Engineering Self-Adaptive Systems & Dynamic Software Product Line

Mahdi Bashari
University of New Brunswick
Ebrahim Bagheri
Ryerson University

Tutorial at SPLC 2013
August 27th 2013
Motivation
Introduction
Adaptation aspects
Overview of three DSPL approaches
Engineering of adaptive systems & DSPL
Research Challenges
Summary
Outline

- Motivation
  - Software Product Line
  - Self-adaptive Systems
  - Synergy
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
Goal:
- Mass production and mass customization at the same time

How:
- Two phases to promote reuse:
  - Domain engineering
  - Application engineering

Benefits:
- Effort, quality, time to Market
SPL Process

(Hallsteinsen et al. 2008)
Domain Engineering
- SPL infrastructure developed for the specific product family
  - Variability model
  - Reference architecture
  - Other PL assets

Application Engineering
- Specific product built using a variability model per customers’ needs
  - Final product
Domain engineering:
- Possible variation of products is captured in products family.
- Variation of product is represented by variation points.
- Variation model is specified for selecting variants of variation points.

Application engineering:
- Customer requirements are elicited.
- Variant decision about each variation point is made.
- Variants are bound to the final product.
## Variability Aspects

<table>
<thead>
<tr>
<th>SPL</th>
<th>Goal of variability</th>
<th>Satisfy individual customer needs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Binding time of variability</td>
<td>Pre-runtime/Runtime</td>
</tr>
<tr>
<td></td>
<td>Decision maker of the variant</td>
<td>Application Engineer</td>
</tr>
<tr>
<td></td>
<td>Method for selecting the variant</td>
<td>reuse of shared assets conforming a variability model</td>
</tr>
<tr>
<td></td>
<td>Model of variability</td>
<td>Variability Model, Architecture model</td>
</tr>
<tr>
<td></td>
<td>Lifetime of the variability model</td>
<td>Pre-runtime</td>
</tr>
</tbody>
</table>
Outline

- Motivation
  - Software Product Line
  - Self-adaptive Systems
  - Synergy
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
Goal:
- Maintaining service in different situations

How:
- Building application in two layers:
  - Adaption logic
  - Application logic

Benefits:
- Flexibility, reliability, efficiency
General structure of self-adaptive systems
Adaptation manager:

- Monitors the context of the system internally and externally
- Analyzes the context for conditions where adaptation is needed.
- Decides about the response plan following adaptation policies.
- Carries out the decided plan by dynamic addition, deletion, or modification of product features, or introducing changes in the architectural structure of the system.
Enables monitoring of the system’s internal state and provides the needed interface for adaptation manager.

Enables runtime adaptation and provides the needed interface for adaptation manager to perform the adaptation.
Development of a self-adaptive system:

- Possible variation of system in runtime is captured.
- Variation of product is represented by variation points.
- Adaptation policy which defines variant selections in response to changes in context is designed.

Adaptation manager at runtime:

- Context of system is captured using monitoring sensors.
- Adaptation manager decides about the changes in variation points according to context and adaptation policy.
- The variability changes is executed in managed application.
## Variability Aspects (2)

<table>
<thead>
<tr>
<th></th>
<th>SPL</th>
<th>Self-adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal of variability</td>
<td>Satisfy individual customer needs</td>
<td>Maintaining service in situation change</td>
</tr>
<tr>
<td>Binding time of variability</td>
<td><strong>Pre-runtime/Runtime</strong></td>
<td>Runtime</td>
</tr>
<tr>
<td>Decision maker of the variant</td>
<td>Application Engineer</td>
<td>Application itself</td>
</tr>
<tr>
<td>Method for selecting the variant</td>
<td>reuse of shared assets conforming a variability model</td>
<td>Following a set of adaptation policies</td>
</tr>
<tr>
<td>Model of variability</td>
<td>Variability model, Architecture model</td>
<td>Adaptation policy, variability model, architecture model, context model</td>
</tr>
<tr>
<td>Lifetime of the variability model</td>
<td>Pre-runtime</td>
<td>Runtime</td>
</tr>
</tbody>
</table>

(Alves et. al 2009)
Motivation
- Software Product Line
- Self-adaptive Systems
- Synergy

Introduction

Adaptation aspects

Overview of three DSPL approaches

Engineering of adaptive systems & DSPL

Research Challenges

Summary
Managing variability self-adaptive systems is a complex task.

More consolidated methods systematically address runtime variability.

SPL & Self-adaptive systems have the similar goal of satisfying specific needs of various environments and users through variability management.
For their rich practices and models for managing variability in software development process, SPL infrastructures are used at runtime to develop self-adaptive systems.

Major uses of SPL in design of self-adaptive system:

- Variability models
- Product derivation approaches
## Variability Aspects (3)

<table>
<thead>
<tr>
<th></th>
<th>SPL</th>
<th>Self-adaptive</th>
<th>DSPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal of variability</td>
<td>Satisfy individual customer needs</td>
<td>Maintaining service in situation change</td>
<td>Maintaining service in situation change</td>
</tr>
<tr>
<td>Binding time of variability</td>
<td>Pre-runtime/Runtime</td>
<td>Runtime</td>
<td>Pre-runtime/Runtime</td>
</tr>
<tr>
<td>Decision maker of the variant</td>
<td>Application Engineer</td>
<td>Application itself</td>
<td>Application Engineer/User/Application itself</td>
</tr>
<tr>
<td>Method for selecting the variant</td>
<td>reuse of shared assets conforming a variability model</td>
<td>Following a set of adaptation policy</td>
<td>Use of SPL assets at runtime</td>
</tr>
<tr>
<td>Model of variability</td>
<td>Variability Model, Architecture model</td>
<td>Adaptation policy, variability model, architecture model, context model</td>
<td>Adaptation policy, variability model, architecture model, context model</td>
</tr>
<tr>
<td>Lifetime of the variability model</td>
<td>Pre-runtime</td>
<td>Runtime</td>
<td>Pre-runtime/Runtime</td>
</tr>
</tbody>
</table>
Motivation

Introduction

Adaptation aspects

Overview of three DSPL approaches

Engineering of adaptive systems & DSPL

Research Challenges

Summary
Utilizes parts of SPL infrastructure to adapt at runtime.

DSPL Properties:
- Runtime reconfiguration
- Self-management *(Optional)*

DSPL Goal:
- Market forces→individual environment/context situation

(Hallsteinsen et al. 2008)
Runtime reconfiguration: DSPL

- allows for dynamic variability: configuration and binding at runtime,
- changes binding several times during its lifetime,
- introduces variation points [which] change during runtime: variation point addition (by extending one variation point),
- deals with unexpected changes (in some limited way), and
- deals with changes by users, such as functional or quality requirements.

(Hallsteinsen et al. 2008)
Self-management (*Optional*):

- “context awareness”
- “autonomic or self-adaptive properties”
- “automatic decision making”

(Hallsteinsen et al. 2008)
When users’ requirements change at runtime

An automated transportation system (Helleboogh et al. 2009)

- The user wants to be able to change the operation mode of transportation system in special situation (e.g. when goods arrive by truck)
When environment changes at runtime

Distributed nodes for Flood Monitoring (Bencomo et al. 2008)

- The node software priority over performance and power efficiency changes according to likelihood of flood
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
Adaptation is change of system properties and behavior at runtime in response to dynamically varying user needs and resource constraints.
Adaptation aspects

- Degree of adaptation
- Degree of autonomy
- Adaptation trigger
- Architectural platform
- Adaptation goal
- Evolution
Behavioral adaptation. The system dynamically changes its behavior within its existing structure. There is no change to the system structure or architecture.
- Parameterization
- Inheritance
- Try...catch

Component adaptation. Dynamic adaptation involves replacing one component with another that has the same interface.

Architectural adaptation. The software architecture has to be modified as a result of the dynamic adaptation.

(Gomma et al. 2011)
Degree of autonomy

- Basic
- Managed
- Predictive
- Adaptive
- Autonomic

(Huebscher et al. 2008)
Change in user requirements
Requires capturing user requirement

Change in running context
Requires capturing running context
Component-based platform

- has good granularity for adaptation
- is not designed to reconfigure at runtime (this problem is handled by architectural styles)

Service-oriented platform

- has been built with the goal of reusing existing assets for developing new products.
- consists of services which are loosely coupled which eases adaptation of the system.
Self-configuration
Self-optimization
Self-healing
Self-protecting

(Kephart et al. 2003)
Evolution is the degree to which unanticipated or unforeseen changes, e.g. adding new features, can be absorbed by the system at runtime.

**Bounded Adaptation.**
- Adaptation policies remain same at runtime.
- Context variation is anticipated at design time and is accommodated by DSPL models.

**Open Adaptation.**
- Adaptation policies can be updated at runtime (e.g. according to feedback).
- New variants can be introduced at runtime (when facing unanticipated situations).
Motivation
Introduction
Adaptation aspects
Overview of three DSPL approaches
- MADAM
- Models@Runtime
- MoRE
Engineering of adaptive systems & DSPL
Research Challenges
Summary
MADAM is a framework for developing mobile and distributed applications that adapt dynamically to changes in context (at launch time and during use).

Applications must adapt to such changes in order to sustain availability, usability and usefulness.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of dynamism</td>
<td>Component adaptation</td>
</tr>
<tr>
<td>Degree of autonomy</td>
<td>autonomous</td>
</tr>
<tr>
<td>Adaptation trigger</td>
<td>Change in running context</td>
</tr>
<tr>
<td>Architecture platform</td>
<td>Component-based</td>
</tr>
<tr>
<td>Adaptation goal</td>
<td>Self-optimizing</td>
</tr>
<tr>
<td>Evolution</td>
<td>Bounded</td>
</tr>
</tbody>
</table>
(Hallsteinsen et al. 2006)
Adaptation in architecture

\[ \text{rsp} = \text{Db.rsp} \]
\[ \text{haf} = \text{Ui.haf} \]

\[ \text{nbw} = \text{Db.nbw} \]

\[ \text{mem} = \text{Ui.mem+Ctrl.mem+Db.mem} \]

(Floch et al. 2006)
Properties and Utility

<table>
<thead>
<tr>
<th>Name</th>
<th>Value range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp</td>
<td>1:100</td>
<td>Response time</td>
</tr>
<tr>
<td>mem</td>
<td>1:100</td>
<td>Amount of memory</td>
</tr>
<tr>
<td>nbw</td>
<td>1:100</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>haf</td>
<td>yes, no</td>
<td>Handsfree operation</td>
</tr>
</tbody>
</table>

\[
\text{utility} = \left( \frac{1}{2} \left( \begin{array}{l}
\text{if} \ \text{usr.rsp} \geq \text{app.rsp} \ \text{then} \ 1 \\
\text{else} \ 1 - \frac{\text{app.rsp} - \text{usr.rsp}}{\text{app.rsp}}
\end{array} \right) + \left( \begin{array}{l}
\text{if} \ (\text{usr.haf} \ \text{and} \ \text{app.haf}) \\
\text{or} \ (\text{usr.haf} \ \text{and} \ \text{app.haf}) \ \text{then} \ 1 \\
\text{else} \ 0
\end{array} \right) \right)
\]

(Floch et al. 2006)
Configurable product bases which are extended to be used at runtime are utilized in building the new configuration of the system after adaption.
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
  - MADAM
  - Models@Runtime
  - MoRE
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
A framework for adaptive applications by adopting ideas from software product line and aspect oriented design at runtime.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of dynamism</td>
<td>Architecture adaptation</td>
</tr>
<tr>
<td>Degree of autonomy</td>
<td>autonomous</td>
</tr>
<tr>
<td>Adaptation trigger</td>
<td>Change in running context</td>
</tr>
<tr>
<td>Architecture platform</td>
<td>Component-based</td>
</tr>
<tr>
<td>Adaptation goal</td>
<td>Self-configuring/Self-optimizing</td>
</tr>
<tr>
<td>Evolution</td>
<td>Open</td>
</tr>
</tbody>
</table>
Architecture

(Morin et al. 2009)
Complex event monitoring: This component observes runtime events generated by probes integrated into the System and updates context model of system accordingly.
Goal-based reasoning engine
Selects features from feature model following the reasoning model adapted to the current context.
Aspect model weaver receives a derived feature model from the reasoning engine. For each of the features of this model, the weaver composes a corresponding aspect to the base model to produce the targeted architecture model.
Configuration invariant checker checks if the architecture model is consistent and revokes the adaptation if it is not.
Configuration manager receives new configuration of the system and is responsible to reconfigure the system using the services offered by the platform.
Feature models are used at runtime to model the variability of the system.

Variants of the system are selected by configuring the feature model at runtime according to context.

The configuration of the final product is built using Aspect Model Weaving from SPL product derivation.
Motivation
Introduction
Adaptation aspects
Overview of three DSPL approaches
  - MADAM
  - Models@Runtime
  - MoRE
Engineering of adaptive systems & DSPL
Research Challenges
Summary
A framework for developing pervasive applications (such as smart home) that adapt dynamically to changes in available services and devices to maintain system service.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of dynamism</td>
<td>Architecture adaptation</td>
</tr>
<tr>
<td>Degree of autonomy</td>
<td>autonomous</td>
</tr>
<tr>
<td>Adaptation trigger</td>
<td>Change in running context</td>
</tr>
<tr>
<td>Architecture platform</td>
<td>Service-oriented</td>
</tr>
<tr>
<td>Adaptation goal</td>
<td>Self-healing</td>
</tr>
<tr>
<td>Evolution</td>
<td>Bounded</td>
</tr>
</tbody>
</table>
Reconfiguration Process
Variability Model

(Cetina et al. 2009)
Conditions are states of system and environment.

- \{NewVolumetricSensor, AlarmFailure, EmptyHome\}

Resolutions are sets of feature activations/deactivations when a condition triggers.

- \[ R_{EmptyHome} = \{(OccupancySimulation, Active), (InHomeDetection, Active), (LightingByOccupancy, Inactive)\} \]
From features to components

- \( \text{LightingByOccupancy} = \{a, g\} \)
- \( \text{OccupancySimulation} = \{1, b, c, d\} \)
- \( \text{InHomeDetection} = \{e, f\} \)
Feature models are used at runtime to model the variability of the system.

Variants of the system are selected by configuring the feature model at runtime according to context.
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
Self-adaptive systems are designed in two layers

- Application logic: focuses on application functionality
- Adaptation logic: focuses on application adaptability

The two-layer design promotes separation of concerns and results in a system

- which is less complex and easier to extend
- whose components are reusable
General structure of a self-adaptive system
Inside of an adaptation manager

(Kephart et al. 2003)
The MAPE-K loop corresponds to product derivation in SPL

- SPL models can be used as knowledge (e.g. variable model, decision making, ...)

- SPL inspires approaches which can be used in planning and executing in the MAPE-K loop (e.g. configurable product family)

- SPL has different motives for creating new variants of software; therefore, the SPL infrastructure is not used in monitoring and analyzing

- The output of product derivation in SPL is a new product, while that of the MAPE-K loop is a set of changes in the current product.
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- **Engineering of adaptive systems & DSPL**
  - Conceptual Model
  - **Architecture**
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
Architecture of an adaptation manager

(Kramer et al. 2007)
Example (Models@Runtime)

Goal Management

Change Management

Component Control

(Morin et al. 2009)
Example (MADAM)

Goal Management

Change Management

Component Control

(Hallsteinsen et al. 2006)
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
involves capturing properties of the environment that are of significance to the adaptive properties of the system.

could be: Physical (e.g. temperature) or Virtual (e.g. CPU utilization, response time).

is done using sensors

have been widely used in networks and distributed systems.
Context models represent formally the parts of the context which are important for making decisions about the configuration of the software application.

OWL (Web Ontology Language)
- Is a Markup language
- Enables context sharing & context reasoning
Example (Context Model)
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
Rule-based. Decision making is performed by following a set of rules which determine what particular action should be performed in each context.
- Event-condition-action (ECA) policies

Goal-based. These approaches specify criteria that characterise the desirable configuration of the system but leave to the system the task of finding out how to achieve the configuration having those characteristics.
- AI Planning

Utility-based. This set of approaches define a quantitative level of desirability for each state.
- Optimization
Analysis determines when a change is required by analyzing the symptoms provided through monitoring and the history of the system.

Analysis implicates the following:

- Event and conditions, in rule-based approaches.
- The current state of the system while achieving goals, in goal-based approaches.
- The current utility and state of the system, in utility-based approaches.
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
Planning involves taking into account the state of the system in order to select a new variant of the managed element.

Planning can be done in two levels in DSPL:
- Feature level
- Architectural level

In DSPL, planning in feature level is similar to product configuration in SPLE.

Feature level planning improves separation of concern.

If feature level is used, feature configuration should be mapped to the architectural level configuration.
Planning (Knowledge)

- Variability Model
- Adaptation Policy
- Architecture Model
Enumeration of the system variants

- Explicitly lists variants of the system and their corresponding configurations are listed.

Pros.

- Simple
- Easier to validate

Cons.

- has limitations for large systems
Types

- The variation of the system is defined by a set of variation points. Variation points allow variants of same type to be replaced for each other.

Pros.

- Simple
- Manageable

Cons.

- Cross-cutting variations constraint
Example (MADAM)

\[
\text{rsp} = \text{Db.rsp} \\
\text{haf} = \text{Ui.haf}
\]

\[
\text{nbw} = \text{Db nbw}
\]

\[
\text{mem} = \text{Ui.mem} + \text{Ctrl.mem} + \text{Db.mem}
\]

(Hallsteinsen et al. 2006)
Feature Model

- A feature model provides a tree-like structure that shows the organization of features. In a feature model, features are hierarchically organized by structural constraints.
- A feature model supports cross-cutting variations using integrity constraints.

Pros.

- Formal representation
- Tool support
Example (MoRE)

(Parra et al. 2009)
Used to define architectural model of the system (e.g. UML component model)
  - Architecture Definition Language (ADL)
  - Domain Specific Modeling Language (DSML)

Used for:
  - Current architecture of the system
  - Target architecture of the system

Usually represented by a graph-like structure where nodes are components and arcs represent binding.

Is available at runtime and should be memory efficient
Rule-based Approaches

- **State Transition Diagram**
  - States represent variants of a system in adaptation
  - Transitions represent the possible transitions between variants
  - Guard of transitions determine when a transition can happen

- **Planning at design time**

- **Pros**
  - Simple
  - Easy validation

- **Cons**
  - State explosion
  - Hard to modify after design (mostly for evolution at runtime)
Example (Bencomo et al. 2008)
Rule-based Approaches

- **Event-Condition-Action rules**
  - Events are changes in environment (e.g. network traffic) or system state (e.g. processor usage) or a combination of these.
  - Condition represent the current conjuration of the system and its context.
  - Actions are changes over features or components.

- **Planning at design time**

- **Pros.**
  - Readability and elegance of each individual rules
  - Efficient process
  - Easy to Modify (by adding removing new rules)

- **Cons.**
  - Scalability (Possibility of conflicting rules)
  - Application of stateless manner is limited
  - Validation
Example (Parra 2009)

class context-aware-asset {
    isDeterminate = 'true';
    clause {
        condition
            expression = 'if value > 14'
            contextNode = 'BANDWIDTH'
        dynamic-adaptation {
            point description = '...'
            change change-actions = '...'
        }
    }
    clause {
        condition
            expression = 'if value < 8'
            contextNode = 'BANDWIDTH'
        dynamic-adaptation {
            point description = '...'
            change change-actions = '...'
        }
    }
}
A goal-based model defines
- How changes (e.g., inclusion/exclusion of features) impact the goal of the system in a specific context.
- When the system should change.

Planning at runtime
Reducing to satisfiability problem (SAT), constraint satisfaction problem (CSP).

Pros.
- More likely to find the best adaptation

Cons.
- Hard to design
- More resource usage
Utility-based Approaches

Utility-based model defines

- A utility function representing desirability of a configuration
- How changes (e.g., inclusion/exclusion of features) impact the goal of the system
- When the system should change according to utility

Planning at runtime

Pros.

- More likely to find the best adaptation

Cons.

- Hard to design
- More resource usage
Example (MADAM)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp</td>
<td>1:100</td>
<td>Response time</td>
</tr>
<tr>
<td>mem</td>
<td>1:100</td>
<td>Amount of memory</td>
</tr>
<tr>
<td>nbw</td>
<td>1:100</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>haf</td>
<td>yes, no</td>
<td>Handsfree operation</td>
</tr>
</tbody>
</table>

\[
\text{utility} = \left( \begin{array}{c}
\text{if } \text{usr.rsp} \geq \text{app.rsp} \text{ then 1} \\
\text{else } 1- (\text{app.rsp-usr.rsp})/\text{app.rsp} \\
+ (\text{if ((usr.haf and app.haf)} \\
\text{or (lusr.haf and lapp.haf) then 1} \\
\text{else 0)} \\
\end{array} \right) / 2
\]

(Floch et al. 2006)
These approaches fill the gap between features and architecture.

No situation specific development is possible unlike SPL (extensive reuse)
- Configurable Product Family

A change in the features of the system is mapped to changes in architecture.
- Direct link
- Transformation rule
- Aspect Model Weaving
- Common Variability Language(CVL)
Direct mapping between features in feature model to architectural fragments in system implementation.

By selecting/deselecting features mapped fragments activated/deactivated

Pros.
- Simplicity
- Realization

Cons.
- Separation of concern
- Not applicable in many cases
The relation between feature model and architecture is represented by using a set of rules (e.g. prepositional logic rules).

After selecting/deselecting features, corresponding changes should be made in architecture such that the rules hold.

Pros
- More complex relations between features and architecture are enabled.

Cons.
- Hard to find rules
- Hard to derive architectures which sustain the rules
Example (Acher et al. 2011)
Automates the generation of detailed architecture from high level designs.

In aspect model weaving:
- The application commonalities are represented in a base model.
- Each feature of the system is mapped to a set of aspect models which are woven into the base model to create the final model of the system.

Examples:
- SMARTADAPTORS
- Kompose
Common Variability Language (CVL)

- CVL: Generic & Standardized
- DSL: Focused on a domain

Variability model:
- Description of possible variations in the system
- Base model: Domain model of a particular family of system
- Execution of CVL
- Resolution models: Selection of a set of options in the variation model
- Resolved models: Family of systems fully described in the domain specific language. All regular DSL tools can be applied to these models.
Motivation
Introduction
Adaptation aspects
Overview of three DSPL approaches
Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
Research Challenges
Summary
Enabling runtime reconfiguration

- Reflective Middleware
  - provides a model of the system in which any change in the system will be reflected in the model and similarly any change in the model is reflected on the system.
  - Introspection:
    - accesses the representation of system behaviour using this model.
  - Intercession:
    - reconfigures the system by modifying this model
Component Model defines
- the semantics of components
- the syntax of components
- the composition of components

Hierarchical component model allows
- each component to be made of smaller components.
- changes in different levels of granularity.
OpenCOM is a language-independent, component-based middleware which supports:

- Hierarchal structure
- Reflection using graph representation of architecture
  - Adding a node to the graph results in the deployment of a new component
  - Removing an arc results in the breaking of an inter-component binding
```csharp
loaded_component loader_cf.load(component_type);
loaded_component loader_cf.load(caplet, component_type);
loaded_component loader_cf.load(loader_name, component_type);
loaded_component loader_cf.load(caplet, loader_name, component_type);

binding binder_cf.bind(interface_instance, receptacle_instance);
binding binder_cf.bind(binder_name, interface_instance, receptacle_instance);
```

(Coulson et al. 2004)
Services
Reflection
Dynamic install, start, restart and uninstall of services
Bundle
- A bundle is the deliverable application
- A bundle registers zero or more services
- Searches can be used to find services registered by other bundles
- OSGi defines a standard set of services for bundle (e.g. for lifecycle management)

OSGi Wire
- connection between a Producer service and a Consumer service.
OSGi Environment

(Marples et al. 2001)
public interface GPS {
    ...
}

public void foo(BundleContext context) {
    Garmin garmin = new Garmin(...);
    Hashtable properties = new Hashtable();
    properties.put("vendor", "garmin");
    ServiceRegistration reg = context.registerService(GPS.class.getName(),
        garmin,
        properties);
}

public void bar(BundleContext context) {
    ServiceReference ref = context.getService(GPS.class.getName());
    GPS gps = context.getService(ref);
    ...
}
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
  - Conceptual Model
  - Architecture
  - MAPE-K Loop
  - Process
  - Research Highlights
- Research Challenges
- Summary
Two phases

- **Domain Engineering:**
  - Defining Context model
  - Defining binding units & binding time
  - Defining feature to architecture link
  - Designing runtime reconfiguration

- **Application Engineering**
  - Analyzing binding time
  - Customizing adaptation manager
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
  - **Engineering of adaptive systems & DSPL**
    - Conceptual Model
    - Architecture
    - MAPE-K Loop
    - Process
    - Research Highlights
- Research Challenges
- Summary
BPEL provides a language for the formal specification of business process behavior based exclusively on Web Services.

- Captures business logic and behavior of service interactions
- Supports service composition at executable level

BPEL draws upon concepts and constructs from imperative programming languages, and extends them to those related to Web services and business processes.
Considering features as fragments of BPEL code

Dynamic Aspect Weaving allows the incorporation of new aspects in runtime in a program.

- A BPEL code is selected as base model
- Every model variant is expressed in terms of *substitutions* that it needs to incorporate into the base model to include that variant.
Example (Baresi et al. 2012)
Ability to introduce the variants which were not considered at design time and introducing them into the system at runtime.

In order to:
- Handle unanticipated situations
- Enhance efficiency

Two types of evolution:
- Evolution on how the system adapts
  - Modifying ECA rules
- Evolution on system variants
  - Discovering new web services addressing features
Example (Perrouin et al. 2012)

Evolution Layer
(Meta-Adaptation Layer)

Knowledge

Requirements

Evolution Layer
(Meta-Adaptation Layer)

A
M
P
E

Adaptation Layer

A
M
P
E

Business System

c1
c2
c3
c4

monitor
modify

monitor
modify

Knowledge
Example (Helleboogh et al. 2009)
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
Research Challenges

- Formalizing concepts
  - Need for exact definition of concepts used in DSPL

- Process
  - Requirement engineering
  - Adaptation engineering

- Decentralized DSPL
  - Interaction between DSPL

- Experience reports
Run-time Verification
  • Ensuring safety of adaptation

Testing and assurance.
  • Different variants of the system work correctly
  • The combinations of adaptations do not make the system inconsistent

Evaluation and quality of adaptation
  • Safety, security, and performance evaluation

Implementation tools
  • To be used in practice
Outline

- Motivation
- Introduction
- Adaptation aspects
- Overview of three DSPL approaches
- Engineering of adaptive systems & DSPL
- Research Challenges
- Summary
In this tutorial we covered:

- Motivation for DSPL
- Adaptation aspects
- Designing a Dynamic Software Product Line
  - Architecture
  - MAPE-K Loop
- Research Challenges


Thank you!