Towards a Framework for Self-Adaptive Component-Based Applications

Pierre-Charles David & Thomas Ledoux

OBASCO Project, EMN / INRIA

Distributed **A**pplications & **I**nteroperable **S**ystems 2003

2003-11-19

Application Development Challenges

- Heterogeneous environments
  - diversity of hardware platforms (PDAs,. . . grids)
  - and of software systems (Win*, various Unix)
- Dynamic environments
  - resources usage (CPU, memory, network. . .)
  - hardware devices availability (USB, Bluetooth. . .)
- Each combination requires a slightly different solution
- Applications must become self-adaptive

Self-Adaptive Applications

- Self-adaptive applications modify themselves to fit their evolving environment

- Three characteristics:

  1. Reconfigurable: can be modified dynamically
  2. Context-aware: knows its environment
  3. Encapsulates adaptation logic: decides when and how to adapt

- Our objective: ease the creation of self-adaptive applications

- Approach: consider adaptation of a software application to its environment as an aspect (in the sense of AOSD)

Self-Adaptive Application Development

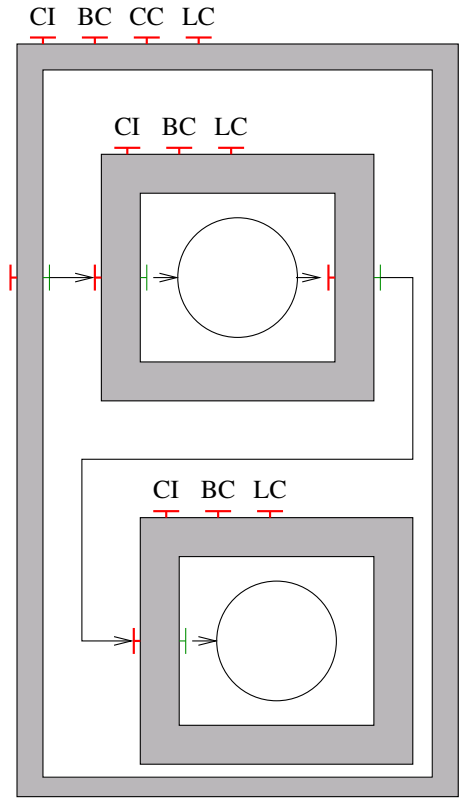
1. Development time: build an adaptable application
   - developers create/reuse business components
   - concentrate on functional concerns
2. Assembly & deployment time: make it self-adaptive
   - development of the adaptation logic in policies
   - integration (weaving), resulting in a self-adaptive application
3. Runtime
   - application adapts itself to the evolving environment
   - addition/removal/tweaking of policies

The Fractal Component Model

- General composition model: not domain specific
- Language independant (reference implementation in Java)
- Extensible model and reference implementation
- Run-time model, no language extension
  - supports reflection & dynamic reconfigurations
- More info at http://fractal.objectweb.org/

Anatomy of a Fractal Component

- Controller part
  - exposes typed interfaces (ports)
  - service interfaces: application specific
  - control interfaces: management
- Content part
  - simple (Java) object ⇒ primitive
  - other Fractal components ⇒ composite



Supported Reconfigurations

- Simple parameterization: `attribute-controller`
  - JavaBean-like
  - supported by Fractal, must be anticipated
- Structural reconfigurations: `binding-controller`, `content-controller`
  - modify bindings and containment relationships
  - supported by Fractal, must be anticipated
- Behavioural reflection: `reflection-controller`
  - MOP-like extension to Fractal
  - supports some unanticipated adaptations

Making Fractal Components Self-Adaptive

- Fractal-based applications are reconfigurable
- Next step: enabling dynamic binding (weaving) of adaptation policies
- New, optional control interface for components: `adaptation-controller`
- Manages the set of adaptation policies applying to the component

```
public interface AdaptationController {
    void addFcPolicy(AdaptationPolicy p);
    void removeFcPolicy(AdaptationPolicy p);
    AdaptationPolicy[]  getFcPolicies();
}
```

E-C-A Based Adaptation Policies

- Policies' role is to detect changes, and if appropriate, apply reconfigurations to adapt the application $\Rightarrow$ E-C-A rules
- Event-Condition-Action "paradigm" comes from Active Databases
  - **Event**: specify circumstances which should trigger an adaptation
  - **Condition**: optional guard
  - **Action**: (sequence of) reconfigurations
- Should enable analyses and simulations
  - detect conflicts, instabilities
  - tooling, IDE integration to help developers

Behaviour of Adaptation Policies

1. When a policy is bound to a component
   - start listening to the appropriate events
2. When one or several events are detected
   - select all the corresponding rules
   - evaluate their condition
3. When a (set of) rule(s) is activated
   - compute the global effect of the reconfigurations
   - apply reconfigurations

Proposed Framework

- Context-Awareness
  - observe the environment and detect meaningful changes
- Adaptation Policies Development Framework
  - classes to build E-C-A rules and policies
- Binding of policies to Fractal components
  - Load-time (ADL)
  - Run-time (`adaptation-controller`)

Context-Awareness

- Provides an up to date and accurate representation of the application's environment
- Enables reasoning and detection of circumstances requiring adaptation
- Three steps:
  1. **Acquisition**: import raw data from OS / hardware using probes
  2. **Structuration**: organize probed values in a dynamic hierarchy of resources described by attributes
  3. **Reasoning**: higher level view of the application environment using synthetic attributes

Specifying and Detecting Events

- Primitive events
  - external: from the execution environment (context-awareness)
  - internal: react to the application's behaviour
- Composite events: reasoning on the evolution of the system
  - sequence, alternative, conjonction of events

Conditions

- Behave as optional guards
- Simple boolean expressions on:
  - execution context (as seen by context-awareness)
  - application state (introspection)
  - captured event data

Reconfiguration Actions

- Sequence of primitive reconfigurations
  - parameters, structure, behaviour, policies
- Limited scope to allow local reasoning
  - only affects `this` component and direct sub-components
- Applied atomically

Example Adaptation Policies

1. An adaptive cache (see paper)
   - Add a generic cache to a component when it is too slow.
   - Adjust the size of the cache depending on the available memory.
   - Choose the appropriate replacement policy depending on the actual access patterns.

2. Disconnected mode management
   - When the user sends an email, if the network connection is down, then add a "meta-component" to store the message instead of sending it.
   - When the network connection comes back, then remove the meta-component (which then send the deferred messages).

Related Work

- Other adaptive component models
  - ACEEL (Cherfour & André)
    - separation of state and behaviour; multiple behaviours
    - forces specific programming pattern
  - K-Components (Dowling): very close to our approach
    - Archi. description + Adaptation Contracts $\Rightarrow$ C++ app.
    - contracts defined at build time
- Adaptive middleware approach
  - dynamicTAO (Kon et al.), QuO (BBN)
  - same objective, different approach

Conclusion and Future Work

- Use an AOSD approach to self-adaptive applications development
- Extended Fractal Component Model
  - Supports behavioural reflection
  - Allow binding of adaptation policies
- Adaptation Policies Development Framework
  - Based on the E-C-A "paradigm"
  - Framework helps to develop policies
- Future
  - Design a DSL/ASL to program policies
  - Provide analyses, conflict checking and tools based on the DSL