Presentation at TOOLS USA 2000

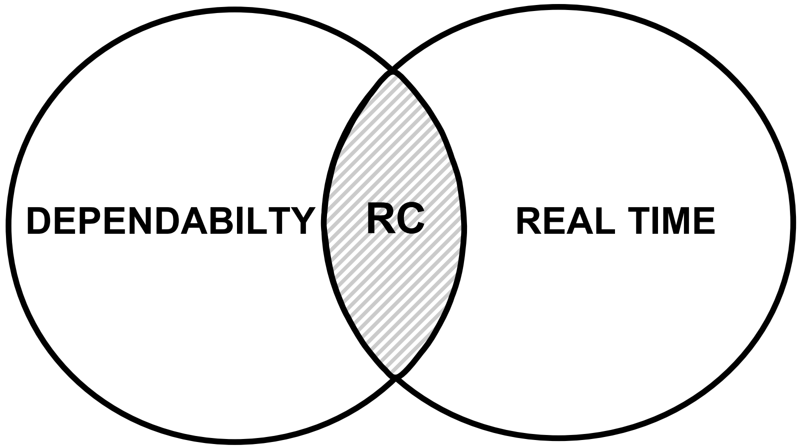
Automatic Generation of Fault-Tolerant CORBA-Services

Andreas Polze, Janek Schwarz and Miroslaw Malek
Department of Computer Science
Humboldt-University of Berlin
apolze@informatik.hu-berlin.de

Overview

- Motivation:
  - Fault-tolerant computing on off-the-shelf components
  - Standard middleware: CORBA
- Description of non-functional component properties
  - Fault-models and protocols
  - Aspect-oriented programming
- Case studies:
  - Automatic generation of fault-tolerant services
  - XML-based aspect description for component replication
- Conclusions

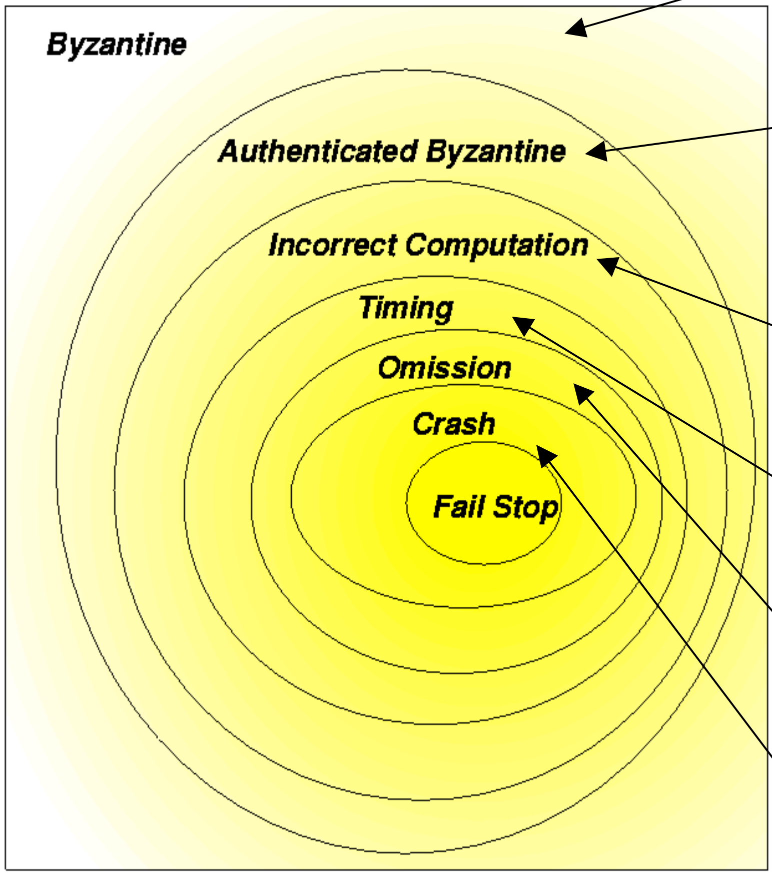
Responsive Computing



RESPONSIVE COMPUTER SYSTEMS

are dependable real-time systems, that deliver satisfactory service in a timely manner under given fault and load hypotheses.

Fault model at the component level



- Every possible fault. This class includes the authenticated Byzantine fault.
- PE behaves in an arbitrary or malicious manner, but is unable to imperceptibly change an authenticated message.
- PE fails to produce a correct output in response to a correct input.
- PE completes an assignment before or after its specified time frame or never.
- PE fails to meet a deadline or to begin a task.
- Processing element (PE) loses its internal state or halts. The processor is silent during the fault

Choosing the appropriate protocols

- A variety of protocols handle different fault classes.
  - Establish a consistent view onto system state (Consensus)
  - Among (non-faulty) processors
- Framework deals with:
  - **crash faults (of components of processors)**
  - **incorrect computation faults**
- The system maps timing and omission faults onto crash faults and stops a faulty CORBA component.
  - (due to limitations inherent in CORBA communication (IIOP))
- No detection mechanisms for Byzantine faults.

Problem: Description of a component‘s fault-assumptions/models

Description of non-functional Properties: Aspect-Oriented Programming

AspectJ: http://www.parc.xerox.com/spl/projects/aop/
Voyager ORB: http://www.objectspace.com

- Objects have been a great success (data-abstraction, encapsulation)
  - Functional-decomposition
- Objects don't seem to help as much for:
  synchronization, multi-object protocols, replication, resource sharing, distribution, memory management,
- Rather than staying well localized within a class, these concerns tend to cross-cut the system's class and module structure.
- Much of the complexity in existing systems appears to stem from the way in which the implementation of these kinds of concerns ends up being intertwined throughout the code.

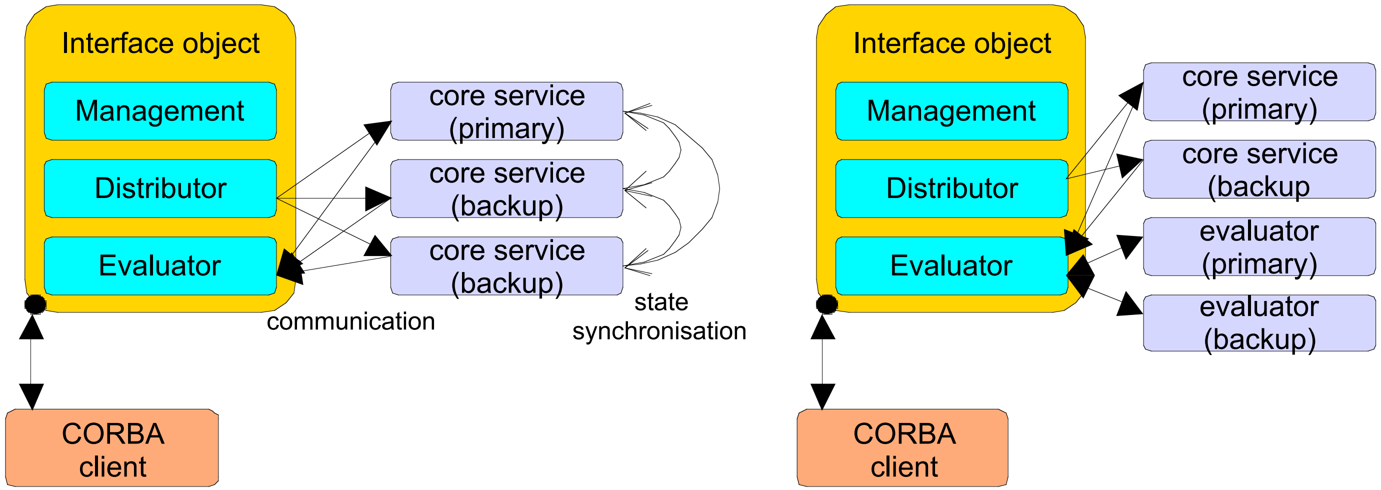
Aspects / Facets

- Aspects are a new unit of software modularity, that appears to provide a better handle on managing cross-cutting concerns.
- aspects are intended to be used in both design and implementation.
- During design the concept of aspect facilitates thinking about cross-cutting concerns as well-defined entities.
- During implementation, aspect-oriented programming languages make it possible to program directly in terms of design aspects.
- Promising way to describe non-functional component properties:
  - **fault-tolerance measures, resource constraints**
  - **timing behavior, security, mobility**

Case study: Automatic Generation of fault-tolerant CORBA Services

- Programmer implements sequential service and gives design time information about possible fault-tolerance measures
- Service configurator starts multiple copies of server objects based on chosen fault-model and available network nodes (replication in space vs. time)
- Client may request some fault tolerance level with each request and depending on actual service configuration the request is either fulfilled or an exception returned
- GUI for service configuration; NT-based implementation

Component Model for a Fault-tolerant Service



- Design-time (programming) vs. Runtime (crash) faults
- Analytic redundancy + consensus protocols
- Hot/warm/cold replication:
  - Group comm., checkpointing to memory/disk

Design time

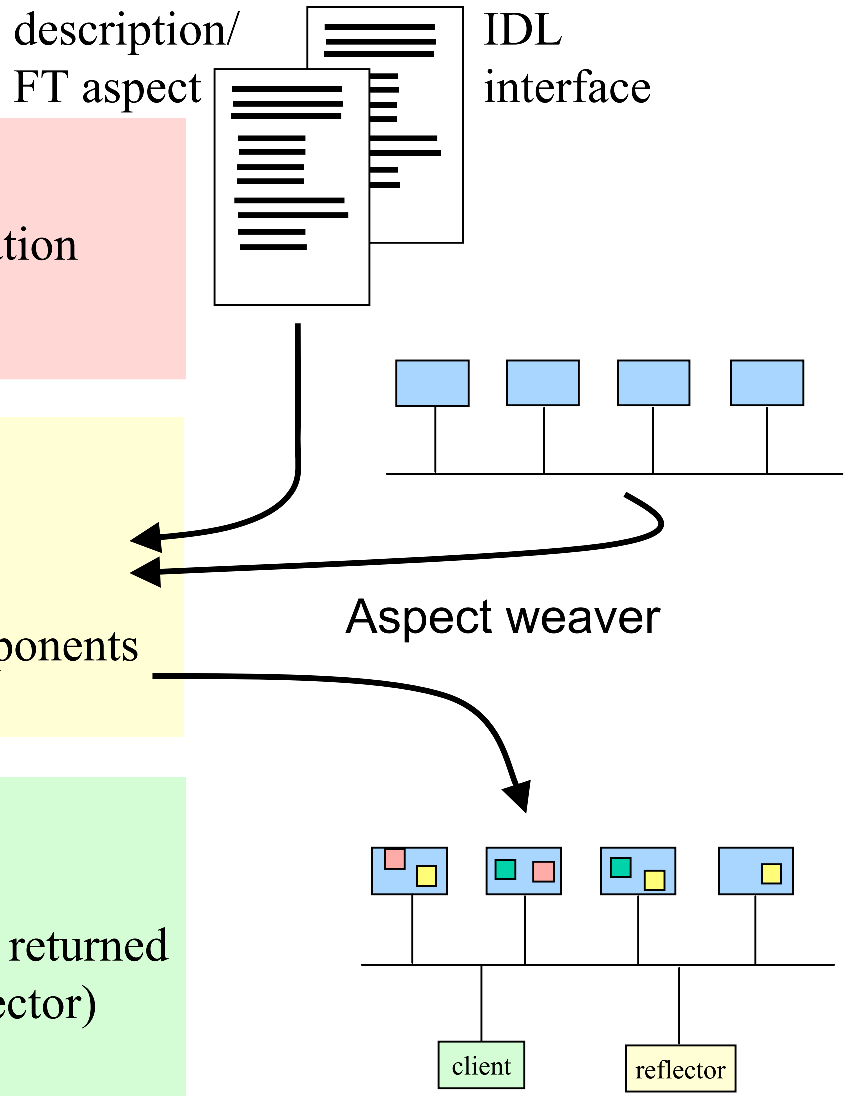
- service description
- sequential service implementation
- state, synchronization

Configuration time

- description of FT requirements
- environment description
- distribution/replication of components

Runtime

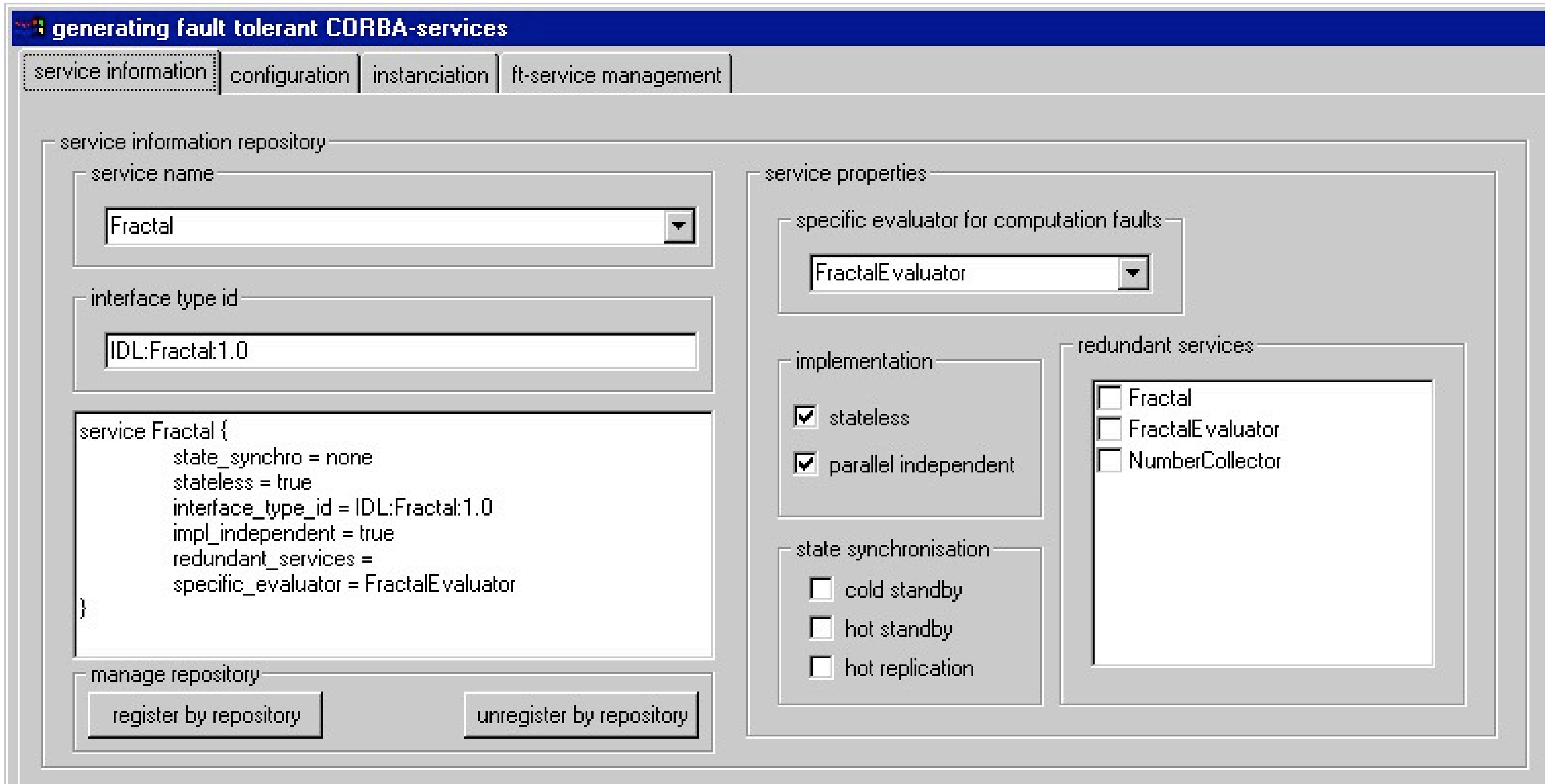
- clients' requests with QoS (FT)
- request is accepted or exception returned
- reflection (implemented by reflector)



Description of a Service

service <Name>	Name of service for registration with implementation repository interface_t

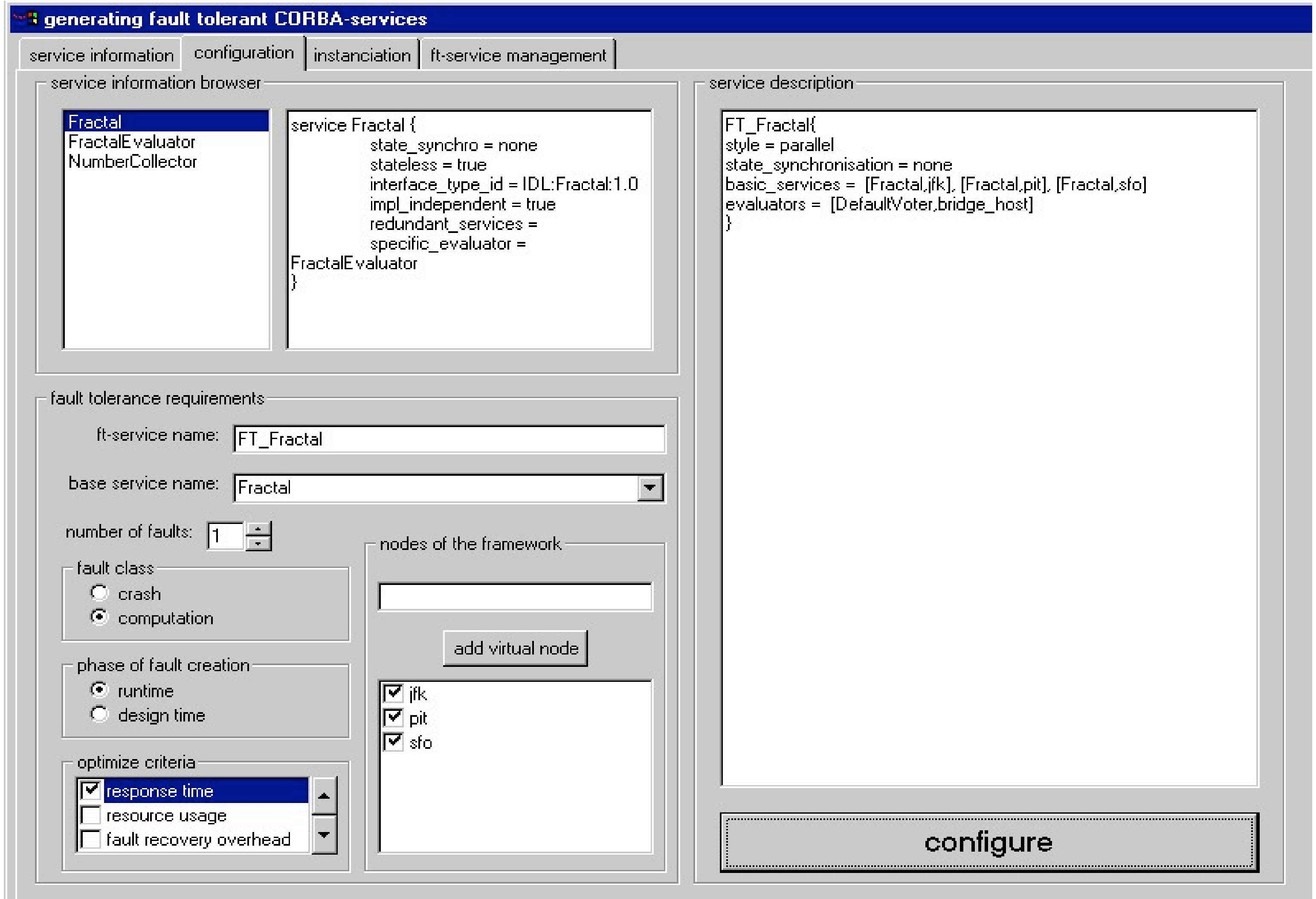
NT-based GUI – Description of a FT Fractal Service



Description of Fault Tolerance Requirements

```
ft_service FT_FractalTest {
   base_service        = Fractal
   fault_class = computation
                              //(crash|computation)
   number_of_faults    = 1
   phase_of_creation   = implementation
                              //(implementation|runtime)
   optimize_criteria   = resource_usage, response_time,
                         fault_recovery_overhead
}
```

Requirements for the FT Fractal service



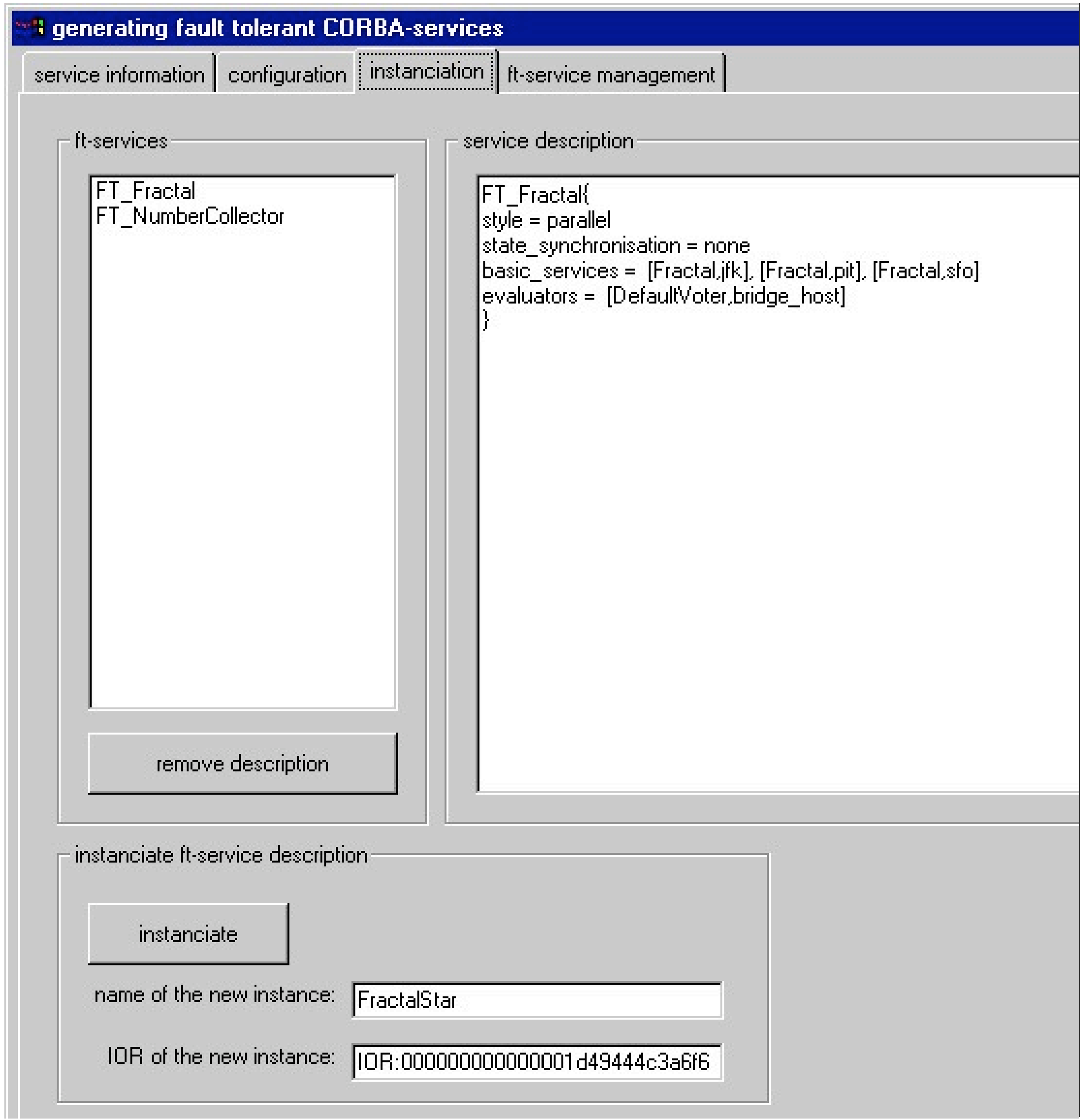
Configuration of FT Service

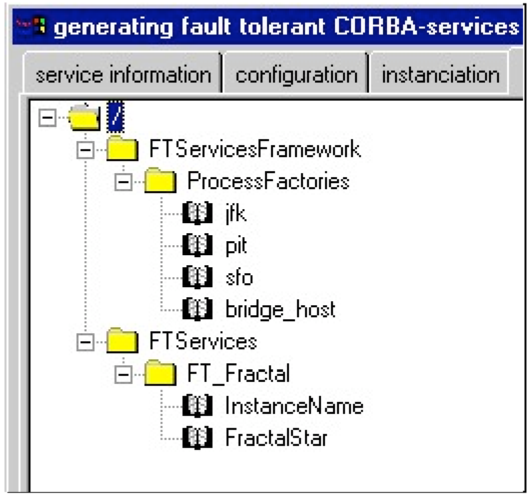
- Generated based on information about environment, FT requirements and service description

```
FT_FractalTest {
        style = sequential
        state_synchronisation = none
        basic_services =  [Fractal,zeus], [Fractal_2,queen]
        evaluators =  [Fractal_eval,zeus], [Fractal_eval,queen]
}
```

- Example shows primary/backup replication without state synchronization based on functional redundancy (multiversion)
- The service may tolerate a single computation fault

Instantiation of the FT Fractal service

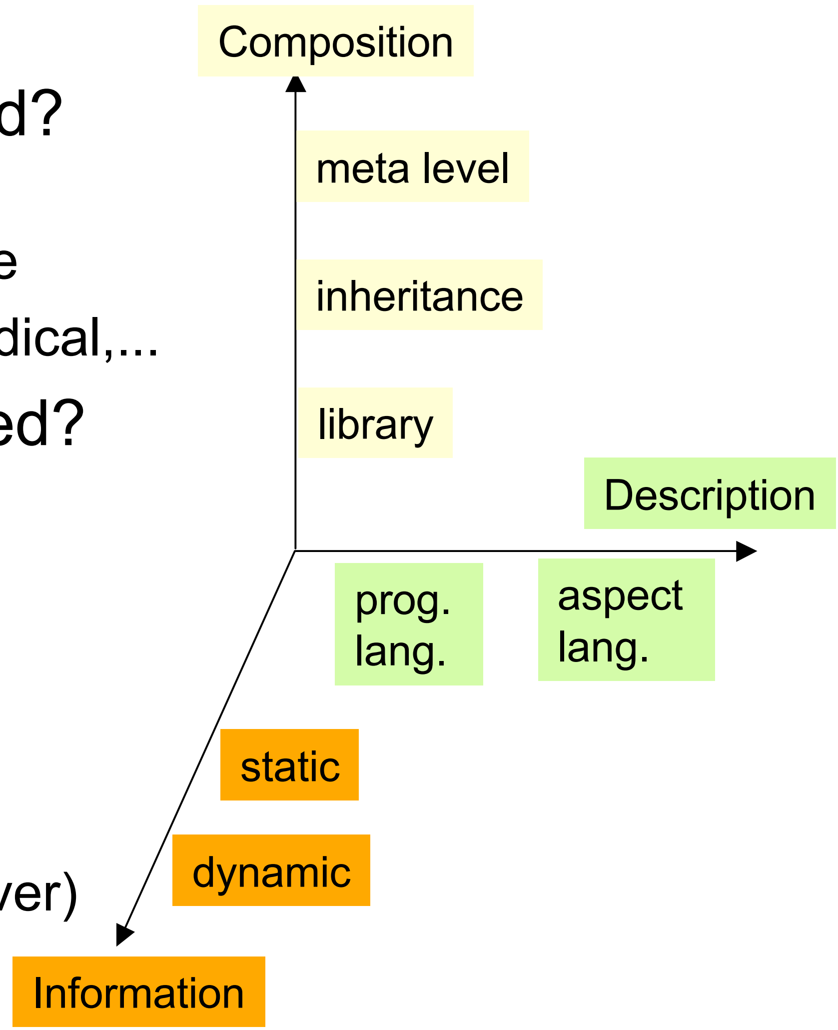




Component Replication as an Aspect

Open questions:

- How can aspects be identified?
  - General: Synchronization, Communication, Fault-tolerance
  - Domain-specific: Business, Medical,...
- How can aspects be described?
  - Language extensions, libraries
  - Separate aspect description language(s?)
- How to combine aspects and program logic?
  - Library, generator (aspect weaver)



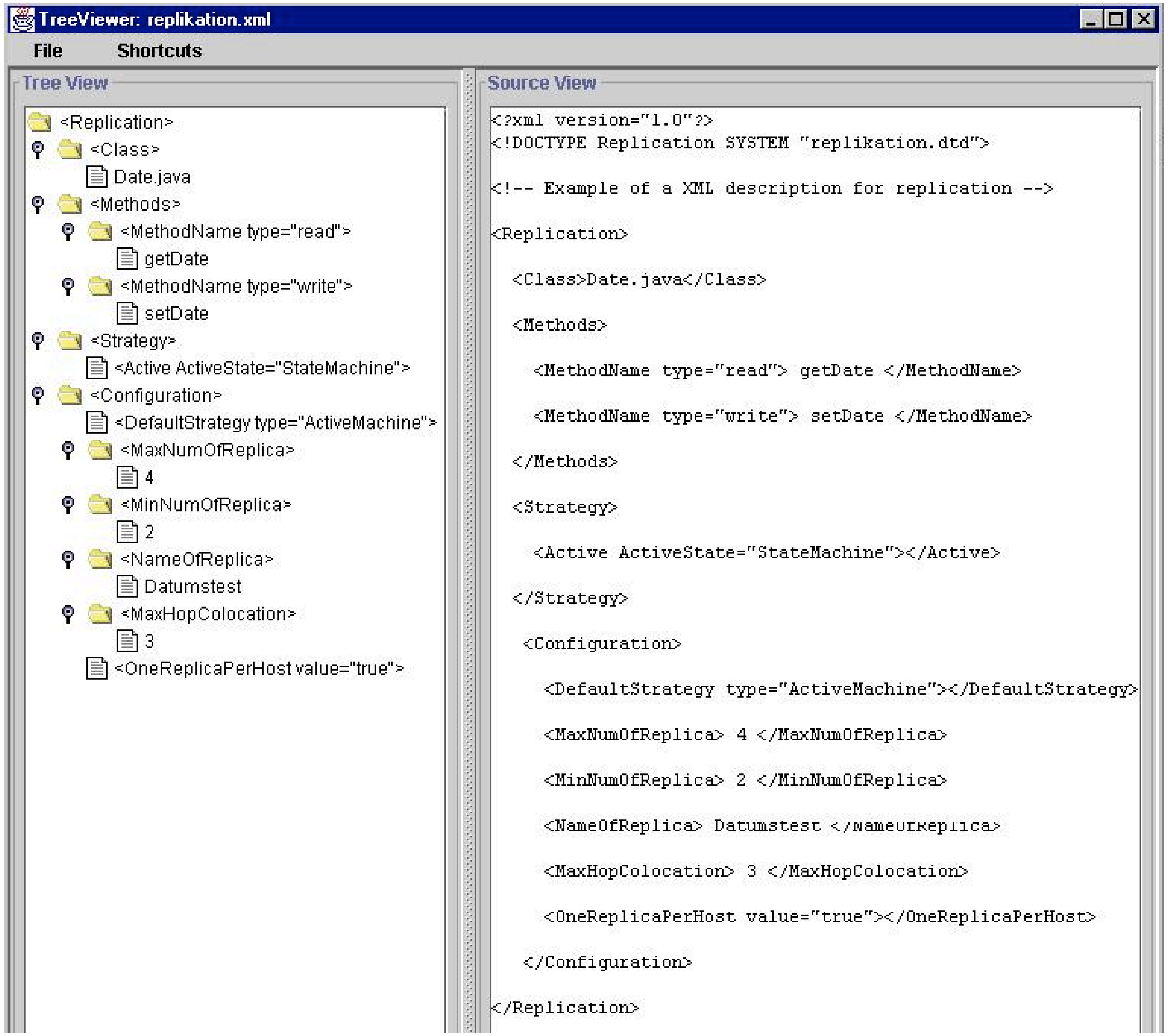
Document Type Description for Replication

```
<?xml encoding="US-ASCII"?>
<!ELEMENT Replication(Class,Methods,Strategy,Configuration)>
    <!ELEMENT Class(#PCDATA)>
    <!ELEMENT Methods(MethodName)+>
          <!ELEMENT MethodName(#PCDATA)>
          <!ATTLIST MethodName type (read|write) #REQUIRED>
    <!ELEMENT Strategy(Active?,Passive?)+>
          <!ELEMENT Active EMPTY>
          <!ATTLIST Active ActiveState(StateMachine|LeaderFollower) #REQUIRED>
          <!ELEMENT Passive EMPTY>
          <!ATTLIST Passive PassiveState(hot|warm|cold) #REQUIRED>
    <!ELEMENT Configuration(DefaultStrategy,MaxNumOfReplica,MinNumOfReplica,
                            NameOfReplica?,HostRequired?,OneReplicaPerHost?)>
          <!ELEMENT DefaultStrategy EMPTY>
          <!ATTLIST DefaultStrategy type(ActiveMachine|ActiveLeader|
                            PassiveHot|PassiveWarm|PassiveCold) #REQUIRED>
          <!ELEMENT MaxNumOfReplica(#PCDATA)>
          <!ELEMENT MinNumOfReplica(#PCDATA)>   ...
```

Aspect Description for a particular Java-class

```
<?xml version=“1.0“?>
<!DOCTYPE Replication SYSTEM “replication.dtd“>
<Replication>
    <Class>Date.java</Class>
    <Methods>
        <MethodName type=“read“> getDate </MethodName>
        <MethodName type=“write“> setDate </MethodName>  </Methods>
    <Strategy>
        <Active ActiveState=“StateMachine“></Active> </Strategy>
    <Configuration>
        <DefaultStrategy type=“ActiveMachine“></DefaultStrategy>
        <MaxNumOfReplica> 4 </MaxNumOfReplica>
        <MinNumOfReplica> 2 </MinNumOfReplica>
        <NameOfReplica> DateTest </NameOfReplica>
        <HostRequired> trave.informatik.hu-berlin.de </HostRequired>
        <OneReplicaPerHost value=“true“></OneReplicaPerHost>  </Configuration>
</Replication>
```

Description of Component Replication using XML



Based on IBM Alphaworks toolkit

Work in Progress

- Definition of a general aspect language for description of non-functional component properties
  - XML-based
- Focus on additional criteria for service configuration: resource usage, security, timing behavior, co-locations
  - Generation of Secure DCOM Services
- Design patterns
  - Software Engineering approach to System Composition based on Non-functional properties

Conclusions

- **Availability will become one of the most sought after qualities for distributed services**
- **Off-the-shelf components and standard middleware are the only feasible approach**
- **Steps towards engineering of software for availability have been presented**