

Reliability of dynamic reconfigurations in the Fractal component model

Marc Léger (PhD student, France Telecom R&D)

Thierry Coupaye (France Telecom R&D)

Thomas Ledoux (OBASCO Group EMN/INRIA)

6th workshop on Adaptive and Reflexive Middleware (ARM'07)

November 26th, 2007

Newport Beach, CA, USA



research & development



Outline

- Context and motivations
- The Fractal component model and associated tools
- A transactional approach to ensure reliable reconfigurations
- Related work and future work

Context

■ Component-based systems

- Models and definition of system consistency

■ Reflexive architectures

- Architecture-based dynamic reconfigurations for system administration
- Runtime analysis (thanks to the causal connection)

■ Dynamic reconfigurations

- Introduction of modifications in a system at run-time without stopping the whole system
- **Distributed reconfigurations:** components and administration are distributed
- **Concurrent reconfigurations:** parallelism (for optimization), several administrators (with possibly different concerns)
- **Non-anticipated reconfigurations:** not known at compile time in opened systems

Motivations and problem

- Some motivations for dynamically reconfiguring a system
 - Correction
 - System optimization
 - Adaptation and extension

- The reliability problem of dynamic reconfigurations
 - **Reliability**: measure of continuous correct service delivery
 - the system must stay consistent after reconfigurations,
 - a reconfiguration must be a valid transformation of the system state.
 - Runtime modifications can let the system in an inconsistent state
 - **Structural** (violation of architectural invariants)
 - **Behavioral** (corrupted component states, lifecycles)

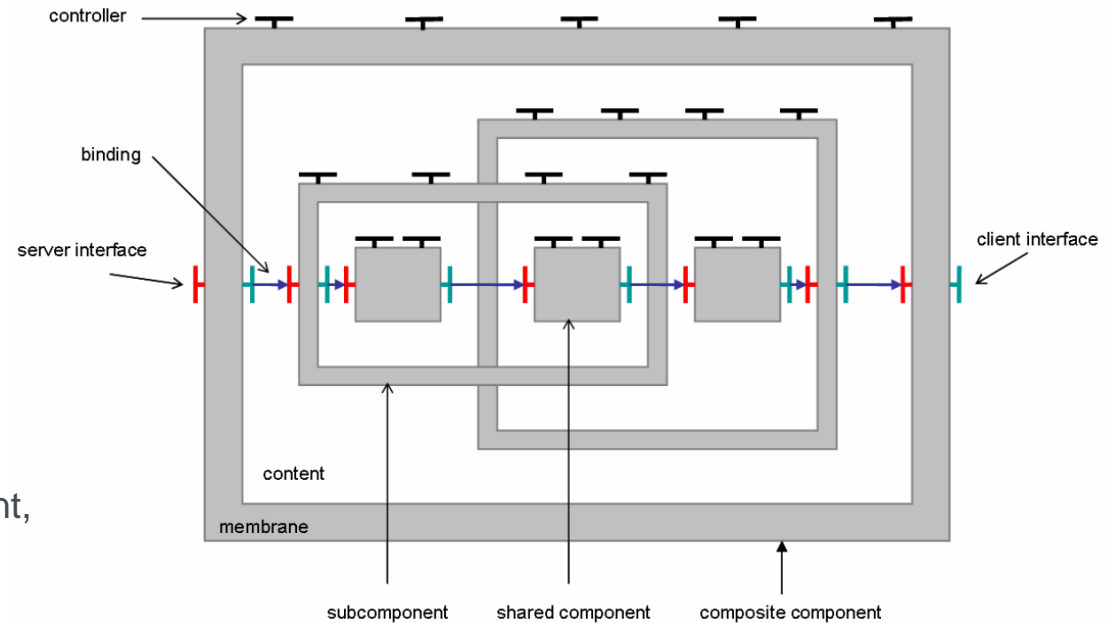
The Fractal component model

■ Model

- Hierarchical with sharing
- Opened
- Reflexive

■ Key concepts of the model

- Component
 - primitive, composite, content, membrane
- Interface
 - functional, control, client / server
- Binding
 - primitive, composite (distributed bindings)



The Fractal ecosystem

- Java implementation

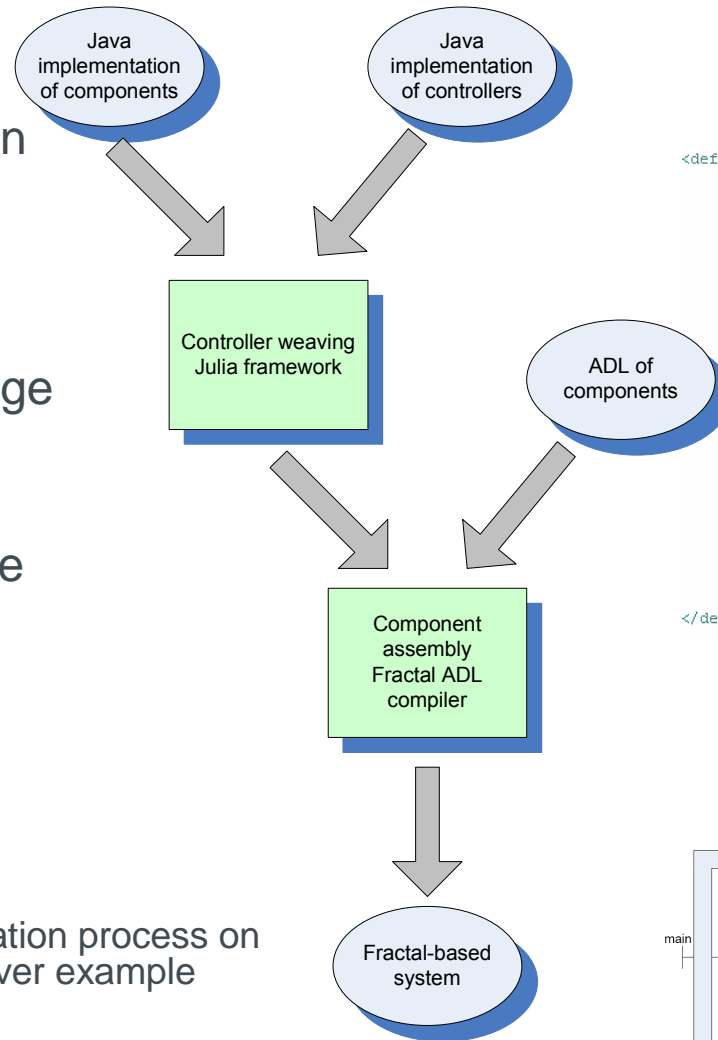
- Julia

- Architecture Description Language

- Fractal ADL

- Navigation language

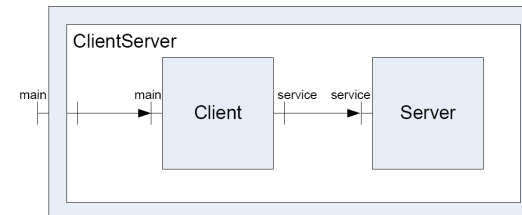
- FPath



```

<definition name="CS">
  <interface name="main" role="server" signature="java.lang.Runnable" />
  <component name="client">
    <interface name="main" role="server" signature="java.lang.Runnable" />
    <content class="impl.ClientImpl" />
    <controller desc="primitive" />
  </component>
  <component name="server">
    <interface name="service" role="server" signature="itf.Service" />
    <content class="impl.ServerImpl" />
    <attributes signature="itf.ServiceAttributes">
      <attribute name="count" value="1" />
      <attribute name="header" value="start" />
    </attributes>
    <controller desc="parametricPrimitive" />
  </component>
  <binding client="this.main" server="client.main" />
  <binding client="client.service" server="server.service" />
  <controller desc="composite" />
</definition>
  
```

Fractal Compilation process on a client-server example



Dynamic reconfigurations in the Fractal component model

■ Reconfiguration types in controllers

■ Structural

- Component addition/removal (ContentController)
- Component interconnection modification (BindingController)
- ...

■ Behavioral

- Attribute values setting (AttributeController)
- Lifecycle modification (LifeCycleController)
- ...

■ Reconfiguration specification

- with general programming language (e.g., Java)
 - Direct API calls to Fractal controllers
- with a DSL (FScript)
 - Language properties (static analyses)
 - Only model elements are manipulated

■ Primitive operations in Fractal controllers

- **Introspection** operations (e.g., *lookupFc* method in the BindingController)
- **Intercession** operations (e.g., *bindFc* method)
- Reconfiguration = operation composition
 - e.g., component hot swap

■ Reconfiguration initiators

- Human-based reconfiguration (interactive reconfigurations)
 - can depend on administrator events
- Application-based reconfiguration (automatic reconfigurations)
 - The system can auto-reconfigure: Autonomic computing

Our transactional approach to ensure reliable reconfigurations

- Definition and guarantee of system Consistency
 - Constraint specification (DSL)
 - Constraint verification (dynamic checker)

- System Recovery
 - recovery from faulty reconfigurations (software faults)
 - recovery from system failures (e.g., hardware crash)

- Concurrency management of distributed reconfigurations
 - Synchronization between reconfigurations (locking, reconfiguration scheduling)

- Implementation
 - A modular framework to guarantee reliable dynamic reconfigurations in Fractal/Julia

Transaction model and properties

■ Model

- Flat transactions (no nested transaction for the moment)
- Short-lived transactions

■ Demarcation

- Language demarcation for Java reconfigurations
- Automatic (with FScript: the execution of a top level action is transactional)

■ Propagation

- A reconfiguration operation is always included in a transaction
- autocommit for non-demarcated operations (configurable)

■ Termination property

- A timeout can be configured to force the rollback of a reconfiguration

■ ACID properties of reconfiguration transactions

■ **Atomicity** of reconfigurations

- Either the system is reconfigured and the transaction commit or it is not and the transaction aborts.

Atomic protocol

■ Atomic commit protocol

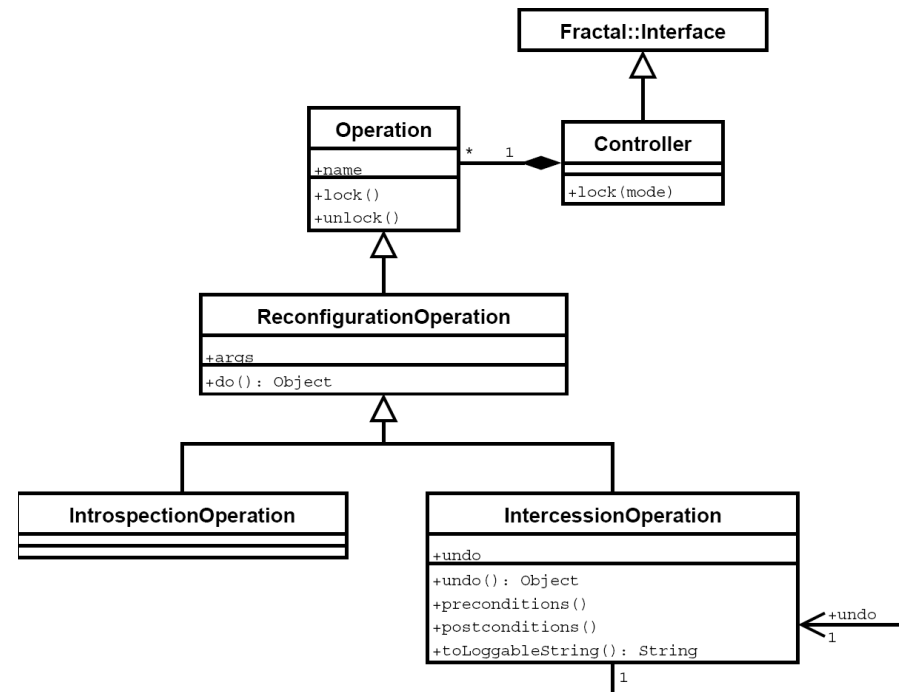
- Two-Phase Commit Protocol to deal with system distribution
- Intercession operations are logged in a transaction journal

■ Rollback recovery policy

- Used to repair transient failures linked to the component model
 - If a reconfiguration transaction fails, it must be rolled back to put the system in its last stable configuration (last commit).
- Reconfiguration operations performed in aborted transactions are undone
 - Undo a reconfiguration ↔ sequentially undo intercession operations in the reverse order

Operation model

- **Reversibility**
 - There is f^{-1} so that $f \circ f^{-1} = Id$
 - Each standard intercession operation is associated to its reverse operation in our model ($bind(A.i, B.i) \neq unbind(A.i, B.i)$)
- **Operation can keep a state to be undone**
 - E.g., the old value of a component attribute
- **Non-reversible operations**
 - Customized treatment of the rollback
 - Definition of compensation operations



Reconfiguration operation model

■ Integrity constraints to ensure system **consistency**

- A reconfiguration must be a valid transformation of the system state.

Integrity constraints

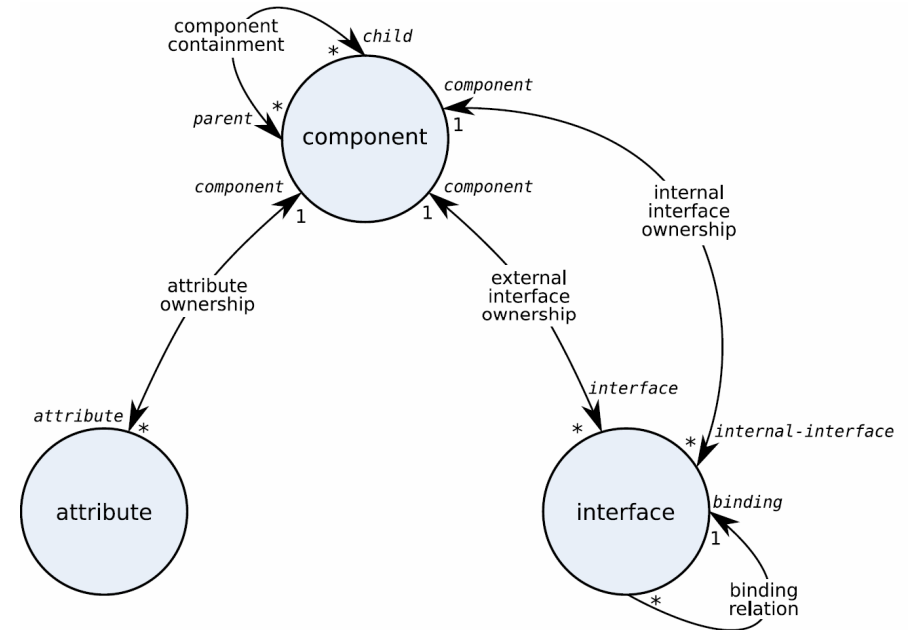
- **Integrity constraint:** a predicate which concerns the validity of an assembly of architectural elements and component state
 - Invariants on the system structure
 - Preconditions and postconditions on reconfiguration operations
- **Types of constraints**
 - Structural constraints: architectural invariants on component assembly
 - Behavioral constraints: on attribute values, component lifecycles
- A system is consistent if all integrity constraints on the system are satisfied

Constraint specification

■ A constraint language (DSL)

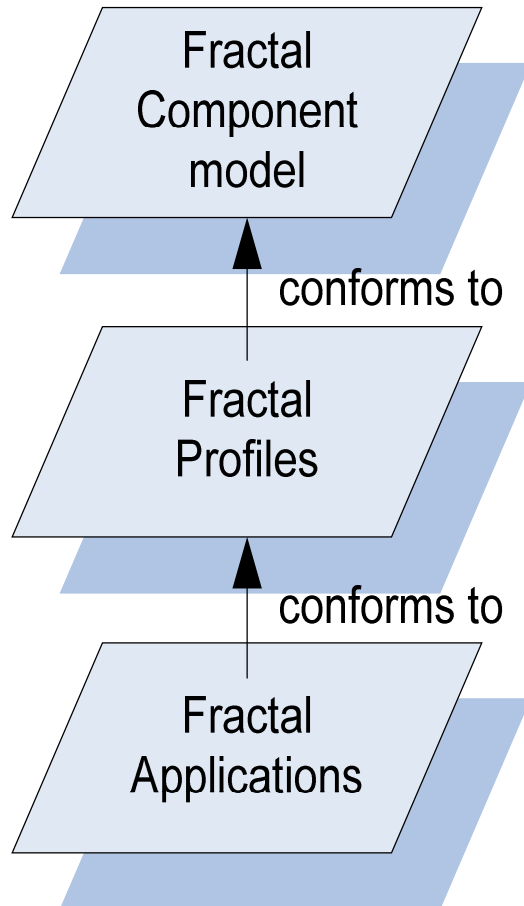
- Language FPath 'à la OCL' for Fractal elements
- Based on a graph representation of systems at runtime
- Only introspection capacities
- Syntax

```
nodeset/axis::name[predicate]  
size($server/parent::*) <= 1
```



A graph-based representation of Fractal

Three levels of integrity constraints



- A set of generic constraints associated to the component model
 - E.g., cycle-free structure
- A well defined semantics for reconfiguration operations
 - Pre/post-conditions on operations
 - E.g., *remove* operation

```
Fractal API: void removeSubComponent(Component child);  
// preconditions:  
not($child/interface::*[bound(.)]);
```

- A set of generic constraints used for a family of applications
 - E.g., forbid component sharing

```
size(./parent::*)<=1
```

- Specific constraints applying directly to instances of Fractal elements in the ADL
 - e.g., the ClientServer component must contain only one component providing the service functional interface

```
<definition name="ClientServer">  
  ...  
  <constraint value="size(./child::*[./interface::service])=1" />  
</definition>
```

Consistency checking

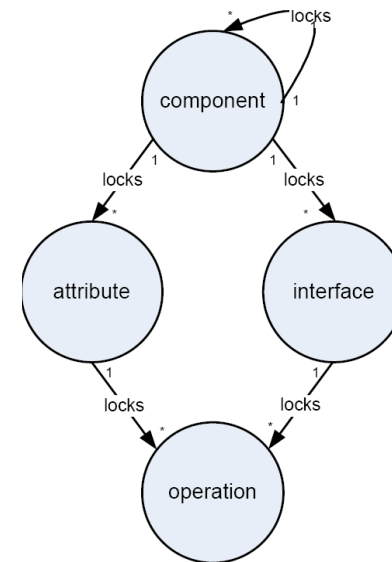
- A constraint checker to detect consistency violation
 - Static checking on a component configuration (ADL)
 - Dynamic checking on a runtime system
 - Application constraints can be dynamically added or removed
 - A constraint violation throws an exception captured by a Recovery Manager (triggering a rollback in our case)
- Pre/Post-conditions of operations
 - Checked at each operation execution
- Invariants
 - Checked at the end of a reconfiguration transaction
 - A system can temporarily violates invariants
 - It must be consistent at the commit of the transaction

■ **Isolation** of reconfigurations to support concurrency

- Reconfigurations are executed as if they were independant (seriability).

Concurrency management

- Pessimistic approach with locking
 - accesses to the Fractal elements in the system must be synchronized
 - Serializable isolation level
- A fine-grained locking model
 - Read/Write, reentrant, with upgrades
 - Avoid dirty reads and fuzzy (non repeatable) reads
 - Hierarchical locking algorithm in a DAG
 - Granularity: Fractal element
- Locking protocol
 - Strict Two-Phase locking algorithm
 - Avoid phantoms (update lost)
 - Deadlock management
 - Cycle in the waiting graph of transactions
 - policy to choose the tx to rollback and retries with a delay



locking graph

Locking aware of operation semantics

■ Objectives

- A transactions should not see modifications in the system due to other non-committed transactions
- Two transactions can introspect the same element by sharing a read lock

■ Avoid semantic conflicts between reconfiguration operations

- Locking depends on the semantics of the operation
 - Introspection operation = read (shared) locks
 - Intercession operation = write (exclusive) locks

■ **Durability** of reconfigurations for recovery

- The result of committed transactions are permanent

Recovery protocol for system failures

- Used to recover from catastrophic failures
 - Once a reconfiguration is finished, the system state is persisted.
 - Assumption: system failures are detectable (e.g., heartbeat detectors)
- Undo/Redo protocol with checkpointing
 - Redo all committed transactions from the last checkpoint in the log (top-down)
 - Undo all uncommitted transactions from the end of the log up to the last checkpoint (bottom-up)

Transaction journalization

- Kept in memory and persisted on disk
 - Optimization for transient failures
- The log contains:
 - Records of every intercession operations
 - Checkpoints records
- Operation logging
 - Fractal elements in operation parameters are identified by absolute FPath expression in the system
 - E.g., `bind(ClientServer/child::client/interface::service, ClientServer/child::server/interface::service)`

Component configuration checkpointing

- Checkpoint frequency
 - e.g. time period, number of reconfigurations, after each commit...
- Persistence of the system state
 - System architecture
 - Fractal ADL dumps
 - State of components = attribute values (functional state) + life cycle
 - Java serialization or ORM
 - Persistent attributes are tagged in the ADL definition

Related work

■ Global approach for reliable dynamic reconfigurations

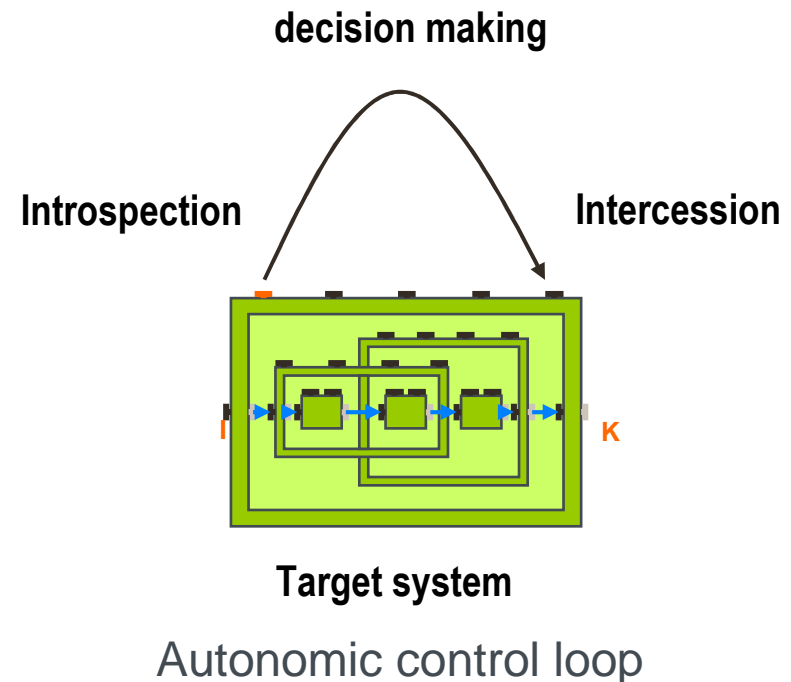
- **FORMAware** (R. S. Moreira, G. S. Blair, and E. Carrapatoso. Supporting adaptable distributed systems with FORMAware. 2004)
 - Constraints are architectural style rules
 - Semantics of reconfiguration operations is not as easily extensible
 - Lock granularity is only component

- **Plastik** (T. V. Batista, A. Joolia, and G. Coulson. Managing dynamic reconfiguration in component-based systems. 2005)
 - Integration of the OpenCOM component model and the ACME/Armani ADL
 - Constraints are fonctionnal and property-based.

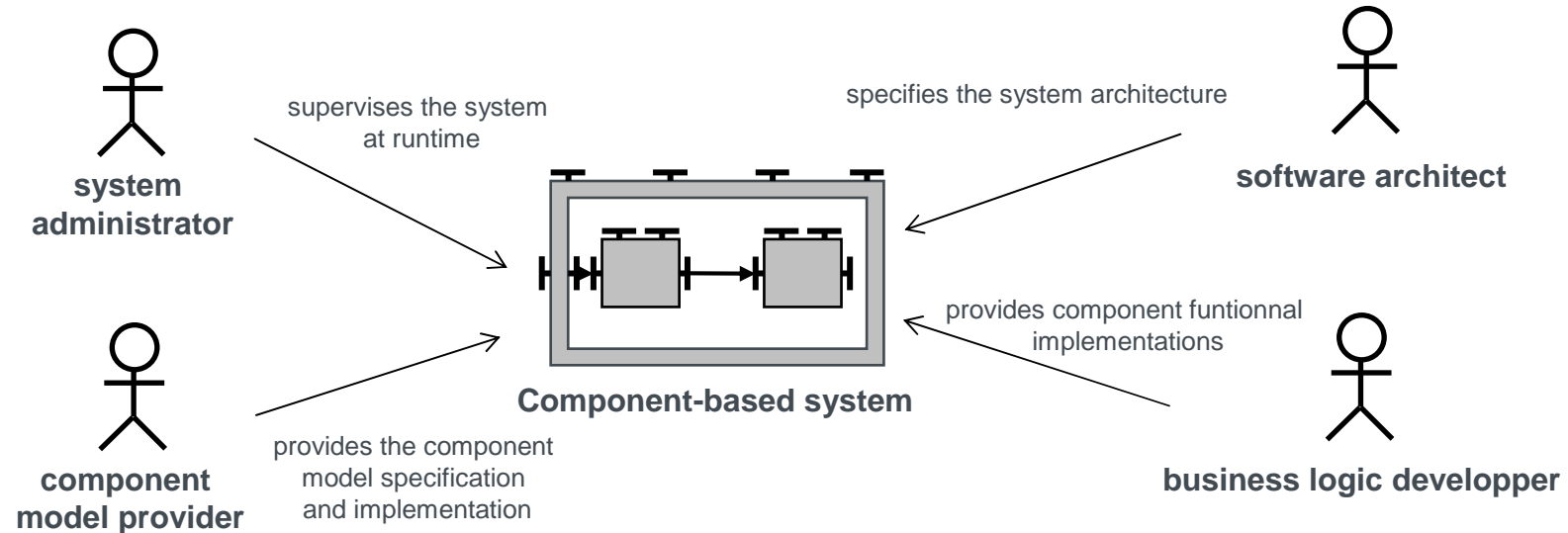
Ongoing and future work

- **Constraint checking**
 - Better check of constraint compatibility
 - Static checking of reconfigurations
- **Concurrency management**
 - Study the optimistic approach with validation
- **Work on security of dynamic reconfigurations**
 - Reconfiguration permissions and access control on reconfiguration operations

- **Integration in an autonomic computing framework**



Context diagram



Role name	Role description
Component model provider	- Specification and implementation of a component profile with generic integrity constraints (constraint language)
Software architect (functional / system)	- Architecture specification (component assembly in ADL) - Definition of non-functional properties (component membranes in GPL) - Definition of application-specific integrity constraints (constraint language) - Specification of automatic reconfigurations of the system (reconfiguration language)
Business logic developer	- Functional implementation of components (general purpose programming language)
System administrator	- Supervision of the system (monitoring) - Dynamic reconfigurations of the system (GPL / reconfiguration language)