A Resource Model For Adaptable Applications

ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)

F.Mancinelli, P.Inverardi

Dipartimento di Informatica
Università dell'Aquila
Italy

May 22, 2006

Talk Outline

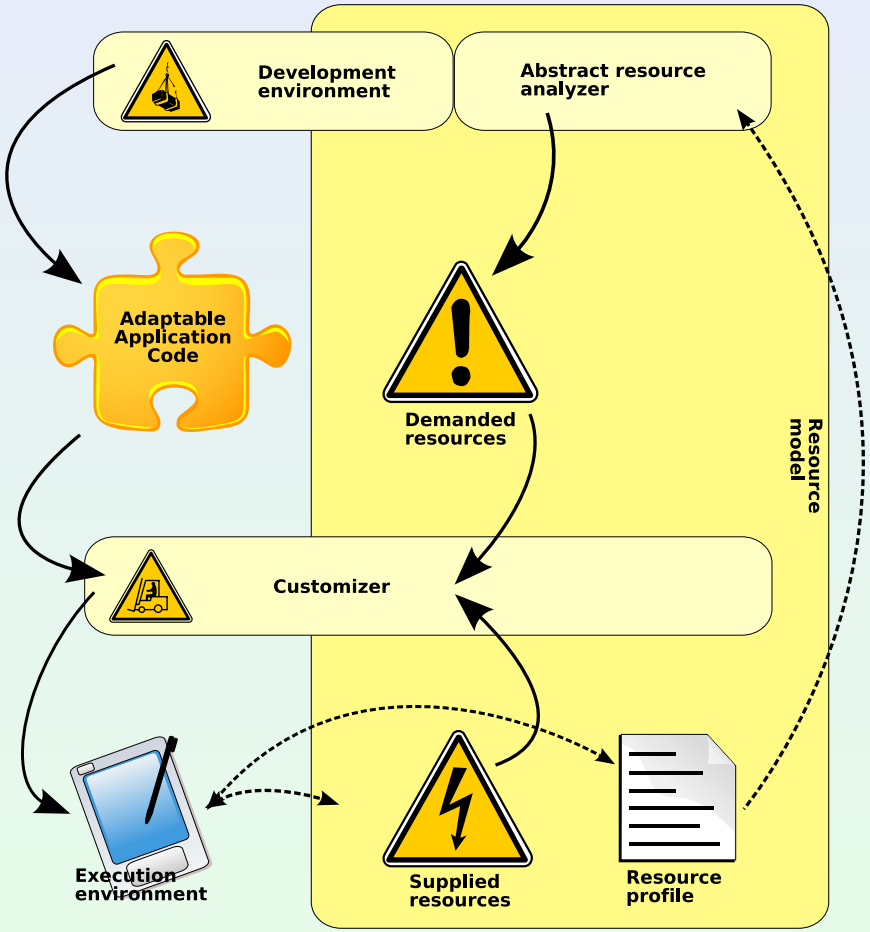
- Application context.
- A framework for resource-aware adaptable software applications.
- The resource model.
  - Resource formalization.
  - *Compatibility*.
  - *Priorities* and *goodness*.
  - Resource profiles.
- Conclusions and future work.

Application Context

We are considering the context in which:

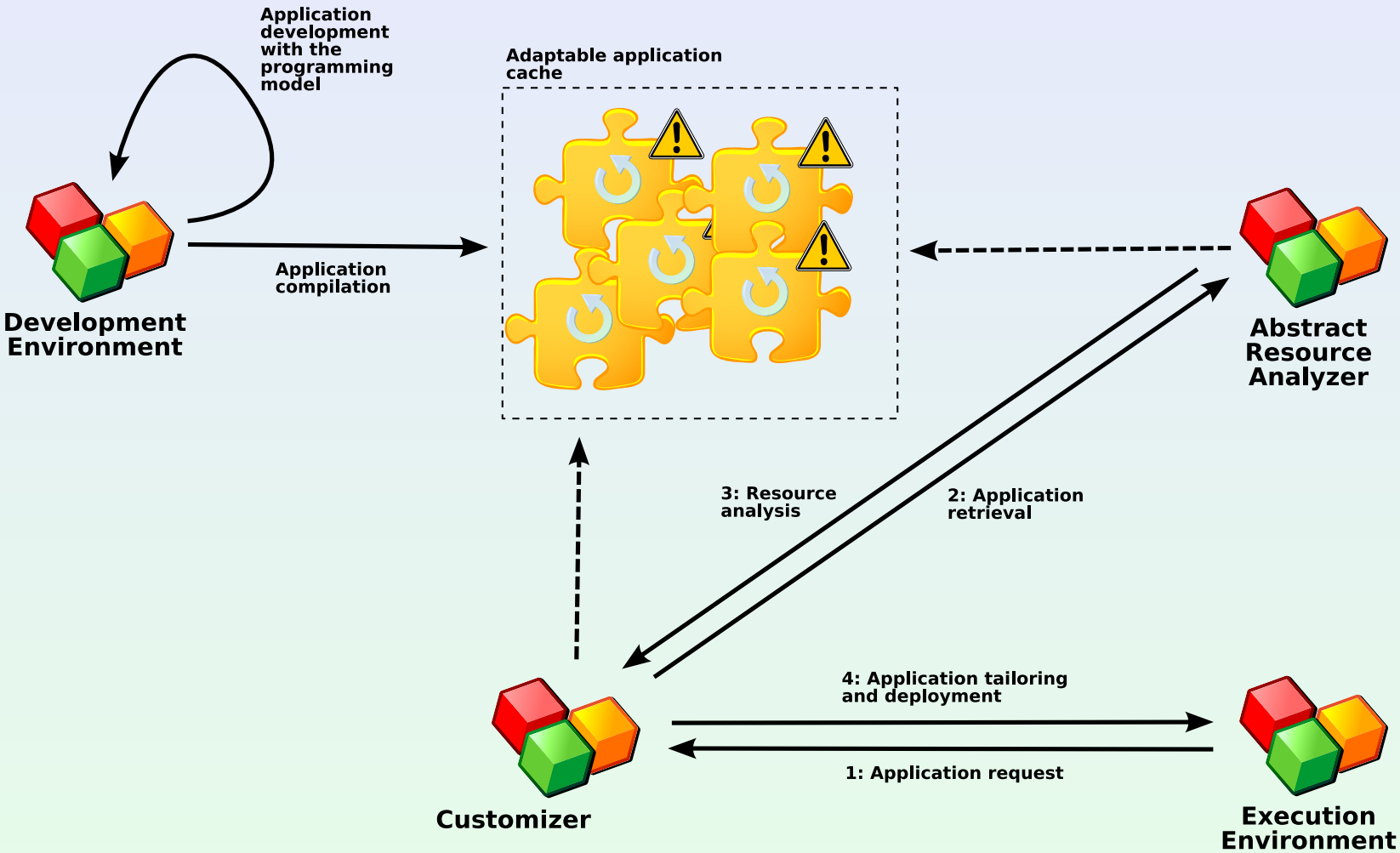
- Mobile and limited devices are requesting small applications through a client/server interaction.
- Applications must be adapted with respect to the characteristics of the execution environment of the device requesting them.
- The delivered applications are tailored applications, i.e., they are the result of a (static) adaptation process that has been formerly applied.
- Applications are written using the Java language.

The Framework



- Development environment. It is supported by a *programming model* for defining how the application could be adapted.
- Abstract resource analyzer. Provides the characterization in terms of *resource demands* of the possible adaptations, according to the characteristics of the execution environment defined through *resource profiles*.
- Customizer. Analyzes the resource demands of the possible adaptations in order to choose the best one with respect to the *resources supplied* by the execution environment and its characteristics.

The Framework Workflow



The Resource Model: Definitions

Definition (Resource)

A **resource** is a typed identifier consisting of naturals, booleans or enumerated values. By being defined with the support of *totally ordered* sets, resources have an implicitly defined total order depending on their type:

- *natural*-typed resources can be compared by using the standard $<$, $\leq$, $\geq$, $>$ relations.
- *boolean*-typed resources can be compared by assuming that $\mathit{false} < \mathit{true}$.
- *enumerated*-typed resources can be compared by assuming a position-wise mapping to naturals, with respect to enumeration order.

The Resource Model: Definitions

Definition (Resource instance)

A **resource instance** is an expression where both the resource and an actual value for that resource are specified in the following form: $Res(Value)$. $Value$ must have the same type of the one associated to the resource Res.

Definition (Resource set)

A **resource set** is a set of resource instances.

Example (Resource set)

$Power : Integer;$
$Bluetooth : Boolean;$
$3Drendering : Boolean;$
$RS = \{Power(100), Bluetooth(true), 3DRendering(true)\}$

The Resource Model and the Framework

Resource sets are used throughout the framework components with different purposes:

- In the programming model: to allow the developer to provide *annotations* to better characterize its applications.
- In the abstract resource analyzer: as the result of the analysis which describe the overall *resource demands* of the different adaptations.
- In the execution environment: to describe the current *resource supplies*.

The Resource Model: Relationships among Resource Sets

In order to be able to reason on adaptation we need a way to relate resource sets in order to decide:

- Whether a *resource set* describing a resource demand (of an adaptation) is compatible with the *resources supplied* by a given *execution environment* or not.
- Whether a *resource set* describing a resource demand of a program adaptation PA_1 is *more convenient* with respect to a resource demand of another program adaptation PA_2.

Resource Model: Compatibility

Definition (Compatibility)

A *resource set* $R_1 = \{Res_{1,1}(v_{1,1}), Res_{1,2}(v_{1,2}), \ldots, Res_{1,i}(v_{1,i})\}$ is compatible with a resource set $R_2 = \{Res_{2,1}(v_{2,1}), Res_{2,2}(v_{2,2}), \ldots, Res_{2,j}(v_{2,j})\}$, i.e., $R_1 \triangleright R_2$ if:

1. **(Availability)** For every *resource instance* $Res_{1,k}(v_{1,k}) \in R_1$ of type $Res_{1,k}$ there exist a *resource instance* $Res_{2,l}(v_{2,l}) \in R_2$ of the same type.
2. **(Wealth)** For every pair of *resource instances* $Res(v_{1,k}) \in R_1$ and $Res(v_{2,l}) \in R_2$ of type Res, $v_{1,k} \leq v_{2,l}$ according to the order relation associated to the type Res.

Resource Model: Compatibility

Example (Compatible resource sets)

$\{Power(45), Bluetooth(true)\} \triangleright$
$\{Power(100), Bluetooth(true), 3DRendering(true)\}$

Example (Incompatible resource sets)

$\{Power(45), Bluetooth(true)\} \not\triangleright \{Power(100), 3DRendering(true)\}$
$\{Power(150), Bluetooth(true)\} \not\triangleright \{Power(100), Bluetooth(true)\}$

Resource Model: Goodness

Definition (Resource priority)

The **resource priority** is a total function $\mathcal{RP}$ that associates to every resource one of the following weights : -1, 0 or 1.

Definition (Goodness)

Given a *resource set* $R = \{r_1(v_1), \ldots, r_i(v_i)\}$ we define its **goodness** the following value:

$$\mathcal{G}(R) = \sum_i \mathcal{RP}(r_i) \cdot v_i$$

Resource Model: Goodness

- By using the goodness function the customizer can discriminate among compatible resource sets.
- The resource priority function can be used to change the "importance" given to a particular resource.
  - If $\mathcal{RP}(r) = -1$ then higher values of the resource r contribute to decrease the overall *goodness* of the resource set. This is the case of that kind of resources whose consumption (or presence) follows "the less the better" principle (e.g., *Power*).
  - If $\mathcal{RP}(r) = 0$ then the resource r is ignored and does not contribute to raise the *goodness* of the resource set.
  - If $\mathcal{RP}(r) = 1$ then higher values of the resource r contribute to increase the overall *goodness* of the resource set. This could be the case of that kind of resources whose consumption (or presence) follows "the more the better" principle (e.g., *Threads*).
- The goodness function does not provide a total order: different resource sets might have the same *goodness* value.

Resource Model: Goodness

Example (Resource set goodness)

$\mathcal{RP}(Power) = -1$
$\mathcal{RP}(...) = 0$

$\mathcal{G}(\{Power(150), \ldots\}) < \mathcal{G}(\{Power(100), \ldots\})$

- Resource priorities can be used to define different *policies* that privilege certain types of resources with respect to others.
- The most simple policy is the *"don't care"* one, where for every resource type R, $\mathcal{RP}(R) = 0$.
- Resource priorities assignments can be created by end users starting from some predefined patterns and changed to reflect the user's preferences.

Resource Model: Resource Profiles

Resource profiles contain the specification of the structural characteristics of the execution environment affecting the adaptation process. In particular it contains:

- The specification of the resource priorities that must be considered when calculating the goodness of the resource sets associated to different adaptations, and that will affect how the customizer will choose an adaptation.
- The resource bindings between single bytecode instructions patterns and the corresponding resource usages that are to be considered by the *abstract resource analyzer* when performing the analysis of the code for calculating the resource demands of the different adaptations.
  Resource bindings allow the execution environment to specify how it is influenced by the execution of the bytecode (the same instructions may have different effects on different devices)

Conclusions and Future Work

In this work we have presented a simple resource model that is used as the basis of a framework for handling adaptable software applications:

- It provides a way to declaratively specify the characteristics of both the adaptable application and the execution environment in which they will be executed.
- It allows the execution environment to define a custom perspective on the interpretation of the resource demands using resource priorities and the goodness function.
- It provides a flexible way to customize the analysis of the adaptable application code with respect to the way the execution environment handles the actual application code by using different resource bindings.

Future work on the resource model would allow the declarative specification of more properties concerning resource oriented information. In particular:

- Resource relationships: conflicts and dependencies.
- More complex bindings defining parametrical resource instances.