Autonomic Management Policy Specification
from UML to DSML

Benoît Combemale   Laurent Broto
Xavier Crégut   Michel Daydé   Daniel Hagimont

Institut de Recherche en Informatique de Toulouse (UMR CNRS 5505)
2, rue Charles Camichel - BP 7122
F-31071 Toulouse Cedex 7
firstname.lastname@irit.fr

October 3rd, 2008
MoDELS 2008

Outline

1. Autonomic Management Policy Specification
   - Autonomic Computing
   - Component-Based Autonomic Computing
   - Management Policy Specification
2. UML-Based Autonomic Computing Policies Specification
   - Motivations
   - A Wrapping Description Language
   - UML-Based Formalism for Architecture Schemas
   - UML-based Formalism for (Re)Configuration Procedures
3. DSML-Based Autonomic Computing Policies Specification
   - The Configuration Description Language
   - The Wrapping Description Language
   - The other DSML
4. Conclusion
   - Lessons Learned
   - Future Works

IRIT
Institut de Recherche en Informatique de Toulouse
CNRS INPT UPS UT1

Plan

1 Autonomic Management Policy Specification
- Autonomic Computing
- Component-Based Autonomic Computing
- Management Policy Specification

2 UML-Based Autonomic Computing Policies Specification
- Motivations
- A Wrapping Description Language
- UML-Based Formalism for Architecture Schemas
- UML-based Formalism for (Re)Configuration Procedures

3 DSML-Based Autonomic Computing Policies Specification
- The Configuration Description Language
- The Wrapping Description Language
- The other DSML

4 Conclusion
- Lessons Learned
- Future Works

Motivations for Autonomic Computing

- Computing environments are becoming increasingly sophisticated:
  - numerous complex software
  - that cooperate in potentially large scale distributed environments
  - heterogeneous programming models
  - specific configuration facilities
  - components from different vendors with proprietary management interfaces
- **Consequence:** Their management is a much complex task
  ⇒ consumes a lot of human resources
- **One solution**: Autonomic computing
  - IBM, *The Vision of Autonomic Computing*. IEEE Computer Magazine, 2003

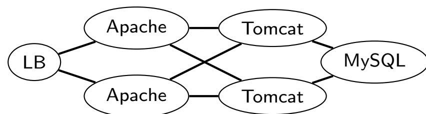
  ⇒ Automatic deployment

  ⇒ Self-management: self-configuration, -optimization, -healing, -protection.

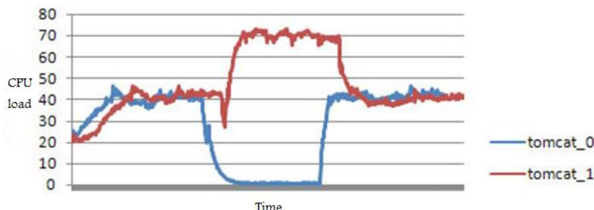
IRIT Institut de Recherche en Informatique de Toulouse CNRS INPT UPS UT1

JEE Use Case: Automatic restart of a failing Tomcat server

- J2EE Use Case
  - multi-tiered application
    - Apache, Tomcat, MySQL
    - Load Balancer
  - self-repair, self-sizing

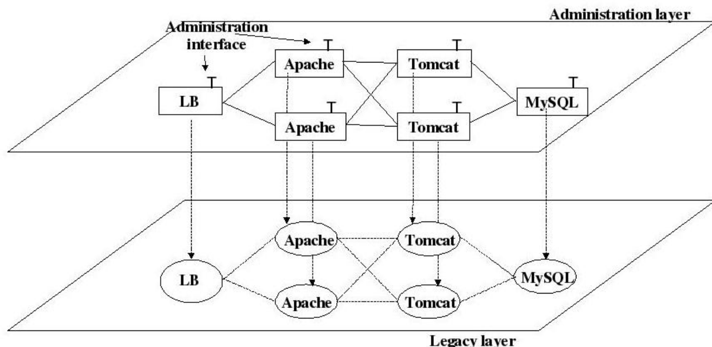


- One scenario : Automatic restart of a tomcat server on software fault



Component-Based Autonomic Computing

- **Basic idea:** to maintain a consistent and homogeneous view of the legacy
  - Relying on a component model
  - Each managed server is encapsulated into a component
  - Software architecture is abstracted as a component architecture
- **Implemented** as a component-based autonomic management system (**Tune**)
- Tune (Toulouse University Network) relies on the **Fractal component model**



Management Policy Specification

First implementation: (Jade, predecessor of Tune):

Management policies directly relying on the Fractal component model:

- A **wrapper** was implemented by a Fractal component, developped in Java
  - reflect management operation onto the legacy software
    - assigning port attribute on the wrapper is reflected in the *http.conf* file
    - setting up a binding between an Apache wrapper and a tomcat one is reflected in the *worker.properties* file
- A Fractal ADL[1] file describes the **software to be deployed** (XML syntax)
  - components/wrappers to instantiate
  - their attributes
  - their relationships
- **Reconfigurations:** administration programs or autonomic managers
  - are developped in Java, relying on the Fractal APIs.
  - may have to navigate on the component model (e.g. configuring Apache)
  - do not need to deal with configuration files or legacy layer (wrappers and ADL)

[1] Architecture Description Language

Plan

1 Autonomic Management Policy Specification
- Autonomic Computing
- Component-Based Autonomic Computing
- Management Policy Specification

2 UML-Based Autonomic Computing Policies Specification
- Motivations
- A Wrapping Description Language
- UML-Based Formalism for Architecture Schemas
- UML-based Formalism for (Re)Configuration Procedures

3 DSML-Based Autonomic Computing Policies Specification
- The Configuration Description Language
- The Wrapping Description Language
- The other DSML

4 Conclusion
- Lessons Learned
- Future Works

Evaluation of the first implementation

Relying directly on the component model is too low level, one has to:

- learn yet another framework (Fractal component model)
- write wrappers and reconfigurations
- write the XML Fractal ADL file describing the deployment in extension
  - contains many similar lines (replica) $\Longrightarrow$ copy/paste!

Consequences:

- lots of work, loss of time and money
- error-prone

$\Longrightarrow$ still consumes a lot of resources! Self-return to initial state :D

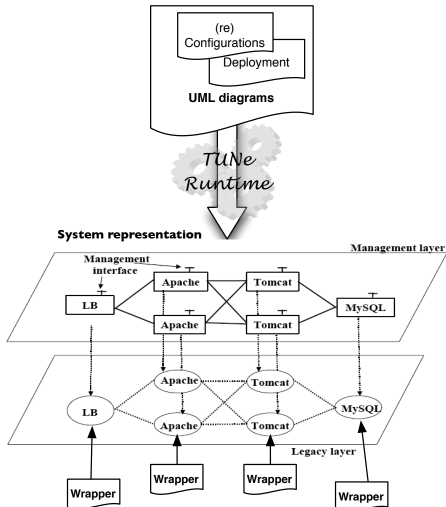
Solution: Leverage the level of abstraction... using the UML notation:

- it is a widely-used graphical notation (and a fashioned one ;)
- it is supported by a great number of tools

A UML-Based Management System

- **Avoid writing wrappers:**
  - Wrapping Description Language (WDL)
  - Generic wrapper Fractal component
  - ... not related to UML
- **Describe deployment in intension**
  - uses the UML class diagram
  - types of software, attributes and bindings
  - much more intuitive than an ADL file
- **Reconfigurations** as workflows
  - uses UML state machines
  - manipulates entities described in the deployment and reconfigurations
- The **Tune runtime automatically**
  - deploys the architecture
  - run reconfigurations

⇒ the Fractal component model is hidden



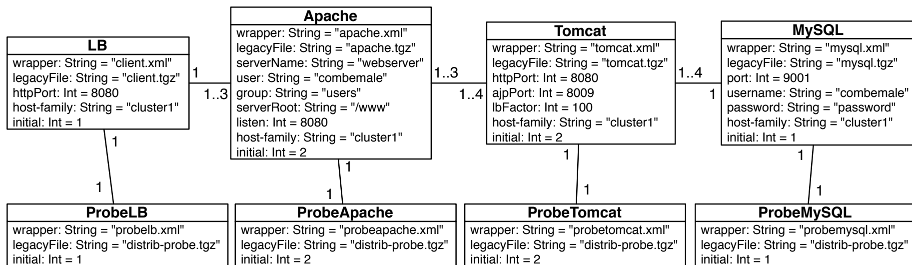
The Wrapping Description Language

An example

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<wrapper name='apache'>
  <method name="start" class="wrapper.util.GenericStart" method="start_with_linux" >
    <param ... /> <param ... /> </method>

  <method name="configure" class="wrapper.util.ConfigurePlainText" method="configure">
    <param ... /> <param ... /> </method>

  <method name="addWorkers" class="wrapper.util.ConfigurePlainText" method="configure">
    <param name="config-file" value="conf/worker.properties" />
    <param name="worker.list" value="Tomcat.nodeName" /> </method>

  <method name="stop" class="appli.wrapper.util.GenericStop" method="stop_with_linux" >
    <param ... /> <param ... /> </method>
</wrapper>
```

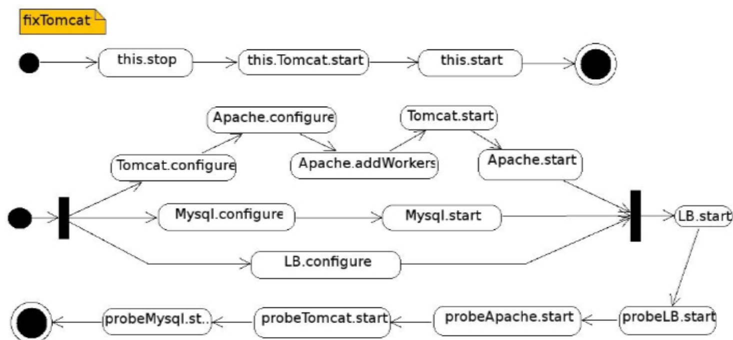
- XML file: easy to parse, not so difficult to write
- parameters values implies a navigation of the architecture schema
  - example: Tomcat.nodeName

Architecture schema for J2EE



- reuse of the class diagram
- architecture described in intension:
  - one classe represent one type of the software elements
  - multiplicity to indicate a constraint on the number of binded replicas
  - *initial* attributes indicates the initial number of replicas of this software element
- common attributes: wrapper and legacyFile
- other attributes specific to the considered legacy software element
- inconsistencies identified by the Tune runtime

State machine diagrams for *repair* and *start*



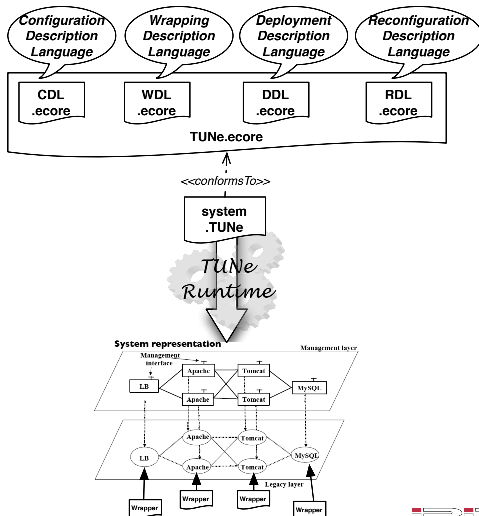
- Describes the workflow of operations that must be called
- An activity diagram would have been a better choice!
- An annotation is used to identify the event which triggers this reconfiguration
- The name of one state is used to describe an operation call
- Operation calls navigate on the architecture

Plan

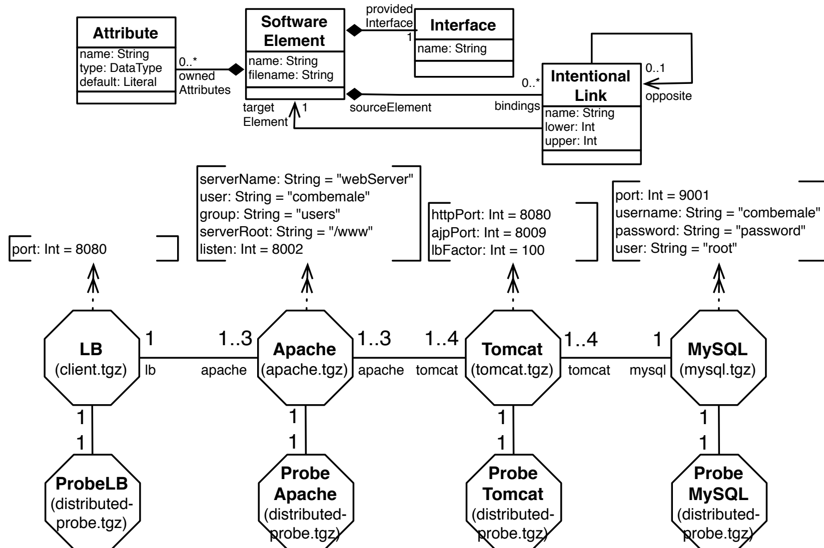
1. Autonomic Management Policy Specification
   - Autonomic Computing
   - Component-Based Autonomic Computing
   - Management Policy Specification
2. UML-Based Autonomic Computing Policies Specification
   - Motivations
   - A Wrapping Description Language
   - UML-Based Formalism for Architecture Schemas
   - UML-based Formalism for (Re)Configuration Procedures
3. DSML-Based Autonomic Computing Policies Specification
   - The Configuration Description Language
   - The Wrapping Description Language
   - The other DSML
4. Conclusion
   - Lessons Learned
   - Future Works

Overall architecture

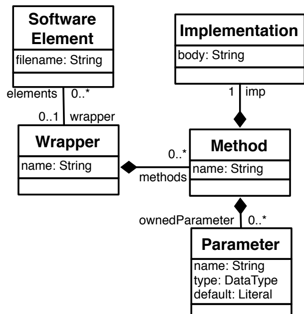
- Each point of view relies on a Domain Specific Modeling Language (DSML),
- Each language is supported by constrained tools (editors).
- The Tune runtime compiles mograms into the Fractal component model.



The Configuration Description Language



The Wrapping Description Language



j2ee.wdl

```
/*
    Definition of the J2EE's model
*/
model j2ee {
    /*
    Definition of the Apache's Wrapper
    */
    wrapper ApacheWrapper {
        -- The Configure method
        method configure() {
            -- implementation ... :
            body = "generic.Configure.java"
        }
        -- A method
        method myMethod(p1 : String = "chaine1", p2 : String = "chaine2") {
            body = "generic.Start.java"
        }
    }

    -- wrapping definition
    software Apache_conf1 { wrapper = ApacheWrapper }
    software Apache_conf2 { /* wrapper not defined, yet... */ }
}
```

Outline

- TuneModel : j2ee
  - SoftwareElement : Apache_conf1
  - SoftwareElement : Apache_conf2
  - Wrapper : ApacheWrapper
    - Method : configure
      - Implementation
    - Method : myMethod
      - Parameter : p1
      - Parameter : p2
      - Implementation

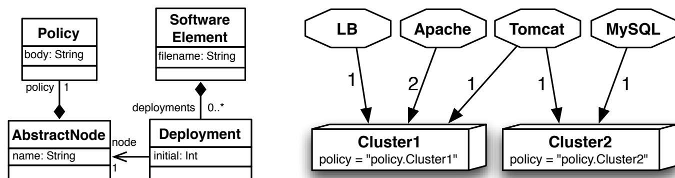
Problems | Properties | Error Log | Console

Property	Value
Body	generic.Configure.java
Comments After	
Comments Before	-- implementation ... :
Location	12:4-12:35

- Textual concrete syntax, easier to use than XML one.
- Developped using TCS (Textual Concrete Syntax)
  - provides a full-featured eclipse editor (color, folding, error detection, etc.)
- Only a couple of hours

The other DSML

- **Deployment Description Language**



- **Reconfiguration Description Language**
  - Not shown here
  - Inspired by the UML activity diagram
  - But simplified with useless elements removed

Plan

1 Autonomic Management Policy Specification
- Autonomic Computing
- Component-Based Autonomic Computing
- Management Policy Specification

2 UML-Based Autonomic Computing Policies Specification
- Motivations
- A Wrapping Description Language
- UML-Based Formalism for Architecture Schemas
- UML-based Formalism for (Re)Configuration Procedures

3 DSML-Based Autonomic Computing Policies Specification
- The Configuration Description Language
- The Wrapping Description Language
- The other DSML

4 Conclusion
- Lessons Learned
- Future Works

Lessons Learned

User viewpoint

- **Higher level of the abstraction** in the definition of the system
  - facilitates the learning and the adoption of the tool
  - improves productivity for new and experimented users
- Tune 1.0 consists in **reusing the UML notation** with one main **advantage**
  - many high quality UML editors are available

  but has several **drawbacks**
  - the *UML semantics is tailored* in a misleading way for UML users
  - the Tune parser for UML models depends on the used UML editor
- TUne 1.1 is **based on DSML** and provides:
  - better focus on domain concepts
  - well-founded notation based on metamodels and OCL constraints
  - **better user assistance** :
    - either by enforcing the construction of consistent models
    - or by pointing out errors in models

  but new editors have to be developped
  - may be generated thanks to editor generators: Topcased, TCS, GMF...

Lessons Learned

Correctness

The adopted approach favors correctness of managed applications :

- the **definition of the management layer** :
  - hides complex configuration files
  - all configurable entities are homogeneously manipulated (reified)
- the **definition of an application pattern** and its **enforcement** :
  - the deployed architecture is generated from the architecture schema
  - the architecture schema may be used to check consistency of reconfigurations
- the **definition of a well-founded user-friendly notation**:
  - it favors understanding of the users on their models
  - consistency of models is enforced or checked by the editors before they are parsed by the Tune runtime

Future works

- Complete the editing tools for all administration points of view
- One DSML with multiple views/diagrams or several related DSML
- Embedded DSML: for navigating on the deployment schema (ADL)
- Use a transformation language to implement the Tune runtime
- Models should not only used to describe policies but also to handle the *management layer*:
  Model-Driven System Management (MDSM)