On the Move to a Homogeneous Programming Model for Replicated and Distributed Objects?

<table>
<thead>
<tr>
<th>Jörg Domaschka</th>
<th>Hans P. Reiser</th>
<th>Rüdiger Kapitza</th>
<th>Franz J. Hauck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ulm University</td>
<td>University of Lisboa</td>
<td>FAU Erlangen</td>
<td>Ulm University</td>
</tr>
<tr>
<td>Germany</td>
<td>Portugal</td>
<td>Germany</td>
<td>Germany</td>
</tr>
</tbody>
</table>
Motivation: Why Replication? Why Objects?

Why Replication?
- Increased reliability
- Increased availability
- Increased performance
Motivation: Why Replication? Why Objects?

Why Replication?
- Increased reliability
- Increased availability
- Increased performance

But:
- Additional costs (hardware)
- Overhead due to inter-replica communication
- Overhead due to runtime overhead
- Restricted programming model
Motivation: Why Replication? Why Objects?

Why Replication?
- Increased reliability
- Increased availability
- Increased performance

But:
- Additional costs (hardware)
- Overhead due to inter-replica communication
- Overhead due to runtime overhead
- Restricted programming model

⇒ Is there a way to circumvent these problems?
Motivation: Why Replication? Why Objects?

Why Objects?
Motivation: Why Replication? Why Objects?

**Why Objects?**
- Why not?
- Legacy code
- Realise a “natural“ and homogeneous system
Motivation: Why Replication? Why Objects?

Why Objects?
• Why not?
• Legacy code
• Realise a “natural“ and homogeneous system

But:
• Current OO middleware is unflexible
• Objects are either local or remote
• “Services“ might require both
Motivation: Why Replication? Why Objects?

Why Objects?
- Why not?
- Legacy code
- Realise a “natural“ and homogeneous system

But:
- Current OO middleware is unflexible
- Objects are either local or remote
- “Services“ might require both

⇒ Is there a way to circumvent these problems?
Use case: A Distributed Code Repository

- Remote repository: shared code base

Remote Repository
Use case: A Distributed Code Repository

• Remote repository: shared code base
• Local repository: track local changes
Use case: A Distributed Code Repository

- Remote repository: shared code base
- Local repository: track local changes
Use case: A Distributed Code Repository

- Remote repository: shared code base
- Local repository: track local changes
- Provide a single service ➔ One object
Use case: A Distributed Code Repository

- Remote repository: shared code base
- Local repository: track local changes
- Provide a single service ➔ One object
Use case: A Distributed Code Repository

- Remote repository: shared code base
- Local repository: track local changes
- Provide a single service ➔ One object
Use case: A Distributed Code Repository

- Remote repository: shared code base
- Local repository: track local changes
- Provide a single service
  ➔ One object

➔ How to obtain transparency for the application developer?
Outline

• Motivation

• **Fragmented Objects:**
  Beyond Remote Method Invocation
  • Semantic Annotations:
    Increased Flexibility for Code Generation
  • **FTflex:**
    Replication Support in FO Middleware
  • Deterministic Multithreading:
    Overcoming Deadlocks and Performance Bottlenecks

• Conclusion

• Work in Progress
• Discussion
Distributed Objects – Remote Objects

Object Implementation:
- Actual functionality
- Well-defined interface
- Represented by remote reference

Object Adapter:
- (un)marshalling, dispatching

Stub/Proxy:
- (un)marshalling
- Location transparency
- Same interface as object implementation
- Generated or created at run-time

Middleware:
- Binding: maps remote reference to stub
Distributed Objects – Fragmented Objects

- Model for *really* distributed objects
  - Fragmented object represented by local fragments
  - Unit with well-defined interface
  - FO not bound to fixed location

⇒ No fixed internal structure
Which Fragment to Load: Extended Binding

**Standard middleware:**
- Default stub associated with reference
Which Fragment to Load: Extended Binding

**Standard middleware:**
- Default stub associated with reference

[Diagram showing remote reference and middleware connection]
Which Fragment to Load: Extended Binding

**Standard middleware:**
- Default stub associated with reference

```
1: Remote Reference
```

```
2: create
```

```
Middleware
```

```
Stub
```

```
Application
```
Which Fragment to Load: Extended Binding

**Standard middleware:**
- Default stub associated with reference

![Diagram](attachment:image.png)

1: Remote Reference
2: create
3: invoke

Application
Middleware
Stub
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment

Diagram:
- Application
  - 1: Remote Reference
  - middleware
    - 2: find
    - DLS
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment

1: Remote Reference

Application

Middleware

DLS

Policy Fragment

3: create

2: find
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment

*Diagram:*
1. Remote Reference
   - Application
   - Middleware
   - Policy Fragment
   - DLS
   - Create
   - Find
   - Find

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment

1. Remote Reference
2. Find
3. Create
4. Find
5. Create

Diagram:
- Application
- Middleware
- Policy Fragment
- Local Fragment
- DLS
Which Fragment to Load: Extended Binding

**FO middleware:**
- Dynamic environments
- Use a dynamic loading service
- Multiple fragments per type
- Pre-load policy fragment

![Diagram showing the process of finding and loading fragments]

1. Remote Reference
2. Find
3. Create Middleware
4. Find Policy Fragment
5. Create Local Fragment
6. Invoke
Fragmented Objects vs. Remote Objects

**Fragmented Objects:**
- Functionality distributed arbitrarily
- State distributed arbitrarily
- Arbitrary inter-fragment communication
  - E.g. streaming
  ➔ Highly flexible and configurable
  ➔ Runtime reconfiguration
Fragmented Objects vs. Remote Objects

**Fragmented Objects:**
- Functionality distributed arbitrarily
- State distributed arbitrarily
- Arbitrary inter-fragment communication
  - E.g. streaming
→ Highly flexible and configurable
→ Runtime reconfiguration

**Remote Objects:**
- Functionality in on location
- State in one location
- Fixed interaction pattern
  - Client-server
  - Enables code generation
→ Less development effort
→ Sufficient for many use cases
Fragmented Objects vs. Remote Objects

**Fragmented Objects:**
- Functionality distributed arbitrarily
- State distributed arbitrarily
- Arbitrary inter-fragment communication
  - E.g. streaming
  ➔ Highly flexible and configurable
  ➔ Runtime reconfiguration

**Remote Objects:**
- Functionality in on location
- State in one location
- Fixed interaction pattern
  - Client-server
  - Enables code generation
  ➔ Less development effort
  ➔ Sufficient for many use cases

➔ Combine both approaches
  - Standard use cases: remote objects
  - Fragmented objects when required
Fragmented Objects vs. Remote Objects

**Fragmented Objects:**
- Functionality distributed arbitrarily
- State distributed arbitrarily
- Arbitrary inter-fragment communication
  - E.g. streaming

→ Highly flexible and configurable
→ Runtime reconfiguration

**Remote Objects:**
- Functionality in on location
- State in one location
- Fixed interaction pattern
  - Client-server
  - Enables code generation

→ Less development effort
→ Sufficient for many use cases

→ Combine both approaches
  - Standard use cases: remote objects
  - Fragmented objects when required

→ Middleware shall support both models
  - Aspectix: FO in CORBA
  - FORMI: FO in Java RMI
Replication with Fragmented Objects

Advantages:
• Easy to integrate in FO middleware
• Access transparency
Outline

• Motivation

• Fragmented Objects:
  Beyond Remote Method Invocation

• **Semantic Annotations:**
  Increased Flexibility for Code Generation

  • Ff lex:
    Replication Support in FO Middleware

  • Deterministic Multithreading:
    Overcoming Deadlocks and Performance Bottlenecks

• Conclusion

• Work in Progress

• Discussion
Development Process

FOs can have an internal structure
• Developer support feasible

Keep development process simple:
1. Interface definition
2. Code generator: (un)marshalling and dispatching code
3. Implement logic
Development Process – Extensions

Object provides service to application
- Service might require local resources
- Structural annotations extend interface definition
  - local: local execution
    - E.g. read from filesystem
  - private: not part of the object interface, yet available at server
    - Multiple local methods use identical server-side functionality
    - breaks encapsulation
  - intercepted: private + local
    - E.g. for caching, encryption
  - Allows to treat service as a black box
Why would you want that?

Use case: A distributed code repository

Replica Fragment -> Smart Stub

Replica Fragment

Replica Fragment

Middleware controlled

Local Repository

Application
Why would you want that?

Use case: A distributed code repository
Outline

• Motivation

• Fragmented Objects:
  Beyond Remote Method Invocation
• Semantic Annotations:
  Increased Flexibility for Code Generation
• FTflex:
  Replication Support in FO Middleware
  Deterministic Multithreading:
  Overcoming Deadlocks and Performance Bottlenecks

• Conclusion

• Work in Progress
• Discussion
FTflex – Overview

System Model:
- Synchronous request-reply interaction
- Each request executed in its own thread
- Replicas can fail with fail-stop
- New replicas can be added dynamically
- Threads communicate exclusively by shared state
- Exclusive access to shared state due to mutexes
On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?

**FTflex Architecture – Client Side**

- Application
  - Access Fragment
    - custom code
    - (un)marshalling
    - failure transparancy
    - consistency
    - invocation
    - replica tracking
On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?

**FTflex Architecture – Replica Side**

- Replica Fragment
  - Scheduler
  - Object Impl
  - Object Adapter
  - Nested Invocation Handling
- Client Comm
  - Replication Protocol
  - Group Comm
  - Membership
On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?

**FTflex Architecture**

Fragmented Object

Access Fragment

Replica Fragment

Replica Fragment

FTflex
Replication Protocols

- **Active Replication**
  - All replicas execute all requests
  - Requires deterministic implementation
  - Consistency: order on conflicting requests
  - Nested invocations become a problem (SOA)

- **Passive replication**
  - One replica executes request
  - Forwards changes
  - Allows non-deterministic implementations
  - Nested invocations still a problem
Outline

• Motivation

• Fragmented Objects: Beyond Remote Method Invocation

• Semantic Annotations: Increased Flexibility for Code Generation

• FT\textit{flex}: Replication Support in FO Middleware

• \textbf{Deterministic Multithreading: Overcoming Deadlocks and Performance Bottlenecks}

• Conclusion

• Work in Progress

• Discussion
Consistency

Replication: keep replicas in consistent states

- Start in same state
- Exclusively use deterministic operations
  - Handle non-deterministic parts like nested invocations
- Execute conflicting methods in same order
  - No semantical knowledge: all methods conflict
  - Ordering by atomic multicast
- Relaxed requirements for read-only methods
  - No need for ab-cast
  - `readonly` keyword in interface definition
Concurrency

Concurrency might destroy determinism
Concurrency

Concurrency might destroy determinism

- Strict sequential execution
  - Deadlock prone
Concurrent Programming Models

Concurrency might destroy determinism

- Strict sequential execution
  - Deadlock prone

Indirect recursion

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Concertivity

Concurrency might destroy determinism

⇒ Strict sequential execution
  • Deadlock prone

indirect recursion

\[
\begin{align*}
A & \xrightarrow{1:m_{A_1}} B \\
& \xleftarrow{3:m_{A_2}} B
\end{align*}
\]

2:m_{R}

cross invocation

\[
\begin{align*}
A & \xrightarrow{1:m_{A}} B \\
& \xleftarrow{1':m_{B}} B
\end{align*}
\]

2:m_{BA}

2':m_{BA}
Concurrence
Concurrence might destroy determinism

- Strict sequential execution
  - Deadlock prone

indirect recursion

\[ A \xrightarrow{1:m_{A1}} B \xrightarrow{2:m_B} A \xrightarrow{3:m_{A2}} B \]

cross invocation

\[ A \xrightarrow{1:m_A} B \xrightarrow{2:m_{BA}} A \xrightarrow{2':m_{BA}} B \]

condition variables

\[ A \xrightarrow{1:wait} B \xrightarrow{2:notify} A \]
Concurrency

Concurrency might destroy determinism

→ Strict sequential execution
  • Deadlock prone

- Poor performance
  • Idle time (nested invocations)
  • No use of multi-core CPUs
Concurrency

Concurrency might destroy determinism

- Strict sequential execution
  - Deadlock prone

- Poor performance
  - Idle time (nested invocations)
  - No use of multi-core CPUs

Other approaches?
Deterministic Scheduling

- Methods not considered atomic anymore
  - Computational blocks
  - Atomic blocks
  - Locking of atomic blocks as scheduling points

- Deterministic schedulers: Basile et al., Napper et al., Reiser et al.
  - Determinism by communication
  - Determinism by round-wise execution
  - Determinism by priorisation
Determinism by Communication

Loose-Synchronization Algorithm (LSA):
- Dedicated leader replica
- Uses arbitrary lock sequence
- Forwards lock sequence to followers
- Mutexes require global ID
- Additional communication costs
- Costly reconfiguration in case of failures
Determinism by Round-wise Execution

Preemptive-Deterministic Scheduling (PDS):
- Round-based
- $n$ threads run in parallel
- Each thread may request up to 2 locks per round
- Threads with non-conflicting locks may run in parallel

- $n$ is a configuration parameter: fixed
- Lack of new client requests impedes start of new rounds
- No decision about locks until $n$-th thread has requested lock
Determinism by Priorisation

Multiple Active Threads (MAT):

- Threads ordered according their arrival
- Thread with highest rank may request locks
- Others may not; lock request blocks thread

- Natively supports nested invocations
- Natively supports condition variable semantics of mutexes
- Takes only pessimistic decisions
- Degree of concurrency depends on implementation
MAT Algorithm

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Scheduler Integration

Goals:
- Homogeneous programming model
- Re-utilisation of legacy code
- Universally applicable

→ Source code transformation

Java:
- Find synchronized statements
- Replace block by invocations to scheduler
- Process with wait/notify accordingly

- Holds for all library classes, too
Conclusion
Conclusion

**Fragmented Object Model**
- Provides location transparency for functionality
- Semantic Annotations for flexible code generation
- Extended binding process for flexible instantiation
Conclusion

**Fragmented Object Model**

- Provides location transparency for functionality
- Semantic Annotations for flexible code generation
- Extended binding process for flexible instantiation

→ Functionality entirely encapsulated in one object
Conclusion

**Fragmented Object Model**
- Provides location transparency for functionality
- Semantic Annotations for flexible code generation
- Extended binding process for flexible instantiation

⇒ Functionality entirely encapsulated in one object

**Deterministic Multithreading**
- Re-introduces blocking operations on shared data
- Enables risk-free use of any object reference
- Software transformation for neatless integration
Conclusion

Fragmented Object Model
• Provides location transaparancy for functionality
• Semantic Annotations for flexible code generation
• Extended binding process for flexible instantiation

⇒ Functionality entirely encapsulated in one object

Deterministic Multithreading
• Re-introduces blocking operations on shared data
• Enables risk-free use of any object reference
• Software transformation for neatless integration

⇒ Less restrictions for service developer
Some Papers...

1. Integrating Fragmented Objects into a CORBA Environment  
   *(Net Object Days 2003)*
2. A flexible and extensible object middleware: CORBA and beyond  
   *(SEM 2005)*
3. Consistent Replication of Multithreaded Distributed Objects  
   *(SRDS 2006)*
4. Deterministic Multithreading for Replicated CORBA Objects  
   *(PDCS 2006)*
5. Fault-tolerant Replication Based on Fragmented Objects  
   *(DAIS 2006)*
6. FORMI: Integrating Adaptive Fragmented Objects into Java RMI  
   *(DSOnline, 10/2006)*
   *(DAIS 2007)*
Outline

• Motivation

• Fragmented Objects: Beyond Remote Method Invocation
• Semantic Annotations: Increased Flexibility for Code Generation
• FTflex: Replication Support in FO Middleware
• Deterministic Multithreading: Overcoming Deadlocks and Performance Bottlenecks

• Conclusion
• Work in Progress

• Discussion
Work in Progress

Replication Protocols:
• Integration and Evaluation of several protocols for passive replication

Concurrency/Multithreading:
• Can data flow analysis reduce pessimistic decisions?
• Support for other ways of synchronisation
  • Locking sequences that do not follow block structure
  • Lock-free synchronisation
  • Introduction of optimism

Reconfiguration:
• User initialised reconfiguration
• Adaptation to a changing environment
Object Replication Using Transactions

Idea:

- Synchronize critical regions using **optimistic** transactions
- Comparable to STMs for multiprocessor architectures
- Request executed at an arbitrary replica
- Decision about commit of a transaction by some protocol

Model:

- One method invocation corresponds to zero or more transactions

Benefits:

- Computational power of all replicas available
- Allows non-deterministic implementations
Object Replication Using Transactions (II)

Consistency:
- Has to deal with nested invocations
- Has to deal with the replica crashing between two transactions
- Each object requires unique, global ID

Questions:
- Heterogeneous programming model feasible?
- Performance comparable to existing replication protocols?
- ...

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Open Question

Condition Variable vs. Object Serialization:
- Absence of threads cannot be guaranteed
- No stable serialization point
- Starvation of newly joining replicas

Multithreading vs. Passive Replication:
- No stable serialization point
- Introduces consistency issues
- Solvable by logs and transaction semantics (rollbacks)?

Adaptability:
- Performance of replication protocol and scheduling depends on application scenario
- Switching at runtime feasible? Which heuristics to use?
- Can a model help?
Open Questions (II)

Concurrent Programming:
- STMs – based on transactions
- Without locks - mostly based on HW-primitives
- Replication feasible on application level?
- Introduction of read and write locks?

Programming Model:
- Allow developer to write Non-replicated programs
- Introduce adaptations by software transformation
Deterministic Schedulers – Evaluation – Bounded Buffers, get
On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?

Deterministic Schedulers – Evaluation – Comp

![Graph showing time per invocation vs. number of clients](image-url)
Deterministic Schedulers – Evaluation – LCU

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Deterministic Schedulers – Evaluation – CLU
Deterministic Schedulers – Evaluation – LUC
Deterministic Schedulers – Evaluation – Complex

On the Move to a Homogenous Programming Model for Replicated, Distributed Objects?
Deterministic Schedulers – Evaluation – Unbounded Buffer

The graph shows the time per invocation (in milliseconds) for different numbers of consumers. The graph compares different scheduling algorithms: SAT, PDS, LSA, MAT, and SEQ. The x-axis represents the number of consumers, ranging from 1 to 10, and the y-axis shows the time per invocation, ranging from 0 to 140 milliseconds. The graph indicates how the time per invocation increases with the number of consumers for each algorithm.