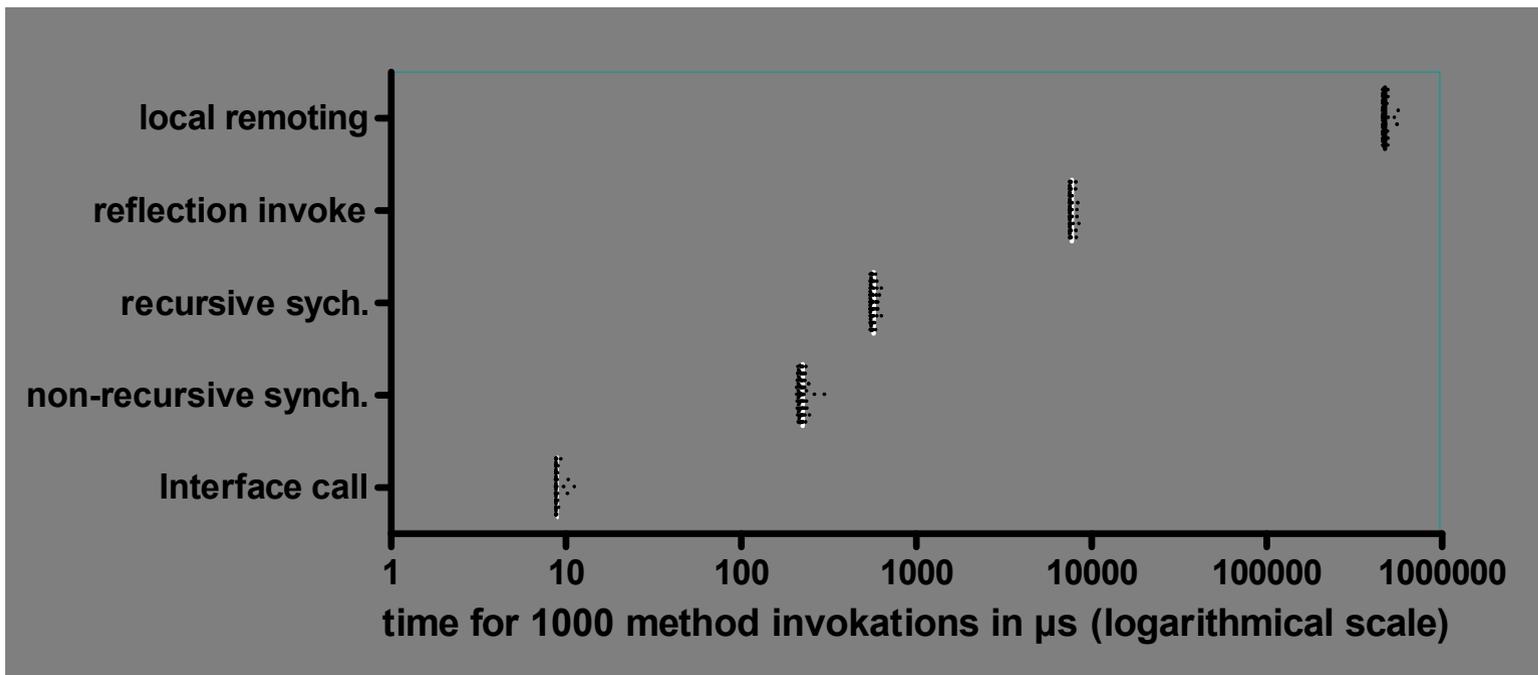
Graphical Components in .NET



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 invocations of int Count(int c), 100 measurements, 1st. skipped

It's not that bad ...

Method Name	% in Method	% with Children	Called	% in Image	Average (us)
CallGetLastWin32Error	1,5	1,5	32.005.044	78,5	0,2
System.IntPtr.op_Equality	4,1	4,1	8.658.035	5,5	1,4
System.RuntimeMethodHandle.Equals	12,9	18,9	8.662.617	17,4	4,5
System.Reflection.RuntimeMethodInfo.CacheEquals	13,2	30,2	8.678.378	17,8	4,6
System.Collections.ArrayList.get_Count	1,0	1,0	2.811.008	1,3	1,0
System.Collections.ArrayList.get_Item	1,1	1,1	2.791.077	1,5	1,2
System.String.EqualsHelper	1,3	1,3	2.694.180	1,7	1,4
System.String.Equals	3,3	4,5	2.687.153	4,4	3,7
System.Reflection.Emit.TypeBuilder.get_FullName	1,0	1,1	2.686.358	1,3	1,1
System.RuntimeType.GetTypeHandleInternal	1,3	1,3	2.805.798	1,7	1,6
CallLeaveCriticalSection	0,1	0,1	1.739.488	7,4	0,2
Interlocked.Decrement	0,1	0,1	1.026.844	12,6	0,2
CallAllocateHeap	0,2	0,2	864.707	8,1	0,5
CallFreeHeap	0,1	0,1	821.428	5,3	0,3
PaintDotNet.MemoryBlock.get_VoidStar	0,4	0,4	891.430	7,8	1,5
PaintDotNet.Surface.GetPaintAddressUnchecked	0,8	1,4	674.026	17,0	3,3
Interlocked.Increment	0,1	0,1	871.512	11,1	0,2
MultiByteToWideChar	0,1	0,1	847.899	18,0	0,5
PaintDotNet.ColorBgra.BgraToUInt32	0,3	0,3	528.745	5,7	1,8
PaintDotNet.ColorBgra.FromBgra	1,1	1,4	528.745	20,2	6,4
System.RuntimeType.IsArrayType	0,5	0,8	505.579	0,7	3,2
System.ModuleHandle..ctor	0,2	0,2	503.863	0,3	1,2
...					
Loom.JoinPoints.Context..ctor(void)	0,0	0,0	7.167	1,0	2,7
System.Threading.ReaderWriterLock.ReleaseReaderLock	0,0	0,0	7.167	0,0	2,4
System.Threading.ReaderWriterLock.AcquireReaderLock	0,0	0,0	7.167	0,0	2,8
DSUP.ReconfigurationAspect.InvokeAll*?<Context, Object>	0,5	12,0	7.138	91,8	193,9
Loom.Runtime.CallContext..ctor(Object)	0,0	0,0	7.138	0,8	4,8
Loom.Runtime.MethodContext..ctor(Object)	0,0	0,0	7.138	0,5	4,1
System.Drawing.Color.FromKnownColor	0,0	0,0	7.137	1,4	7,1
System.Drawing.SizeF..ctor	0,0	0,0	7.128	0,3	1,4
ContextWindowForm.ToolPalette.get_DefaultPadding	0,0	0,0	7.107	0,1	1,0

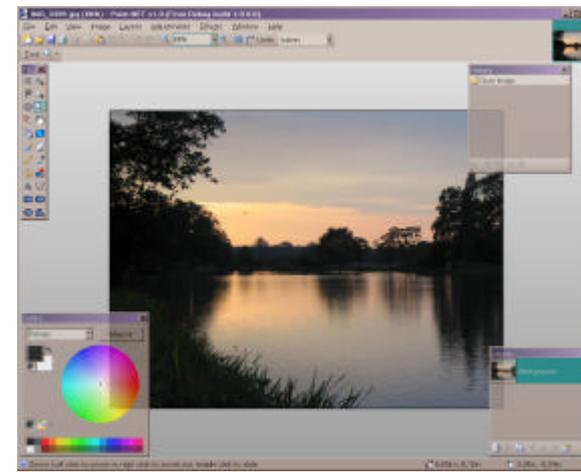
Call Graph

- DSUP.ReconfigurationAspect.InvokeAll*?<Context, Object> (1,8%)
 - 1,8% is spent by the function body.
 - 22,7% to _Call_17.Invoke (0,1%)
 - 12,2% to _Call_19.Invoke (0,2%)
 - 11,4% to _Call_20.Invoke (0,2%)
 - 9,6% to _Call_26.Invoke (0,2%)
 - 5,6% to _Call_28.Invoke (0,2%)
 - 3,3% to _Call_14.Invoke (0,0%)
 - 1,0% to _Call_14.Invoke (0,0%)
 - 1,2% to _Call_14.Invoke (0,0%)

- Only a few method invocations 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

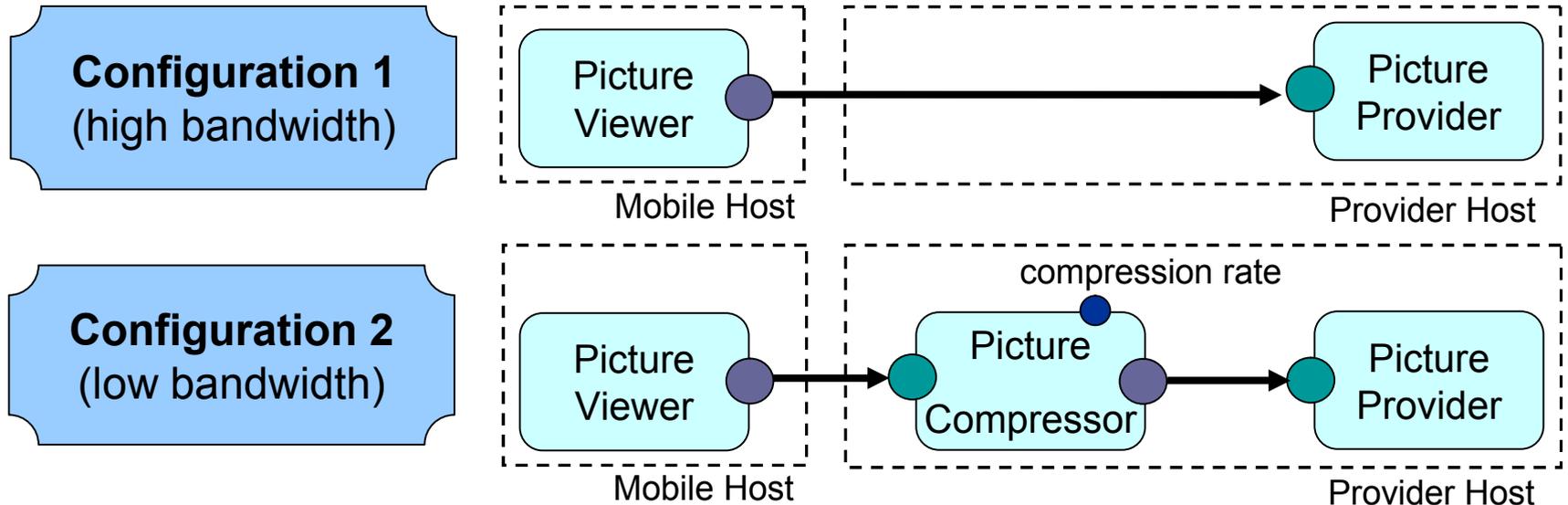
- Changed lines of code: ca. 30
- Synchronization overhead: not noticeable

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 Mitgliedern
- Hinzufügen statischer Variablen
- Entfernen von Mitgliedern
- 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
- 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