A Resource Model for Adaptable Applications

Fabio Mancinelli, Paola Inverardi
Dipartimento di Informatica
Università dell’Aquila
Via Vetoio, 1
67010 Coppito - L’Aquila
{mancinel,inverardi}@di.univaq.it

ABSTRACT

Adaptable applications are software applications that can be adapted with respect to the environment that will host their execution. In order to be able to perform an adaptation it is essential to provide an actual way to model the characteristics of both the application and the execution environment. In this paper we describe a resource model that is the basis of a framework for the development and the deployment of adaptable applications targeted to mobile and limited devices (e.g., smart phones, PDAs, etc.). By using this resource model we are able to reason about the resources required by an application (and its possible adaptations) and the ones supplied by the hosting environment. In this model we introduce the notions of compatibility and goodness. The first one is used to verify that the supplied resources are enough to satisfy the application needs (and thus to choose the correct adaptations). The second one provides a flexible mechanism that can be used to choose the “best” adaptation.

Categories and Subject Descriptors: D.2.9 [Software Engineering]: Management: Software configuration management

General Terms: Management, Design

Keywords: Resource models, Adaptable applications

1. INTRODUCTION

One of the main issues to face when dealing with today’s computational platforms is related to resource management. During the personal computing era, resources were not given too much attention because the computing platforms were supposed to always have a sufficient supply of them. Applications were built to be executed in well known and restricted environments and failures were due, above all, by mistakes made by the users and not by an inherently faulty environment: failures or resource shortage were seen as a very exceptional event. With the advent of distributed computing, Software Engineering has had to discover new ways of building applications that can experience failures in the behavior of the underlying infrastructure (i.e., the network), due to the inherently faulty nature of it. For example, in such an environment, it was not obvious that a method call would always be successful.

With the introduction of new communication infrastructures, namely wireless and mobile networks, and their integration with the existing ones, new interconnected platforms have been deployed in order to run applications on. Typically these platforms are made by mobile devices, characterized by their heterogeneity and limitedness. In such a context, application may be deployed on an even more faulty ground, furthermore characterized by a scarcity of computational resources, where each computational environment offers a different set of capabilities to the applications.

Resource awareness and adaptation have become two important aspects in the development of applications suited to be executed in such an environment. Resource awareness identifies the capability of being aware of the resources offered by an execution environment, in order to decide whether that environment is suited to receive and execute the application. Adaptation identifies the capability of changing the application in order to comply with the current resource conditions of the execution environment. In this paper we focus on resources by proposing a resource model that is the basis of a framework for the development and the deployment of adaptable applications targeted to mobile and limited devices (e.g., smart phones, PDAs, etc.). The resource model can be profitably used in order to specify the characteristics of both the execution environments and the applications and their adaptations and it allows us to reason on them in order to discriminate among the different ways of adapting the application. This resource model, and the surrounding framework, provides a declarative way of developing, handling and characterizing adaptable applications and represents a novel attempt to organize and rationalize the resource specification.

The paper is structured as follows: Section 2 briefly introduces the framework the resource model is part of; Section 3 details the actual resource model and introduces the notions of compatibility and goodness in order to provide an actual way to discriminate among application adaptation; Section 4 shows how to use the resource model to describe the characteristics of the applications and the environments in which they are executed; Section 5 shows how to use the resource model in order to reason on the possible ways to perform the adaptation; finally, Sections 6 and 7 close the paper by discussing related work and by drawing conclusions.
2. THE FRAMEWORK

In this section we briefly describe the framework [4] we have designed in order to support the development and deployment of adaptable applications. Figure 1 shows the framework architecture, which is composed of the following components:

- **Development environment**: this is the environment where the developer builds the actual adaptable application. The application development is supported by a set of patterns that are used to specify the possible ways the application can be adapted. These patterns are reflected in the compiled code through additional information that is used by the other framework components in the adaptation process.

- **Abstract resource analyzer**: an abstract interpreter based on a resource aware operational semantics [10] that extracts a resource demand from the application code with respect to a resource profile provided by the execution environment. The abstract resource analyzer embodies all the logic needed to explore the space of possible adaptations and evaluates, for each of them, the demanded resources.

- **Customizer**: a component that performs the actual adaptation of the application code with respect to the execution environment. The customizer actually choose the correct adaptation with respect to the information provided through the resource model and is able to assemble the final application code, tailored for the target execution environment and ready to be deployed and run.

- **Execution environment**: the physical device that requests the adaptable application and provides the runtime support.

- **Resource profile and supplied resource**: a declarative description of the resources available in the execution environment. The customizer uses this information, together with the resource demand of the adaptable program, to reason about the adaptation.

The framework has been conceived with a special focus on the small and limited devices, such as smart phones and PDAs. Moreover the programming model that is at its basis is the one of the J2ME (Java2 Micro Edition) [15] platform. The reference architecture depicted in Figure 1 does not suggest any localization of these components on an actual architecture. Depending on the application domain, the components may be deployed in different ways. For example, a typical client-server deployment scenario would deploy the execution environment and its properties on the client side, while the customizer, the adaptable application code, its declarative descriptions and the development environment would be installed on the server.

In order to better explain the relationships between the resource model and the framework components, we point out that the machinery that analyzes the adaptable applications (i.e., the abstract resource analyzer) and associates to it resource information is based on an abstract operational semantics of the Java Bytecode (the reference language for the J2ME platform). The granularity at which the adaptation can be specified is at the bytecode level and, therefore, resource information can be attached down to the single bytecode instruction (see Section 4).

3. RESOURCE MODEL

Broadly speaking a resource is any item that can be used and that is required to accomplish an activity or complete a task. In the real world resources can be people, equipments, facilities, funding or anything else needed to perform the work of a project.

From our perspective, a resource is a device-related item or feature that enables the application to work correctly and to make it achieve what it has been programmed for.

Notice that resources are different in their nature. Some resources are subject to consumption (e.g., power, heap space, etc.), while others, if present, are always available and never exhausted (e.g., software resources such as communication protocol stacks, hardware devices, etc.)

A non-consumable resource, actually, has always some kind of requirements with respect to consumable ones. For example an hardware device needs a certain amount of power in order to work, and the library that implements its driver needs some computational resources in order to make the device usable: there exists a classification of resources that can be labeled as basic and derived ones.

3.1 Basic definitions

For our purposes, we choose to model a resource as a typed identifier that can vary over natural, boolean or enumerated values. Natural values are used for those resources that can have different degrees of availability, that can be measured and that might vary during the execution of a given task (i.e., that are subject to consumption). Boolean values are used to define resources that can be either present or not and that do not have an associated notion of availability (non-consumable ones). Enumerated values can be used to define resources as in the natural case (enumerated values can be mapped to naturals straightforwardly) and provide a restricted set of admissible values.
In order to better present our resource model, we introduce the following definitions:

**Definition 1 (Enumerated Type).** An enumerated type is a scalar type consisting of a specified set of ordered values.

**Definition 2 (Resource).** A resource is a typed identifier whose values can vary on naturals, booleans or enumerated values.

**Definition 3 (Resource Instance).** A resource instance is an expression where both the resource and an actual value for it are specified in the following form: \( \text{Res}(\text{Value}) \). Values must have the same type of the one associated to the resource \( \text{Res} \).

We can think that resources are defined by using a mechanism similar to the C language `typedef` construct [5], i.e., in order to introduce a new resource a new identifier is declared and treated as a synonym of the natural or boolean basic types or as an enumerated type.

**Definition 4 (Resource Type).** The resource type of a resource is the actual type (natural, boolean or enumerated) of the given resource. We use the `typeof` operator to indicate it.

By being defined with the support of `totally ordered` sets, resources have an implicitly defined total ordering depending on their type. This is important because the total ordering is necessary in order to compare resource instances and sets of resource instances.

Natural-typed resources can be compared by using the standard \( <, \leq, \geq, > \) relations. For boolean-typed resources we assume that \( \text{false} < \text{true} \). Finally, enumerated resources are ordered by mapping the enumerated type elements to naturals and then using the standard order relations on naturals. The mapping is done position-wise from left to right with respect to the declaration order, i.e., the lastmost declared identifier is mapped to 1 and the identifiers that follows are mapped to strictly one-stepped increasing naturals. For instance, let \( E \) be an enumerated type \( E = \{id_1, id_2, \ldots, id_i\} \), then the mapping would be \( id_1 \rightarrow 1, id_2 \rightarrow 2, \ldots, id_i \rightarrow i \).

### 3.2 Resource maps and sets

We can use resources in order to describe what is provided by a given environment and what is needed by a given application in order to be executed. Usually this cannot be done by using a single resource because often these descriptions are made of a remarkable number of them.

In order to provide a characterization in term of resources of the needs of a program and of the environmental supplies, we introduce the definition of resource map:

**Definition 5 (Resource Map).** A resource map is a function that maps resources (identifiers or types) to resource instances of the same type:

\[
\text{RM} : \text{ResId} \rightarrow \text{ResId}(v) \quad \text{where} \quad v \in \text{typeof}(\text{ResId})
\]

\[
\forall r \in \text{ResId}, \text{RM}(r) = r(v)
\]

Besides resource maps we also define resource sets:

**Definition 6 (Resource Set).** A resource set is a set of resource instances, where every resource type may appear at most one time.

We can consider resource sets as the image of some resource map. For example, the resource set \( R_{S1} \) in Figure 2 would be the image of the resource map function \( R_{M_{S1}} \) that describes the following mapping:

\[
\begin{align*}
\text{Power} & \rightarrow \text{Power}(100) \\
\text{Bluetooth} & \rightarrow \text{Bluetooth}(\text{true}) \\
\text{3DCaps} & \rightarrow \text{3DCaps}(\text{true})
\end{align*}
\]

When dealing with resource sets, we need a way to relate them in order to decide:

- Whether a resource set describing a resource demand is compatible with the resource supplied by a given environment or not.
- Whether a resource set describing a resource demand of a program \( P_1 \) (i.e., an adaptation of program \( P \)) is more convenient with respect to a resource demand of a program \( P_2 \).

Since resource instances are comparable elements due to the total ordering defined on their associated resource types, we can use this fact in order to compare resource sets.

**Definition 7 (Supporting Set).** A supporting set of a resource set \( R \) is the set of all the resource types that appear instantiated in \( R \). We will write \( S(R) \) in order to indicate the supporting set of \( R \).

We point out that the supporting set of a resource set \( R \) can be seen as the co-image of a resource map having as its image the resource set \( R \). Figure 3 shows the relations among the definitions we have given so far.

### 3.3 Resource set compatibility

Now we introduce a notion of compatibility between resource sets:

**Definition 8 (Compatibility).** A resource set \( R_1 = \{\text{Res}_{1,1}(v_{1,1}), \text{Res}_{1,2}(v_{1,2}), \ldots, \text{Res}_{1,i}(v_{1,i})\} \) is compatible with a resource set \( R_2 = \{\text{Res}_{2,1}(v_{2,1}), \text{Res}_{2,2}(v_{2,2}), \ldots, \text{Res}_{2,j}(v_{2,j})\} \) if

1. (Availability). For every resource instance \( \text{Res}_{1,k}(v_{1,k}) \in R_1 \) of type \( \text{Res}_{1,k} \), there exist a resource instance \( \text{Res}_{2,l}(v_{2,l}) \in R_2 \) of the same type. That is \( S(\text{R}_1) \subseteq S(\text{R}_2) \).
2. (Wealth). For every pair of resource instances \( \text{Res}(v_{1,k}) \in R_1 \) and \( \text{Res}(v_{2,l}) \in R_2 \) of type \( \text{Res} \), \( v_{1,k} \leq v_{2,l} \) according to the ordering relation associated to \( \text{typeof}(\text{Res}) \).

\[
R_{S1} = \{\text{Power}(100), \text{Bluetooth}(\text{true}), \text{3DCaps}(\text{true})\}
\]

(a) A valid resource set

\[
R_{S2} = \{\text{Power}(50), \text{Bluetooth}(\text{true}), \text{Bluetooth}(\text{false})\}
\]

(b) An invalid resource set

**Figure 2: Resource sets**
The previous definition of compatibility between two resource sets is established in order to verify that a given resource demand is compatible with the corresponding resource supply, provided by an environment. The availability condition ensures that each demanded resource type is also supplied (i.e., the supporting set of the demanded resources is included in the one of the supplied resources). The wealth condition ensures that for every needed resource type a sufficient amount of that resource is provided.

We point out that the availability condition in definition 8 can be also ruled out by assuming that in every resource set all the missing resource types are instantiated to the bottom value of the total ordering associated to their relative resource type. So, for example, every missing boolean-typed resource $R_i$ would be considered present in the resource set instantiated as $R_i(false)$. In a similar way, missing natural-typed resources $R_i$ would be implicitly instantiated as $R_i(0)$.

The meaning of the addition and multiplication operators are overloaded in order to cope with the non-natural resource types. In particular, in the case of boolean-typed resources we consider the boolean value true as if it was the natural number 1 and the boolean value false as if it was the natural number 0.

Enumerated resources are straightforwardly mapped to naturals and so, in presence of an enumerated value $e$ we consider it as if it was the natural number given by position(e) where position is the position-wise mapping of the enumeration identifiers to naturals.

The resource priority function can be used to change the “importance” given to a particular resource in the computation of the goodness of a given resource set; thus, it provides a flexible way to express a preference on the resources that must be considered.

By looking at the Definition 10 we show how the resource priority affects its calculation:

- If $RP(r) = -1$ then higher values of the resource $r$ contributes 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 $RP(r) = 0$ then the resource $r$ is ignored and does not contribute to raise the goodness of the resource set.
• If \( R_P(r) = 1 \) 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).

By having defined the notion of goodness, we can use it to discriminate resource set according to their goodness value, choosing the ones with higher values.

We point out that the goodness associated to the resource sets does not provide a total ordering. In fact, different resource sets might have the same goodness value. In this case, discrimination among the equally-good resource sets must be delegated to some other mechanism (e.g., random choice, user advice, etc.)

4. RESOURCE SPECIFICATIONS

The resource model permits to express the resources demanded by an adaptable application and the ones supplied by the execution environment, and it enables the customizer to reason on how to adapt the application before delivering it. In order to integrate the resource model to the framework described in Section 2, we introduce three different kinds of descriptions and we use XML [16] files (Figure 7) to specify them. The corresponding XML schemas are available in a separate document [9].

Resource declarations

Resource declaration files contain the declarations of all the resources that are used in the other resource specification files. Such a file is used in order to define resource types, as described in Definition 2. Figure 6 shows a simple resource declaration file, which defines, for example, the resources that are mentioned in the resource set in Figure 2 or 4. By means of resource tags it is possible to create new resources and associate to them a type (e.g., integer), using the type attribute. An enumeration tag allows to define new enumerated types and their different values. Once defined, an enumeration can be used as a resource type in resource declarations, by simply specifying the enumeration id as the resource type.

```
<declaration>
  <enumeration id="res">
    <value id="low"/>
    <value id="medium"/>
    <value id="high"/>
  </enumeration>
  <resource id="Power" type="integer"/>
  <resource id="Bluetooth" type="boolean"/>
  <resource id="3DCaps" type="boolean"/>
  <resource id="ScreenRes" type="res"/>
  <resource id="Thread" type="integer"/>
</declaration>
```

Figure 6: Resources declaration

Resource declarations are kept in separate files in order to make them reusable by the other specifications that are described later.

Resource supplies and demands

Resource supplies and demands specifications are used to describe resource sets (Definition 6). Figure 8 shows the specification of a resource set which models the resources demanded by a particular application adaptation. By using the declaration attribute of the main resources tag it is possible to specify a file containing all the declaration of the resources used in the file itself. The type attribute, on the other hand, allows you to specify if the file is actually describing a resource supply of an execution environment or a resource demand of an application adaptation. The resource tags, finally, allows you to specify resource instances with their types and their values. Figure 8 shows the specification of a resource set like the one in Figure 2 and it represents the resource demanded by some adaptation of an application (i.e., type="demand").

Resource profiles

Resource profiles are the glue that allows to link the resource model to the framework. By using resource profiles, it is possible specify two different aspects:

- **Bindings**: the impact that computational elements (i.e., the Java Bytecode Language constructs) have on the resources.
- **Priorities**: the specification of the resource priority function (Definition 9) that contributes to calculate the goodness of a given resource demand.

Resource profiles are associated to execution environments, specify their characteristics and are used by the framework in order to reason about how to adapt the application with respect to them. Alternatively, we could consider a resource profile as a specification that allows the execution environment to customize the framework in order to match its settings and needs.

```
<profile declaration="res.xml">
  <bind element="render" resource="3DCaps" value="true"/>
  <bind element="getLocalDevice" resource="Bluetooth" value="true"/>
  <bind element="Thread" resource="thread" value="1"/>
  <bind element="new" resource="Memory" value="100"/>
  ...
  <priority resource="Thread" priority="0"/>
  <priority resource="Memory" priority="0"/>
</profile>
```

Figure 7: A resource profile

Figure 7 shows a sample resource profile. The bind tag is used to associate resource information to the machinery used in the Abstract resource analyzer in order to adjust it to
the current execution environment. In particular, the bind tag associates a resource consumption to particular patterns of the Java bytecode language \[7\] and their usage. The element attribute of the bind tag allows the specification and customization of the effects of the computation in the execution environment with respect to the resource demand. Since the bytecode is a verbose language, the element attribute contains a string that is pattern-matched against the symbolic name of the Java bytecode instructions. This allows the resource binding to be associated to both basic instructions (e.g., ipush, iload, etc.) and to complex ones such as method calls. Resource priorities, as already described in Section 3.4, are declared using the priority tag, and assigning to each resource type the desired weights. In figure 7, for example, the Bluetooth resource is bound to the getLocalDevice string that, when the application is analyzed, will match against the getLocalDevice method invocations of the bluetooth library. Of course, the resource value associated to this binding is true, resulting in a resource demand containing the Bluetooth resource.

We point out that resource profiles are not supposed to be exhaustive on all the possible resource types and instructions. There always exists a default resource profile that is present in the abstract resource analyzer and that is (in part) overridden by the provided resource profile. It is clear that the more the resource profile is detailed, the more the resulting resource demand will be accurate. This is a trade-off that is present everywhere in the framework. We use abstraction in order to examine the adaptable application, otherwise it would be impossible to handle the complexity present even in the simplest Java programs. The abstraction provides a reasonable approximation of the resource demand. By using more detailed profiles, this approximation can be reduced.

5. REASONING ON ADAPTATION

Once an adaptable application has been developed and a resource profile has been provided, before the deployment the Abstract resource analyzer uses the information contained in both the adaptable application and the resource profile to extract the resource demand. Actually, since the application might be adapted with respect to the resources supplied by the execution environment, there will be a different resource demand for each possible adaptation (Figure 9). Figure 8 shows the XML file that specifies the resources demanded by a particular adaptation of a given application. Practically a resources supply specification has the same format. The type attribute is used to discriminate their semantics. We point out that the declaration attribute is used to reference a resource declaration specification (Figure 6) that contains all the resource types used in the file.

<resources declaration="res.xml" type="demand">
  <resource id="Power" value="100"/>
  <resource id="Screenres" value="low"/>
  <resource id="3DCaps" value="true"/>
</resources>

Figure 8: A resource demand

Figure 9: Resource specifications and the framework

The resource supply specification provided by the execution environment is used to discriminate the adaptations that are compatible with that supply. The compatibility relation (Definition 8) is used to perform this task. Among all the compatible adaptations, the goodness of each of them is computed by taking into account the priorities specified in the resource profile. The “best” adaptation is then chosen automatically. In Figure 9 this task is carried on by the goodness evaluator, which is a part of the customizer.

However, when several adaptations have an equal goodness value, the choice is delegated to something else. For example, there could be a request to the end user to perform that choice on the basis of relevant differences among the demanded resources of the adaptation. Otherwise, a random choice could be also an acceptable way to discriminate in the case in which the user interaction is not desired, impossible or too expensive.

6. RELATED WORK

The MRG Project \[1\] (now continued in the MoBIUS [2] one) proposes a framework for giving correct guarantees that programs are free from run-time violations of resource bounds. The MRG approach is based on a resource-counting semantics that takes into account, though a resource component, of the number of executed instruction and the maximum size of the stack frame. Our approach is somewhat similar but in MRG they restricted the resource model in order to deal with only a specific set of them. This allows them to be more precise with respect to the properties they prove and the guarantees they ensure but, on the other hand, it doesn’t allow to handle adaptable applications and it makes
harder to accommodate new resource types. Another area where some work has been done in formalizing a resource model is in the embedded systems area. In [14] an abstract resource model is proposed and attached to a process algebra, namely CCS, that is used to actually model the system. This approach is similar to ours, except for the level of abstraction since we reason at the bytecode level and not at the system model level.

We must mention also the resource models that are used both in the Windows Registry [11] or in other platform like Software Dock [3]. These models provide to the application a tree-structured description of what are the characteristics of the hosting environments. This information can be used directly by the application in order to adapt its behavior to the execution environment. This is a similar to our approach however, since we focus on the specific context of limited devices we do not need the complexity of such descriptions. Moreover, the resource model we have proposed does not need to be used on the application side, but can be used only on the framework side. In the case of limited devices it is natural to put all the adaptation logic outside the application. So once deployed, the application does not need to access to such descriptions.

It is also worth mentioning the efforts in formalizing what is known as Service Level Agreement (SLA). WSLA [8] is an effort for specifying all the details that are present in a SLA using an XML format that can be automatically processed. WSLA enables the service provider to specify the policy that will regulate the service supply to the users. By using this format it is possible to specify declaratively the behavior that will be enforced during the service supplying. SLAng [6], is another formalization of service level agreements which uses XML as well but, with respect to WSLA it doesn’t provide policy specification. On this subject it is also worth to mention Tivoli [12] that provides a Resource Modeler tool that enables the specification of resources and allows the automatic monitoring of them by instrumenting the environment in which programs are executed.

Of course we should mention Proof Carrying Code (PCC) [13] that is the root from which a lot of research work has been derived. Even though it doesn’t provide any explicit resource model (it was in case embedded in the abstract machine modeling the execution environment), it has been used as inspiration in several works that dealt with resource oriented issues (i.e., MRG).

All these approaches give a resource model as we do, but differently from us, they do not use these models in the context of adaptation and, moreover, they do not provide the flexibility for reasoning on the resources in the perspective of program adaptation.

7. CONCLUSION

In this paper we have presented a flexible resource model that has been used as the basis for our adaptation framework. The resource model has been conceived by having in mind adaptable applications, in a setting where these application can be exploited in order to face the issues that arise in a resource-restricted execution environment. The resource model and its realization through the different resource specification file types is the way we declaratively specify information for adapting application to resource constrained environments. The notion of compatibility and goodness, finally, enables the framework to both discriminate correct adaptation and to choose the best one by taking into account the user preferences: sometimes what is “good” from the user perspective could not be as “good” from the execution device perspective. As a future work, the resource specifications could be refined in order to allow the execution environment to specify more expressive bindings (e.g., bindings matched against code patterns) and to express resource declarations in a more object-oriented fashion, in order to reflect the natural hierarchy that is present in the actual resource domain.

8. REFERENCES


