IST-2001-32360

Architecting Evolvable Software

An Introduction to the ArchWare Project

Flavio Oquendo
flavio.oquendo@univ-savoie.fr
Radu Mateescu
radu.mateescu@inria.fr

http://www.arch-ware.org http://www.architecture-ware.org

Outline

- Introduction:
  - Objectives and Planned Results
  - The Approach
  - Languages, Framework and Tools
- Demo in SFM'03: Architecture Analysis by Model-checking
  - Overview
  - Demo

The ArchWare IST European Project
The Partners

IST-2001-32360

3 years:
2002-2005

Manpower:
50 person-years

Scientific Coordinators:

Flavio Oquendo & Brian Warboys

The Players:

- F Gallo: Consorzio Pisa Ricerche, Pisa - Italy
- C Occhipinti: Engineering, Rome - Italy
- F Oquendo: University of Savoie, Annecy - France
- R Dindeleux: Thésame, Annecy - France
- B Warboys: University of Manchester - UK
- R Morrison: University of St. Andrews - UK
- H Garavel: INRIA R/A, Grenoble - France

The Objectives
What ArchWare aims to?

The project aims to:

- **design formal languages** for describing architectural structure, behaviour, qualities, and evolution of software systems
- **implement a customisable environment** for process-driven architecture-based software engineering

The focus is:

- **formal development and evolution of compliant architectures**

IST-2001-32360

The Objectives
ArchWare supports “compliance”

- What is compliant?
  - “compliant” means “property preserving”, thereby the architecture of a compliant system is designed with the goal of fitting the architecture to the needs of the application
- What does it imply?
  - if application requirements change then the architecture must be able to evolve in order to cope with the new needs
  - if the implementation of the application does not verify the properties of the architecture then the application must be able to evolve at runtime in order to cope with the new needs

The Planned Results
What ArchWare delivers?

The ArchWare Project delivers:

- Architectural Languages and Tools
- A Persistent Architecture Framework
- Process Models for Refinement and Evolution
- Customisable Architecture-based Environments
- Customised Architecture-based Environments for the two Industrial Partners

The Approach
What approach to build results?

- Results are built based on the following principles:
  - languages are formal, compositional and reflective
  - The framework provides persistent run-time support for formal compositional reflective languages
  - tools support architecture-based processes

The ArchWare Languages

- ArchWare Architectural Languages:
  a family of integrated languages
- Architecture Description Language
- Architecture Analysis Language
- Architecture Exchange Language
- Architecture Refinement Language

… all are architectural languages of the same family: all are designed based on the Architecture Base Language (π-ADL)

The ArchWare Languages
π-ADL: The Base Language

- π-ADL has as formal foundation the higher-order typed π-calculus
  - a calculus for communicating and mobile systems
- π-ADL is itself a formal language defined as a domain-specific extension of the higher-order typed π-calculus
  - it is a well-formed extension for defining a calculus of communicating and mobile architectural components
- π-ADL has been designed as a formal layered language
  - To be compositional
  - To be persistent
  - To be reflective
  - To be executable
  - To enable analysis
  - To enable refinement
  - To enable evolution
- π-ADL is based on earlier work from the Univ. of Savoie at Annecy (SPACE and PIE Projects), Manchester (PIE and CSA Projects), and St.-Andrews (CSA Project)

The ArchWare Languages Architecture Analysis Language

- μπ-AAL has as formal foundation the modal μ-calculus and the higher order predicate calculus
  - it provides a property specification language
  - it supports for automated checking of property satisfaction by theorem proving (through higher-order logic) and model checking (through compositional model checking)
- μπ-AAL has been designed as an integrated part of the ArchWare Architecture Base Language (π-ADL)
- μπ-AAL is based on earlier work:
  - on temporal logics by INRIA R/A (Vasy)
  - on modal logics and architectural quality attributes by the Univ. of Savoie at Annecy

The ArchWare Languages
Architectural Style Language

- The ArchWare Architecture Description Language:
  - it provides a style-based language
  - it is built on top of the Architecture Base Language and the Architecture Analysis Language

$\sigma\pi$-ADL = π-ADL + $\mu\pi$-AAL
+ *style constructs*

- $\sigma\pi$-ADL is a formal architectural style description language:
  - for describing architectures taking into account reuse of existing styles
  - enabling formal analysis of architectures against defined style properties

The ArchWare Framework Persistence and Reflection

IST-2001-32360

- The ArchWare Framework provides:
  - virtual machine for executing compositional descriptions
    - orthogonal persistence: models of values independent of longevity
    - linguistic reflection: to allow reflective descriptions
  - hyper-coding (through persistence and reflection): one representation of a value throughout its lifetime
  - enabling evolution by composition via hyper-coding

The ArchWare Framework
Persistence and Reflection

IST-2001-32360

- Orthogonal persistence
  - all data resides in a strongly-typed persistent repository
- First-class code (components, ports, etc.)
  - code is a part of this data
  - resides in the same repository as other data
- Hyper-code: no user-level distinction between code and data

IST-2001-32360

The ArchWare Tools
Process-enabled Tools

- Visual Architecture Modeller
- Visual Architecture Animator
- Style-aware Architectural Design Repository
- Architecture Analysis Tools
  - Theorem-Proving Tool
  - Model-Checking Tool
  - Model-Specific Evaluation Tools
- Refinement Engine
- Code Synthesiser
- Style-based Customiser

Supporting Architecture-based Engineering of Evolvable Software

- Central feature
  - ArchWare applies an innovative approach to the engineering of compliant software systems that sets composition and the ability to evolve by decomposition-recomposition as its central characteristic
  - Evolution arises in response to changes to requirements in the problem domain as well as to changes in the implementation domain
- Unified representation
  - Hyper-code: the architect sees only a single representation form throughout the software engineering process

Overview of the ArchWare AAL Architecture Analysis Language

IST-2001-32360

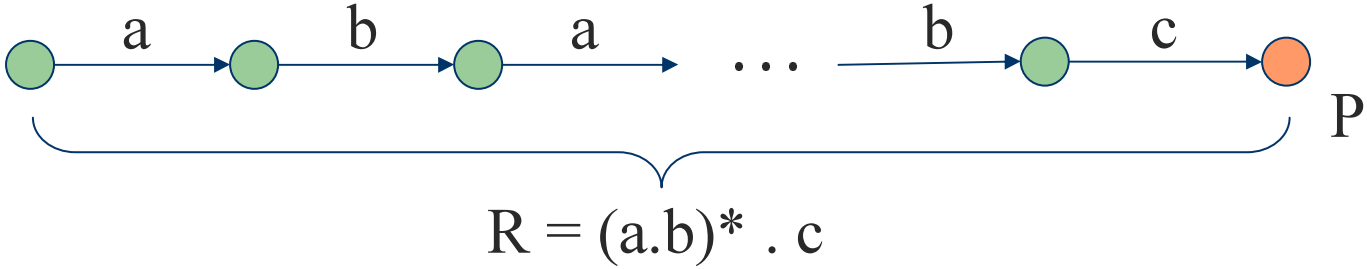
- Designed to express and check properties of
  - Architectural styles (*style architect*)
    - Structural properties (cardinality, topology, …)
  - Architectural descriptions (*application architect*)
    - Behavioural properties (safety, liveness, …)
- Design choices
  - First-order logic combined with modal μ-calculus
  - Data-handling mechanisms
  - Regular expressions over execution sequences

AAL: possibility modality

- Concrete syntax:

  some sequence { R } leads to state { P }

  there exists an execution sequence whose concatenated actions form a word of the regular language R and that leads to a state satisfying P



Possibility: example

- « A transport demand will be potentially sent »

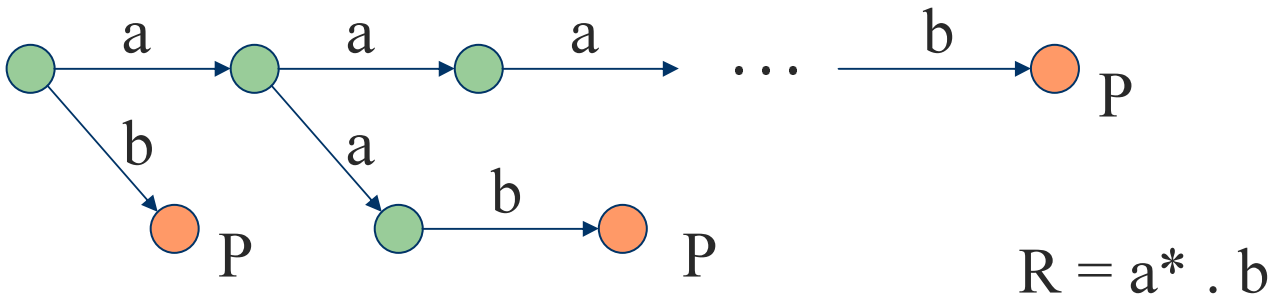
```
some sequence {
    true* .
    via transportDemand send any
} leads to state { true }
```

AAL: necessity modality

- Concrete syntax:

  every sequence { R } leads to state { P }

  any execution sequence whose concatenated actions form a word of the regular language R must lead to a state satisfying P



Necessity: example

- « A document not present in the repository cannot be part of a query result »

```
{ forall d:Document |
    every sequence {
        (not via addDocument receive d)* .
        via documentsOut send d
    } leads to state { false }
}
```

AAL: minimal fixed point

- Concrete syntax:

  finite tree X given by { P }

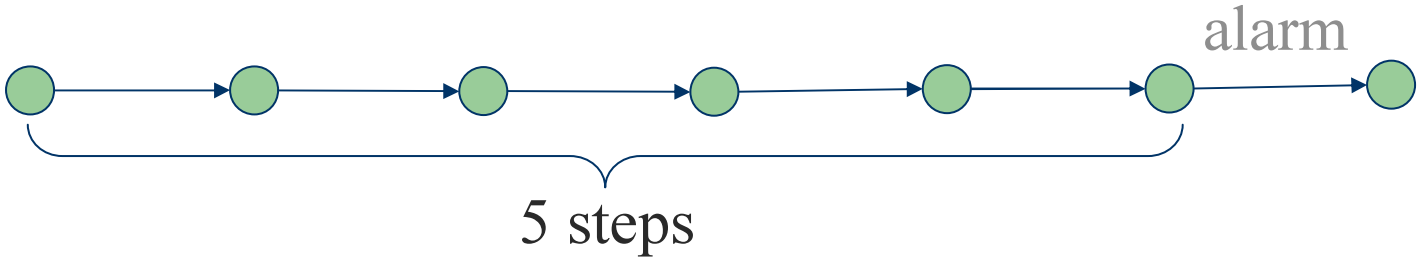
  denotes the least solution of the equation X = P where propositional variable X occurs in P
- Informal semantics:

  « recursive function » which explores forward the execution tree starting from the current state, and stops after a finite number of actions

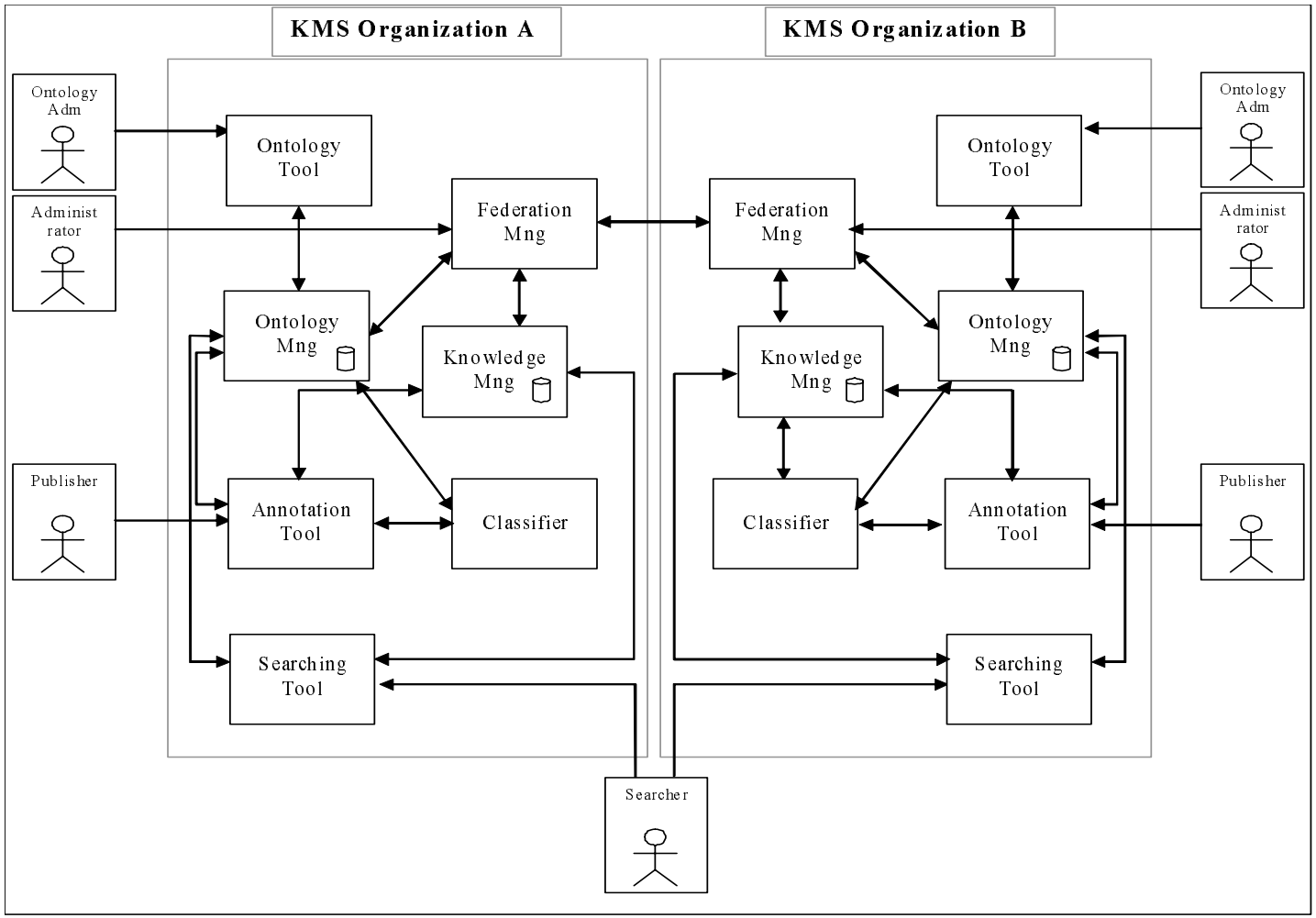
Minimal fixed point: example

- « Action alarm occurs after 5 steps »

  finite tree Alarm (c:integer) is {
    c > 0 implies some sequence { true }
                  leads to state { Alarm (c – 1) }
    and
    c = 0 implies some sequence { alarm }
                  leads to state { true }
  } (5)



Demo: model-checking behavioural AAL properties



Web site: www.arch-ware.org

IST-2001-32360

Several tools will be available in open source (from spring 2004)